Studies in Computational Intelligence 516

Xin-She Yang *Editor*

Cuckoo Search and Firefly Algorithm

Theory and Applications

Springer

Studies in Computational Intelligence

Volume 516

Series Editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland
e-mail: kacprzyk@ibspan.waw.pl

For further volumes:
http://www.springer.com/series/7092

About this Series

The series "Studies in Computational Intelligence" (SCI) publishes new developments and advances in the various areas of computational intelligence—quickly and with a high quality. The intent is to cover the theory, applications, and design methods of computational intelligence, as embedded in the fields of engineering, computer science, physics and life sciences, as well as the methodologies behind them. The series contains monographs, lecture notes and edited volumes in computational intelligence spanning the areas of neural networks, connectionist systems, genetic algorithms, evolutionary computation, artificial intelligence, cellular automata, self-organizing systems, soft computing, fuzzy systems, and hybrid intelligent systems. Of particular value to both the contributors and the readership are the short publication timeframe and the world-wide distribution, which enable both wide and rapid dissemination of research output.

Xin-She Yang
Editor

Cuckoo Search and Firefly Algorithm

Theory and Applications

Springer

Editor
Xin-She Yang
School of Science and Technology
Middlesex University
London
UK

ISSN 1860-949X ISSN 1860-9503 (electronic)
ISBN 978-3-319-02140-9 ISBN 978-3-319-02141-6 (eBook)
DOI 10.1007/978-3-319-02141-6
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013953202

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Many modelling and optimization problems require sophisticated algorithms to solve. Contemporary optimization algorithms are often nature-inspired, based on swarm intelligence. In the last two decades, there have been significant developments in the area of metaheuristic optimization and computational intelligence. Optimization and computational intelligence have become ever-increasingly more important. One of the core activities of the computational intelligence is that "intelligent" evolutionary algorithms play a vital role. Accompanying the progress of computational intelligence is the emergence of metaheuristic algorithms. Among such algorithms, swarm-intelligence-based algorithms form a large part of contemporary algorithms, and these algorithms are becoming widely used in classifications, optimization, image processing, business intelligence as well as in machine learning and computational intelligence.

Most new nature-inspired optimization algorithms are swarm-intelligence-based, with multiple interacting agents. They are flexible, efficient and easy to implement. For example, firefly algorithm (FA) was developed in late 2007 and early 2008 by Xin-She Yang, based on the flashing behaviour of tropical fireflies, and FA has been proved to be very efficient in solving multimodal, nonlinear, global optimization problems. It is also very efficient in dealing with classification problems and image processing. As another example, cuckoo search (CS) was developed by Xin-She Yang and Suash Deb in 2009, based on the brooding parasitism of some cuckoo species, in combination with Lévy flights, and CS is very efficient as demonstrated in many studies by many researchers with diverse applications. In fact, at the time of the writing in July 2013, there have been more than 440 research papers on cuckoo search and 600 pagers on firefly algorithm in the literature, which shows that these algorithms are indeed an active, hot research area.

This book strives to provide a timely summary of the latest developments concerning cuckoo search and firefly algorithm with many contributions from leading experts in the field. Topics include cuckoo search, firefly algorithm, classifications, scheduling, feature selection, travelling salesman problem, neural network training, semantic web service, multi-objective manufacturing process optimization, parameter-tuning, queuing, randomization, reliability problem, GPU optimization, shape optimization and others. This unique book can thus serve as an ideal reference for both graduates and researchers in computer science, evolutionary computing, machine learning, computational intelligence and optimization,

as well as engineers in business intelligence, knowledge management and information technology.

I would like to thank our Editors, Drs. Thomas Ditzinger and Holger Schaepe, and staff at Springer for their help and professionalism. Last but not least, I thank my family for the help and support.

London, July 2013 Xin-She Yang

Contents

Contributors

Abdul Hanan Abdullah Faculty of Computing, Universiti Teknologi Malaysia, Skudai, Malaysia

B. Ahiod LRIT, Associated Unit to the CNRST (URAC 29), Mohammed V-Agdal University, Rabat, Morocco

Tiago. N. S. Almeida Department of Computing, UNESP, Univ Estadual Paulista, Bauru, SP, Brazil

Jose Ambrosio Intelligent Systems Group, Universidad La Salle, Col. Hipódromo Condesa, Mexico

Mohammed Bakri Bashir Faculty of Computing, Universiti Teknologi Malaysia, Skudai, Malaysia

Erkan Besdok Faculty of Engineering, Department of Geomatic Engineering, Erciyes University, Kayseri, Turkey

Ivona Brajevic University of Belgrade, Belgrade, Serbia

Adrián Bonilla-Petriciolet Department of Chemical Engineering, Instituto Tecnológico de Aguascalientes, Aguascalientes, México

Janez Brest Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia

Viorica Rozina Chifu Computer Science Department, Technical University of Cluj-Napoca, Cluj-Napoca, Romania

Pinar Civicioglu Department of Aircraft Electrics and Electronics, College of Aviation, Erciyes University, Kayseri, Turkey

Adelano Esposito Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil

Seif-Eddeen K. Fateen Department of Chemical Engineering, Cairo University, Giza, Egypt

Bogusław Filipowicz AGH University of Science and Technology, Krakow, Poland

Dušan Fister Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia

Iztok Fister Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia

Iztok Fister Jr. Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia

Herbert Martins Gomes Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil

K. A. Hawick Department of Computer Science, Massey University, Auckland, New Zealand

A. V. Husselmann Department of Computer Science, Massey University, Auckland, New Zealand

Joanna Kwiecień AGH University of Science and Technology, Krakow, Poland

Sulaiman Mohd Nor Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Skudai, Malaysia

Aziz Ouaarab LRIT, Associated Unit to the CNRST (URAC 29), Mohammed V-Agdal University, Rabat, Morocco

João Paulo Papa Department of Computing, UNESP, Univ Estadual Paulista, Bauru, SP, Brazil

L. A. M. Pereira Department of Computing, UNESP, Univ Estadual Paulista, Bauru, SP, Brazil

Cristina Bianca Pop Computer Science Department, Technical University of Cluj-Napoca, Cluj-Napoca, Romania

Douglas Rodrigues Department of Computing, UNESP, Univ Estadual Paulista, Bauru, SP, Brazil

Caio C. O. Ramos Department of Computing, UNESP, Univ Estadual Paulista, Bauru, SP, Brazil

Ioan Salomie Computer Science Department, Technical University of Cluj-Napoca, Cluj-Napoca, Romania

Guillermo Sandoval Intelligent Systems Group, Universidad La Salle, Col. Hipódromo Condesa, Mexico

André N. Souza Department of Computing, UNESP, Univ Estadual Paulista, Bauru, SP, Brazil

Anna Syberfeldt University of Skövde, Skövde, Sweden

Milan Tuba Megatrend University Belgrade, Belgrade, Serbia

Ehsan Valian Faculty of Electrical and Computer Engineering, University of Sistan and Baluchestan, Sistan and Baluchestan, Iran

Roberto A. Vazquez Intelligent Systems Group, Universidad La Salle, Col. Hipódromo Condesa, Mexico

Xin-She Yang School of Science and Technology, Middlesex University, London, UK

Adil Yousif Faculty of Computing, Universiti Teknologi Malaysia, Skudai, Malaysia

Cuckoo Search and Firefly Algorithm: Overview and Analysis

Xin-She Yang

Abstract Firefly algorithm (FA) was developed by Xin-She Yang in 2008, while cuckoo search (CS) was developed by Xin-She Yang and Suash Deb in 2009. Both algorithms have been found to be very efficient in solving global optimization problems. This chapter provides an overview of both cuckoo search and firefly algorithm as well as their latest developments and applications. We analyze these algorithms and gain insight into their search mechanisms and find out why they are efficient. We also discuss the essence of algorithms and its link to self-organizing systems. In addition, we also discuss important issues such as parameter tuning and parameter control, and provide some topics for further research.

Keywords Algorithm · Cuckoo search · Firefly algorithm · Metaheuristic · Optimization · Self-organization

1 Introduction

Optimization and computational intelligence are active research areas with rapidly expanding literature. For most applications, time, money and resources are always limited, and thus their optimal use becomes increasingly important. In modern design applications, it requires a paradigm shift in thinking and design to find energy-saving and greener design solutions. However, to obtain optimal solutions to design problems are non-trivial, and many real-world optimization problems can be really hard to solve. For example, it is well-known that combinatorial optimization problems such as the travelling salesman problem (TSP) are NP-hard, which means that there are no efficient algorithms in general.

X.-S. Yang (✉)
School of Science and Technology, Middlesex University, London NW4 4BT, UK
e-mail: xy227@cam.ac.uk; x.yang@mdx.ac.uk

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_1,

Even without efficient algorithms, we still have to solve these challenging problems in practice. This leads to the development of various exotic techniques and problem-specific methods so that problem-specific knowledge (such as gradients) can be used to guide better search procedures. In general, algorithms that use problem-related knowledge should perform better than black-box-type methods that do not use problem knowledge at all. But the incorporation of problem-specific knowledge often limits the use of a method or an algorithm. In addition, it is not easy or too computationally extensive to incorporate such knowledge. Sometimes, it may be impossible to incorporate such knowledge, either because we may not have any insight into the problem or because we do not know how to. In this case, the only option is to use black-box-type algorithms that do not assume any knowledge of the problem of interest.

In most cases, for NP-hard problems, the only alternative is to use heuristic or metaheuristic methods by trial and error. Heuristic methods are to search for solutions by trial and error, while metaheuristic methods can be considered as higher-level heuristic methods that use information and selection of the solutions to guide the search process. Optimization algorithms are the tools and techniques for solving optimization problems with the aim to find its optimality, though such optimality is not always reachable. This search for optimality is complicated further by the fact that uncertainty is almost always present in the real-world systems. Therefore, we seek not only the optimal design but also robust design in engineering and industry. Optimal solutions, which are not robust enough, are not practical in reality. Suboptimal solutions or good robust solutions are often the choice in such cases. In the last twenty years, nature-inspired metaheuristic algorithms have gained huge popularity in optimization, artificial intelligence, machine learning, computational intelligence, data mining and engineering applications [40, 65, 78].

The aim of this chapter is to review both cuckoo search and firefly algorithm and their applications. Therefore, the chapter is organized as follows: Sect. 2 outlines briefly the basic formulation of optimization problems, while Sect. 3 discusses the essence of an optimization algorithm. Section 4 provides a detailed introduction to cuckoo search, and Sect. 5 introduces the firefly algorithm in greater detail. Section 6 highlights the importance of the right amount of randomization, while Sect. 7 touches the challenging issues of parameter tuning and parameter control. Finally, Sect. 8 draw conclusions briefly.

2 Optimization Problems

Optimization problems can be formulated in many ways. For example, the commonly used method of least-squares is a special case of maximum-likelihood formulations. By far the most widely used formulation is to write a nonlinear optimization problem as

$$\text{minimize} \quad f_m(\boldsymbol{x}), (m = 1, 2, ..., M), \tag{1}$$

subject to the constraints

$$h_j(\boldsymbol{x}) = 0, \quad (j = 1, 2, ..., J), \tag{2}$$

$$g_k(\boldsymbol{x}) \leq 0, \quad (k = 1, 2, ..., K), \tag{3}$$

where f_m, h_j and g_k are in general nonlinear functions. Here the design vector or design variables $\boldsymbol{x} = (x_1, x_2, ..., x_d)$ can be continuous, discrete or mixed in d-dimensional space. The functions f_m are called objective or cost functions, and when $M > 1$, the optimization problem is called multiobjective or multicriteria [65].

It is worth pointing out that here we write the problem as minimization, it can also be written as maximization by simply replacing $f_m(\boldsymbol{x})$ by $-f_m(\boldsymbol{x})$. When all functions are nonlinear, we are dealing with nonlinear constrained problems. In some special cases when f_m, h_j and g_k are linear, the problem becomes linear, and we can use the widely linear programming techniques such as the simplex method. When some design variables can only take discrete values (often integers), while other variables are real continuous, the problem is of mixed type, which is often difficult to solve, especially for large-scale optimization problems.

In general, multiobjective optimization requires multiobjective techniques to obtain the Pareto fronts of a problem of interest. However, it is possible to combine different objectives into a single objective by using weighted sum

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m f_m, \quad \alpha_m > 0, \quad \sum_{m=1}^{M} \alpha_m = 1. \tag{4}$$

This is a simple approach that allows us to use algorithms for single-objective optimization without major modifications. However, this approach does have some disadvantages as it can only deal with convex Pareto fronts. In addition, true multiobjective approaches to multiobjective optimization can give far more information and insight into the problem.

3 The Essence of an Optimization Algorithm

Before we go any further to discuss nature-inspired algorithms such as cuckoo search and firefly algorithm, let us analyze the essence of an optimization algorithm.

In essence, optimization is a process of searching for the optimal solutions to a particular problem of interest, and this search process can be carried out using multiple agents which essentially form a system of evolving agents. This system can evolve by iterations according to a set of rules or mathematical equations. Consequently, such a system will show some emergent characteristics, leading to self-organizing states which correspond to some optima of the objective landscape. Once the self-organized states are reached, we say the system converges. Therefore, to design

an efficient optimization algorithm is equivalent to mimicking the evolution of a self-organizing system [4, 38].

3.1 The Essence of an Algorithm

Mathematically speaking, an algorithm is a procedure to generate outputs for a given set of inputs. From the optimization point of view, an optimization algorithm generates a new solution $\boldsymbol{x}^{t+1}$ to a given problem from a know solution $\boldsymbol{x}^t$ at iteration or time t. That is

$$\boldsymbol{x}^{t+1} = \boldsymbol{A}(\boldsymbol{x}^t, \boldsymbol{p}(t)), \tag{5}$$

where A is a nonlinear mapping from a given solution (i.e., a d-dimensional vector) $\boldsymbol{x}^t$ to a new solution vector $\boldsymbol{x}^{t+1}$. The algorithm A has k algorithm-dependent parameters $p(t) = (p_1, ..., p_k)$ which can be time-dependent and can thus be tuned.

To find the optimal solution $\boldsymbol{x}_*$ to a given optimization problem S with an often infinitely number of states is to select some desired states ϕ from all states ψ, according to some predefined criterion D. We have

$$S(\psi) \xrightarrow{\boldsymbol{A}(t)} S(\phi(x_*)), \tag{6}$$

where the final converged state ϕ corresponds to an optimal solution $\boldsymbol{x}_*$ of the problem of interest. However, the converged states can be multiple, and each corresponds to a local optimum. Among all the optima, there should be the global optimality. This self-organization process does not give an indication that it will converge to the global optimal solution. The aim is to tune the algorithm and control its behaviour so that true global optima (states) can be selected out of many possible states.

The selection of the system states in the design space is carried out by running the optimization algorithm $\boldsymbol{A}$. The behavior of the algorithm is controlled by the parameters $\boldsymbol{p}$, the initial solution $\boldsymbol{x}^{t=0}$ and the stopping criterion D. We can view the combined $S + \boldsymbol{A}(t)$ as a complex system with a self-organizing capability.

The change of states or solutions of the problem of interest is through the algorithm $\boldsymbol{A}$. In many classical algorithms such as hill-climbing, gradient information of the problem S is used so as to select states, say, the minimum value of the landscape, and the stopping criterion can be a given tolerance or accuracy, or zero gradient, etc.

An algorithm can act like a tool to tune a complex system. If an algorithm does not use any state information of the problem, then the algorithm is more likely to be versatile to deal with many types of problems. However, such black-box approaches can also imply that the algorithm may not be efficient as it could be for a given type of problem. For example, if the optimization problem is convex, algorithms that use such convexity information will be more efficient than the ones that do not use such information. In order to be efficient to select states/solutions efficiently, the information from the search process should be used to enhance the search process. In many case, such information is often fed into the selection mechanism of an

algorithm. By far the most widely used selection mechanism to select or keep the best solution found so far. That is, the simplest form of 'survival of the fittest'.

From the schematic representation (6) of the optimization process, we can see that the performance of an algorithm may also depend on the type of problem S it solves. On the other hand, the final, global optimality is achievable or not (within a given number of iterations) will also depend on the algorithm used. This may be another way of stating the so-called no-free-lunch theorems.

Optimization algorithms can very diverse with several dozen widely used algorithms. The main characteristics of different algorithms will only depend on the actual, nonlinear, often implicit form of $\boldsymbol{A}(t)$ and its parameters $\boldsymbol{p}(t)$.

3.2 An Ideal Algorithm?

The number of iterations t needed to find an optimal solution for a given accuracy largely determines the overall computational efforts and the performance of an algorithm. A better algorithm should use less computation and fewer iterations.

In an extreme case for an iterative algorithm or formula (5), an ideal algorithm should only take one iteration $t = 1$. You may wonder if such an algorithm exists in practice. The answer is "yes, it depends". In fact, for a quadratic function such as $f(x) = ax^2$ where $a > 0$, the well-know Newton-Raphson method

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \tag{7}$$

is the ideal algorithm. If we start the iteration from any random guess $x_0 = b$, we have

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)} = b - \frac{2ab}{2a} = 0, \tag{8}$$

which gives the global optimal solution $f_*(0) = 0$ at $x_* = 0$.

Obviously, we can extend this to all class of quadratic functions. Even it is possible to extend to more generalized convex functions. However, many problems are not convex, and certainly not quadratic. Therefore, the so-called ideal algorithm does not exist in general. As we have mentioned earlier, there is no good algorithm for solving NP-hard problems.

There are many optimization algorithms in the literature and no single algorithm is suitable for all problems [64]. Even so, the search for efficient algorithm still forms the major efforts among researchers, and this search for the 'Holy Grail' continues, unless some proves analytically otherwise.

3.3 Metaheuristic Algorithms

Algorithms can be classified as deterministic or stochastic. If an algorithm works in a mechanically deterministic manner without any random nature, it is called deterministic. For such an algorithm, it will reach the same final solution if we start with the same initial point. Hill-climbing and downhill simplex are good examples of deterministic algorithms. On the other hand, if there is some randomness in the algorithm, the algorithm will usually reach a different point every time we run the algorithm, even though we start with the same initial point. Genetic algorithms and hill-climbing with a random restart are good examples of stochastic algorithms.

Analyzing current metaheuristic algorithms in more detail, we can single out the type of randomness that a particular algorithm is employing. For example, the simplest and yet often very efficient method is to introduce a random starting point for a deterministic algorithm. The well-known hill-climbing with random restart is a good example. This simple strategy is both efficient in most cases and easy to implement in practice. A more elaborate way to introduce randomness to an algorithm is to use randomness inside different components of an algorithm, and in this case, we often call such algorithms heuristic or more often metaheuristic [65, 70].

Metaheuristic algorithms are often nature-inspired, and they are now among the most widely used algorithms for optimization. They have many advantages over conventional algorithms [28, 65]. Metaheuristic algorithms are very diverse, including genetic algorithms, simulated annealing, differential evolution, ant and bee algorithms, bat algorithm, particle swarm optimization, harmony search, firefly algorithm, flower pollination algorithm, cuckoo search and others [29, 39, 70, 73–75, 82]. Here we will introduce cuckoo search and firefly algorithm in great detail.

4 Cuckoo Search and Analysis

4.1 Cuckoo Search

Cuckoo search (CS) is one of the latest nature-inspired metaheuristic algorithms, developed in 2009 by Xin-She Yang and Suash Deb [76, 80, 81]. CS is based on the brood parasitism of some cuckoo species. In addition, this algorithm is enhanced by the so-called Lévy flights [48], rather than by simple isotropic random walks. Recent studies show that CS is potentially far more efficient than PSO and genetic algorithms [76]. Cuckoo are fascinating birds, not only because of the beautiful sound they can make, but also because of their aggressive reproduction strategy. Some species such as the *ani* and *Guira* cuckoos lay their eggs in communal nests, though they may remove others' eggs to increase the hatching probability of their own eggs. Quite a number of species engage the obligate brood parasitism by laying their eggs in the nests of other host birds (often other species).

For simplicity in describing the standard Cuckoo Search, we now use the following three idealized rules:

- Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest;
- The best nests with high-quality eggs will be carried over to the next generations;
- The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in (0, 1)$. In this case, the host bird can either get rid of the egg, or simply abandon the nest and build a completely new nest.

As a further approximation, this last assumption can be approximated by replacing a fraction p_a of the n host nests with new nests (with new random solutions). For a maximization problem, the quality or fitness of a solution can simply be proportional to the value of the objective function. Other forms of fitness can be defined in a similar way to the fitness function in genetic algorithms.

From the implementation point of view, we can use the following simple representations that each egg in a nest represents a solution, and each cuckoo can lay only one egg (thus representing one solution), the aim is to use the new and potentially better solutions (cuckoos) to replace a not-so-good solution in the nests. Obviously, this algorithm can be extended to the more complicated case where each nest has multiple eggs representing a set of solutions. Here, we will use the simplest approach where each nest has only a single egg. In this case, there is no distinction between an egg, a nest or a cuckoo, as each nest corresponds to one egg which also represents one cuckoo.

This algorithm uses a balanced combination of a local random walk and the global explorative random walk, controlled by a switching parameter p_a. The local random walk can be written as

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \alpha s \otimes H(p_a - \epsilon) \otimes (\boldsymbol{x}_j^t - \boldsymbol{x}_k^t), \tag{9}$$

where $\boldsymbol{x}_j^t$ and $\boldsymbol{x}_k^t$ are two different solutions selected randomly by random permutation, $H(u)$ is a Heaviside function, ϵ is a random number drawn from a uniform distribution, and s is the step size. On the other hand, the global random walk is carried out by using Lévy flights

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \alpha L(s, \lambda), \tag{10}$$

where

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, \quad (s \gg s_0 > 0). \tag{11}$$

Here $\alpha > 0$ is the step size scaling factor, which should be related to the scales of the problem of interest. In most cases, we can use $\alpha = O(L/10)$, where L is the characteristic scale of the problem of interest, while in some cases $\alpha = O(L/100)$ can be more effective and avoid flying too far. Obviously, the α value in these two updating

equations can be different, thus leading to two different parameters α_1 and α_2. Here, we use $\alpha_1 = \alpha_2 = \alpha$ for simplicity.

The above equation is essentially the stochastic equation for a random walk. In general, a random walk is a Markov chain whose next state/location only depends on the current location (the first term in the above equation) and the transition probability (the second term). However, a substantial fraction of the new solutions should be generated by far field randomization and their locations should be far enough from the current best solution; this will make sure that the system will not be trapped in a local optimum [76, 80].

The literature on cuckoo search is expanding rapidly. It has received a lot of attention and there are many recent studies using cuckoo search with a diverse range of applications [14, 15, 19–21, 28, 37, 83]. For example, Walton et al. improved the algorithm by formulating a modified cuckoo search algorithm [62], while Yang and Deb extended it to multiobjective optimization [81].

4.2 Special Cases of Cuckoo Search

Cuckoo search as a metaheuristic algorithm has surprisingly rich characteristics. If we look at the updating Eqs. (9) and (10) more close, we can discover such subtle richness. From (9), we can group some factors together by setting $Q = \alpha s \otimes H(p_a - \epsilon)$, then we have $Q > 0$. As a result, equation (9) becomes the major updating equation of differential evolution (DE). Furthermore, we further replace $\boldsymbol{x}_j^t$ by the current best g^* and set $k = i$, we have

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + Q(g^* - \boldsymbol{x}_i^t), \tag{12}$$

which is essential a variant of the particle swarm optimization without individual historical best. In this case, the above case is very similar to the accelerated particle particle swarm optimization (APSO) developed by Yang et al. [78].

On the other hand, from (10), this random walk is in fact the simulated annealing (SA) with a Lévy-flight transition probability. In this case, we have a simulated annealing with a stochastic cooling scheduling controlled by p_a.

Therefore, differential evolution, particle swarm optimization and simulated annealing can be considered as special cases of cuckoo search. Conversely, we can say that cuckoo search is a good and efficient combination of DE, PSO and SA in one algorithm. Therefore, it is no surprise that cuckoo search is very efficient.

4.3 Why Cuckoo Search is so Efficient?

In addition to the analysis of the previous section showing that DE, PSO and SA are specially cases of cuckoo search, recent theoretical studies also indicate that cuckoo search has global convergence [63], which will be outlined in the next subsection.

Theoretical studies of particle swarm optimization have suggested that PSO can converge quickly to the current best solution, but not necessarily the global best solutions [16, 36, 63]. In fact, some analyses suggest that PSO updating equations do not satisfy the global convergence conditions, and thus there is no guarantee for global convergence. On the other hand, it has proved that cuckoo search satisfy the global convergence requirements and thus has guaranteed global convergence properties [63]. This implies that for multimodal optimization, PSO may converge prematurely to a local optimum, while cuckoo search can usually converge to the global optimality.

Furthermore, cuckoo search has two search capabilities: local search and global search, controlled by a switching/discovery probability. As mentioned in Sect. 3.1, the local search is very intensive with about 1/4 of the search time (for $p_a = 0.25$), while global search takes about 3/4 of the total search time. This allows that the search space can be explored more efficiently on the global scale, and consequently the global optimality can be found with a higher probability.

A further advantage of cuckoo search is that its global search uses Lévy flights or process, rather than standard random walks. As Lévy flights have infinite mean and variance, CS can explore the search space more efficiently than algorithms using standard Gaussian processes. This advantage, combined with both local and search capabilities and guaranteed global convergence, makes cuckoo search very efficient. Indeed, various studies and applications have demonstrated that cuckoo search is very efficient [17, 28, 29, 56, 62, 80].

4.4 Global Convergence: Brief Mathematical Analysis

Wang et al. provided a mathematical proof of global convergence for the standard cuckoo search, and their approach is based on the Markov chain theory [63]. Their proof can be outlined as follows:

As there are two branches in the updating formulas, the local search step only contributes mainly to local refinements, while the main mobility or exploration is carried out by the global search step. In order to simplify the analysis and also to emphasize the global search capability, we now use a simplified version of cuckoo search. That is, we use only the global branch with a random number $r \in [0, 1]$, compared with a discovery/switching probability p_a. Now we have

$$\begin{cases} \boldsymbol{x}_i^{(t+1)} \leftarrow x_i^{(t)} & \text{if } r < p_a, \\ \boldsymbol{x}_i^{(t+1)} \leftarrow \boldsymbol{x}_i^{(t)} + \alpha \otimes L(s, \lambda) & \text{if } r > p_a. \end{cases} \tag{13}$$

As our cuckoo search algorithm is a stochastic search algorithm, we can summarize it as the following key steps:

1. Randomly generate an initial population of n nests at the positions, $\boldsymbol{X} = \{\boldsymbol{x}_1^0, \boldsymbol{x}_2^0, ..., \boldsymbol{x}_n^0\}$, then evaluate their objective values so as to find the current global best g_t^0.
2. Update the new solutions/positions by

$$\boldsymbol{x}_i^{(t+1)} = \boldsymbol{x}_i^{(t)} + \alpha \otimes L(\lambda). \tag{14}$$

3. Draw a random number r from a uniform distribution [0, 1]. Update $\boldsymbol{x}_i^{(t+1)}$ if $r > p_a$. Then, evaluate the new solutions so as to find the new, global best g_t^*.
4. If the stopping criterion is met, then g_t^* is the best global solution found so far. Otherwise, return to step (2).

The global convergence of an algorithm. If f is measurable and the feasible solution space Ω is a measurable subset on $\Re^n$, algorithm A satisfies the above two conditions with the search sequence $\{x_k\}_{k=0}^{\infty}$, then

$$\lim_{k\to\infty} P(x_k \in R_{\epsilon,M}) = 1. \tag{15}$$

That is, algorithm A can converge globally. Here $P(x_k \in R_{\epsilon,M})$ is the probability measure of the kth solution on $R_{\epsilon,M}$ at the kth iteration.

The state and state space. The positions of a cuckoo/nest and its global best solution g in the search history forms the states of cuckoos: $y = (x, g)$, where $x, g \in \Omega$ and $f(g) \leq f(x)$. The set of all the possible states forms the state space, denoted by

$$Y = \{y = (x, g) | x, g \in \Omega, f(g) \leq f(x)\}. \tag{16}$$

The states and state space of the cuckoo group/population. The states of all n cuckoos/nests form the states of the group, denoted by $q = (y_1, y_2, ..., y_n)$. All the states of all the cuckoos form a state space for the group, denoted by

$$Q = \{q = (y_1, y_2, ..., y_n), y_i \in Y, 1 \leq i \leq n\}. \tag{17}$$

Obviously, Q contains the historical global best solution g^* for the whole population and all individual best solutions $g_i (1 \leq i \leq n)$ in history. In addition, the global best solution of the whole population is the best among all g_i, so that $f(g^*) = \min(f(g_i)),\ 1 \leq i \leq n$.

The transition probability from state y_1 to y_2 in cuckoo search is

$$P(T_y(y_1) = y_2) = P(x_1 \to x_1') P(g_1 \to g_1') P(x_1' \to x_2) P(g_1' \to g_2), \tag{18}$$

where $P(x_1 \to x_1')$ is the transition probability at Step 2 in cuckoo search, and $P(g_1 \to g_1')$ is the transition probability for the historical global best at this step. $P(x_1' \to x_2)$ is the transition probability at Step 3, while $P(g_1' \to g_2)$ is the transition probability of the historical global best.

For globally optimal solution g_b for an optimization problem $< \Omega, f >$, the optimal state set is defined as $R = \{y = (x, g) | f(g) = f(g_b), y \in Y\}$.

For the globally optimal solution g_b to an optimization problem $< \Omega, f >$, the optimal group state set can be defined as

$$H = \{q = (y_1, y_2, ..., y_n) | \exists y_i \in R, 1 \leq i \leq n\}. \tag{19}$$

Further detailed mathematical analysis proves that when the number of iteration approaches sufficiently large, the group state sequence will converge to the optimal state/solution set H. Therefore, the cuckoo search has guaranteed global convergence.

4.5 Applications

Cuckoo search has been applied in many areas of optimization and computational intelligence with promising efficiency. For example, in the engineering design applications, cuckoo search has superior performance over other algorithms for a range of continuous optimization problems such as spring design and welded beam design problems [28, 29, 80].

In addition, a modifed cuckoo search by Walton et al. [62] has demonstrated to be very efficient for solving nonlinear problems such as mesh generation. Vazquez [61] used cuckoo search to train spiking neural network models, while Chifu et al. [14] optimized semantic web service composition processes using cuckoo search. Furthermore, Kumar and Chakarverty [41] achieved optimal design for a reliable embedded system using cuckoo search, and Kaveh and Bakhshpoori [37] used CS to successfully design steel frames. Yildiz [83] has used CS to select optimal machine parameters in milling operation with enhanced results, and while Zheng and Zhou [86] provided a variant of cuckoo search using Gaussian process.

On the other hand, a discrete cuckoo search algorithm has been proposed by Tein and Ramli [58] to solve nurse scheduling problems. Cuckoo search has also been used to generate independent paths for software testing and test data generation [15, 49, 56]. In the context of data fussion and wireless sensor network, cuckoo search has been shown to be very efficient [19, 20]. Furthermore, a variant of cuckoo search in combination with quantum-based approach has been developed to solve Knapsack problems efficiently [42]. From the algorithm analysis point of view, a conceptural comparison of cuckoo search with particle swarm optimization (PSO), differential evolution (DE), artificial bee colony (ABC) by Civicioglu and Desdo [17] suggested that cuckoo search and differential evoluton algorithms provide more robust results than PSO and ABC. Gandomi et al. [28] provided a more extensive comparison study for solving various sets of structural optimization problems and concluded that cuckoo search obtained better results than other algorithms such as PSO and gentic algorithms (GA). Speed [55] modified the Lévy cuckoo search and shown that CS can deal with very large-scale problems. Among the diverse

applications, an interesting performance enhancement has been obtained by using cuckoo search to train neural networks as shown by Valian et al. [59] and reliability optimization problems [60].

For complex phase equilibrium applications, Bhargava et al. [9] have shown that cuckoo search offers a reliable method for solving thermodynamic calculations. At the same time, Bulatović et al. [11] have solved a six-bar double dwell linkage problem using cuckoo search, and Moravej and Akhlaghi [43] have solved the DG allocation problem in distribution networks with good convergence rate and performance. Taweewat and Wutiwiwatchi have combined cuckoo search and supervised neural network to estimate musical pitch with reduced size and higher accuracy [57].

As a further extension, Yang and Deb [81] produced the multiobjective cuckoo search (MOCS) for design engineering appplications. For multiobjective scheduling problems, very progress was made by Chandrasekaran and Simon [12] using cuckoo search algorithm, which demonstrated the superiority of their proposed methodology. Recent studies have demonstrated that cuckoo search can performance significantly better than other algorithms in many applications [28, 45, 83, 86].

5 Firefly Algorithm and Analysis

5.1 Firefly Algorithm

Firefly Algorithm (FA) was first developed by Xin-She Yang in late 2007 and 2008 at Cambridge University [65, 70], which was based on the flashing patterns and behaviour of fireflies. The FA literature has expanded significantly in the last 5 years with several hundred papers published about the firefly algorithms, and Fister et al. provided a comprehensive review [27]. In essence, FA uses the following three idealized rules:

- Fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex.
- The attractiveness is proportional to the brightness, and they both decrease as their distance increases. Thus for any two flashing fireflies, the less brighter one will move towards the brighter one. If there is no brighter one than a particular firefly, it will move randomly.
- The brightness of a firefly is determined by the landscape of the objective function.

As a firefly's attractiveness is proportional to the light intensity seen by adjacent fireflies, we can now define the variation of attractiveness β with the distance r by

$$\beta = \beta_0 e^{-\gamma r^2}, \tag{20}$$

where β_0 is the attractiveness at $r = 0$.

The movement of a firefly i is attracted to another more attractive (brighter) firefly j is determined by

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2}(\boldsymbol{x}_j^t - \boldsymbol{x}_i^t) + \alpha_t\, \epsilon_i^t, \tag{21}$$

where the second term is due to the attraction. The third term is randomization with α_t being the randomization parameter, and ϵ_i^t is a vector of random numbers drawn from a Gaussian distribution or uniform distribution at time t. If $\beta_0 = 0$, it becomes a simple random walk. On the other hand, if $\gamma = 0$, FA reduces to a variant of particle swarm optimisation [65]. Furthermore, the randomization $\boldsymbol{\epsilon}_i^t$ can easily be extended to other distributions such as Lévy flights [65]. A demo version of firefly algorithm implementation by Xin-She Yang, without Lévy flights for simplicity, can be found at Mathworks file exchange web site.[1]

5.2 Parameter Settings

As α_t essentially controls the randomness (or, to some extent, the diversity of solutions), we can tune this parameter during iterations so that it can vary with the iteration counter t. So a good way to express α_t is to use

$$\alpha_t = \alpha_0 \delta^t, \quad 0 < \delta < 1, \tag{22}$$

where α_0 is the initial randomness scaling factor, and δ is essentially a cooling factor. For most applications, we can use $\delta = 0.95$ to 0.97 [65].

Regarding the initial α_0, simulations show that FA will be more efficient if α_0 is associated with the scalings of design variables. Let L be the average scale of the problem of interest, we can set $\alpha_0 = 0.01L$ initially. The factor 0.01 comes from the fact that random walks requires a number of steps to reach the target while balancing the local exploitation without jumping too far in a few steps [67, 70]. The parameter β controls the attractiveness, and parametric studies suggest that $\beta_0 = 1$ can be used for most applications. However, γ should be also related to the scaling L. In general, we can set $\gamma = 1/\sqrt{L}$. If the scaling variations are not significant, then we can set $\gamma = O(1)$.

For most applications, we can use the population size $n = 15$ to 100, though the best range is $n = 25$ to 40 [65, 70].

[1] http://www.mathworks.com/matlabcentral/fileexchange/29693-firefly-algorithm

5.3 Algorithm Complexity

Almost all metaheuristic algorithms are simple in terms of complexity, and thus they are easy to implement. Firefly algorithm has two inner loops when going through the population n, and one outer loop for iteration t. So the complexity at the extreme case is $O(n^2 t)$. As n is small (typically, $n = 40$), and t is large (say, $t = 5000$), the computation cost is relatively inexpensive because the algorithm complexity is linear in terms of t. The main computational cost will be in the evaluations of objective functions, especially for external black-box type objectives. This latter case is also true for all metaheuristic algorithms. After all, for all optimisation problems, the most computationally extensive part is objective evaluations.

If n is relatively large, it is possible to use one inner loop by ranking the attractiveness or brightness of all fireflies using sorting algorithms. In this case, the algorithm complexity of firefly algorithm will be $O(nt \log(n))$.

5.4 Special Cases of FA

Firefly algorithm is indeed rich in many ways. First, it uses attraction to influence the behavior of the population. As local attraction tends to be stronger than long-distance attraction, the population in FA can automatically subdivide into subgroups, depending on the modality of the problem, which enables FA to deal with multimodal, nonlinear optimization problems naturally.

Furthermore, we look at the updating equation (21) more closely, this nonlinear equation provides much richer characteristics. Firstly, if γ is very large, then attractiveness or light intensity decreases too quickly, this means that the second term in (21) becomes negligible, leading to the standard simulated annealing (SA). Secondly, if γ is very small (i.e., $\gamma \to 0$), then the exponential factor $\exp[-\gamma r_{ij}^2] \to 1$. We have

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \beta_0(\boldsymbol{x}_j^t - \boldsymbol{x}_i^t) + \alpha_t \epsilon_i^t. \tag{23}$$

Here, if we further set $\alpha_t = 0$, then the above equation (23) becomes a variant of differential evolution. On the other hand, if we replace $\boldsymbol{x}_i^t$ by the current global best solution g^*, we have

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \beta_0(\boldsymbol{g}^* - \boldsymbol{x}_i^t) + \alpha_t \epsilon_i^t, \tag{24}$$

which is essentially the accelerated particle swarm optimization (APSO) [78].

Thirdly, we set $\beta_0 = 0$, and let ϵ_i^t is related to $\boldsymbol{x}_i$, then (23) becomes a pitch adjustment variant of harmony search (HS).

Therefore, we can essentially say that DE, APSO, SA and HS are special cases of firefly algorithm. Conversely, FA can be considered as a good combination of all the four algorithms (DE, APSO, SA and HS), to a certain extent. Furthermore, FA uses nonlinear updating equation, which can produce rich behaviour and higher

convergence than the linear updating equations used in standard PSO and DE. Consequently, it is again no surprise that FA can outperform other algorithms in many applications such as multimodal optimization, classifications, image processing and feature selection as we will see later in the applications.

5.5 Variants of Firefly Algorithm

For discrete problems and combinatorial optimisation, discrete versions of firefly algorithm have been developed with superior performance [22, 26, 31, 35, 53], which can be used for travelling-salesman problems, graph colouring and other applications.

In addition, extension of firefly algorithm to multiobjective optimisation has also been investigated [3, 79].

A few studies show that chaos can enhance the performance of firefly algorithm [18, 77], while other studies have attempted to hybridize FA with other algorithms to enhance their performance [30, 33, 34, 51].

5.6 Attraction and Diffusion

The novel idea of attraction via light intensity as an exploitation mechanism was first used by Yang in the firefly algorithm (FA) in 2007 and 2008. In FA, the attractiveness (and light intensity) is intrinsically linked with the inverse-square law of light intensity variations and the absorption coefficient. As a result, there is a novel but nonlinear term of $\beta_0 \exp[-\gamma r^2]$ where β_0 is the attractiveness at the distance $r = 0$, and $\gamma > 0$ is the absorption coefficient for light [65].

Other algorithms also used inverse-square laws, derived from nature. For example, the charged system search (CSS) used Coulomb's law, while the gravitational search algorithm (GSA) used Newton's law of gravitation.

The main function of such attraction is to enable an algorithm to converge quickly because these multi-agent systems evolve, interact and attract, leading to some self-organized behaviour and attractors. As the swarming agents evolve, it is possible that their attractor states will move towards to the true global optimality.

This novel attraction mechanism is the first of its kind in the literature of nature-inspired computation and computational intelligence. This also motivated and inspired others to design similar or other kinds of attraction mechanisms. Whatever the attraction mechanism may be, from the metaheuristic point of view, the fundamental principles are the same: that is, they allow the swarming agents to interact with one another and provide a forcing term to guide the convergence of the population.

Attraction mainly provides the mechanisms only for exploitation, but, with proper randomization, it is also possible to carry out some degree of exploration. However, the exploration is better analyzed in the framework of random walks and diffusive

randomization. From the Markov chain point of view, random walks and diffusion are both Markov chains. In fact, Brownian diffusion such as the dispersion of an ink drop in water is a random walk. For example, the most fundamental random walks for an agent or solution $\boldsymbol{x}_i$ can be written as the following form:

$$\boldsymbol{x}_i^{(t+1)} = \boldsymbol{x}_i^{(t)} + \epsilon, \tag{25}$$

where t is a counter of steps. Here, ϵ is a random number drawn from a Gaussian normal distribution with a zero mean. This gives an average diffusion distance of a particle or agent that is a square root of the finite number of steps t. That is, the distance is the order of $\sqrt{Dt}$ where D is the diffusion coefficient. To be more specific, the variance of the random walks in a d-dimensional case can be written as

$$\sigma^2(t) = |v_0|^2 t^2 + (2dD)t, \tag{26}$$

where v_0 is the drift velocity.

This means it is possible to cover the whole search domain if t is sufficiently large. Therefore, the steps in the Brownian motion $B(t)$ essentially obeys a Gaussian distribution with zero mean and time-dependent variance. A diffusion process can be viewed as a series of Brownian motion, which obeys a Gaussian distribution. For this reason, standard diffusion is often referred to as the Gaussian diffusion. If the motion at each step is not Gaussian, then the diffusion is called non-Gaussian diffusion. On the other hand, random walks can take many forms. If the step lengths obey other distributions, we have to deal with more generalized random walks. A very special case is when step lengths obey the Lévy distribution, such a random walk is called Lévy flights or Lévy walks.

5.7 Why SA is Efficient

As the literature about firefly algorithm expands and new variants have emerged, all pointed out the firefly algorithm can outperform many other algorithms. Now we may ask naturally "Why is it so efficient?" To answer this question, let us briefly analyze the firefly algorithm itself.

FA is swarm-intelligence-based, so it has the similar advantages that other swarm-intelligence-based algorithms have. In fact, a simple analysis of parameters suggests that some PSO variants such as Accelerated PSO [78] are a special case of firefly algorithm when $\gamma = 0$ [65].

However, FA has two major advantages over other algorithms: automatical subdivision and the ability of dealing with multimodality. First, FA is based on attraction and attractiveness decreases with distance. This leads to the fact that the whole population can automatically subdivide into subgroups, and each group can swarm around each mode or local optimum. Among all these modes, the best global solution can

be found. Second, this subdivision allows the fireflies to be able to find all optima simultaneously if the population size is sufficiently higher than the number of modes. Mathematically, $1/\sqrt{\gamma}$ controls the average distance of a group of fireflies that can be seen by adjacent groups. Therefore, a whole population can subdivide into subgroups with a given, average distance. In the extreme case when $\gamma = 0$, the whole population will not subdivide. This automatic subdivision ability makes it particularly suitable for highly nonlinear, multimodal optimisation problems.

In addition, the parameters in FA can be tuned to control the randomness as iterations proceed, so that convergence can also be sped up by tuning these parameters. These above advantages makes it flexible to deal with continuous problems, clustering and classifications, and combinatorial optimisation as well.

As an example, let us use two functions to demonstrate the computational cost saved by FA. For details, please see the more extensive studies by Yang [70]. For De Jong's function with $d = 256$ dimensions

$$f(\boldsymbol{x}) = \sum_{i=1}^{256} x_i^2. \tag{27}$$

Genetic algorithms required 25412 ± 1237 evaluations to get an accuracy of 10^{-5} of the optimal solution, while PSO needed 17040 ± 1123 evaluations. For FA, we achieved the same accuracy by 5657 ± 730. This save about 78 and 67 % computational cost, compared to GA and PSO, respectively.

For Yang's forest function

$$f(\boldsymbol{x}) = \Big(\sum_{i=1}^{d} |x_i|\Big) \exp\Big[-\sum_{i=1}^{d} \sin(x_i^2)\Big], \quad -2\pi \le x_i \le 2\pi, \tag{28}$$

GA required 37079 ± 8920 with a success rate of 88 % for $d = 16$, and PSO required 19725 ± 3204 with a success rate of 98 %. FA obtained a 100 % success rate with just 5152 ± 2493. Compared with GA and PSO, FA saved about 86 and 74 %, respectively, of overall computational efforts.

In short, FA has three distinct advantages:

- Automatic subdivision of the whole population into subgroups.
- The natural capability of dealing with multi-modal optimization.
- High ergodicity and diversity in the solutions.

All these advantages make FA very unique and very efficient.

5.8 Applications

Firefly algorithm has attracted much attention and has been applied to many applications [3, 13, 31–33, 53, 71]. Horng et al. demonstrated that firefly-based algorithm used the least computation time for digital image compression [32, 33], while Banati and Bajaj used firefly algorithm for feature selection and showed that firefly algorithm produced consistent and better performance in terms of time and optimality than other algorithms [5].

In the engineering design problems, Gandomi et al. [28] and Azad and Azad [2] confirmed that firefly algorithm can efficiently solve highly nonlinear, multimodal design problems. Basu and Mahanti [7] as well as Chatterjee et al. have applied FA in antenna design optimisation and showed that FA can outperform artificial bee colony (ABC) algorithm [13]. In addition, Zaman and Matin have also found that FA can outperform PSO and obtained global best results [85].

Sayadi et al. developed a discrete version of FA which can efficiently solve NP-hard scheduling problems [53], while a detailed analysis has demonstrated the efficiency of FA over a wide range of test problems, including multobjective load dispatch problems [3, 70]. Furthermore, FA can also solve scheduling and travelling salesman problem in a promising way [35, 46, 84].

Classifications and clustering are another important area of applications of FA with excellent performance [50, 54]. For example, Senthilnath el al. provided an extensive performance study by compared FA with 11 different algorithms and concluded that firefly algorithm can be efficiently used for clustering [54]. In most cases, firefly algorithm can outperform all other 11 algorithms. In addition, firefly algorithm has also been applied to train neural networks [44].

For optimisation in dynamic environments, FA can also be very efficient as shown by Farahani et al. [24, 25] and Abshouri et al. [1].

6 Right Amount of Randomization

As we mentioned earlier, all metaheuristic algorithms have to use stochastic components (i.e., randomization) to a certain degree. Randomness increases the diversity of the solutions and thus enables an algorithm to have the ability to jump out of any local optimum. However, too much randomness may slow down the convergence of the algorithm and thus can waste a lot of computational efforts. Therefore, there is some tradeoff between deterministic and stochastic components, though it is difficult to gauge what is the right amount of randomness in an algorithm? In essence, this question is related to the optimal balance of exploration and exploitation, which still remains an open problem.

6.1 How to do Random Walks

As random walks are widely used for randomization and local search in metaheuristic algorithms [65, 68], a proper step size is very important. Typically, we use the following generic equation

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t + s\boldsymbol{\epsilon}_t, \tag{29}$$

where $\boldsymbol{\epsilon}_t$ is drawn from a standard normal distribution with a zero mean and unity standard deviation. Here, the step size s determines how far a random walker (e.g., an agent or a particle in metaheursitics) can go for a fixed number of iterations. Obviously, if s is too large, then the new solution $\boldsymbol{x}^{t+1}$ generated will be too far away from the old solution (or more often the current best). Then, such a move is unlikely to be accepted. If s is too small, the change is too small to be significant, and consequently such search is not efficient. So a proper step size is important to maintain the search as efficient as possible.

From the theory of simple isotropic random walks [48, 66, 67], we know that the average distance r traveled in the d-dimension space is

$$r^2 = 2dDt, \tag{30}$$

where $D = s^2/2\tau$ is the effective diffusion coefficient. Here, s is the step size or distance traveled at each jump, and τ is the time taken for each jump. The above equation implies that

$$s^2 = \frac{\tau\, r^2}{t\, d}. \tag{31}$$

For a typical scale L of dimensions of interest, the local search is typically limited in a region of $L/10$. That is, $r = L/10$. As the iterations are discrete, we can take $\tau = 1$. Typically in metaheuristics, we can expect that the number of generations is usually $t = 100$ to 1000, which means that

$$s \approx \frac{r}{\sqrt{td}} = \frac{L/10}{\sqrt{t\, d}}. \tag{32}$$

For $d = 1$ and $t = 100$, we have $s = 0.01L$, while $s = 0.001L$ for $d = 10$ and $t = 1000$. As step sizes could differ from variable to variable, a step size ratio s/L is more generic. Therefore, we can use $s/L = 0.001$ to 0.01 for most problems.

6.2 Accuracy and Number of Iterations

Let us suppose that we wish to achieve an accuracy of $\delta = 10^{-5}$, then we can estimate that the number of steps or iterations $N_{\max}$ needed by pure random walks. This is essentially the upper bound for $N_{\max}$

$$N_{\max} \approx \frac{L^2}{\delta^2 d}. \tag{33}$$

For example, for $L = 10$ and $d = 10$, we have

$$N_{\max} \approx \frac{1}{(10^{-5})^2 \times 10} \approx 10^{10}, \tag{34}$$

which is a huge number that is not easily achievable in practice. However, this number is still far smaller than that needed by a uniform or brute force search method. It is worth pointing out the above estimate is the upper limit for the worst-case scenarios. In reality, most metaheuristic algorithms require far fewer numbers of iterations.

On the other hand, the above formula implies another interesting fact that the number of iterations will not affect much by dimensionality. In fact, higher-dimensional problems do not necessarily increase the number of iterations significantly. This may lead to a rather surprising possibility that random walks may be efficient in higher dimensions if the optimization lanscape is highly multimodal. This provides some hints for designing better algorithms by cleverly using random walks and other randomization techniques.

If we use Lévy flights instead of Gaussian random walks. Then, we have an estimate [67]

$$N_{\max} \approx \left(\frac{L^2}{\delta^2 d}\right)^{1/(3-\lambda)}. \tag{35}$$

If we use $\lambda = 1.5$, we have

$$N_{\max} \approx 2 \times 10^7. \tag{36}$$

We can see that Lévy flights can reduce the number of iterations by about 4 orders $[O(10^4)]$.

7 Parameter Tuning and Parameter Control

The most challenging issue for metaheuristic algorithms is probably to control exploration and exploitation properly, which is still an open question. It is possible to control attraction and diffusion in algorithms that use such features so that the performance of an algorithm can be influenced in the right way. Ideally we should have some mathematical relationship that can explicitly show how parameters can be affect the performance of an algorithm, but this is an un-resolved problem. In fact, unless for very simple cases under very strict, sometimes, unrealistic assumptions, there is no theoretical results at all.

7.1 Parameter Tuning

As an algorithm is a set of interacting Markov chains, we can in general write an algorithm as

$$\begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^{t+1} = A[\boldsymbol{x}_1, ..., \boldsymbol{x}_n, \epsilon_1, ..., \epsilon_m, p_1(t), ..., p_k(t)] \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^{t}, \tag{37}$$

which generates a set of new solutions $(\boldsymbol{x}_1, ..., \boldsymbol{x}_n)^{t+1}$ from the current population of n solutions. This behaviour of an algorithm is largely determined by the eigenvalues of the matrix A that are in turn controlled by the parameters $p_k(t)$ and the randomness vector $\boldsymbol{\epsilon} = (\epsilon_1, ..., \epsilon_m)$. From the Markovian theory, we know that the first eigenvalue is typically 1, and therefore the convergence rate of an algorithm is mainly controlled by the second eigenvalue λ_2 of A. However, it is extremely difficult to find this eigenvalue in general. Therefore, the tuning of parameters become a very challenging task.

In fact, parameter-tuning is an important topic under active research [23]. The aim of parameter-tuning is to find the best parameter setting so that an algorithm can perform most well for a wider range of problems. At the moment, parameter-tuning is mainly carried out by detailed, extensive parametric studies, and there is no efficient method in general. In essence, parameter-tuning itself is an optimization problem which requires higher-level optimization methods to tackle.

7.2 Parameter Control

Related to parameter-tuning, there is another issue of parameter control. Parameter values after parameter-tuning are often fixed during iterations, while parameters should vary for parameter control. The idea of parameter control is to vary the parameters so that the algorithm of interest can provide the best convergence rate and thus may achieve the best performance. Again, parameter control is another tough optimization problem to be yet resolved. In the bat algorithm, some basic form of parameter control has been attempted and found to be very efficient [68]. By controlling the loudness and pulse emission rate, BA can automatically switch from explorative moves to local exploitation that focuses on the promising regions when the global optimality may be nearby. Similarly, the cooling schedule in simulated annealing can be considered as a form of basic parameter control.

On the other hand, eagle strategy (ES) is a two-stage iterative strategy with iterative switches [69]. ES starts with a population of agents in the explorative mode, and then switch to the exploitation stage for local intensive search. Then, it starts again with

another set of explorative moves and subsequently turns into a new exploitation stage. This iterative, restart strategy has been found to be very efficient. Both parameter-tuning and parameter control are under active research. More efficient methods are highly need for this purpose.

8 Discussions and Concluding Remarks

Swarm intelligence based algorithms such as cuckoo search and firefly algorithm are very efficient in solving a wide range of nonlinear optimization problems, and thus have diverse applications in sciences and engineering. Some algorithms (e.g., cuckoo search) can have very good global convergence. However, there are still some challenging issues that need to be resolved in future studies.

One key issue is that there is a significant gap between theory and practice. Most metaheuristic algorithms have good applications in practice, but mathematical analysis of these algorithms lacks far behind. In fact, apart from a few limited results about the convergence and stability about algorithms such as particle swarm, genetic algorithms and cuckoo search, many algorithms do not have theoretical analysis. Therefore, we may know they can work well in practice, we hardly understand why it works and how to improve them with a good understanding of their working mechanisms.

Another important issue is that all metaheuristic algorithms have algorithm-dependent parameters, and the actual values/setting of these parameters will largely influence the performance of an algorithm. Therefore, the proper parameter-tuning itself becomes an optimization problem. In fact, parameter-tuning is an important area of research [23], which deserves more research attention.

In addition, even with very good applications of many algorithms, most of these applications concern the cases with the number of design variables less than a few hundreds. It would be more beneficial to real-world applications if the number of variables can increase to several thousands or even to the order of millions.

It is worth pointing out that new research efforts should focus on the important questions such as those outlined above, it should not focus on developing various new algorithms for distractions. Nature has evolved over billions of years, providing a vast source of inspiration; this does not mean that we should develop all new kind of algorithms, such as grass algorithm, tiger algorithm, tree algorithm, sky algorithm, wind algorithm, ocean algorithm or universe algorithm. These could lead to misunderstanding and confusion of the true nature of metaheuristic algorithms and optimization. In fact, studies should try to answer truly important questions, including optimal balance of exploration and exploitation, global convergence, optimal parameter control and tuning, and large-scale real-world applications.

All these challenging issues may motivate more further research. There is no doubt that more applications of cuckoo search and firefly algorithm will be seen in the expanding literature in the near future. It is highly expected that more theoretical results about metaheuristic optimization will appear in the coming years.

References

1. Abshouri, A.A., Meybodi, M.R., Bakhtiary, A.: New firefly algorithm based on multiswarm and learning automata in dynamic environments. Third international conference on signal processing systems (ICSPS 2011), pp. 73–77. Yantai, China, 27–28 Aug 2011
2. Azad, S.K., Azad, S.K.: Optimum design of structures using an improved firefly algorithm. Int. J. Optim. Civ. Eng. **1**(2), 327–340 (2011)
3. Apostolopoulos, T., Vlachos, A.: Application of the firefly algorithm for solving the economic emissions load dispatch problem. Int. J. Comb. 2011, (2011). Article ID 523806. http://www.hindawi.com/journals/ijct/2011/523806.html
4. Ashby, W.R.: Princinples of the self-organizing sysem. In: Von Foerster, H., Zopf Jr, G.W. (eds.) Pricinples of Self-Organization: Transactions of the University of Illinois Symposium, pp. 255–278. Pergamon Press, London, UK (1962)
5. Banati, H., Bajaj, M.: Firefly based feature selection approach. Int. J. Comput. Sci. Issues **8**(2), 473–480 (2011)
6. Bansal, J.C., Deep, K.: Optimisation of directional overcurrent relay times by particle swarm optimisation. In: Swarm intelligence symposium (SIS 2008), pp. 1–7. IEEE Publication (2008)
7. Basu, B., Mahanti, G.K.: Firefly and artificial bees colony algorithm for synthesis of scanned and broadside linear array antenna. Prog. Electromagn. Res. B **32**, 169–190 (2011)
8. Bénichou, O., Loverdo, C., Moreau, M., Voituriez, R.: Two-dimensional intermittent search processes: An alternative to Lévy flight strategies. Phys. Rev. **E74**, 020102(R) (2006)
9. Bhargava, V., Fateen, S.E.K., Bonilla-Petriciolet, A.: Cuckoo search: a new nature-inspired optimization method for phase equilibrium calculations. Fluid Phase Equilib. **337**, 191–200 (2013)
10. Blum, C., Roli, A.: Metaheuristics in combinatorial optimisation: overview and conceptural comparision. ACM Comput. Surv. **35**, 268–308 (2003)
11. Bulatović, R.R., Bordević, S.R., Dordević, V.S.: Cuckoo search algorithm: a metaheuristic approach to solving the problem of optimum synthesis of a six-bar double dwell linkage. Mech. Mach. Theory **61**, 1–13 (2013)
12. Chandrasekaran, K., Simon, S.P.: Multi-objective scheduling problem: hybrid appraoch using fuzzy assisted cuckoo search algorithm. Swarm Evol. Comput. **5**(1), 1–16 (2012)
13. Chatterjee, A., Mahanti, G.K., Chatterjee, A.: Design of a fully digital controlled reconfigurable switched beam conconcentric ring array antenna using firefly and particle swarm optimisation algorithm. Prog. Elelectromagn. Res. B **36**, 113–131 (2012)
14. Chifu, V.R., Pop, C.B., Salomie, I., Suia, D.S., Niculici, A.N.: Optimizing the semantic web service composition process using cuckoo search. In: Intelligent distributed computing V, studies in computational intelligence vol. 382, pp. 93–102 (2012)
15. Choudhary, K., Purohit, G.N.: A new testing approach using cuckoo search to achieve multi-objective genetic algorithm. J. Comput. textbf3(4), 117–119 (2011)
16. Clerc, M., Kennedy, J.: The particle swarm—explosion, stability, and convergence in a multi-dimensional complex space. IEEE Trans. Evol. Comput. **6**(1), 58–73 (2002)
17. Civicioglu, P., Besdok, E.: A conception comparison of the cuckoo search, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artif. Intell. Rev. (2011). doi:10.1007/s10462-011-92760
18. dos Santos Coelho, L., de Andrade Bernert, D.L., Mariani, V.C.: A chaotic firefly algorithm applied to reliability-redundancy optimisation. In: 2011 IEEE Congress on evolutionary computation (CEC'11), pp. 517–521 (2011)
19. Dhivya, M., Sundarambal, M., Anand, L.N.: Energy efficient computation of data fusion in wireless sensor networks using cuckoo based particle approach (CBPA). Int. J. Commun. Netw. Syst. Sci. **4**, 249–255 (2011)
20. Dhivya, M., Sundarambal, M.: Cuckoo search for data gathering in wireless sensor networks. Int. J. Mobile Commun. **9**, 642–656 (2011)
21. Durgun, I., Yildiz, A.R.: Structural design optimization of vehicle components using cuckoo search algorithm. Mater. Test. **3**, 185–188 (2012)

22. Durkota, K.: Implementation of a discrete firefly algorithm for the QAP problem within the sage framework. B.Sc. thesis, Czech Technical University (2011)
23. Eiben, A.E., Smit, S.K.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol. Comput. **1**, 19–31 (2011)
24. Farahani, S.M., Abshouri, A.A., Nasiri, B., Meybodi, M.R.: A Gaussian firefly algorithm. Int. J. Mach. Learn. Comput. **1**(5), 448–453 (2011)
25. Farahani, S.M., Nasiri, B., Meybodi, M.R.: A multiswarm based firefly algorithm in dynamic environments. In: Third international conference on signal processing systems (ICSPS2011), pp. 68–72. Yantai, China, 27–28 Aug 2011
26. Fister Jr, I., Fister, I., Brest, J., Yang, X.S.: Memetic firefly algorithm for combinatorial optimisation. In: Filipič, B., Šilc, J. (eds.) Bioinspired Optimisation Methods and Their Applications (BIOMA2012), pp. 75–86. Bohinj, Slovenia, 24–25 May 2012
27. Fister, I., Fister Jr., I., Yang, X.S., Brest, J.: A comprehensive review of firefly algorithms. Swarm Evol. Comput. **6** (in press) (2013). http://dx.doi.org/10.1016/j.swevo.2013.06.001
28. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Cuckoo search algorithm: a meteheuristic approach to solve structural optimization problems. Engineering with Computers **29**(1), 17–35 (2013). doi:10.1007/s00366-011-0241-y
29. Gandomi, A.H., Yang, X.S., Talatahari, S., Deb, S.: Coupled eagle strategy and differential evolution for unconstrained and constrained global optimization. Comput. Math. Appl. **63**(1), 191–200 (2012)
30. Giannakouris, G., Vassiliadis, V., Dounias, G.: Experimental study on a hybrid nature-inspired algorithm for financial portfolio optimisation, SETN 2010. Lecture Notes in Artificial Intelligence (LNAI 6040), pp. 101–111 (2010)
31. Hassanzadeh, T., Vojodi, H., Moghadam, A.M.E.: An image segmentation approach based on maximum variance intra-cluster method and firefly algorithm. In: Proceedings of 7th International Conference on Natural Computation (ICNC2011), pp. 1817–1821 (2011)
32. Horng, M.-H., Lee, Y.-X., Lee, M.-C., Liou, R.-J.: Firefly metaheuristic algorithm for training the radial basis function network for data classification and disease diagnosis. In: Parpinelli, R., Lopes, H.S. (eds.) Theory and New Applications of Swarm Intelligence, pp. 115–132 (2012)
33. Horng, M.-H.: Vector quantization using the firefly algorithm for image compression. Expert Syst. Appl. **39**, 1078–1091 (2012)
34. Horng, M.-H., Liou, R.-J.: Multilevel minimum cross entropy threshold selection based on the firefly algorithm. Expert Syst. Appl. **38**, 14805–14811 (2011)
35. Jati, G.K., Suyanto, S.: Evolutionary discrete firefly algorithm for travelling salesman problem, ICAIS2011. Lecture Notes in Artificial Intelligence (LNAI 6943), pp. 393–403 (2011)
36. Jiang, M., Luo, Y.P., Yang, S.Y.: Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. Inf. Process. Lett. **102**, 8–16 (2007)
37. Kaveh, A., Bakhshpoori, T.: Optimum design of steel frames using cuckoo search algorithm with Levy flights. Struct. Des. Tall. Spec. Build. **21**, (online first) (2011). http://onlinelibrary.wiley.com/doi/10.1002/tal.754/abstract
38. Keller, E.F.: Organisms, machines, and thunderstorms: a history of self-organization, part two. Complexity, emergenece, and stable attractors. Hist. Stud. Nat. Sci. **39**(1), 1–31 (2009)
39. Kennedy, J., Eberhart, R.C.: Particle swarm optimization., In: Proceedings of IEEE International Conference on Neural Networks, pp. 1942–1948. Piscataway, NJ (1995)
40. Koziel, S., Yang, X.S.: Computational Optimization, Methods and Algorithms. Springer, Germany (2011)
41. Kumar A., Chakarverty, S.: Design optimization for reliable embedded system using Cuckoo Search. In: Proceedings of 3rd International Conference on Electronics Computer Technology (ICECT2011), pp. 564–568 (2011)
42. Layeb, A.: A novel quantum-inspired cuckoo search for Knapsack problems. Int. J. Bio-inspired Comput. **3**(5), 297–305 (2011)
43. Moravej, Z., Akhlaghi, A.: A novel approach based on cuckoo search for DG allocation in distribution network. Electr Power Energy Syst. **44**, 672–679 (2013)

44. Nandy, S., Sarkar, P.P., Das, A.: Analysis of nature-inspired firefly algorithm based back-propagation neural network training. Int. J. Comput. Appl. **43**(22), 8–16 (2012)
45. Noghrehabadi, A., Ghalambaz, M., Vosough, A.: A hybrid power series—Cuckoo search optimization algorithm to electrostatic deflection of micro fixed-fixed actuators. Int. J. Multi. Sci. Eng. **2**(4), 22–26 (2011)
46. Palit, S., Sinha, S., Molla, M., Khanra, A., Kule, M.: A cryptanalytic attack on the knapsack cryptosystem using binary Firefly algorithm. In: 2nd International Conference on Computer and Communication Technology (ICCCT), pp. 428–432. India, 15–17 Sept 2011
47. Parpinelli, R.S., Lopes, H.S.: New inspirations in swarm intelligence: a survey. Int. J. Bio-Inspired Comput. **3**, 1–16 (2011)
48. Pavlyukevich, I.: Lévy flights, non-local search and simulated annealing. J. Comput. Phys. **226**, 1830–1844 (2007)
49. Perumal, K., Ungati, J.M., Kumar, G., Jain, N., Gaurav, R., Srivastava, P.R.: Test data generation: a hybrid approach using cuckoo and tabu search, Swarm, Evolutionary, and Memetic Computing (SEMCCO2011). Lecture Notes in Computer Sciences vol. 7077, pp. 46–54 (2011)
50. Rajini, A., David, V.K.: A hybrid metaheuristic algorithm for classification using micro array data. Int. J. Sci. Eng. Res. **3**(2), 1–9 (2012)
51. Rampriya, B., Mahadevan, K., Kannan, S.: Unit commitment in deregulated power system using Lagrangian firefly algorithm. In: Proceedings of IEEE International Conference on Communication Control and Computing Technologies (ICCCCT2010), pp. 389–393 (2010)
52. Ren, Z.H., Wang, J., Gao, Y.L.: The global convergence analysis of particle swarm optimization algorithm based on Markov chain. Control Theory Appl. (in Chinese) **28**(4), 462–466 (2011)
53. Sayadi, M.K., Ramezanian, R., Ghaffari-Nasab, N.: A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. Int. J. Ind. Eng. Comput. **1**, 1–10 (2010)
54. Senthilnath, J., Omkar, S.N., Mani, V.: Clustering using firely algorithm: performance study. Swarm Evol. Comput. **1**(3), 164–171 (2011)
55. Speed, E.R.: Evolving a Mario agent using cuckoo search and softmax heuristics. In: Proceedings of the Games Innovations Conference (ICE-GIC), pp. 1–7 (2010)
56. Srivastava, P.R., Chis, M., Deb, S., Yang, X.S.: An efficient optimization algorithm for structural software testing. Int. J. Artif. Intell. **9**(S12), 68–77 (2012)
57. Taweewat, P., Wutiwiwatchai, C.: Musical pitch estimation using a supervised single hidden layer feed-forward neural network. Expert Syst. Appl. **40**, 575–589 (2013)
58. Tein, L.H., Ramli, R.: Recent advancements of nurse scheduling models and a potential path. In: Proceedings of 6th IMT-GT Conference on Mathematics, Statistics and its Applications (ICMSA 2010), pp. 395–409 (2010)
59. Valian, E., Mohanna, S., Tavakoli, S.: Improved cuckoo search algorithm for feedforward neural network training. Int. J. Artif. Intell. Appl. **2**(3), 36–43 (2011)
60. Valian, E., Tavakoli, S., Mohanna, S., Haghi, A.: Improved cuckoo search for reliability optimization problems. Comput. Ind. Eng. **64**, 459–468 (2013)
61. Vazquez, R.A.: Training spiking neural models using cuckoo search algorithm. In: 2011 IEEE Congress on Eovlutionary Computation (CEC'11), pp. 679–686 (2011)
62. Walton, S., Hassan, O., Morgan, K., Brown, M.R.: Modified cuckoo search: a new gradient free optimization algorithm. Chaos, Solitons Fractals **44**(9), 710–718 (2011)
63. Wang, F., He, X.-S., Wang, Y., Yang, S.M.: Markov model and convergence analysis based on cuckoo search algorithm. Comput. Eng. **38**(11), 180–185 (2012)
64. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)
65. Yang, X.S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press, UK (2008)
66. Yang, X.S.: Introduction to Computational Mathematics. World Scientific Publishing, Singapore (2008)
67. Yang, X.S.: Engineering Optimisation: An Introduction with Metaheuristic Applications. John Wiley and Sons, USA (2010)

68. Yang, X.S.: A new metaheuristic bat-inspired algorithm. In: Gonzalez, J.R. et al. (eds.) Nature Inspired Cooperative Strategies for Optimisation (NICSO 2010). Studies in Computational Intelligence, vol. 28, 4 pp. 65–74. Springer, Berlin (2010)
69. Yang, X.S., Deb, S.: Eagle strategy using Lévy walks and firefly algorithm for stochastic optimization. In: Gonzalez, J.R. et al. (eds.) Nature-Inspired Cooperative Strategies for Optimization (NICSO 2010), Studies in Computational Intelligence, vol. 284, pp. 101–111. Springer, Berlin (2010)
70. Yang, X.S.: Firefly algorithms for multimodal optimization. In: Stochastic Algorithms: Foundations and Applications, SAGA 2009. Lecture Notes in Computer Sciences, vol. 5792, pp. 169–178 (2009)
71. Yang, X.-S.: Firefly algorithm, stochastic test functions and design optimisation. Int. J. Bio-inspired Comput. **2**(2), 78–84 (2010)
72. Yang, X.S., Deb, S., Fong, S.: Accelerated particle swarm optimization and support vector machine for business optimization and applications, Networked Digital Technologies 2011. Commun. Comput. Inf. Sci. **136**, 53–66 (2011)
73. Yang, X.S., Gandomi, A.H.: Bat algorithm: a novel approach for global engineering optimization. Eng. Comput. **29**(5), 1–18 (2012)
74. Yang, X.S.: Flower pollination algorithm for global optimization. In: Unconventional Computation and Natural Computation, pp. 240–249. Springer (2012)
75. Yang, X.S., Karamanoglu, M., He, X.S.: Multi-objective flower algorithm for optimization. Procedia Comput. Sci. **18**, 861–868 (2013)
76. Yang, X.S., Deb, S.: Cuckoo search via Lévy flights. In: Proceeings of World Congress on Nature and Biologically Inspired Computing (NaBIC 2009), pp. 210–214. IEEE Publications, USA (2009)
77. Yang, X.S.: Chaos-enhanced firefly algorithm with automatic parameter tuning. Int. J. Swarm Intell. Res. **2**(4), 1–11 (2011)
78. Yang, X.S., Deb, S., Fong, S.: Accelerated particle swarm optimization and support vector machine for business optimization and applications, Networked Digital Technologies (NDT'2011). Commun. Comput. Inform. Sci. **136**(Part I), 53–66 (2011)
79. Yang, X.S.: Multiobjective firefly algorithm for continuous optimization. Engineering with Computers **29**(2), 175–184 (2013)
80. Yang, X.S., Deb, S.: Engineering optimization by cuckoo search. Int. J. Math. Model. Num. Opt. **1**(4), 330–343 (2010)
81. Yang, X.S., Deb, S.: Multiobjective cuckoo search for design optimization. Comput. Oper. Res. **40**(6), 1616–1624 (2013)
82. Yang, X.S., Cui, Z.H., Xiao, R.B., Gandomi, A.H., Karamanoglu, M.: Swarm Intelligence and Bio-inspired Computation: Theory and Applications. Elsevier, Waltham (2013)
83. Yildiz, A.R.: Cuckoo search algorithm for the selection of optimal machine parameters in milling operations. Int. J. Adv. Manuf. Technol. (2012). doi:10.1007/s00170-012-4013-7
84. Yousif, A., Abdullah, A.H., Nor, S.M., Abdelaziz, A.A.: Scheduling jobs on grid computing using firefly algorithm. J. Theor. Appl. Inform. Technol. **33**(2), 155–164 (2011)
85. Zaman, M.A., Matin, M.A.: Nonuniformly spaced linear antenna array design using firefly algorithm. Int. J. Microw. Sci. Technol. 2012, 8 (2012). Article ID: 256759, doi:10.1155/2012/256759
86. Zheng, H.Q., Zhou, Y.: A novel cuckoo search optimization algorithm based on Gauss distribution. J. Comput. Inform. Syst. **8**, 4193–4200 (2012)

On the Randomized Firefly Algorithm

Iztok Fister, Xin-She Yang, Janez Brest and Iztok Fister Jr.

Abstract The firefly algorithm is a stochastic meta-heuristic that incorporates randomness into a search process. Essentially, the randomness is useful when determining the next point in the search space and therefore has a crucial impact when exploring the new solution. In this chapter, an extensive comparison is made between various probability distributions that can be used for randomizing the firefly algorithm, e.g., Uniform, Gaussian, Lévi flights, Chaotic maps, and the Random sampling in turbulent fractal cloud. In line with this, variously randomized firefly algorithms were developed and extensive experiments conducted on a well-known suite of functions. The results of these experiments show that the efficiency of a distributions largely depends on the type of a problem to be solved.

Keywords Chaos · Firefly algorithm · Randomization · Random sampling in turbulent fractal cloud · Swarm intelligence

I. Fister (✉) · J. Brest · I. Fister Jr.
Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia
e-mail: iztok.fister@uni-mb.si

J. Brest
e-mail: janez.brest@uni-mb.si

I. Fister Jr.
e-mail: iztok.fister@guest.arnes.si

X.-S. Yang
School of Science and Technology, Middlesex University, London, UK
e-mail: x.yang@mdx.ac.uk

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_2,

1 Introduction

Automatic problem solving with a digital computer has been the eternal quest of researchers in mathematics, computer science and engineering. The majority of complex problems (also NP-hard problems [1]) cannot be solved using exact methods by enumerating all the possible solutions and searching for the best solution (minimum or maximum value of objective function). Therefore, several algorithms have been emerged that solve problems in some smarter (also heuristic) ways. Nowadays, designers of the more successful algorithms draw their inspirations from Nature. For instance, the collective behavior of social insects like ants, termites, bees and wasps, or some animal societies like flocks of bird or schools of fish have inspired computer scientists to design intelligent multi-agent systems [2].

For millions of years many biological systems have solved complex problems by sharing information with group members [3]. These biological systems are usually very complex. They consists of particles (agents) that are definitely more complex than molecules and atoms, and are capable of performing autonomous actions within an environment. On the other hand, a group of particles is capable of intelligent behavior which is appropriate for solving complex problems in Nature. Therefore, it is no coincidence that these biological systems have also inspired computer scientists to imitate their intelligent behavior for solving complex problems in mathematics, physics, engineering, etc. Moreover, interest in researching various biological systems has increased recently. These various biological systems have been influenced by swarm intelligence (SI) that can be viewed as an artificial intelligence (AI) discipline concerned with the designing of intelligent systems.

It seems that the first use of the term 'swarm intelligence' was probably by Beni and Wang [4] in 1989 in the context of a cellular robotic system. Nowadays, this term also extends to the field of optimization, where techniques based on swarm intelligence have been applied successfully. Examples of notable swarm intelligence optimization techniques are ant colony optimization [5], particle swarm optimization [6], and artificial bees colony (ABC) [7, 8]. Today, the more promising swarm intelligence optimization techniques include the firefly algorithm (FA) [9–14], the cuckoo search [15], and the bat algorithm [16, 17].

Stochastic optimization searches for optimal solutions by involving randomness in some constructive way [18]. In contrast, if optimization methods provide the same results when doing the same things, these methods are said to be deterministic [19]. If the deterministic system behaves unpredictably, it arrives at a phenomenon of chaos [19]. As a result, randomness in SI algorithms plays a huge role because this phenomenon affects the exploration and exploitation in search process [20]. These companions of stochastic global search represent the two cornerstones of problem solving, i.e., exploration refers to moves for discovering entirely new regions of a search space, while exploitation refers to moves that focus searching the vicinity of promising, known solutions to be found during the search process. Both components are also referred to as intensification and diversification in another terminology [21]. However, these refer to medium- to long- term strategies based on the usage of mem-

ory (Tabu search), while exploration and exploitation refer to short-term strategies tied to randomness [20].

Essentially, randomness is used in SI algorithms in order to explore new points by moving the particles towards the search space. In line with this, several random distributions can be helpful. For example, uniform distribution generates each point of the search space using the same probability. On the other hand, Gaussian distribution is biased towards the observed solution, that is that the smaller modifications occur more often than larger ones [22]. On the other hand, the appropriate distribution depends on the problem to be solved, more precisely, on a fitness landscape that maps each position in the search space into fitness value. When the fitness landscape is flat, uniform distribution is more preferable for the stochastic search process, whilst in rougher fitness landscapes Gaussian distribution should be more appropriate.

This chapter deals with an FA algorithm that uses different randomization methods, like Uniform, Gaussian, Lévy flights, Chaotic maps, and the Random sampling in turbulent fractal cloud. Whilst the observed randomization methods are well-known and widely used (e.g., uniform, Gaussian, Lévi flights, and chaotic maps), the random sampling in turbulent fractal cloud is taken from astronomy and, as we know, it is the first time used for the optimization purposes. Here, two well-known chaotic maps are also taken into account, i.e., Kent and Logistic.

The goal of our experimental work is to show how the different randomized methods influence the results of the modified FA. In line with this, a function optimization problem was solved by this algorithm. A suite of ten well-known functions were defined, as defined in the literature.

The structure of the rest of the chapter is as follows: Sect. 2 describes a background information of various randomization methods. In Sect. 3, the randomized firefly algorithms are presented. Experiments and results are illustrated in Sect. 4. The chapter is finished summarizing our work and the directions for further development are outlined.

2 Background Information

This section describes the background information about generating random variates and computing their probability distributions, as follows:

- Uniform,
- Normal or Gaussian,
- Lévy flights,
- Chaotic maps,
- Random sampling in turbulent fractal cloud.

Continuous random number distributions [23] are defined by a probability density function $p(x)$, such that the probability of x occurring within the infinitesimal range x to $x + dx$ is $p \cdot dx$. The cumulative distribution function for the lower tail $P(x)$ is defined as the integral

$$P(x) = \int_{-\infty}^{x} p(x) \cdot dx, \tag{1}$$

which gives the probability of a variate taking a value of less than x. The cumulative distribution function for the upper tail $Q(x)$ is defined as the integral

$$Q(x) = \int_{x}^{+\infty} p(x) \cdot dx, \tag{2}$$

which provides the probability of a value greater than x.

The upper and lower cumulative distribution functions are related by $P(x) + Q(x) = 1$ and satisfy the following limitations $0 \leq P(x) \leq 1$ and $0 \leq Q(x) \leq 1$. In the remainder of this section, these randomized methods are presented in more detail.

2.1 Uniform Distribution

Uniform continuous distribution has the density function, as follows:

$$p(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that each possible value of the uniform distributed random variable is within optional interval $[a, b]$, on which the probability of each sub-interval is proportional to its length. If $a \leq u < v \leq b$ then the following relation holds:

$$P(u < x < v) = \frac{v-u}{b-a}. \tag{4}$$

Normally, the uniform distribution is obtained by a call to the random number generator [23]. Note that the discrete variate functions always return a value of type unsigned which on most platforms means a random value from the interval $[0, 2^{32} - 1]$. In order to obtain the random generated value within the interval [0, 1], the following mapping is used:

$$r = ((double)rand()/((double)(\text{RAND\_MAX}) + (double)(1))), \tag{5}$$

where r is the generated random number, the function $rand()$ is a call of the random number generator, and the RAND_MAX is the maximal number of the random value $(2^{32} - 1)$.

2.2 Normal or Gaussian Distribution

Normal or Gaussian distribution is defined with the following density function:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-a}{\sigma})^2}. \tag{6}$$

The distribution depends on parameters $a \in \mathbb{R}$ and $\sigma > 0$. This distribution is denoted as $\mathrm{N}(a, \sigma)$. The standardized normal distribution is obtained, when the distribution has an average value of zero with standard deviation of one, i.e., $\mathrm{N}(0, 1)$. In this case, the density function is simply defined as:

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}. \tag{7}$$

The Gaussian distribution has the property that approximately $2/3$ of the samples drawn lie within one standard deviation [22]. That is, the most of the modifications made on the virtual particle will be small, whilst there is a non-zero probability of generating very large modifications, because the tail of distribution never reaches zero [22].

2.3 Lévy Flights

In reality, resources are distributed non-uniformly in Nature. This means, that the behavior of a typical forager needing to find these resources as fast as possible does not obey Gaussian distribution. In order to simulate foragers search strategies, Lévy flights is closer to their behavior [24]. It belongs to a special class of α-stable distributions defined by a Fourier transform [23]:

$$p(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-itx - |ct|^\alpha}. \tag{8}$$

The α-stable means that it exhibits the same probability density distributions for each randomly generated variable. This density function has two parameters: scale $c \in \mathbb{R}$ and exponent $\alpha \in [0, 2]$. A main characteristic of this distribution is that it has an infinite variance.

Interestingly, for $\alpha = 1$ the density function reduces to the Cauchy distribution, whilst for $\alpha = 2$ it is a Gaussian distribution with $\sigma = \sqrt{2c}$. For $\alpha < 1$ the tails of the distribution become extremely wide [24]. The more appropriate setting of this parameter for optimization is therefore $\alpha \in (1.0, 2.0)$, where the distribution is non-Gaussian with no variance (as in the case of Lévy flights) or with no mean, variance or higher moments defined (as in the case of Cauchy distribution). Essentially, the difference between non-Gaussian and Gaussian distributions is that the tail of the

distribution by the former is wider than by the latter. This means, the probability of generating very large modifications is much higher by Lévy flights than by Gaussian distribution.

2.4 Chaotic Maps

Chaos is a phenomenon encountered in science and mathematics wherein a deterministic (rule-based) system behaves unpredictably [19]. Let us assume a Logistic equation defined as a map:

$$x_{n+1} = r x_n (1 - x_n), \tag{9}$$

where $x_n \in (0, 1)$ and parameter r is a parameter. A generated sequence of numbers by iterating a Logistic map (also orbit) with $r = 4$ are *chaotic*. That is, it posses the following propositions [19]:

- the dynamic rule of generating the sequence of numbers is deterministic,
- the orbits are aperiodic (they never repeat),
- the orbits are bounded (they stay between upper and lower limits, normally, within the interval [0, 1]),
- the sequence has sensitive dependence on the initial condition (also SDIC).

Similar behavior can also be observed by the Kent map [25]. The Kent map is one of the more studied chaotic maps that has been used to generate pseudo-random numbers in many applications, like secure encryption. It is defined as follows:

$$x(n+1) = \begin{cases} \frac{x(n)}{m}, & 0 < x(n) \leq m, \\ \frac{(1-x(n))}{1-m}, & m < x(n) < 1, \end{cases} \tag{10}$$

where $0 < m < 1$. Hence, if $x(0) \in (0, 1)$, for all $n \geq 1, x(n) \in [0, 1]$. In accordance with the propositions in [25], $m = 0.7$ was used in our experiments.

2.5 Random Sampling in Turbulent Fractal Cloud

Stars formation begins with a random sampling of mass in a fractal cloud [26]. This random sampling is performed by the initial mass function (IMF) and represents a basis for a new sampling method named as random sampling in turbulent fractal cloud (RSiTFC).

The method can be described as follows. Let us consider a fractal cloud that is divided into a hierarchical structure consisting of several levels with a certain number of cloud pieces containing a certain number of sub-pieces. Then, a sampling method randomly samples a cloud piece from any level. The sampled pieces at the top of this

hierarchical structure are denser than the pieces at the bottom. When the cloud piece is chosen the initial mass of that piece is identified and the piece representing the formed star is removed from the hierarchy. This process is repeated until all of the cloud is chosen [26]. This mentioned method is formally illustrated in Algorithm 1.

The algorithm RSiTFC consists of six parameters: scaling factor L, number of levels H, number of sub-pieces for each piece N, fractal cloud pieces x, level h, and piece number i. The scaling factor is determined as $S = L^{-h}$ when calculated from the fractal dimension expressed as $D = \frac{\log N}{\log L}$. The number of levels H determines the depth of the hierarchical structure. The number of pieces increases with level h according to Poisson distribution $P_N(h) = N^h e^{-N}/h!$. The length of fractal cloud pieces x is limited by N^H and consists of elements representing the initial mass of the star to be formed. Level h denotes the current hierarchical level, whilst i determines the star to be formed.

Algorithm 1 RSiTFC(L, H, N, x, h, i)

Input: L scaling factor, H number of levels, N number of sub-pieces
Input: $*x$ fractal cloud, h current level, $*i$ piece number

```
if(i == 0)
  x = new double [N^H];
  for(j = 0; j < N^h; j++)
    x[*i] = 2 * (U(0, 1) − 0.5)/L^h + 0.5;
    *i = *i + 1;
  end for
else
  for(j = 0; j < N^h; j++)
    x[*i] = x[*i] + 2 * (U(0, 1) − 0.5)/L^h;
    *i = *i + 1;
  end for
end if
if(h < H)
  return RSiTFC(L, H, N, x, h + 1, i);
end if
return x;
```

For example, let $N = 2$ be a constant number of sub-pieces for each piece. Then, there is one cloud at the top level $h = 0$, with two pieces inside this cloud at $h = 1$, etc. For $H = 4$, the total number of pieces is expressed as $1 + 2 + 4 + 8 + 16 = 31$ [26].

3 Randomized Firefly Algorithms

Fireflies (Coleoptera: Lampyridae) are well known for bioluminescent signaling, which is used for species recognition and mate choosing [27]. Bioluminiscence that comprises a complicated set of chemical reactions is not always a sexual signal only but also warns off potential predators. In the remainder of the chapter, an original FA

algorithm is described that captures the bioluminiscent behavior of fireflies within the fitness function. Further, this algorithm is then modified with various randomized methods.

3.1 Original Firefly Algorithm

The light-intensity I of the flashing firefly decreases as the distance from source r decreases in terms of $I \propto 1/r^2$. Additionally, air absorption causes the light to become weaker and weaker as the distance from the source increases. This flashing light represented the inspiration for developing the FA algorithm by Yang [9] in 2008. Here, the light-intensity is proportional to the objective function of the problem being optimized (i.e., $I(\mathbf{s}) \propto f(\mathbf{s})$, where $\mathbf{s} = S(\mathbf{x})$ represent a candidate solution).

In order to formulate the FA, some flashing characteristics of fireflies were idealized, as follows:

- All fireflies are unisex.
- Their attractiveness is proportional to their light intensity.
- The light intensity of a firefly is affected or determined by the landscape of the objective function.

Note that light-intensity I and attractiveness are in some way synonymous. While the intensity I is referred to as an absolute measurement of emitted light by firefly, the attractiveness is a relative measurement of the light that should be seen in the eyes of beholders and judged by the other fireflies [9]. The light intensity I varies with distance r is expressed by the following equation

$$I(r) = I_0 e^{-\gamma r^2}, \tag{11}$$

where I_0 denotes the intensity of the light at the source, and γ is a fixed light absorption coefficient. Similarly, the attractiveness β that also depends on the distance r is calculated according to the following generalized equation

$$\beta(r) = \beta_0 e^{-\gamma r^2}. \tag{12}$$

The distance between two fireflies i and j is represented as the Euclidian distance

$$r_{ij} = \|\mathbf{s}_i - \mathbf{s}_j\| = \sqrt{\sum_{k=1}^{D} s_{ik} - s_{jk}}, \tag{13}$$

where s_{ik} is the k-th element of the i-th firefly position within the search-space, and D denotes the dimensionality of a problem. Each firefly i moves to another more attractive firefly j, as follows

$$\mathbf{s}_i = \mathbf{s}_i + \beta_0 e^{-\gamma r_{ij}^2}(\mathbf{s}_j - \mathbf{s}_i) + \alpha \cdot N_i(0, 1). \tag{14}$$

Equation (14) consists of three terms. The first term determines the position of the i-th firefly. The second term refers to the attractiveness, while the third term is connected with the randomized move of the i-th firefly within the search-space. This term consists of the randomized parameter α, and the random numbers $N_i(0, 1)$ drawn from a Gaussian distribution. The scheme of FA is sketched in Algorithm 1. The FA algorithm (Algorithm 2) runs on the population of fireflies $P^{(t)}$ that are

Algorithm 2 Original FA algorithm

1: $t = 0$; $\mathbf{s}^* = \emptyset$; $\gamma = 1.0$; // initialize: gen.counter, best solution, attractiveness
2: $\mathbf{P}^{(t)}$=InitFA(); // initialize the firefly population $\mathbf{s_i^{(0)}} \in \mathbf{P}^{(0)}$
3: **while** $t \leq$ MAX_GEN **do**
4: $\alpha^{(t)}$ = AlphaNew(); // determine a new value of α
5: EvaluateFA($\mathbf{P}^{(t)}$, $f(\mathbf{s})$); // evaluate $\mathbf{s}_i^{(t)}$ according to $f(\mathbf{s}_i)$
6: OrderFA($\mathbf{P}^{(t)}$, $f(\mathbf{s})$); // sort $\mathbf{P}_i^{(t)}$ according to $f(\mathbf{s}_i)$
7: s^* = FindTheBestFA($\mathbf{P}^{(t)}$, $f(\mathbf{s})$); // determine the best solution $\mathbf{s}^*$
8: $\mathbf{P}^{(t+1)}$=MoveFA($\mathbf{P}^{(t)}$); // vary attractiveness according Eq. (14)
9: $t = t + 1$;
10: **end while**
11: return $\mathbf{s}^*$, $f(\mathbf{s})$; // post process

represented as real-valued vectors $\mathbf{s}^{(t)}{}_i = s_{i0}^{(t)}, \ldots s_{in}^{(t)}$, where $i = 1 \ldots NP$ and NP denotes the number of fireflies in population $P^{(t)}$ at generation t. Note that each firefly $\mathbf{s}_i^{(t)}$ is of dimension D. The population of fireflies is initialized randomly (function InitFA) according to equation

$$s_{ij}^{(0)} = (ub_i - lb_i) \cdot \text{rand}(0, 1) + lb_i, \tag{15}$$

where ub_i and lb_i denote the upper and lower bounds, respectively. The main loop of the firefly search process that is controlled by the maximum number of generations *MAX_GEN* consists of the following functions. Firstly, the new values for the randomization parameter α is calculated according to the following equation (function AlphaNew):

$$\begin{aligned} \Delta &= 1 - 10^{-4}/0.9^{1/MAX\_GEN}, \\ \alpha^{(t+1)} &= 1 - \Delta \cdot \alpha^{(t)}, \end{aligned} \tag{16}$$

where Δ determines the step size of changing the parameter $\alpha^{(t+1)}$. Note that this parameter monotony descends with the increasing of generation counter t. Secondly, the new solution $\mathbf{s}_i^{(t)}$ is evaluated according to a fitness function $f(\mathbf{s}^{(t)})$, where $\mathbf{s}_i^{(t)} = S(\mathbf{x}_i^{(t)})$ (function EvaluateFA). Thirdly, solutions $\mathbf{s}_i^{(t)}$ for $i = 1 \ldots NP$ were ordered with respect to the fitness function $f(\mathbf{s}_i^{(t)})$ ascending, where $\mathbf{s}_i^{(t)} = S(\mathbf{x}_i^{(t)})$ (function

OrderFA). Fourthly, the best solution the best solution $\mathbf{s}^* = \mathbf{s}_0^{(t)}$ is determined in the population $P^{(t)}$ (function FindTheBestFA). Finally, the virtual fireflies are moved (function MoveFA) towards the search space according to the attractiveness of their neighbors' solution (Eq. 14).

In the remainder of this paper, the randomized FA is discussed in more detail.

3.2 Variants of the Randomized Firefly Algorithm

Randomized FA (RFA) is based on the original FA that is upgraded using the mentioned randomized methods. In place of the original Eq. (14), the modified equation is used by RFA, as follows:

$$\mathbf{s}_i = \mathbf{s}_i + \beta_0 e^{-\gamma r_{ij}^2}(\mathbf{s}_j - \mathbf{s}_i) + \alpha \cdot R_i, \tag{17}$$

where R_i denotes one of the randomized methods presented in Table 1.
As can be seen from Table 1, six randomized methods are used in the RFA algorithm that also differ according to the interval of generated random values. The former returns the random values in an interval [0, 1], like Uniform distribution, and both chaotic maps, whilst the latter within an interval $(-\infty, +\infty)$, like Gaussian, Lévy flights, and RSiTFC. When the random value is generated within the interval [0, 1], this value is extended to the interval $[-1, 1]$ using a formula $r_i = 2(r_i - 0.5)$. However, the generated solution value s_{ij} is verified to lie within the valid interval $s_{ij} \in [lb_j, ub_j]$, in both cases. Interestingly, some implementations of random generators can be found in Standard C-library (as a standard random generator for generating uniformly distributed random values), another in the GNU Scientific Library [23] (as random generators for generating the Gaussian and Lévy flights distributed random values), and the rest were developed from scratch (as chaotic maps and RSiTFC). According to the used randomized method, six different variants of the RFA algorithm are developed (as UFA, NFA, LFA, CFA1, CFA2, and FFA).

Table 1 Variants of the RFA algorithm

Randomization method	Random generator	Implementation	RFA variant
Uniform distributed	$U_i(0, 1)$	Standard C-library	UFA
Gaussian distributed	$N_i(-\infty, +\infty)$	GSL-library	NFA
Lévy flights	$L_i(-\infty, +\infty)$	GSL-library	LFA
Kent chaotic map	$C_i^K(0, 1)$	From scratch	CFA1
Logistic chaotic map	$C_i^L(0, 1)$	From scratch	CFA2
RSiTFC	$F_i(-\infty, +\infty)$	From scratch	FFA

4 Experiments and Results

An aim of our experimental work was to show that the developed randomized methods has a substantial (if not significant) impact on the results of the RFA algorithm. In line with this, six variants of the RFA algorithm (UFA, NGA, LFA, CFA1, CFA2, and FFA) were compared on a suite of ten well-known functions taken from publications. In the remainder of this chapter, the test suite is presented, then an experiment setup is defined that is followed by a description of a PC configuration, on which the experiments were performed. Finally, the results of the experiments are presented, in detail.

4.1 Test Suite

The test suite consisted of ten functions which were selected from two references. The first five functions were taken from Karaboga's paper [7], in which the ABC algorithm was introduced, while the last five were from the paper of Yang [28] that proposed a set of optimization functions suitable for testing the newly-developed algorithms.

The functions within the test suite can be divided into *unimodal* and *multimodal*. The multimodal functions have two or more local optima. The function is *separable*, when the set of variables can be rewritten as a sum of the function of just one variable. The separable and multimodal functions are more difficult to solve. The more complex functions are those that have an exponential number of local optima randomly distributed within the search space. The definitions and characteristics of functions constituting the test suite, can be summarized as follows:

- Griewangk's function:

$$f_1(\mathbf{s}) = -\prod_{i=1}^{D} \cos\left(\frac{s_i}{\sqrt{i}}\right) + \sum_{i=1}^{D} \frac{s_i^2}{4000} + 1, \tag{18}$$

  where $s_i \in [-600, 600]$. The function has the global minimum $f^* = 0$ at $\mathbf{s}^* = (0, 0, \ldots, 0)$. It is highly multimodal, when the number of variables is higher than 30.
- Rastrigin's function:

$$f_2(\mathbf{s}) = D * 10 + \sum_{i=1}^{D} (s_i^2 - 10\cos(2\pi s_i)), \tag{19}$$

  where $s_i \in [-15, 15]$. The function has the global minimum $f^* = 0$ at $\mathbf{s}^* = (0, 0, \ldots, 0)$ and is also highly multimodal.
- Rosenbrock's function:

$$f_3(\mathbf{s}) = \sum_{i=1}^{D-1} 100\,(s_{i+1} - s_i^2)^2 + (s_i - 1)^2, \tag{20}$$

where $s_i \in [-15, 15]$ and whose global minimum $f^* = 0$ is at $\mathbf{s}^* = (1, 1, \ldots, 1)$. This function, also known as the 'banana function' has several local optima. Gradient-based algorithms are especially difficult to converge to the global optima by optimizing this function.

- Ackley's function:

$$f_4(\mathbf{s}) = \sum_{i=1}^{D-1} (20 + e - 20e^{-0.2\sqrt{0.5(s_{i+1}^2 + s_i^2)}} - e^{0.5(\cos(2\pi s_{i+1}) + \cos(2\pi s_i))}), \tag{21}$$

where $s_i \in [-32.768, 32.768]$. The function has the global minimum $f^* = 0$ at $\mathbf{s}^* = (0, 0, \ldots, 0)$ and it is highly multimodal.

- Schwefel's function:

$$f_5(\mathbf{s}) = 418.9829 * D - \sum_{i=1}^{D} s_i \sin(\sqrt{|s_i|}), \tag{22}$$

where $s_i \in [-500, 500]$. The Schwefel's function has the global minimum $f^* = 0$ at $\mathbf{s}^* = (1, 1, \ldots, 1)$ and is highly multimodal.

- De Jong's sphere function:

$$f_6(\mathbf{s}) = \sum_{i=1}^{D} s_i^2, \tag{23}$$

where $s_i \in [-600, 600]$ and whose global minimum $f^* = 0$ is at $\mathbf{s}^* = (0, 0, \ldots, 0)$. The function is unimodal and convex.

- Easom's function:

$$f_7(\mathbf{s}) = -(-1)^D \left(\prod_{i=1}^{D} \cos^2(s_i) \right) \exp\left[-\sum_{i=1}^{D} (s_i - \pi)^2 \right], \tag{24}$$

where $s_i = [-2\pi, 2\pi]$. The function has several local minimum and the global minimum $f^* = -1$ at $\mathbf{s}^* = (\pi, \pi, \ldots, \pi)$.

- Michalewicz's function:

$$f_8(\mathbf{s}) = -\sum_{i=1}^{D} \sin(s_i) \left[\sin\left(\frac{i s_i^2}{\pi} \right) \right]^{2 \cdot 10}, \tag{25}$$

where $s_i = [0, \pi]$. The function has the global minimum $f^* = -1.8013$ at $\mathbf{s}^* = (2.20319, 1.57049)$ within two-dimensional parameter space. In general, it has several local optima.

- Xin-She Yang's function:

$$f_9(\mathbf{s}) = \left(\sum_{i=1}^{D} |s_i|\right) \exp\left[-\sum_{i=1}^{D} \sin(s_i^2)\right], \tag{26}$$

where $s_i = [-2\pi, 2\pi]$. The function is not smooth because it has several local optima and the global minimum $f^* = 0$ at $\mathbf{s}^* = (0, 0, \ldots, 0)$.

- Zakharov's function:

$$f_{10}(\mathbf{s}) = \sum_{i=1}^{D} s_i^2 + \left(\frac{1}{2}\sum_{i=1}^{D} i s_i\right)^2 + \left(\frac{1}{2}\sum_{i=1}^{D} i s_i\right)^4, \tag{27}$$

where $s_i = [-5, 10]$. The function has the global minimum $f^* = 0$ at $\mathbf{s}^* = (0, 0, \ldots, 0)$ with no local optima.

The lower and upper bounds of the design variables determine intervals that limit the size of the search space. The wider this interval, the wider the search space. Note that the intervals were selected so that the search space was wider than those proposed in the standard literature. Another difficulty was represented by the dimensions of the functions. Typically, the higher the dimensional function, the more difficult to optimize. Note that functions with dimensions $D = 10$, $D = 30$ and $D = 50$ were employed in our experiments.

4.2 Experimental Setup

The characteristics of RFA algorithms are illustrated in Table 2, from which it can be seen that the specific FA parameters were set according to the propositions in [9], i.e., $\alpha = 0.1$, $\beta = 0.2$, and $\gamma = 0.9$. The population size was set as $NP = 100$, because extensive experiments have shown that this value represents the good balance between exploration and exploitation within the FA search process.

The number of fitness function evaluations depends on a dimension of problem D and was limited to $MAX\_FEs = 1,000 \cdot D$. However, the number of generations used as termination condition in RFA can simply be expressed as $MAX\_GEN = MAX\_FES/NP$. Because all the algorithms have stochastic natures they were run 25 times. The results from these algorithms were accomplished according to five standard measures, as follows: the *Best*, the *Worst*, the *Mean*, the *StDev*, and the *Median* values.

Table 2 Characteristics of the RFA algorithms

Parameter	Designation	Setting
Randomized parameter	α	0.1
Attractiveness	β	0.2
Light absorption	γ	0.9
Population size	NP	100
Maximum number of evaluations	MAX_FEs	$1,000 \cdot D$
Maximum number of generations	MAX_GEN	MAX_FEs/NP
Maximum number of runs	MAX_RUN	25

4.3 PC Configuration

All runs were made on a HP Compaq using the following configurations:

1. Processor—Intel Core i7-2600 3.4 (3.8) GHz
2. RAM—4GB DDR3
3. Operating system—Linux Mint 12

All versions of the tested algorithms were implemented within the Eclipse Indigo CDT framework.

4.4 Results

The following experiments were conducted, in order to show how the various randomized methods influenced the results of the RFA algorithms:

- analyzing the characteristics of the various randomization methods,
- verifying the impact of these randomizing methods on the results of the RFA algorithms,
- comparing the results of the RFA algorithms with other well-known metaheuristics, like BA, DE, and ABC.

The results of the mentioned experiments are observed in the remainder of this chapter.

4.4.1 Characteristics of the Various Randomized Methods

In this experiment, characteristics of the randomized methods, such as:

- Uniform distributed,
- Gaussian distributed,
- Lévy flights,

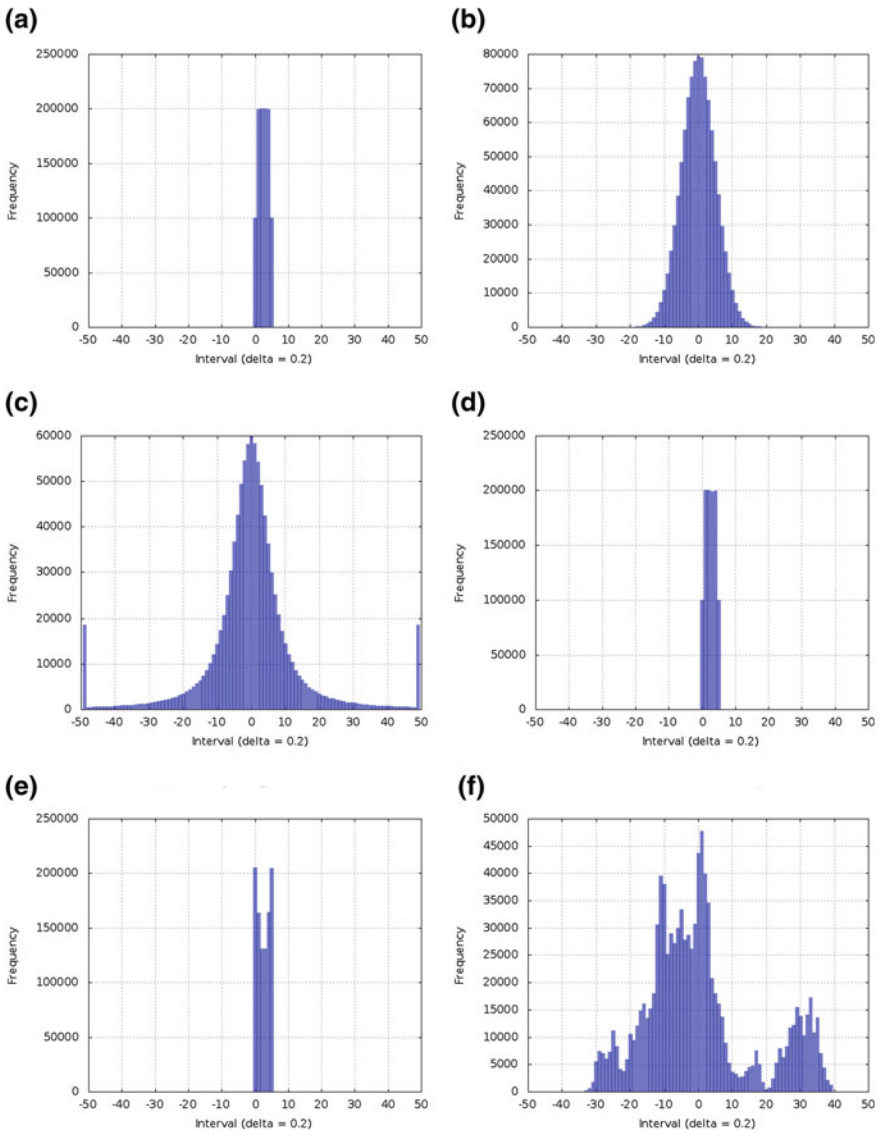

Fig. 1 Results of the various randomized methods

- Kent chaotic maps,
- Logistic chaotic map, and
- Random sampling in turbulent fractal cloud

are taken into account. The results are illustrated in Fig. 1 that is divided into six diagrams. Each of these presents a specific randomized method.

The results diagrams were obtained as follows. A million random numbers were generated for each method. Then, these were represented in a histogram plot that is a natural graphical representation of the distribution of random values. Each histogram consists of intervals coated on the x-axis denoting the value of the statistic variable, and their frequencies coated on the y-axis. Indeed, the range of real values $[-9.9, 9.9]$ is divided into 101 intervals each of width 0.2. The interval zero comprises the range $[-0.1, 0.1]$, whilst intervals -50 and 50 capture values < -9.9 and > 9.9, respectively. However, the total sum of the frequencies is one million.

The following characteristics can be summarized from the presented plots:

1. At first glance, a Kent chaotic map is similar to the uniform distribution, but the frequencies of the intervals slightly differ between each other by the former. On the other hand, Logistic chaotic map is inversion of both previously mentioned, because the border intervals 0 and 5 exhibit higher frequencies than the inner.
2. Gaussian distribution is more compact than Lévy flights, because the latter enables the generating the random numbers that are outside the intervals -50 and 50. This phenomenon can be seen in Fig. 1c as peeks in the first and last intervals.
3. The RSiTFC plot exhibits more peaks. This behavior might be useful by the optimization of multi-modal problems.

In summary, three randomized methods generates random numbers into interval [0, 1], whilst the other three into an interval that is much wider than mentioned. This finding complies with Table 1.

4.4.2 Impact of Randomized Methods on the Results of the RFA Algorithms

In this experiment, an impact of various randomized methods on the results of the RFA algorithms was verified. Therefore, the six variants of the RFA algorithms, i.e., UFA, NFA, LFA, CFA1, CFA2, and FFA were applied to the test suite as defined in Sect. 4.1. Indeed, the experimental setup was employed as presented in Sect. 4.2. Although the functions with dimensions $D = 10$, $D = 30$, and $D = 50$ were optimized, only those results optimizing the functions with dimension $D = 30$ are presented in Table 3, because of the limitation of the chapter's length. The best results are bold in this table.

As can be seen from Table 3, the FFA variant of RFA achieved the best results by optimizing functions f_1, f_2, f_4, f_6, and f_7, the LFA by functions f_5, f_9, and f_{10}, whilst the UFA outperformed the other algorithms by optimizing the function f_3 and the CFA1 by f_8. Indeed, the functions f_1, f_2, and f_4 are highly multi-modal.

The Friedman test was conducted in order to estimate the quality of the results. The Friedman test [29, 30] compares the average ranks of the algorithms. A null-hypothesis states that two algorithms are equivalent and, therefore, their ranks should be equal. If the null-hypothesis is rejected, i.e., the performance of the algorithms is statistically different, the Bonferroni-Dunn test [31] is performed that calculates the critical difference between the average ranks of those two algorithms. When the statistical difference is higher than the critical difference, the algorithms

Table 3 Detailed results of the RFA algorithms (D = 30)

Function	Measure	UFA	NFA	LFA	CFA1	CFA2	FFA
f_1	Mean	6.65E−001	1.08E+000	1.05E+000	6.55E−001	8.89E−001	**3.09E−001**
	Stdev	6.40E−001	1.07E+000	1.05E+000	6.96E−001	8.48E−001	3.32E−001
f_2	Mean	2.44E+002	3.51E+002	4.35E+002	2.30E+002	3.01E+002	**2.18E+002**
	Stdev	2.35E+002	3.53E+002	4.42E+002	2.31E+002	2.94E+002	2.16E+002
f_3	Mean	**1.12E+002**	2.25E+004	2.00E+005	3.95E+002	3.14E+002	5.89E+002
	Stdev	1.01E+002	1.72E+004	2.39E+005	3.71E+002	2.35E+002	5.88E+002
f_4	Mean	2.11E+001	2.05E+001	2.02E+001	2.10E+001	2.10E+001	**1.45E+001**
	Stdev	2.11E+001	2.05E+001	2.02E+001	2.11E+001	2.10E+001	1.43E+001
f_5	Mean	6.78E+003	1.46E+003	**2.55E+002**	7.17E+003	5.44E+003	8.64E+003
	Stdev	6.75E+003	−1.39E+003	−3.37E+002	7.12E+003	5.41E+003	8.45E+003
f_6	Mean	5.19E+000	2.64E+002	9.92E−001	3.67E+000	7.54E+000	**3.61E−001**
	Stdev	5.14E+000	2.63E+002	6.69E−001	3.09E+000	7.20E+000	3.61E−001
f_7	Mean	−3.81E−030	−6.58E−069	−8.56E−014	−1.81E−036	−4.07E−039	**−1.29E−078**
	Stdev	−3.73E−030	−4.94E−069	−9.68E−018	−1.38E−036	−2.59E−039	−4.00E−084
f_8	Mean	−5.15E+000	−1.20E+001	−7.60E+000	**−7.73E+000**	−4.82E+000	−2.60E+000
	Stdev	−5.35E+000	−1.23E+001	−8.24E+000	−7.94E+000	−4.76E+000	−2.63E+000
f_9	Mean	1.70E−004	6.72E−005	**9.11E−006**	1.15E−004	9.21E−005	4.69E−003
	Stdev	4.72E−005	3.13E−005	2.37E−010	5.59E−005	7.81E−005	4.09E−003
f_{10}	Mean	1.32E+004	2.27E+002	**1.12E+002**	6.09E+002	1.41E+003	3.12E+005
	Stdev	1.32E+004	2.28E+002	8.40E+001	6.09E+002	1.41E+003	3.12E+005

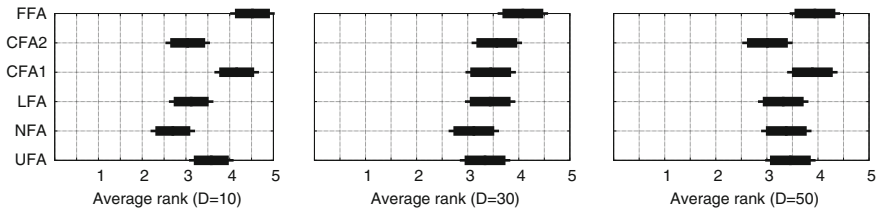

Fig. 2 Results of the Friedman non-parametric test on different variants of RFA algorithms

are significantly different. The equation for the calculation of critical difference can be found in [31].

Friedman tests were performed using a significance level 0.05. The results of the Friedman non-parametric test are presented in Fig. 2 being divided into three diagrams that show the ranks and confidence intervals (critical differences) for the algorithms under consideration. The diagrams are organized according to the dimensions of functions. Two algorithms are significantly different if their intervals in Fig. 2 do not overlap.

The first diagram in Fig. 1 shows that the FFA and KFA variants of RFA significantly outperform the results of all other variants of RFA (i.e., NFA, LFA, CFA2), except UFA, according to dimension $D = 10$. However, the results became insignificant when these were compared in regard to the dimensions $D = 30$ and $D = 50$. In fact, the results of FFA and KFA variants of RFA still remained substantially better, but this difference was not significant.

In summary, we can conclude that the selection of the randomized method has a great impact on the results of RFA. Moreover, in some cases the selection of the more appropriate randomized method can even significantly improve the results of the original FA (Gaussian NFA). Indeed, the best results were observed by the RSiTFC and Kent chaotic map.

4.4.3 Comparative Study

In this experiment, the original FA algorithm (NFA) was compared with other well-known algorithms as bat algorithm (BA), differential evolution (DE), and artificial bees colony (ABC) as well as the FFA algorithm that was exhibited as the most promising variant of the developed RFA algorithms.

The specific BA parameters were set as follows: the loudness $A_0 = 0.5$, the pulse rate $r_0 = 0.5$, minimum frequency $Q_{max} = 0.0$, and maximum frequency $Q_{max} = 0.1$. The DE parameters were configured as follows: the amplification factor of the difference vector $F = 0.9$, and the crossover control parameter $CR = 0.5$. The percentage of onlooker bees for the ABC algorithm was 50 % of the colony,

Table 4 Comparing algorithms (D = 10)

Function	Measure	NFA	FFA	BA	DE	ABC
f_1	Mean	7.19E−001	**6.03E−002**	8.99E+000	2.62E+000	6.26E−001
	Stdev	7.00E−001	6.25E−002	6.44E+000	4.32E−001	1.99E−001
f_2	Mean	6.59E+001	4.29E+001	1.46E+002	9.01E+001	**1.24E+001**
	Stdev	6.99E+001	4.46E+001	7.44E+001	9.10E+000	4.36E+000
f_3	Mean	5.51E+005	**4.08E+001**	8.91E+004	1.53E+004	2.05E+002
	Stdev	9.62E+004	3.36E+001	1.45E+005	8.94E+003	2.62E+002
f_4	Mean	2.02E+001	9.73E+000	1.02E+001	8.55E+000	**4.08E+000**
	Stdev	2.02E+001	9.72E+000	3.23E+000	9.78E−001	9.03E−001
f_5	Mean	1.37E+003	4.00E+003	2.19E+003	1.08E+003	**5.99E+002**
	Stdev	1.38E+004	4.01E+003	2.58E+002	1.26E+002	1.25E+002
f_6	Mean	8.02E+001	**5.88E−002**	2.38E+004	6.90E+003	1.23E+001
	Stdev	8.03E+001	5.66E−002	1.75E+004	2.05E+003	1.68E+001
f_7	Mean	−3.89E−018	**−5.15E−034**	−6.46E−002	−3.96E−001	−1.13E−002
	Stdev	−3.88E−018	−4.11E−034	2.18E−001	1.40E−001	5.67E−002
f_8	Mean	−4.79E+000	**−6.37E−001**	−5.99E+000	−6.69E+000	−8.60E+000
	Stdev	−5.63E+000	−6.38E−001	1.04E+000	3.24E−001	2.89E−001
f_9	Mean	3.49E−002	1.32E−001	1.39E−003	1.92E−003	**6.10E−004**
	Stdev	2.15E−002	1.30E−001	6.95E−004	1.31E−004	5.96E−005
f_{10}	Mean	6.98E+001	4.99E+003	**2.05E+001**	2.93E+001	2.38E+001
	Stdev	6.96E+001	4.99E+003	2.03E+001	8.29E+000	7.80E+000

the employed bees represented another 50 % of the colony, whilst one scout bee was generated in each generation (i.e., *limits* = 100, when the population size is $NP = 100$).

The results of comparing the mentioned algorithms by optimizing the functions with dimension $D = 10$, are presented in Table 4. Again, only one instance of data is illustrated in the table, although the experiments were conducted on all three observed dimensions. The best results of the algorithms are written in bold.

As can be seen from Table 4, the FFA algorithm outperformed the results of the other algorithms when solving the functions f_1, f_3, f_6, f_7, and f_8. Again the majority of these functions are highly multi-modal. The ABC algorithm was the best by solving the functions f_2, f_4, f_5, and f_9, whilst the BA algorithm excellently solved the function f_{10}.

Also here, the Friedman tests using the significance level 0.05 were conducted according to all the observed dimensions of the functions. The results of these tests are presented in Fig. 3, which is divided into three diagrams.

The first diagram illustrates the results of the Friedman test that observes the results obtained by optimizing the functions with dimensions $D = 10$. It can be shown from this diagram that ABC and FFA outperformed the results of all other algorithms in test (i.e., NFA, BA, and DE) significantly. Furthermore, the ABC outperformed the results of the same algorithms when also optimizing the functions with dimension $D = 30$, whilst the FFA algorithm was substantially better than those on the same

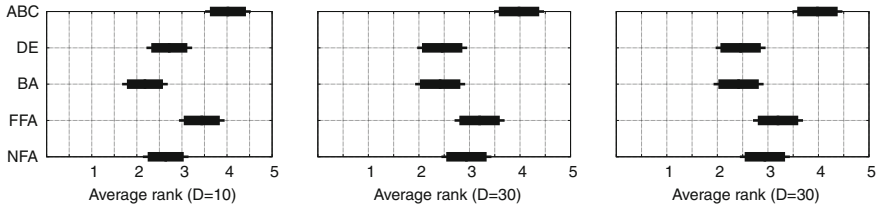

Fig. 3 Results of the Friedman non-parametric test on suite of test algorithms

instances. Finally, on the functions with dimension $D = 50$, the ABC achieved significantly better results than BA and DE. Here, both firefly algorithms exhibited good results.

In summary, the FFA variant of RFA outperforms the results of the original FA algorithm significantly by optimizing the test functions with dimension $D = 10$, and substantially by optimizing the test functions with dimensions $D = 30$ and $D = 50$. Indeed, the ABC algorithm outperforms significantly all the other experiments in tests except FFA. In general, the results of experiments have been shown that the Gaussian distribution method is appropriately selected by the original FA algorithm.

5 Conclusion

In this chapter, an extensive comparison of various probability distributions is performed that can be used to randomize the firefly algorithm, e.g., uniform, Gaussian, Lévi flights, chaos maps and the random sampling in turbulent fractal cloud. In line with this, various firefly algorithms with various randomized methods were developed and extensive experiments were conducted on well-known suite of functions.

The goal of the experiments were threefold. Firstly, the mentioned randomized methods were analyzed. Secondly, an impact of randomized methods on the results of the RFA algorithms were verified. Finally, the results of the original FA algorithm and FFA variant of RFA were compared with the other well-known algorithms like ABC, BA, and DE.

In summary, the selection of an appropriate randomized method has a great impact on the results of RFA. Moreover, this selection depends on the nature of the problem to be solved. On the other hand, selecting the appropriate randomized method can improve the results of the original FA significantly.

In the future, further experiments should be performed with the random sampling in turbulent fractal cloud that exhibits the excellent results especially by optimizing the multi-modal functions.

References

1. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co, New York (1979)
2. Blum, C., Li, X.: Swarm intelligence in optimization. In: Blum, C., Merkle, D. (eds.) Swarm Intelligence: Introduction and Applications, pp. 43–86. Springer, Heidelberg (2008)
3. Beekman, M., Sword, G.A., Simpson, S.J.: Biological foundations of swarm intelligence. In: Blum, C., Merkle, D. (eds.) Swarm Intelligence: Introduction and Applications, pp. 3–41. Springer, Berlin (2008)
4. Beni, G., Wang, J.: Swarm intelligence in cellular robotic systems. Proceedings of NATO Advanced Workshop on Robots and Biological Systems, pp. 26–30. Tuscany, Italy (1989)
5. Dorigo, M., Di Caro, G.: The ant colony optimization meta-heuristic. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 11–32. McGraw Hill, London (1999)
6. Kennedy, J., Eberhart, R.C.: The particle swarm optimization: social adaptation in information processing. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 379–387. McGraw Hill, London (1999)
7. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Global Optim. **39**, 459–471 (2007)
8. Fister, I., Fister, I. Jr., Brest, J., Žumer, V.: Memetic artificial bee colony algorithm for large-scale global optimization. In: IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp. 3038–3045. IEEE Publications (2012)
9. Yang, X.-S.: Firefly algorithm. In: Yang, X.-S. (ed.) Nature-Inspired Metaheuristic Algorithms, pp. 79–90. Wiley Online, Library (2008)
10. Yang, X.-S.: Firefly algorithms for multimodal optimization. In: Stochastic Algorithms: Foundations and Applications, pp. 169–178. Springer, Berlin (2009)
11. Fister, I. Jr., Yang, X.-S., Fister, I., Brest, J.: Memetic firefly algorithm for combinatorial optimization. In: Filipič, B., Šilc, J. (eds.) Bioinspired optimization methods and their applications : proceedings of the Fifth International Conference on Bioinspired Optimization Methods and their Applications—BIOMA 2012, pp. 75–86. Jožef Stefan Institute (2012)
12. Gandomi, A.H., Yang, X.-S., Talatahari, S., Alavi, A.H.: Firefly algorithm with chaos. Commun. Nonlinear Sci. Numer. Simul. **18**(1), 89–98 (2013)
13. Fister, I., Yang, X.-S., Brest, J., Fister Jr, I.: Memetic self-adaptive firefly algorithm. In: Yang, X.-S., Xiao, R.Z.C., Gandomi, A.H., Karamanoglu, M. (eds.) Swarm Intelligence and Bio-Inspired Computation: Theory and Applications, pp. 73–102. Elsevier, Amsterdam (2013)
14. Fister, I., Fister Jr., I., Yang, X.-S., Brest, J.: A comprehensive review of firefly algorithms. Swarm and Evolutionary Computation (2013). Available via ScienceDirect. http://www.sciencedirect.com/science/article/pii/S2210650213000461. Cited 03 Jul 2013
15. Yang, X.-S., Deb, S.: Cuckoo search via Levy flights. In: World Congress on Nature and Biologically Inspired Computing (NaBIC 2009), pp. 210–214. IEEE Publications (2009)
16. Yang, X.-S.: A new metaheuristic bat-inspired algorithm. In: Cruz, C., Gonzlez, J.R., Krasnogor, N., Pelta, D.A., Terrazas, G. (eds.) Nature Inspired Cooperative Strategies for Optimization (NISCO 2010), vol. 284, pp. 65–74. Springer, Berlin (2010)
17. Fister Jr, I., Fister, D., Yang, X.-S.: A Hybrid bat algorithm. Electrotech. Rev. **80**, 1–7 (2013)
18. Hoos, H.H., Stützle, T.: Stochastic local search: Foundations and applications. Morgan Kaufmann, San Francisco (2004)
19. Feldman, D.P.: Chaos and Fractals: An Elementary Introduction. Oxford University Press, Oxford (2012)
20. Črepinšek, M., Mernik, M., Liu, S.H.: Analysis of exploration and exploitation in evolutionary algorithms by ancestry trees. Int. J. Innovative Comput. Appl. **3**, 11–19 (2011)
21. Hertz, A., Taillard, E., de Werra, D.: Tabu search. In: Aarts, E., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, pp. 121–136. Princeton University Press, New Jersey (2003)
22. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Berlin (2003)

23. Galassi, D., et al.: GNU Scientific Library: Reference Manual, Edn. 1.15. Network Theory Ltd, Bristol (2011)
24. Jamil, M.: Zepernick: Lévy flights and global optimization. In: Yang, X.-S., Xiao, R.Z.C., Gandomi, A.H., Karamanoglu, M. (eds.) Swarm Intelligence and Bio-Inspired Computation: Theory and Applications, pp. 49–72. Elsevier, Amsterdam (2013)
25. Zhou, Q., Li, L., Chen, Z.-Q., Zhao, J.-X.: Implementation of LT codes based on chaos. Chin. Phys. B **17**(10), 3609–3615 (2008)
26. Elmegreen, B.G.: The initial stellar mass function from random sampling in a turbulent fractal cloud. Astrophys. J. **486**, 944–954 (1997)
27. Long, S.M., Lewis, S., Jean-Louis, L., Ramos, G., Richmond, J., Jakob, E.M.: Firefly flashing and jumping spider predation. Anim. Behav. **83**, 81–86 (2012)
28. Yang, X.-S.: Appendix A: Test Problems in Optimization. In: Yang, X.-S. (ed.) Engineering Optimization, pp. 261–266. John Wiley and Sons, Inc., New York (2010)
29. Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J. Am. Stat. Assoc. **32**, 675–701 (1937)
30. Friedman, M.: A comparison of alternative tests of significance for the problem of m rankings. An. Math. Stat. **11**, 86–92 (1940)
31. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)

Cuckoo Search: A Brief Literature Review

Iztok Fister Jr., Xin-She Yang, Dušan Fister and Iztok Fister

Abstract Cuckoo search (CS) was introduced by Xin-She Yang and Suash Deb in 2009, and it has attracted great attention due to its promising efficiency in solving many optimization problems and real-world applications. In the last few years, many papers have been published regarding cuckoo search, and the relevant literature has expanded significantly. This chapter summarizes briefly the majority of the literature about cuckoo search in peer-reviewed journals and conferences found so far. These references can be systematically classified into appropriate categories, which can be used as a basis for further research.

Keywords Cuckoo search · Engineering optimization · Metaheuristic · Nature-inspired algorithm · Scheduling

1 Introduction

Since the first introduction of Cuckoo Search (CS) by Xin-She Yang and Suash Deb in 2009 [109], the literature of this algorithm has exploded. Cuckoo search, which drew its inspiration from the brooding parasitism of cuckoo species in Nature, were firstly

I. Fister Jr. (✉) · D. Fister · I. Fister
Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia
e-mail: iztok.fister2@uni-mb.si

D. Fister
e-mail: dusan.fister@uni-mb.si

I. Fister
e-mail: iztok.fister@uni-mb.si

X.-S. Yang
School of Science and Technology, Middlesex University, London, UK
e-mail: x.yang@mdx.ac.uk

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_3,

proposed as a tool for numerical function optimization and continuous problems. Researchers tested this algorithm on some well-known benchmark functions and compared with PSO and GA, and it was found that cuckoo search achieved better results than the results by PSO and GA. Since then, the original developers of this algorithm and many researchers have also applied this algorithm to engineering optimization, where Cuckoo search also showed promising results.

Nowadays cuckoo search has been applied in almost every area and domain of function optimization, engineering optimization, image processing, scheduling, planning, feature selection, forecasting, and real-world applications. A quick search using Google scholar returned 440 papers, while the original paper by Yang and Deb [109] has been cited 223 times at the time of writing of this chapter. A search using Scirus returned 616 hits with 126 journal papers recorded up to July 2013. While many papers may be still in press, it is not possible to get hold of all these papers. Consequently, we will focus on the full papers we can get and thus 114 papers are included in this chapter, which may be one fraction of the true extent of the literature, but they should be representative and useful.

The aim of this chapter is to provide readers with a brief and yet relatively comprehensive list of literature in the last few years. This helps to gain insight into all the major studies concerning this hot and active optimization algorithm. The structure of this chapter is divided in four different parts. Section 2 presents all the main variants of the cuckoo search variants, including those studies that have been carried out in numerical and multi-objective optimization. Hybrid algorithms are also included in this part. Section 3 focuses on engineering optimization, while Sect. 4 summarizes all the major applications and their relevant literature. Then, Sect. 5 discusses implementation and some theoretical studies. Finally, Sect. 6 concludes with some suggestions for further research topics.

2 Cuckoo Search: Variants and Hybrids

2.1 Variants

The original cuckoo search was first tested using numerical function optimization benchmarks. Usually, this kind of problems represents a test bed for new developed algorithms. In line with this, standard benchmark function suites [33, 110] have been developed in order to make comparison between algorithms as fair as possible. For example, some original studies in this area are:

- Cuckoo search via Lévy flights [109].
- An efficient cuckoo search algorithm for numerical function optimization [61].
- Multimodal function optimisation [34].

Cuckoo search can deal with multimodal problems naturally and efficiently. However, researchers have also attempted to improve its efficiency further so as to obtained

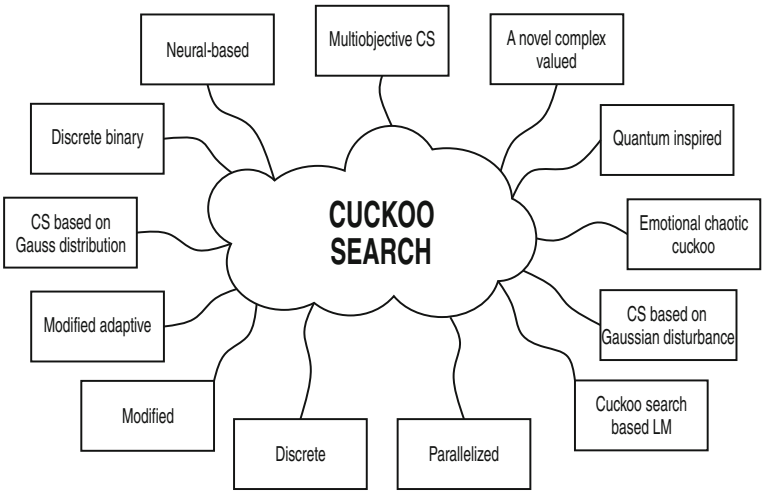

Fig. 1 Variant of cuckoo search

better solutions than those in the literature [20], and one such study that is worth mentioning is by Jamil and Zepernick [34].

Since the first appearance of cuckoo search in 2009, many variants of the cuckoo search algorithm have been developed by many researchers. The major variants are summarized in Fig. 1 and Table 1.

Table 1 Variants of cuckoo search

Name	Author	Reference
Discrete binary CS	Gherboudj et al.	[26]
Discrete CS	Jati and Manurung	[35]
Discrete CS for TSP	Ouaarab et al.	[62]
Neural-based CS	Khan and Sahai	[42]
Quantum inspired CS	Layeb	[46]
Emotional chaotic cuckoo	Lin et al.	[50]
Cuckoo search based LM	Nawi et al.	[60]
Parallelized CS	Subotic et al.	[83]
Modified CS	Tuba et al.	[87]
Modified CS	Walton et al.	[95]
Modified adaptive CS	Zhang et al.	[115]
Multiobjective CS	Yang and Deb	[112]
A novel complex valued	Zhou and Zheng	[117]
CS based on Gauss distribution	Zheng and Zhou	[116]
CS based on Gaussian disturbance	Wang et al.	[99]

Table 2 Hybrid cuckoo search

Name	Author	Reference
Hybrid CS/GA	Ghodrati and Lotfi	[27, 28]
Hybrid CS	Li and Yin	[48]

2.2 Hybrid Algorithms

For many continuous optimization problems, cuckoo search can find the desired solutions very efficiently. However, sometimes, some difficulty may arise, which is true for all nature-inspired algorithms when the appropriate solutions could not be found for some other optimization problems. This is consistent with the so-called No-Free-Lunch theorem [104]. To circumvent this theorem, hybridization has been applied to optimization algorithms for solving a given set of problems. In line with this, cuckoo search has been hybridized with other optimization algorithms, machine learning techniques, heuristics, etc. Hybridization can take place in almost every component of the cuckoo search. For example, initialization procedure, evaluation function, moving function and others have all been tried. Some of the hybrid variants are summarized in Table 2.

2.3 Multi-objective Optimization

Multi-objective optimization consists of more than one objective, and these objectives may be conflicting one another. Many real-world optimization problems require design solutions according to many criteria. Single objective optimization searches for a single optimal solution, whilst multi-objective optimization requires a set of many (potentially infinite), optimal solutions, namely the Pareto front [72, 91]. Obviously, there are many issues and approaches for multi-objective optimization; however, two goals in multi-objective optimization are worth noting:

- to obtain solutions as close to the true Pareto front as possible;
- to generate solutions as diversely as possible on the non-dominated front.

Various variants have been developed to extend the standard cuckoo search into multi-objective cuckoo search. The following list presents some main variants on multi-objective optimization using CS.

- Multi-objective CS [112].
- Multi-objective scheduling problem [9].
- Multi-objective cuckoo search algorithm for Jiles-Atherton vector hysteresis parameters estimation [14].
- Pareto archived cuckoo search [32].
- Hybrid multiobjective optimization using modified cuckoo search algorithm in linear array synthesis [68].
- Multi-objective cuckoo search for water distribution systems [103].

3 Engineering Optimization

Among the diverse applications of cuckoo search, by far the largest fraction of literature may have focused on the engineering design applications. In fact, cuckoo search and its variants have become a crucial technology for solving problems in engineering practice as shown in Fig. 2. Nowadays, there are applications from almost every engineering domain. Some of these research papers are summarized in Table 3.

4 Applications

Obviously, engineering optimization is just part of the diverse applications. In fact, cuckoo search and its variants have been applied into almost every area of sciences, engineering and industry. Some of the application studies are summarized in Fig. 3 and also in Table 4.

5 Theoretical Analysis and Implementation

As we have seen, the applications of cuckoo search are very diverse. In contrast, the theoretical studies are very limited. This brief summary may highlight the need for further research in theoretical aspects of cuckoo search.

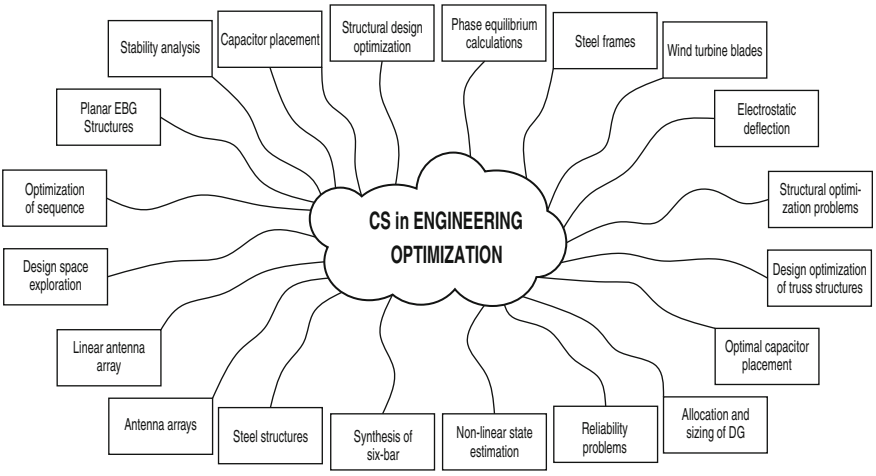

Fig. 2 Engineering optimization

Table 3 Cuckoo search in engineering optimization

Problem	Author	Reference
Engineering optimization	Yang and Deb	[110]
Capacitor placement	Arcanjo et al.	[3]
Synthesis of six-bar	Bulatovič et al.	[8]
Wind turbine blades	Ernst et al.	[22]
Design of truss structures	Gandomi et al.	[24]
Structural optimization problems	Gandomi et al.	[25]
Electrostatic deflection	Goghrehabadi et al.	[30]
Steel frames	Kaveh and Bakhspoori	[39]
Steel structures	Kaveh et al.	[40]
Antenna arrays	Khodier	[43]
Design space exploration	Kumar and Chakarverty	[44, 45]
Optimization of Sequence	Lim et al.	[49]
Planar EBG Structures	Pain et al.	[63]
Stability analysis	Rangasamy and Manickam	[66]
Linear antenna array	Rani and Malek	[67, 69]
Optimal Capacitor Placement	Reddy and Manohar	[70]
Allocation and sizing of DG	Tan et al.	[85]
Reliability problems	Valian et al.	[88, 89]
Non-linear state estimation	Walia and Kapoor	[93]
Phase equilibrium calculations	Bhargava et al.	[6]
Phase equilibrium (comments)	Walton et al.	[96]
Structural design optimization	Durgun and Yildiz	[19]

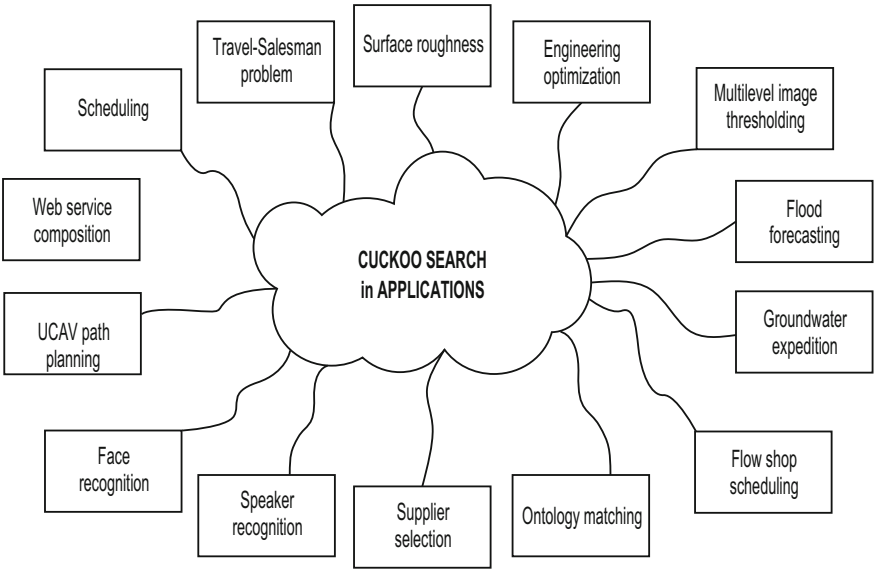

Fig. 3 Cuckoo search in applications

Table 4 Cuckoo search in applications

Application	Author	Reference
Multilevel image thresholding	Brajevic et al.	[7]
Flood forecasting	Chaowanawatee and Heednacram	[10]
Wireless sensor networks	Dhivya and Sundarambal	[16]
Data fusion	Dhivya et al.	[17]
Cluster in wireless networks	Dhivya et al.	[18]
Clustering	Goel et al.	[29]
Groundwater expedition	Gupta et al.	[31]
Supplier selection	Kanagaraj et al.	[38]
Load forecasting	Kavousi-Fard and Kavousi-Fard	[41]
Surface roughness	Madic et al.	[51]
Flow shop scheduling	Marichelvam	[52]
Optimal replacement	Mellal et al.	[53]
DG allocation in network	Moravej and Akhlaghi	[54]
Optimization of bloom filter	Natarajan et al.	[56–58]
BPNN neural network	Nawi et al.	[59]
Travelling salesman problem	Ouaarab et al.	[62]
Web service composition	Pop et al.	[64]
Web service composition	Chifu et al.	[11, 12]
Ontology matching	Ritze and Paulheim	[71]
Speaker recognition	Sood and Kaur	[77]
Automated software testing	Srivastava et al.	[80–82]
Manufacturing optimization	Syberfeldt and Lidberg	[84]
Face recognition	Tiwari	[86]
Training neural models	Vázquez	[90]
Non-convex economic dispatch	Vo et al.	[92]
UCAV path planning	Wang et al.	[101, 102]
Business optimization	Yang et al.	[113]
Machining parameter selection	Yildiz	[114]
Job scheduling in grid	Prakash et al.	[65]
Quadratic assignment	Dejam et al.	[15]
Sheet nesting problem	Elkeran	[21]
Query optimization	Joshi and Srivastava	[36]
n-Queens puzzle	Sharma and Keswani	[74]
Computer games	Speed	[78, 79]

5.1 Theory and Algorithm Analysis

It may be difficult to decide if a study should be classified into a theoretical category or not because the contents may sometime include both simulations and some analysis of the algorithm. So the following categorization may not be rigorous. Even so, some theoretical studies about cuckoo search in the current literature can be summarized, as follows:

- A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms [13].
- Enhancing the performance of cuckoo search algorithm using orthogonal learning method [47].
- Starting configuration of cuckoo search algorithm using centroidal Voronoi tessellations [75].
- Reduced order mesh optimisation using proper orthogonal decomposition and a modified cuckoo search [94, 97].
- Bat algorithm and cuckoo search: a tutorial [106, 107].
- Metaheuristic algorithms for inverse problems [105, 108, 111].
- Markov model and convergence analysis of cuckoo search [100].
- Towards the improvement of cuckoo search algorithm [76].

5.2 Improvements and Other Studies

As mentioned earlier, it is not always clear how to classify certain papers. Many research studies concern the improvements of the standard cuckoo search algorithm. So we loosely put some papers here and thus summarized them as follows:

- Tsallis entropy [1].
- Improved scatter search using cuckoo search [2].
- Cuckoo search via Lévy flights for optimization of a physically-based runoff-erosion model [23].
- Improved differential evolution via cuckoo search operator [55].
- Cuckoo search with the conjugate gradient method [73].
- Cuckoo search with PSO [98].

5.3 Implementations

Whatever the algorithms may be, proper implementations are very important. Yang provided a standard demo implementation of cuckoo search.[1] Important implementations such as object-oriented approach and parallelization have been carried out, which can be summarized as follows:

- Object oriented implementation of CS [4, 5].
- Parallelization of CS [37].

[1] http://www.mathworks.co.uk/matlabcentral/fileexchange/29809-cuckoo-search-cs-algorithm

6 Conclusion

In this brief review, a relatively comprehensive bibliography regarding cuckoo search algorithm has been presented. References have been systematically sorted into proper categories. The rapidly expanding literature implies that cuckoo search is a very active, hot research area. There is no doubt that more studies on cuckoo search will appear in the near future.

From the above review, it is worth pointing out that there are some important issues that need more studies. One thing is that theoretical analysis should be carried out so that insight can be gained into various variants of the cuckoo search algorithm. In addition, it may be very useful to carry out parameter tuning in some efficient variants and see how parameters can affect the behaviour of an algorithm. Furthermore, applications should focus on large-scale real-world applications.

References

1. Agrawal, S., Panda, R., Bhuyan, S., Panigrahi, B.K.: Tsallis entropy based optimal multilevel thresholding using cuckoo search algorithm. Swarm Evol. Comput. **11**(1):16–30 (2013)
2. Sadiq Al-Obaidi, A.T.: Improved scatter search using cuckoo search. Int. J. Adv. Res. Artif. Intell. **2**(2):61–67 (2013)
3. Arcanjo, D.J., Pereira, J.L.R., Oliveira, E.J., Peres, W., de Oliveira, L.W., da Silva, I.C., Jr.: Cuckoo search optimization technique applied to capacitor placement on distribution system problem. In: 10th IEEE/IAS International Conference on Industry Applications (INDUSCON), 2012, pp. 1–6 IEEE (2012)
4. Bacanin, N.: An object-oriented software implementation of a novel cuckoo search algorithm. In: Proceedings of the 5th European Conference on European Computing Conference (ECC11), pp. 245–250 (2011)
5. Bacanin, N.: Implementation and performance of an object-oriented software system for cuckoo search algorithm. Int. J. Math. Comput. Simul. **6**(1), 185–193 (2012)
6. Bhargava, V., Fateen, S.E.K., Bonilla-Petriciolet, A.: Cuckoo search: a new nature-inspired optimization method for phase equilibrium calculations. Fluid Phase Equilib. **337**, 191–200 (2013)
7. Brajevic, I., Tuba, M., Bacanin, N.: Multilevel image thresholding selection based on the cuckoo search algorithm. In: Proceedings of the 5th International Conference on Visualization, Imaging and Simulation (VIS'12), Sliema, Malta, pp. 217–222 (2012)
8. Bulatović, R.R., Djordjević, S.R., Djordjević, V.S.: Cuckoo search algorithm: a metaheuristic approach to solving the problem of optimum synthesis of a six-bar double dwell linkage. Mech. Mach. Theory **61**, 1–13 (2013)
9. Chandrasekaran, K., Simon, S.P.: Multi-objective scheduling problem: Hybrid approach using fuzzy assisted cuckoo search algorithm. Swarm Evol. Comput. **5**, 1–16 (2012)
10. Chaowanawatee, K., Heednacram, A.: Implementation of cuckoo search in rbf neural network for flood forecasting. In: 2012 Fourth International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), pp. 22–26. IEEE (2012)
11. Chifu, V.R., Pop, C.B., Salomie, I., Dinsoreanu, M., Niculici, A.N., Suia, D.S.: Bio-inspired methods for selecting the optimal web service composition: Bees or cuckoos intelligence? Int. J. Bus. Intell. Data Min. **6**(4), 321–344 (2011)
12. Chifu, V.R., Pop, C.B., Salomie, I., Suia, D.S., Niculici, A.N.: Optimizing the semantic web service composition process using cuckoo search. In: Intelligent Distributed Computing V, pp. 93–102. Springer (2012)

13. Civicioglu, P., Besdok, E.: A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artif. Intell. Rev. **39**(4), 315–346 (2013)
14. Coelho, L.S., Guerra, F.A., Batistela, N.J., Leite, J.V.: Multiobjective cuckoo search algorithm based on duffings oscillator applied to jiles-atherton vector hysteresis parameters estimation. IEEE Trans. Magn. **49**(5), 1745 (2013)
15. Dejam, S., Sadeghzadeh, M., Mirabedini, S.J.: Combining cuckoo and tabu algorithms for solving quadratic assignment problems. J. Acad. Appl. Stud. **2**(12), 1–8 (2012)
16. Dhivya, M., Sundarambal, M.: Cuckoo search for data gathering in wireless sensor networks. Int. J. Mob. Commun. **9**(6), 642–656 (2011)
17. Dhivya, M., Sundarambal, M., Anand, L.N.: Energy efficient computation of data fusion in wireless sensor networks using cuckoo based particle approach (cbpa). IJCNS **4**(4), 249–255 (2011)
18. Dhivya, M., Sundarambal, M., Vincent, J.O.: Energy efficient cluster formation in wireless sensor networks using cuckoo search. In: Swarm, Evolutionary, and Memetic Computing, pp. 140–147. Springer (2011)
19. Durgun, I., Yildiz, A.R.: Structural design optimization of vehicle components using cuckoo search algorithm. MP Mater. Test. **54**(3), 185 (2012)
20. Eiben, A.E., Smith, J.E.: Introduction to evolutionary computing (natural computing series). Springer, Berlin (2003)
21. Elkeran, A.: A new approach for sheet nesting problem using guided cuckoo search and pairwise clustering. Eur. J. Oper. Res. (2013) http://dx.doi.org/10.1016/j.ejor.2013.06.020
22. Ernst, B., Bloh, M., Seume, J.R., González, A.G.: Implementation of the cuckoo search algorithm to optimize the design of wind turbine rotor blades. In: Proceedings of the European Wind Energy Association (EWEA) 2012 Annual Event. Copenhagen, Denmark:[sn] (2012)
23. Freire, P.K.M.M., Santos, C.A.G., Mishra, S.K.: Cuckoo search via lévy flights for optimization of a physically-based runoff-erosion model. J. Urban Env. Eng. **6**(2), 123–131 (2012)
24. Gandomi, A.H., Talatahari, S., Yang, X.-S., Deb, S.: Design optimization of truss structures using cuckoo search algorithm. Str. Des. Tall Spec. Build. (2012). doi:10.1002/tal.1033
25. Gandomi, A.H., Yang, X.-S., Alavi, A.H.: Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng. Comput. **29**(1), 17–35 (2013)
26. Gherboudj, Amira, Layeb, Abdesslem, Chikhi, Salim: Solving 0–1 knapsack problems by a discrete binary version of cuckoo search algorithm. Int. J. Bio-Inspired Comput. **4**(4), 229–236 (2012)
27. Ghodrati, A., Lotfi, S.: A hybrid cs/ga algorithm for global optimization. In: Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011), 20–22 December 2011, pp. 397–404. Springer (2012)
28. Ghodrati, A., Lotfi, S.: A hybrid cs/pso algorithm for global optimization. In: Intelligent Information and Database Systems, pp. 89–98. Springer (2012)
29. Goel, S., Sharma, A., Bedi, P.: Cuckoo search clustering algorithm: a novel strategy of biomimicry. In: World Congress on Information and Communication Technologies (WICT), 2011, pp. 916–921. IEEE (2011)
30. Goghrehabadi, A., Ghalambaz, M., Vosough, A.: A hybrid power series-cuckoo search optimization algorithm to electrostatic deflection of micro fixed-fixed actuators. Int. J. Multidiscip. Sci. Eng. **2**(4), 22–26 (2011)
31. Gupta, D., Das, B., Panchal, V.K.: Applying case based reasoning in cuckoo search for the expedition of groundwater exploration. In: Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012), pp. 341–353. Springer (2013)
32. Hanoun, S., Nahavandi, S., Creighton, D., Kull, H.: Solving a multiobjective job shop scheduling problem using pareto archived cuckoo search. In: IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA), 2012, pp. 1–8. IEEE (2012)
33. Jamil, M., Yang, X.-S.: A literature survey of benchmark functions for global optimisation problems. Int. J. Math. Model. Numer. Optim. **4**(2), 150–194 (2013)

34. Jamil, M., Zepernick, H.: Multimodal function optimisation with cuckoo search algorithm. Int. J. Bio-Inspired Comput. **5**(2), 73–83 (2013)
35. Jati, G.K., Manurung, H.M., Suyanto, S.: Discrete cuckoo search for traveling salesman problem. In: 7th International Conference on Computing and Convergence Technology (ICCCT2012), pp. 993–997. IEEE (2012)
36. Joshi, M., Srivastava, P.R.: Query optimization: an intelligent hybrid approach using cuckoo and tabu search. Int. J. Intell. Inf. Technol. (IJIIT) **9**(1), 40–55 (2013)
37. Jovanovic, R., Tuba, M., Brajevic, I.: Parallelization of the cuckoo search using cuda architecture. In: Institute of Physics Recent Advances in Mathematics, pp. 137–142 (2013)
38. Kanagaraj, G., Ponnambalam, S.G., Jawahar, N.: Supplier selection: reliability based total cost of ownership approach using cuckoo search. In: Trends in Intelligent Robotics, Automation, and Manufacturing, pp. 491–501. Springer (2012)
39. Kaveh, A., Bakhshpoori, T.: Optimum design of steel frames using cuckoo search algorithm with lévy flights. Str. Des. Tall Spec. Build. (2011). doi:10.1002/tal.754
40. Kaveh, A., Bakhshpoori, T., Ashoory, M.: An efficient optimization procedure based on cuckoo search algorithm for practical design of steel structures. Iran Univ. Sci. Technol. **2**(1), 1–14 (2012)
41. Kavousi-Fard, A., and Kavousi-Fard, F.: A new hybrid correction method for short-term load forecasting based on arima, svr and csa. J. Exp. Theor. Artif. Intell. **2013**(ahead-of-print), 1–16 (2013)
42. Khan, K., Sahai, A.: Neural-based cuckoo search of employee health and safety (hs). Int. J. Intell. Syst. Appl. (IJISA) **5**(2), 76–83 (2013)
43. Khodier, M.: Optimisation of antenna arrays using the cuckoo search algorithm. IET Microwaves Antennas Propag. **7**(6), 458–464 (2013)
44. Kumar, A., Chakarverty, S.: Design optimization for reliable embedded system using cuckoo search. In: 3rd International Conference on Electronics Computer Technology (ICECT), 2011, vol. 1, pp. 264–268. IEEE (2011)
45. Kumar, A., Chakarverty, S.: Design optimization using genetic algorithm and cuckoo search. In: IEEE International Conference on Electro/Information Technology (EIT), 2011, pp. 1–5. IEEE (2011)
46. Layeb, A.: A novel quantum inspired cuckoo search for knapsack problems. Int. J. Bio-Inspired Comput. **3**(5), 297–305 (2011)
47. Li, X., Wang, J., Yin, M.: Enhancing the performance of cuckoo search algorithm using orthogonal learning method. Neural Comput. Appl. **23**, 1–15 (2013)
48. Li, X., Yin, M.: A hybrid cuckoo search via lévy flights for the permutation flow shop scheduling problem. Int. J. Prod. Res. 2013(ahead-of-print), 1–23 (2013)
49. Lim, W.C.E., Kanagaraj, G., Ponnambalam, S.G.: Cuckoo search algorithm for optimization of sequence in pcb holes drilling process. In: Emerging Trends in Science, Engineering and Technology, pp. 207–216. Springer (2012)
50. Lin, J.H., Lee, H.C., et al.: Emotional chaotic cuckoo search for the reconstruction of chaotic dynamics. Latest advances in systems science & computational intelligence. WSEAS Press, Athens (2012)
51. Madić, M., Radovanović, M.: Application of cuckoo search algorithm for surface roughness optimization in co_2 laser cutting. Ann. Fac. Eng. Hunedoara Int. J. Eng. **11**(1), 39–44 (2013)
52. Marichelvam, M.K.: An improved hybrid cuckoo search (ihcs) metaheuristics algorithm for permutation flow shop scheduling problems. Int. J. Bio-Inspired Comput. **4**(4), 200–205 (2012)
53. Mellal, M.A., Adjerid, S., Williams, E.J., Benazzouz, D.: Optimal replacement policy for obsolete components using cuckoo optimization algorithm based-approach: dependability context. J. Sci. Ind. Res. **71**, 715–721 (2012)
54. Moravej, Z., Akhlaghi, A.: A novel approach based on cuckoo search for dg allocation in distribution network. Int. J. Electr. Power Energy Syst. **44**(1), 672–679 (2013)
55. Musigawan, P., Chiewchanwattana, S., Sunat, K.: Improved differential evolution via cuckoo search operator. In: Neural Information Processing, pp. 465–472. Springer (2012)

56. Natarajan, A., Subramanian, P.K., et al.: An enhanced cuckoo search for optimization of bloom filter in spam filtering. Global J. Comput. Sci. Technol. **12**(1), 1–9 (2012)
57. Natarajan, A., Subramanian, S.: Bloom filter optimization using cuckoo search. In: International Conference on Computer Communication and Informatics (ICCCI), 2012, pp. 1–5. IEEE (2012)
58. Natarajan, A., Subramanian, S., Premalatha, K.: A comparative study of cuckoo search and bat algorithm for bloom filter optimisation in spam filtering. Int. J. Bio-Inspired Comput. **4**(2), 89–99 (2012)
59. Nawi, N.M., Khan, A., Rehman, M.Z.: A new back-propagation neural network optimized with cuckoo search algorithm. In: Computational Science and Its Applications-ICCSA 2013, pp. 413–426. Springer (2013)
60. Nawi, N.M., Khan, A., Rehman, M.Z.: A new cuckoo search based levenberg-marquardt (cslm) algorithm. In: Computational Science and Its Applications-ICCSA 2013, pp. 438–451. Springer (2013)
61. Ong, P., Zainuddin, Z.: An efficient cuckoo search algorithm for numerical function optimization. In: AIP Conference Proceedings, vol. 1522, p. 1378 (2013)
62. Ouaarab, A., Ahiod, B., Yang, X.-S.: Discrete cuckoo search algorithm for the travelling salesman problem. Neural Comput. Appl. 1–11. Springer (2013)
63. Pani, P.R., Nagpal, R.K., Malik, R., Gupta, N.: Design of planar ebg structures using cuckoo search algorithm for power/ground noise suppression. Prog. Electromagn. Res. M **28**, 145–155 (2013)
64. Pop, C.B., Chifu, V.R., Salomie, I., Vlad, M.: Cuckoo-inspired hybrid algorithm for selecting the optimal web service composition. In: IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2011, pp. 33–40. IEEE (2011)
65. Prakash, M., Saranya, R., Jothi, K.R., Vigneshwaran, A.: An optimal job scheduling in grid using cuckoo algorithm. Int. J. Comput. Sci. Telecomm. **3**(2), 65–69 (2012)
66. Rangasamy, S., Manickam, P.: Stability analysis of multimachine thermal power systems using nature inspired modified cuckoo search algorithm. Turk. J. Electr. Eng. Comput. Sci. (2013). doi:10.3906/elk-1212-39
67. Abdul Rani, K.N., Abd Malek, M.F., Neoh, S.: Nature-inspired cuckoo search algorithm for side lobe suppression in a symmetric linear antenna array. Radioengineering **21**(3), 865 (2012)
68. Rani, K.N., Malek, M.F.A., Neoh, S.C., Jamlos, F., Affendi, N.A.M., Mohamed, L., Saudin, N., Rahim, H.A.: Hybrid multiobjective optimization using modified cuckoo search algorithm in linear array synthesis. In: Antennas and Propagation Conference (LAPC), 2012 Loughborough, pp. 1–4. IEEE (2012)
69. Abdul Rani, K.N., Malek, F.: Symmetric linear antenna array geometry synthesis using cuckoo search metaheuristic algorithm. In: 17th Asia-Pacific Conference on Communications (APCC), 2011, pp. 374–379. IEEE (2011)
70. Usha Reddy, V., Manohar, T.G.: Optimal capacitor placement for loss reduction in distribution systems by using cuckoo search algorithm. ITSI Trans. Electr. Electron. Eng. (ITST-TEEE) **1**(2), 68–70 (2013)
71. Ritze, D., Paulheim, H.: Towards an automatic parameterization of ontology matching tools based on example mappings. In: Proceedings of the Sixth International Workshop on Ontology Matching at ISWC, vol. 814, p. 37 (2011)
72. Robič, T., Filipič, B.: Demo: Differential evolution for multiobjective optimization. In: Evolutionary Multi-Criterion Optimization, pp. 520–533. Springer (2005)
73. Salimi, H., Giveki, D., Soltanshahi, M.A., Hatami, J.: Extended mixture of mlp experts by hybrid of conjugate gradient method and modified cuckoo search. arXiv, preprint arXiv:1202.3887 (2012)
74. Sharma, R.G., Keswani, B.: Impelementation of n-queens puzzle using metaheuristic algorithm (cuckoo search). Int. J. Latest Trends Eng. Technol. (IJLTET) **2**(2), 343–347 (2013)
75. Shatnawi, M., Nasrudin, M.F.: Starting configuration of cuckoo search algorithm using centroidal voronoi tessellations. In: 11th International Conference on Hybrid Intelligent Systems (HIS), 2011, pp. 40–45. IEEE (2011)

76. Soneji, H., Sanghvi, R.C.: Towards the improvement of cuckoo search algorithm. In: World Congress on Information and Communication Technologies (WICT), 2012, pp. 878–883. IEEE (2012)
77. Sood, M., Kaur, G.: Speaker recognition based on cuckoo search algorithm. Int. J. Innovative Technol. Explor. Eng. (IJITEE) **2**(5), 311–313 (2013)
78. Speed, E.R.: Evolving a mario agent using cuckoo search and softmax heuristics. In: International IEEE Consumer Electronics Society's Games Innovations Conference (ICE-GIC), 2010, pp. 1–7. IEEE (2010)
79. Speed, E.: Artificial intelligence for games. US Patent App. 13/309,036, 1 Dec 2011
80. Srivastava, P.R., Khandelwal, R., Khandelwal, S., Kumar, S., Ranganatha, S.S.: Automated test data generation using cuckoo search and tabu search (csts) algorithm. J. Intell. Syst. **21**(2), 195–224 (2012)
81. Srivastava, P.R., Singh, A.K., Kumhar, H., Jain, M.: Optimal test sequence generation in state based testing using cuckoo search. Int. J. Appl. Evol. Comput. (IJAEC) **3**(3), 17–32 (2012)
82. Srivastava, P.R., Varshney, A., Nama, P., Yang, X.-S.: Software test effort estimation: a model based on cuckoo search. Int. J. Bio-Inspired Comput. **4**(5), 278–285 (2012)
83. Subotic, M., Tuba, M., Bacanin, N., Simian, D.: Parallelized cuckoo search algorithm for unconstrained optimization. In: Proceedings of the 5th WSEAS Congress on Applied Computing Conference, and Proceedings of the 1st International Conference on Biologically Inspired Computation, pp. 151–156. World Scientific and Engineering Academy and Society (WSEAS) (2012)
84. Syberfeldt, A., Lidberg, S.: Real-world simulation-based manufacturing optimization using cuckoo search. In: Proceedings of the Winter Simulation Conference, pp. 1–12. Winter Simulation Conference (2012)
85. Tan, W.S., Hassan, M.Y., Majid, M.S., Rahman, H.A.: Allocation and sizing of dg using cuckoo search algorithm. In: IEEE International Conference on Power and Energy (PECon), 2012, pp. 133–138. IEEE (2012)
86. Tiwari, V.: Face recognition based on cuckoo search algorithm. Image **7**(8), 9 (2012)
87. Tuba, M., Subotic, M., Stanarevic, N.: Modified cuckoo search algorithm for unconstrained optimization problems. In: Proceedings of the 5th European Conference on European Computing Conference, pp. 263–268. World Scientific and Engineering Academy and Society (WSEAS) (2011)
88. Valian, E., Tavakoli, S., Mohanna, S., Haghi, A.: Improved cuckoo search for reliability optimization problems. Comput. Ind. Eng. **64**(1), 459–468 (2013)
89. Valian, E., Valian, E.: A cuckoo search algorithm by lévy flights for solving reliability redundancy allocation problems. Eng. Optim. (ahead-of-print)1–14 (2012) doi:10.1080/0305215X.2012.729055
90. Vázquez, R.A.: Training spiking neural models using cuckoo search algorithm. In: IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 679–686. IEEE (2011)
91. Veldhuizen, D.A.V., Lamont, G.B.: Multiobjective evolutionary algorithms: analyzing the state-of-the-art. Evol. Comput. **8**(2), 125–147 (2000)
92. Vo, D.N., Schegner, P., Ongsakul, W: Cuckoo search algorithm for non-convex economic dispatch. IET Gener. Transm. Distrib. **7**(6), 645–654 (2013)
93. Walia, G.S., Kapoor, R.: Particle filter based on cuckoo search for non-linear state estimation. In: IEEE 3rd International Advance Computing Conference (IACC), 2013, pp. 918–924. IEEE (2013)
94. Walton, S., Hassan, O., Morgan, K.: Reduced order mesh optimisation using proper orthogonal decomposition and a modified cuckoo search. Int. J. Numer. Meth. Eng. **93**(5), 527–550 (2013)
95. Walton, S., Hassan, O., Morgan, K., Brown, M.R.: Modified cuckoo search: a new gradient free optimisation algorithm. Chaos, Solitons Fractals **44**(9), 710–718 (2011)
96. Walton, S., Brown, M.R., Hassan, O., Morgan, K.: Comment on cuckoo search: A new nature-inspired optimization method for phase equilibrium calculation by v. bhargava, s. fateen, a. bonilla-petriciolet. Fluid Phase Equilib. **352**, 64–64 (2013)

97. Walton, S., Hassan, O., Morgan, K.: Selected engineering applications of gradient free optimisation using cuckoo search and proper orthogonal decomposition. Arch. Comput. Methods Eng. **20**(2), 123–154 (2013)
98. Wang, F., He, X.-S., Luo, L., Wang, Y.: Hybrid optimization algorithm of pso and cuckoo search. In: 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011, pp. 1172–1175. IEEE (2011)
99. Wang, F., He, X.-S., Wang, Y.: The cuckoo search algorithm based on gaussian disturbance. J. Xi'an Polytech. Univ. **4**, 027 (2011)
100. Wang, F., He, X.-S., Wang, Y., Yang, S.-M.: Markov model and convergence analysis based on cuckoo search algorithm. Jisuanji Gongcheng/ Comput. Eng. **38**(11), 180–182 (2012)
101. Wang, G., Guo, L., Duan, H., Liu, L., Wang, H., Wang, B.: A hybrid meta-heuristic de/cs algorithm for ucav path planning. J. Inf. Comput. Sci. **5**(16), 4811–4818 (2012)
102. Wang, G., Guo, L., Duan, H., Wang, H., Liu, L., Shao, M.: A hybrid metaheuristic de/cs algorithm for ucav three-dimension path planning. Sci. World J. (2012) doi:10.1100/2012/583973
103. Wang, Q., Liu, S., Wang, H., Savić, D.A.: Multi-objective cuckoo search for the optimal design of water distribution systems. In: Civil Engineering and Urban Planning 2012, pp. 402–405. ASCE (2012)
104. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)
105. Yang, X.-S.: Cuckoo search for inverse problems and simulated-driven shape optimization. J. Comput. Methods Sci. Eng. **12**(1), 129–137 (2012)
106. Yang, X.-S.: Metaheuristic algorithms for self-organizing systems: a tutorial. In: IEEE Sixth Interational Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012), pp. 249–250. IEEE Conference Publications (2012)
107. Yang, X.-S.: Bat algorithm and cuckoo search: a tutorial. In: Artificial Intelligence, Evolutionary Computing and Metaheuristics, pp. 421–434. Springer (2013)
108. Yang, X.-S.: Metaheuristic algorithms for inverse problems. Int. J. Innovative Comput. Appl. **5**(2), 76–84 (2013)
109. Yang, X.-S., Deb, S.: Cuckoo search via lévy flights. In: World Congress on Nature & Biologically Inspired Computing, 2009. NaBIC 2009, pp. 210–214. IEEE (2009)
110. Yang, X.-S., Deb, S.: Engineering optimisation by cuckoo search. Int. J. Math. Model. Numer. Optim. **1**(4), 330–343 (2010)
111. Yang, X.-S., Deb, S.: Cuckoo search for inverse problems and topology optimization. In: Proceedings of International Conference on Advances in Computing, pp. 291–295. Springer (2012)
112. Yang, X.-S., Deb, S.: Multiobjective cuckoo search for design optimization. Comput. Oper. Res. **40**(6), 1616–1624 (2013)
113. Yang, X.-S., Deb, S., Karamanoglu, M., He, X.: Cuckoo search for business optimization applications. In: Computing and Communication Systems (NCCCS), 2012, pp. 1–5. IEEE (2012)
114. Yildiz, A.R.: Cuckoo search algorithm for the selection of optimal machining parameters in milling operations. Int. J. Adv. Manuf. Technol. **64**(1–4), 55–61 (2013)
115. Zhang, Y., Wang, L.: Modified adaptive cuckoo search (macs) algorithm and formal description for global optimisation. Int. J. Comput. Appl. Technol. **44**(2), 73–79 (2012)
116. Zheng, H., Zhou, Y.: A novel cuckoo search optimization algorithm based on gauss distribution. J. Comput. Inf. Syst. **8**, 4193–4200 (2012)
117. Zhou, Y., Zheng, H.: A novel complex valued cuckoo search algorithm. Sci. World J. (2013) doi:10.1155/2013/597803

Improved and Discrete Cuckoo Search for Solving the Travelling Salesman Problem

Aziz Ouaarab, Belaïd Ahiod and Xin-She Yang

Abstract Improved and Discrete Cuckoo Search (DCS) algorithm for solving the famous travelling salesman problem (TSP), an NP-hard combinatorial optimization problem, is recently developed by Ouaarab, Ahiod, and Yang in 2013, based on the cuckoo search (CS), developed by Yang and Deb in 2009. DCS first reconstructs the population of CS by introducing a new category of cuckoos in order to improve its search efficiency, and adapts it to TSP based on the terminology used either in inspiration source of CS or in its continuous search space. The performance of the proposed DCS is tested against a set of benchmarks of symmetric TSP from the well-known TSPLIB library. The results of the tests show that DCS is superior to some other metaheuristics.

Keywords Nature-inspired metaheuristic · Cuckoo search · Lévy flights · Tour improvement algorithm · Combinatorial optimization · Travelling salesman problem

1 Introduction

Combinatorial optimization problems are to find one (or more) best feasible solutions (which can be combinations, sequences, choice of objects, subsets, subgraphs, etc.), in a finite or countable infinite set. The criterion of best solution is defined by an objective function. As cited by Hochbaum [17], the most difficult

A. Ouaarab (✉) · B. Ahiod
LRIT, Associated Unit to the CNRST (URAC 29), Mohammed V-Agdal University,
B.P. 1014 Rabat, Morocco
e-mail: aziz.ouaarab@gmail.com

B. Ahiod
e-mail: ahiod@fsr.ac.ma

X.-S. Yang
School of Science and Technology, Middlesex University, The Burroughs,
London NW4 4BT, UK
e-mail: x.yang@mdx.ac.uk

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_4,

combinatorial optimization problems to solve are said to be NP-hard and cannot be solved efficiently by any known algorithm in a practically acceptable time scale. In fact, many seemingly simple problems are very difficult to solve because the number of combinations increases exponentially with the size of the problem of interest. The most famous example is probably the travelling salesman problem (TSP) in which a salesperson intends to visit a number of cities exactly once, and returning to the starting point, while minimizing the total distance travelled or the overall cost of the trip.

NP-hard problems such as TSP do not have an efficient algorithm to solve all their instances. It is practically very difficult to get at the same time a solution of optimal quality and in a reduced runtime. In fact, most classical algorithms make the choice between a highest quality solution and an exponential runtime, or a moderate quality solution and a polynomial runtime. This leads to the third choice which offers good (not necessarily optimal) solutions in a reasonable runtime. This choice is presented by approximate algorithms such as metaheuristics, which studied and discussed by Blum et al. and Glover et al. in [2, 15].

Metaheuristic algorithms use search strategies to explore the search space more effectively, often focusing on some promising regions of the search space. These methods begin with a set of initial solutions or an initial population, and then, they examine step by step a sequence of solutions to reach, or hope to approach, the optimal solution to the problem of the interest. Metaheuristic algorithms have many advantages over traditional algorithms. Two of the advantages are simplicity and flexibility. Metaheuristics are usually simple to implement, but they often can solve complex problems and can thus be adapted to solve many real-world optimization problems, from the fields of operations research, engineering to artificial intelligence, as given by Yang and Yang et al. in [47, 49], Gandomi et al. in [10–12], and Yang and Gandomi in [48]. In addition, these algorithms are very flexible, and they can deal with problems with diverse objective function properties, either continuous, or discrete, or mixed.

Among the most popular metaheuristics performed to solve TSP, we can name genetic algorithms (GA), referring to Grefenstette et al. and Potvin in [16, 32], Tabu search (TS), simulated annealing (SA) which are presented by Kochenberger in [22], ant colonies optimization (ACO) by Dorigo and Di caro in [9], and particle swarm optimization (PSO) by Kennedy and Eberhart in [19], bee colonies optimization (BCO) by Teodorovic et al. [39], harmony search algorithm (HS) by Geem et al. in [13], firefly algorithm (FA) by Jati in [18], and cuckoo search (CS) recently developed by Ouaarab et al. [29].

In solving travelling salesman problem, these metaheuristics are characterized by their relative ease of implementation, a proper consideration of its constraints using control parameters, and the most important is producing high quality solutions in a relatively reduced runtime. However, the development of new metaheuristics by strengthen their robustness, remains a great challenge, especially for tough NP-hard problems.

This chapter discusses an improved variant of cuckoo search algorithm (CS) that solves efficiently the symmetric TSP. CS is a metaheuristic search algorithm which is

recently developed by Yang and Deb in 2009 [45]. Inspired by the obligate brood parasitic behaviour of some cuckoo species, combined with Lévy flights which describe the foraging patterns adopted by many animals and insects, it has been shown to be effective in solving continuous optimization problems. It provided effective results for multimodal functions in comparison with both genetic algorithms (GA) and particle swarm optimization (PSO). In this context, the Discrete improved Cuckoo Search algorithm (DCS) is compared with discrete particle swarm optimization (DPSO) proposed by Shi et al. in [36], and genetic simulated annealing ant colony system with particle swarm optimization techniques (GSA-ACS-PSOT) proposed by Chen in [4].

The remainder of this chapter is organized as follows: Sect. 2 introduces the TSP with some solution approaches. Section 3 first briefly describes the standard CS, then discusses the improvement carried out on the source of inspiration of CS, and proposes the discrete CS to solve symmetric TSP. Section 4 presents in detail the results of numerical experiments on a set of benchmarks of the so called Euclidean symmetric TSP from the TSPLIB library [33]. Finally, Sect. 5 concludes with some discussions.

2 Travelling Salesman Problem

2.1 Description of TSP

We assume that a salesperson has to visit a list of cities and return to his departure city. In order to find a way to calculate the best tour in term of distance, and before starting the trip, he will first fix some rules. Each city on the list must be visited exactly once, for each pair of cities, he knows the distance between both cities. These rules form a problem that will take as name *"The travelling salesman problem"*. Lawler et al. mentioned in [24] that the first citation of this term dates back to 1832 in a book entitled *"Der Handlungsreisende, wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geshäften gewiss zu sein. Von einem alten Commis-Voyageur"* (*"The travelling Salesman, how he should be and what he should do to get Commissions and to be Successful in his Business. By a veteran Travelling Salesman"*). But, Tucker in 1983 [41] said that the first use of the term in mathematical circles may have been in 1931–1932: *"I cannot confirm or deny the story that I heard of the TSP from Hassler Whitney. If I did (as Flood says), it would have occurred in* 1931–1932…"

Given n as the number of cities to visit in the list, the total number of possible tours covering all cities can be seen as a set of feasible solutions of the TSP and is given as $n!$. Formally, a statement of TSP according to Davendra in [7] is as follows:

Let $C = \{c_1, \ldots, c_n\}$ be the set of distinct cities, $E = \{(c_i, c_j) : i, j \in \{1, \ldots, n\}\}$ be the edge set, and $d_{c_i c_j}$ be a cost measure associated with edge $(c_i, c_j) \in E$. TSP is to find the minimal length of closed tour that visits each city once. Cities $c_i \in C$ are presented by their coordinates (c_{ix}, c_{iy}) and $d_{c_i c_j} = \sqrt{(c_{ix} - c_{jx})^2 + (c_{iy} - c_{jy})^2}$

is the Euclidean distance between c_i and c_j. A tour can be represented as a cyclic permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(N))$ of cities from 1 to N if $\pi(i)$ is interpreted to be the city visited in step i, $i = 1, \ldots, N$. The cost of a permutation (tour) is defined as :

$$f(\pi) = \sum_{i=1}^{N-1} d_{\pi(i)\,\pi(i+1)} + d_{\pi(N)\,\pi(1)} \tag{1}$$

If $d_{c_i c_j} = d_{c_j c_i}$ we talk about a Symmetric Euclidean TSP and if $d_{c_i c_j} \neq d_{c_j c_i}$ for at least one (c_i, c_j) then the TSP becomes an Asymmetric Euclidean TSP. In this chapter, TSP refers to the symmetric TSP.

During all these years, Travelling Salesman Problem (TSP) still attracts mathematicians in order to design an algorithm that detects the best solution for the majority of its instances in a reasonable time. It is characterized by its simple appearance and its statement that requires no mathematical background to understand it, and no thinking skills to find a solution. It can be seen as a theoretical problem of a travelling salesman seeking the shortest tour to winning some energy and time in order to make more money. But reality shows that TSP has a great importance, either in the academic or practical fields. Lenstra et al. and Reinelt [25, 34] gave some direct or indirect applications of TSP in several industrial and technological areas, such as drilling problem of printed circuit boards (PCBs), overhauling gas turbine engines, x-ray crystallography, computer wiring, order-picking problem in warehouses, vehicle routing, and mask plotting in PCB production. On the other hand, Arora [1] has shown, based on the demonstration of Papadimitriou [30] that Euclidian TSP belongs to the class of NP-hard optimization problems, whose computational complexity increases exponentially with the number of cities.

2.2 Approximate Approaches to Solve TSP

All NP-hard problems are attacked by exact or approximate methods. In the case of exact methods, find computationally an exact solution in a reasonable time, historically, is the first challenge facing the travelling salesman problem. This challenge is won in the case where inputs have a restricted size. Current exact methods work practically fast for small size problems. This efficiency is an aim that seems will never be achieved for large NP-hard problems. The running time of exact algorithms for large TSP instances depends on the computing power which is doubled every year. However, these algorithms complexity still remains exponential. So, solving NP-hard problems with only exact methods becomes less practical. To read more about exact methods, Laporte and Davendra provide an extensive discussion and references in [7, 23].

On the other side, approximate methods jump over the time and complexity constraints by the negligence of the optimality notion. They look intelligently for a good solution which could not be proved to be optimal, but its runtime remains accept-

able. Approximate algorithms are divided into three categories: tour construction algorithms, tour improvement algorithms, and metaheuristics.

2.2.1 Tour Construction Algorithms

Tour construction algorithms are less demanding, regarding the quality of the constructed solution in a relatively reduced time. They stop when a solution is found and never try to improve it. The idea is building a tour by iteratively adding cities to the sub-tour.

Nearest Neighbour algorithm is the simplest TSP tour construction heuristic. In this algorithm a salesman looks iteratively for the nearest city. He finds the last city (which is the departure city) with a polynomial complexity $O(n^2)$. In the case of greedy algorithm, the salesman grow the tour by repeatedly selecting the shortest edge and adding it to the tour without creating a cycle, as he has not reached edges or augmenting the degree of a node to 3 (to have one closed path, each node must be connected in the same time to less than 3 nodes). Computational time of greedy heuristic is $O(n^2 \log_2(n))$. Another example is insertion heuristics which starts with an initial edge or a closed sub-tour (often a triangle), and then inserting the rest by some heuristics. It complexity is given as $O(n^2)$. We gave these examples (others tour construction methods are provided by Davendra [7]) to show that this class of heuristics is not interested to find a good solution, but just a solution with moderate quality in a short time. Despite this low performance, tour construction heuristics are still widely used, but as an auxiliary of several heuristics. They are introduced to generate initial solutions before starting the process of other heuristics.

2.2.2 Tour Improvement Algorithms

Tour improvement algorithms start with a solution previously performed using tour construction algorithms. They minimize the length of the tour by improvement operators until reaching a tour that cannot be improved. Generally, they are based on simple tour modifications. Considering that, each modification or improvement, on a tour leads to another neighbour tour in the solution space. So, they search for locally improved tours by moving to a neighbour until no better neighbours exist. The most famous tour modification is 2-opt. It removes two edges from the current tour, and reconnects the new two paths created, in another possible way. In a minimization case, this is done only if the new tour is shorter than the current one. So, the process is repeated till no further improvement is possible or in a given number of steps. 2-opt can be generalized by replacing 2 edges by 3 or 4. In more complex improvement algorithms, such as Lin Kernighan, 2-opt, 3-opt and 4-opt are all intelligently introduced. It decides by a parameter k (k-opt where $k = 2, 3$ and 4), the value of the suitable move in each iteration. A detailed discussion is provided by Lin and Kernighan in [20]. Lin Kernighan is therefore relatively able to avoid local minima (the local minimum is the smallest solution in the neighbourhood). To

overcome this problem Tabu search (TS) algorithm describing by Glover and Laguna [14] had implemented in several ways a tabu list. It moves from the current solution to its best neighbour solution even if it comes from a bad 2-opt move (uphill move). To avoid cycling TS uses tabu list to store the recently performed moves which are prohibited during some iterations. In the other hand, uphill moves are introduced by Kirkpatrick in simulated annealing [21] not only when we have a local optimum, but at all the time. It chooses a solution randomly in the neighbourhood. Acceptance of this solution depends on its quality and on a control parameter called the temperature. This parameter decreases, and consequently the probability of accepting the solution decreases during iterations of the algorithm.

Local improvement algorithms are typically incomplete algorithms. The search may stop even if the best solution found by the algorithm is not optimal, while the optimal solution can lie far from the neighbourhood of the current solutions. Tabu search and simulated annealing, outlined by Malek et al. in [26] are considered as metaheuristics. But, they are single solution metaheuristics which modify or improve one candidate solution. However, metaheuristics discussed in the next sub-section adopt population-based approach.

2.2.3 Metaheuristics

No efficient algorithm exists for the TSP and all its relevant variants or problems of the same class. The need to quickly find good (not necessarily optimal) solutions to these problems has led to the development of various approximation algorithms such as metaheuristics [2, 15, 38]. In fact, metaheuristic algorithms have demonstrated their potential and effectiveness in solving a wide variety of optimization problems and have many advantages over tour construction/improvement algorithms. Metaheuristics are usually simple to implement, in order to solve complex problems and can be adapted to solve many real-world optimization problems. In addition, these algorithms are very flexible, and they can deal with problems with diverse objective function properties, either continuous, or discrete, or mixed. Such flexibility also enables them to be applied to deal a large number of parameters simultaneously.

Due to its high ability to solve complex problems by the simplest ways, nature is the main source of inspiration for designing metaheuristics. Most of these metaheuristics, which are generally based on populations, are inspired by the collective behaviour of groups, colonies and swarms of several species in nature. These collective behaviours are adopted by swarms with the aim of finding a partner, a food source or to avoid a predator in a relatively large space using relatively simple tools for sharing useful information and experiences.

Inspired from nature, metaheuristic algorithms use search strategies to explore the search space and then effectively exploit some promising regions of the search space. The following two discussed metaheuristic examples, entitled “Particle swarm optimization-based algorithms for TSP and generalized TSP” (DPSO) and “Solving the traveling salesman problem based on the genetic simulated annealing

ant colony system with particle swarm optimization techniques" (GSA-ACS-PSOT) are developed respectively by Shi et al. in [36], and Chen and Chien in [4].

Genetic Algorithm (GA)

Genetic Algorithm (GA) for TSP which detailed by Grefenstette et al. in [16], starts with a randomly generated population of candidate solutions (tours). Best (or all) candidates or parents are then mated to produce offspring or children. A percentage of the child tours are selected for mutation, and the new child tours are inserted into the population replacing the longer tours. These steps are repeated until a stop criterion. We notice that during reproduction phase, GA is mainly based on two operators to generate solutions: crossover and mutation. There are various types of crossover (Cycle crossover (CX), Partially-mapped crossover (PMX), Order crossover (OX), etc.) and mutation (swap and scramble, etc.) operators in the literature according to the addressed problem and the encoding used in different versions of GA such as those proposed to solve TSP by Potvin in [32] and Grefenstette in [16].

An example of applying crossover (CX) and mutation (swap) operators is shown respectively in Figs. 1 and 2. To produce an offspring, CX search a cycle of cities between parents. Then, cities in the found cycle (2, 3, 5) are fixed for both parents, and each parent copies the other cities from the second one. Swap mutation operator exchanges two cities chosen randomly.

Particle Swarm Optimization (PSO)

Particle swarm optimization considers solutions in the search space as particles moving with a variable velocity towards their own best solution *pBest* in the past and the global best solution *gbest* showing as follows :

$$X_i{}^{(k+1)} = X_i{}^{k} + V_i{}^{(k+1)}, \tag{2}$$

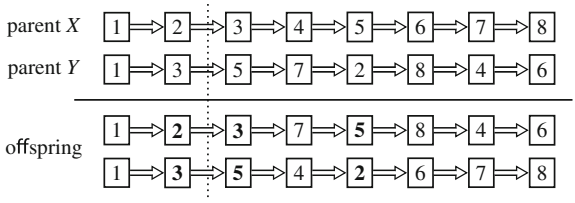

Fig. 1 The cycle crossover operator

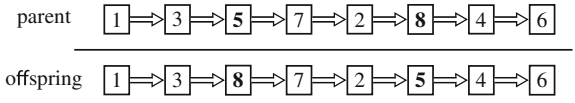

Fig. 2 The swap mutation operator

$$V_i^{(k+1)} = wV_i^{(k)} + c_1 r_1 (P_i - X_i^{(k)}) + c_2 r_2 (P_g - X_i^{(k)}), \quad (3)$$

where, in $(k+1)$ iteration, X_i $(i = 1, 2, \ldots, m)$ is the ith among m particles, which moves with the velocity V_i. P_i and P_g are respectively $pBest$ and $gBest$. w, c_1 and c_2 present parameters that selected, in the interval [0, 1], to control the behaviour of PSO. r_1 and r_2 are randomly generated in [0, 1].

The velocity decreased while the particle is approaching to $pbest$ or $gbest$. PSO has proved its efficiency to face several problems, but it is not very effective in the case of the travelling salesman problem. The biggest constraint is how to introduce the concept of velocity in a combinatorial space. Several attempts have succeeded in making adaptation of PSO to solve TSP. But during the transition from continuous space to the combinatorial space, these adaptations are unable to keep the same properties between the two spaces. This constraint is always found in all adaptations of distance, speed or movement in a continuous space to a combinational one. In order to expand the scale of the solved problems, according to Shi et al. in [36], inspired by the "Swap operator" developed by Wang et al. in [42], they introduced the permutation concept and an uncertainty searching strategy (to speed up the convergence speed) into a proposed discrete PSO algorithm for TSP. The main idea of this adaptation is interpreting velocity by some permutation operators. So, they are focused on the redefinition of the substraction of two particle positions. The test results of this approach are compared with the Discrete Cuckoo Search (DCS) developed by Ouaarab et al. in [29] in the rest of this chapter in "Cuckoo search and Discrete cuckoo search" section.

Ant Colony Optimization (ACO)

It is considered as a solution construction metaheuristic which is inspired by the behaviour of ants in optimizing their paths length between the colony and a food source. According to Dorigo and Di caro [9], its basic version is designed to solve TSP. An ant starts the process by a random displacement, from city c_i to city c_{i+1}, passing through the arc that connects the two cities, it leaves pheromone trail. Other ants that will pass on this arc had the choice to consider the pheromone deposited on the arc or take another arc randomly. To promote arcs of good solutions, ACO by a centralized control, adds a small amount of pheromone. To forget bad previous decisions, it uses a parameter that controls pheromone evaporation on the arcs.

Chen and Chien proposed a method in [4] for solving TSP by a combination of ACO, GA, PSO, and simulated annealing, called "the genetic simulated annealing ant colony system with particle swarm optimization techniques". They first, generate the initial population of the genetic algorithm by ant colony system, a variant of ACO proposed by Dorigo et al. in [8]. Then, to gain better solutions, genetic algorithms are performed with simulated annealing mutation techniques. Particle swarm optimization is introduced after every C cycles (generations), where c is a predefined number. It is used to exchange the pheromone information between groups.

A cycle presents one execution of the ant colony system and some executions of the genetic algorithms with simulated annealing mutation techniques. More details and descriptions are provided by Chen and Chien in [4]. This method is compared with the discrete cuckoo search in the next section.

3 Cuckoo Search and Discrete Cuckoo Search

3.1 Basic CS

Among many interesting features of cuckoos, a so-called brood parasitism feature adopted by some species is the most studied and discussed. Female cuckoos lay eggs in the previously observed nests of another bird species to let host birds hatching and brooding young cuckoo chicks. To increase the probability of having a new cuckoo and reduce the probability of abandoning eggs by the host birds, cuckoos use several strategies that are described by Payne [31].

The behaviour of cuckoos is combined in cuckoo search algorithm with Lévy flights in order to effectively search a new nest. Lévy flights, named by the French mathematician Paul Lévy, represent a model of random walks characterized by their step lengths which obey a power-law distribution. Several scientific studies have shown that the search for preys by hunters follows typically the same model of Lévy flights. This model is commonly represented by small random steps followed in the long term by large jumps [3, 37, 45].

CS is a metaheuristic search algorithm which was recently developed by Xin-She Yang and Suash Deb in 2009, initially designed for solving multimodal functions. It is summarized around the following ideal rules: (1) Each cuckoo lays one egg at a time and selects a nest randomly; (2) The best nest with the highest quality egg can pass onto the new generations; (3) The number of host nests is fixed, and the egg laid by a cuckoo can be discovered by the host bird with a probability $p_\alpha \in [0, 1]$.

A cuckoo i generates a new solution $x_i^{(t+1)}$ via Lévy flights, according to Eq. (4)

$$x_i{}^{(t+1)} = x_i{}^{(t)} + \alpha \oplus Levy(s, \lambda) \tag{4}$$

where α is the step size that follows the Lévy distribution that is shown in Eq. (5):

$$Levy(s, \lambda) \sim s^{-\lambda}, \quad (1 < \lambda \leq 3) \tag{5}$$

which has an infinite variance with an infinite mean [45]. Here s is step size drawn from a Lévy distribution.

3.2 Improved CS

The strength of CS is the way how to exploit and explore the solution space by a cuckoo. This cuckoo can have some "intelligence" so as to find much better solutions. Ouaarab et al. [29] consider in their improvement a cuckoo as the first level in controlling intensification and diversification, and since such a cuckoo is an individual of a population, so this population can be qualified as a second level of control, which can be restructured by adding a new category of cuckoos smarter and more efficient in their search.

Studies show that cuckoos can also engage a kind of surveillance on nests likely to be a host. This behaviour can serve as an inspiration to create a new category of cuckoos that have the ability to change the host nest during incubation to avoid abandonment of eggs (Payne) [31]. These cuckoos use mechanisms before and after brooding such as the observation of the host nest to decide if the nest is the best choice or not (so, it looks for a new nest much better for the egg). In this case, we can talk about a kind of local search performed by a fraction of cuckoos around current solutions.

Inspired from this observed behaviour, the mechanism adopted by this new fraction of cuckoos, can be divided into two main steps: (1) a cuckoo, initially moves by Lévy flights towards a new solution (which represents a new area); (2) from the current solution, the cuckoo in the same area seeks a new, better solution (in this step it can perform a local search). According to these two steps, the population of improved CS algorithm can be structured by three types of cuckoos:

1. A cuckoo, seeking (from the best position) areas which may contain new solutions that are much better than the solution of an individual can be randomly selected in the population;
2. A fraction p_a of cuckoos seeks new solutions far from the best solution;
3. A fraction p_c of cuckoos search for solutions from the current position and try to improve them. They move from one region to another via Lévy flights to locate the best solution in each region without being trapped in a local optimum.

We can note that the population, in its search process, is guided by: the best solution, the solutions found locally, and the solutions found far from its best solution. That improves intensive search around various best solutions, and at the same time, randomization is properly performed to explore new areas using Lévy flights. Thus, an extension to the standard CS is, as shown in bold in Algorithm 1 the addition of a method that handles the fraction p_c of smart cuckoos. It allowing CS to perform more efficiently with fewer iterations, and giving better resistance against any potential traps and stagnation in local optima in the case of TSP.

The new process added to the CS algorithm can be illustrated by its steps as follows. Assume that the value of the fraction p_c is set to 0.5, where this fraction is a set of good solutions of the population without the best solution. From each solution, a cuckoo performs a search starting by a random step via Lévy flights around the current solution, and then he tries to find the best solution in this region using local search.

Algorithm 1 Improved Cuckoo Search

Objective function $f(x), x = (x_1, \ldots, x_d)^T$
Generate initial population of n host nests x_i $(i = 1, \ldots, n)$
while (t <MaxGeneration) or (stop criterion) **do**
  Start searching with a fraction (p_c) **of smart cuckoos**
  Get a cuckoo randomly by Lévy flights
  Evaluate its quality/fitness F_i
  Choose a nest among n (say, j) randomly
  if ($F_i > F_j$) **then**
    replace j by the new solution;
  end if
  A fraction (p_a) of worse nests are abandoned and new ones are built;
  Keep the best solutions (or nests with quality solutions);
  Rank the solutions and find the current best
end while
Postprocess results and visualization

3.3 Discrete Cuckoo Search for Travelling Salesman Problem

In CS adaptation to TSP, Ouaarab et al. [29] are focused mainly on the reinterpretation of terminology used in the CS and its inspiration sources. This terminology is the key to any passage from a continuous space to a combinatorial one. If we take TSP as an example of a combinatorial optimization problem, before solving this problem by metaheuristics designed in continuous space, we need to discuss the following concepts: position, displacement, distance, and objective function. These notions should be clearly defined and well explained by TSP (combinatorial space), and well captured by the metaheuristic (designed for a continuous space) after interpretation (adaptation). To adapt CS to TSP these notions will appear in the discussion of the following five main elements: egg, nest, objective function, search space, and Lévy flights.

3.3.1 The Egg

If we assume that a cuckoo lays a single egg in one nest, we can say that one egg in a nest is a solution represented by one individual (nest) in the population. An egg can also be one new candidate solution for a place/location reserved by an individual in the population.

We can say that, in TSP an egg is the equivalent of a Hamiltonian cycle as shown in Fig. 3, (for more details about Hamiltonian cycle, works of Sahni and Gonzalez can be consulted [35]). Here, we neglect the need to take a departure city for all circuits and also the direction of the tour taken by the salesman.

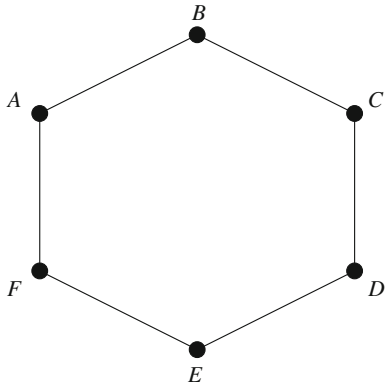

Fig. 3 In TSP, an egg appears as a Hamiltonian cycle

3.3.2 The Nest

In CS, the number of nests is fixed, and it is the size of the population. A nest is an individual of the population and its abandonment involves its replacement in the population by a new one.

By the projection of these features on TSP, we can say that a nest is shown as an individual in the population with its own Hamiltonian cycle. Obviously, a nest can have multiple eggs for future extensions. In the present chapter, each nest contains only one egg.

3.3.3 Objective Function

Each solution in the search space is associated with a numeric objective value. So the quality of a solution is proportional to the value of the objective function. In CS, a nest egg of better quality will lead to new generations. This means that the quality of a cuckoo's egg is directly related to its ability to give a new cuckoo.

In the case of the travelling salesman problem, the quality of a solution is related to the length of the Hamiltonian cycle. The best solution is the one with the shortest Hamiltonian cycle.

3.3.4 Search Space

In the case of two dimensions, the search space represents the positions of potential nests. These positions are $(x, y) \in \mathbb{R} \times \mathbb{R}$. To change the position of a nest, we only have to modify the actual values of its coordinates. It is obvious that moving nests or locations of the nests do not impose real constraints. This is the case in most continuous optimization problems, which can be considered as an advantage that avoids many technical obstacles such as the representation of the coordinates in the

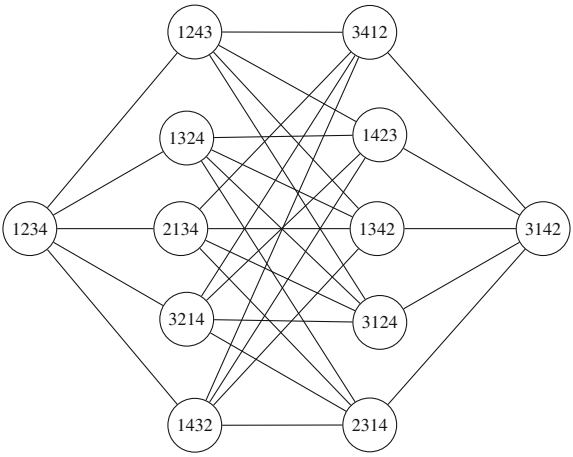

Fig. 4 Solutions in the search space by 2-opt move. An instance of five cities

solution space of TSP, especially in the mechanism of moving a solution from one neighbourhood to another. In TSP we have fixed coordinates of the visited cities; however, the visiting order between the cities can be changed.

The search space in TSP, as shown in the 2-opt perturbation case in Fig. 4 is a set of points. Each point is representative of a tour which appears as a potential solution with $n = 5$ cities (city "0" does not appear in the graph because it is considered as the departure city). These solutions are positioned in space according to the orders of their cities. For this example, we have 12 distinct solutions in the all search space (structured by 2-opt move) and each solution is directly connected with $n(n-3)/2$ neighbours.

Moving in the search space

Since the coordinates of cities are fixed, the movements are based on the order of visited cities. There are several methods, operators, or perturbations that generate a new solution from another existing solution by changing the order of visited cities.

In the adaptation of CS to TSP, there is a discrete CS, where perturbations used to change the order of visited cities are 2-opt moves (which detailed with more details by Croes in [6]), and double-bridge moves, described by Martin in [27]. 2-opt move is used for small steps, and large jumps are made by double-bridge move. A 2-opt move, as shown in Fig. 5, removes two edges from a tour (solution or Hamiltonian

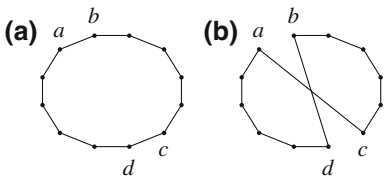

Fig. 5 2-opt move. **a** Initial tour. **b** The tour created by 2-opt move [the edges (a, b) and (c, d) are removed, while the edges (a, c) and (b, d) are added]

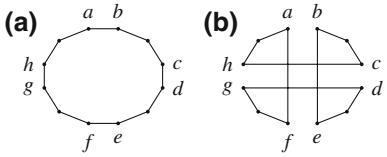

Fig. 6 Double-bridge move. **a** Initial tour. **b** The tour created by double-bridge move [the edges (a,b), (c,d), (e,f) and (g,h) are replaced by the edges (a,f), (c,h), (e,b) and (g,d), respectively]

cycle) and reconnects the two paths created. A double-bridge move cuts four edges and introduces four new ones as shown in Fig. 6.

Moving a solution to another, in the search space (combinatorial space) is made by small steps in a small area around the current solution. Therefore, carrying several steps leads to farther solutions. But, if we want to change the area of search and point to another far area, we perturb the solution by double-bridge move.

The neighbourhood

In continuous problems, the meaning of neighbourhood is obvious. However, for combinatorial problems, the notion of neighbourhood requires that the neighbour of a given solution must be generated by the smallest perturbation. This perturbation must make the minimum changes on the solution. This leads to the 2-opt move, because, for a new solution, and the minimum number of non-contiguous edges that we can delete is two. So, 2-opt move is a good candidate for this type of perturbation.

The step

The step of a movement is the distance between two solutions. It is based on the space topology and the concept of neighbourhood. The step length is proportional to the number of successive 2-opt moves on a solution. A big one step is represented by a double-bridge move.

3.3.5 Lévy flights

Lévy flights have as a characteristic of an intensive search around a solution, followed by occasional big steps in the long run. According to Yang and Deb [45], in some optimization problems, the search for a new best solution is more efficient via Lévy flights. In order to improve the quality of search, the step length will be associated to the value generated by Lévy flights as outlined in the standard CS.

The search space (solution space) must contain a notion of steps, well-defined and stable. The step unit, selected for moving from a solution to another is 2-opt

move. To move in the search space we have a choice between a small step, a number k of steps, and a big step. To facilitate the control of these steps via Lévy flights, we associate them with an interval between 0 and 1. Therefore, according to the value given by the Lévy flights in this interval we can choose the appropriate step length. If the value of Lévy is in:

1. $[0, i[$ so we have one step (2-opt move),
2. $[(k-1) \times i, k \times i[$ we move by k steps,
3. $[k \times i, 1[$ we perform a big step by double bridge-move.

The value of i in this process is: $i = (1/(n+1))$ where n is the max number of steps; and k in $\{2, \ldots, n\}$.

Assume that $n = 4$, so $i = 0.2$, so our interval is divided into five parts.

Lévy in $[0, i[\longrightarrow [0, 0.2[\longrightarrow$ one step by 2opt move
Lévy in $[i, i \times 2[\longrightarrow [0.2, 0.4[\longrightarrow$ two steps by 2opt move
Lévy in $[i \times 2, i \times 3[\longrightarrow [0.4, 0.6[\longrightarrow$ three steps by 2opt move
Lévy in $[i \times 3, i \times 4[\longrightarrow [0.6, 0.8[\longrightarrow$ four steps by 2opt move
Lévy in $[i \times 4, 1[\longrightarrow [0.8, 1[\longrightarrow$ one step by double-bridge move

4 Experimental Results

This version of discrete cuckoo search (DCS) is tested, first without and then with the improvement, on some instances (benchmarks) of TSP taken from the publicly available electronic library TSPLIB of TSP problems by Reinelt in [33]. Forty-one instances are considered with sizes ranging from 51 to 1379 cities. As shown by Reinelt, all these TSP instances belong to the Euclidean distance type. Based on the test results, we have made some comparisons between the basic and the improved DCS. Then, the improved DCS algorithm is compared with some other recent methods (genetic simulated annealing ant colony system with particle swarm optimization techniques (GSA-ACS-PSOT) [4] and discrete particle swarm optimization (DPSO) [36]).

We have implemented basic/standard and improved DCS algorithms using Java language under 32 bit Vista operating system. Experiments are conducted on a laptop with Intel(R) Core^{TM} 2 Duo 2.00 GHz CPU, and 3 GB of RAM.

After some preliminary trials, the selected parameter settings used in the experiments in both algorithms (basic and improved DCS) are shown in Table 1. In each case study, 30 independent runs of the algorithms with these parameters are carried out. Figure 7 shows that the maximum number of iterations (MaxGeneration) can be set to 500 for both algorithms.

It can be seen from Fig. 8 that the improved DCS is superior to basic DCS regarding to PDav(%).The high performance of the improved DCS in relation to the basic DCS may be due to the improvement applied on the basic DCS by the new category of

Table 1 Parameter settings for both algorithms, basic and improved DCS

Parameter	Value	Meaning
n	20	Population size
p_a	0.2	Portion of bad solutions
p_c	0.6	Portion of intelligent cuckoos (only for the improved DCS)
MaxGeneration	500	Maximum number of iterations

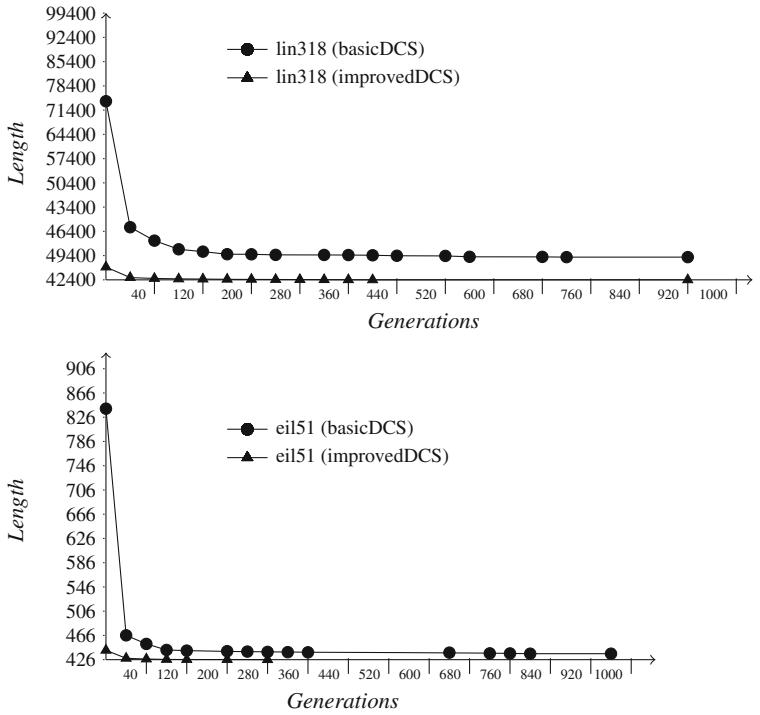

Fig. 7 Average length of the best solutions of 10 runs for eil51(opt = 426) and lin318(opt = 42029)

cuckoos which have an efficient method of generating new solutions by moving from one area to another to find best solutions for each area.

Table 2 summarizes the experiments results, where the first column shows the name of the instance, the column 'opt' shows the optimal solution length taken from the TSPLIB, the column 'best' shows the length of the best solution found by each algorithm, the column 'worst' shows the length of the worst solution found by each algorithm, the column 'average' gives the average solution length of the 30 independent runs of each algorithm, the column 'SD' denotes the standard deviation which takes the value 0.00 shown in bold when all solutions found have the same

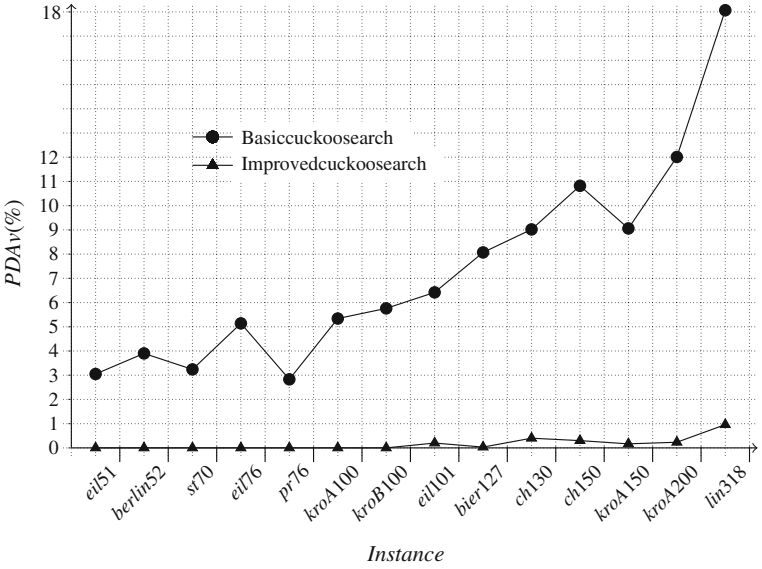

Fig. 8 PDAv(%)(over 30 runs) for 14 TSPLIB instances

length over the 30 runs, while the column '$C_1\%/C_{opt}$' gives the number of solutions that are within 1 % optimality (over 30 runs)/the number of the optimal solutions, the column 'PDav(%)' denotes the percentage deviation of the average solution length over the optimal solution length of 30 runs, the column 'PDbest(%)' gives the percentage deviation of the best solution length over the optimal solution length of 30 runs, and the column 'time' shows the average time in seconds for the 30 runs. The percentage deviation of a solution to the best known solution (or optimal solution if known) is given by Eq. (6):

$$\text{PDsolution}(\%) = \frac{\text{solution length} - \text{best known solution length}}{\text{best known solution length}} \times 100 \quad (6)$$

Table 2 shows the computational results of improved DCS algorithm on 41 TSPLIB instances. Based on PDbest(%) column, we can say that 90.24 % of the values of PDbest(%) are less than 0.5 %, which means that the best solution found, of the 30 trials, approximates less than 0.5 % of the best known solution, while the value of 0.00 shown in bold in column PDav(%) indicates that all solutions found on the 30 trials have the same length of the best known solution. All these numerical values presented in Table 2 show that the improved DCS can indeed provide good solutions in reasonable time.

In Fig. 9 and Table 3, the experimental results of the improved DCS algorithm are compared with the both methods GSA-ACS-PSOT/DPSO. The results of these two methods are directly summarized from original papers [4, 36]. It can be seen clearly

Table 2 Computational results of improved DCS algorithm for 41 TSP benchmark instances for TSPLIB

Instance	Opt	Best	Worst	Average	SD	PDav(%)	PDbest(%)	$C_{1\%}/C_{opt}$	Time
eil51	426	**426**	**426**	**426**	**0.00**	**0.00**	**0.00**	**30/30**	1.16
berlin52	7542	**7542**	**7542**	**7542**	**0.00**	**0.00**	**0.00**	**30/30**	0.09
st70	675	**675**	**675**	**675**	**0.00**	**0.00**	**0.00**	**30/30**	1.56
pr76	108159	**108159**	**108159**	**108159**	**0.00**	**0.00**	**0.00**	**30/30**	4.73
eil76	538	**538**	539	538.03	0.17	0.00	**0.00**	30/29	6.54
kroA100	21282	**21282**	**21282**	**21282**	0.00	**0.00**	**0.00**	**30/30**	2.70
kroB100	22141	**22141**	22157	22141.53	2.87	**0.00**	**0.00**	30/29	8.74
kroC100	20749	**20749**	**20749**	**20749**	**0.00**	**0.00**	**0.00**	**30/30**	3.36
kroD100	21294	**21294**	21389	21304.33	21.79	0.04	**0.00**	30/19	8.35
kroE100	22068	**22068**	22121	2281.26	18.50	0.06	**0.00**	30/18	14.18
eil101	629	**629**	633	630.43	1.14	0.22	**0.00**	30/6	18.74
lin105	14379	**14379**	**14379**	**14379**	**0.00**	**0.00**	**0.00**	**30/30**	5.01
pr107	44303	**44303**	44358	44307.06	12.90	0.00	**0.00**	30/27	12.89
pr124	59030	**59030**	**59030**	**59030**	**0.00**	**0.00**	**0.00**	**30/30**	3.36
bier127	118282	**118282**	118730	118359.63	12.73	0.06	**0.00**	30/18	25.50
ch130	6110	**6110**	6174	6135.96	21.24	0.42	**0.00**	28/7	23.12
pr136	96772	96790	97318	97009.26	134.43	0.24	0.01	30/0	35.82
pr144	58537	**58537**	**58537**	**58537**	**0.00**	**0.00**	**0.00**	**30/30**	2.96
ch150	6528	**6528**	6611	6549.9	20.51	0.33	**0.00**	29/10	27.74
kroA150	26524	**26524**	26767	26569.26	56.26	0.17	**0.00**	30/7	31.23
kroB150	26130	**26130**	26229	26159.3	34.72	0.11	**0.00**	30/5	33.01
pr152	73682	**73682**	**73682**	**73682**	**0.00**	**0.00**	**0.00**	**30/30**	14.86
rat195	2323	2324	2357	2341.86	8.49	0.81	0.04	20/0	57.25
d198	15780	15781	15852	15807.66	17.02	0.17	0.00	30/0	59.95
kroA200	29368	29382	29886	29446.66	95.68	0.26	0.04	29/0	62.08
kroB200	29437	29448	29819	29542.49	92.17	0.29	0.03	28/0	64.06
ts225	126643	**126643**	126810	126659.23	44.59	0.01	**0.00**	30/26	47.51
tsp225	3916	**3916**	3997	3958.76	20.73	1.09	**0.00**	9/1	76.16
pr226	80369	**80369**	80620	80386.66	60.31	0.02	**0.00**	30/19	50.00
gil262	2378	2382	2418	2394.5	9.56	0.68	0.16	22/0	102.39
pr264	49135	**49135**	49692	49257.5	159.98	0.24	**0.00**	28/13	82.93
a280	2579	**2579**	2623	2592.33	11.86	0.51	**0.00**	25/4	115.57
pr299	48191	48207	48753	48470.53	131.79	0.58	0.03	27/0	138.20
lin318	42029	42125	42890	42434.73	185.43	0.96	0.22	15/0	156.17
rd400	15281	15447	15704	15533.73	60.56	1.65	1.08	0/0	264.94
fl417	11861	11873	11975	11910.53	20.45	0.41	0.10	30/0	274.59
pr439	107217	107447	109013	107960.5	438.15	0.69	0.21	22/0	308.75
rat575	6773	6896	7039	6956.73	35.74	2.71	1.81	0/0	506.67
rat783	8806	9043	9171	9109.26	38.09	3.44	2.69	0/0	968.66
pr1002	259045	266508	271660	268630.03	1126.86	3.70	2.88	0/0	1662.61
nrw1379	56638	58951	59837	59349.53	213.89	4.78	4.08	0/0	3160.47

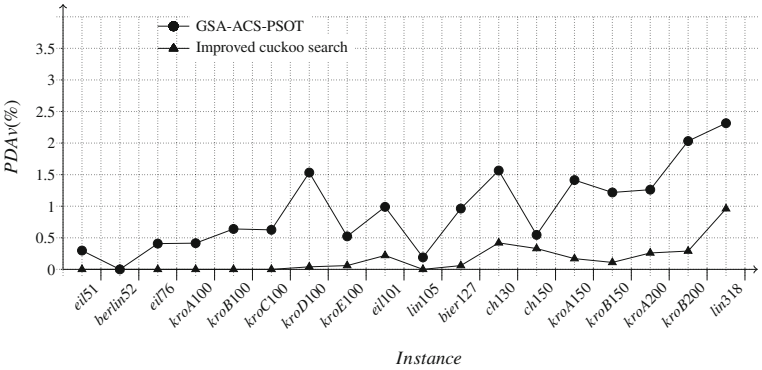

Fig. 9 PDAv(%)(over 30 runs) for 18 TSPLIB instances

Table 3 Comparison of experimental results of the improved DCS with DPSO [36]

Instance		DPSO			Improved DCS		
	Opt	Best	Worst	PDAv(%)	Best	Worst	PDAv(%)
eil51	426	427	452	2.57	**426**	**426**	**0.00**
berlin52	7542	**7542**	8362	3.84	**7542**	**7542.0**	**0.00**
st70	675	**675**	742	3.34	**675**	**675**	**0.00**
pr76	108159	108280	124365	3.81	**108159**	**108159**	**0.00**
eil76	538	546	579	4.16	**538**	539	0.00

from Fig. 9 and Table 3 that improved DCS outperforms the other two algorithms (GSA-ACS-PSOT and DPSO) in solving all the eighteen/five tested TSP instances. From Fig. 9, the lower curve which is associated with the improved DCS algorithm is better, in terms of solution quality. This can be explained basically by the strengths of CS: a good balance between exploitation and exploration, which appears on the population structure and the new category of cuckoo distinguished by its technical of research that is relatively intelligent compared to other ordinary cuckoos. An intelligent use of Lévy flights and the reduced number of parameters are also considered as an advantage.

5 Conclusion

In this chapter, we have studied and discussed an improved and discrete version of cuckoo search (CS) via Lévy flights by reconstructing its population and introducing a new cuckoo category which is more intelligent, and adapting its key points to solve the symmetric travelling salesman problem (TSP). This adaptation is based on a study of terminology interpretation used in CS and in its inspiration sources. Discrete CS (improved CS adapted to TSP) has been implemented and tested on

forty-one benchmark TSP instances, in order to be compared with GSA-ACS-PSOT and DPSO.

According to the comparison results, it is clear that Discrete CS outperforms all two other methods for solving TSP. It comes from the proper preservation of the advantages of CS during adaptation process to TSP problem and the improvement carried by the new population structure. This can be seen from the adoption of Lévy flights to move in the search space, and of a variety of cuckoos that use multiple research methods. Thus, the independence of our cuckoo new category, relatively to the best solution, in its search for a new solution may provide better strategies in generating new solutions than the simple use of one current best solution, thus leading to a more efficient algorithm. Another advantage, which the use of local perturbations introduced in the proposed Discrete CS, can provide more flexibility to solve other combinatorial optimization problems. Further references studies can be fruitful if we can focus on the parametric studies and applications of DCS into other combinatorial problems such as scheduling and routing.

In this chapter, we discussed the first discrete version of CS for solving TSP with the aim of challenging the constraint of passing from a continuous search space, and keeping some properties to the combinatorial one. We also focused on the development of a more controllable algorithm and less complex by the proposition of a new category of cuckoos and a few number of parameters, even in the case of a combinatorial optimization problem. In fact, based on this work we can contribute some extensions either on CS through its improvement or on DCS through the adaptation performed to TSP.

References

1. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. J. ACM (JACM) **45**(5), 753–782 (1998)
2. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. (CSUR) **35**(3), 268–308 (2003)
3. Brown, C.T., Liebovitch, L.S., Glendon, R.: Lévy flights in dobe ju/'hoansi foraging patterns. Hum Ecol **35**(1), 129–138 (2007)
4. Chen, S.M., Chien, C.Y.: Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. Expert Syst. Appl. **38**(12), 14439–14450 (2011)
5. Clerc, M.: Discrete particle swarm optimization, illustrated by the traveling salesman problem. In: Babu, B.V, Onwubolu, G.C. (Eds.) New optimization techniques in engineering, pp. 219–239. Springer, Berlin (2004)
6. Croes, G.A.: A method for solving traveling salesman problems. Oper. Res. **6**(6), 791–812 (1958)
7. Davendra, D.: Traveling Salesman Problem, Theory and Applications. InTech Publisher, Rijeka (2010)
8. Dorigo, M., Maniezzo, V., Colorni, A.: Positive feedback as a search strategy. Technical report. pp. 99–016. Dipartimento di Electtonica, Polotecnico di Milano, Italy (1992)
9. Dorigo, M., Di Caro, G.: Ant colony optimization: a new metaheuristic. In: Proceedings of the 1999 Congress on Evolutionary Computation, 1999, CEC 99, (Vol. 2). IEEE (1999)

10. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Mixed variable structural optimization using firefly algorithm. Comput. Struct. **89**(23–24), 2325–2336 (2011)
11. Gandomi, A.H., Talatahari, S., Yang, X.S., Deb, S.: Design optimization of truss structures using cuckoo search algorithm. Struct. Des Tall Special Build. (2012). doi:10.1002/tal.1033
12. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng. Comput. **29**(1), 17–35 (2013)
13. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A new heuristic optimization algorithm: harmony search. Simulation **76**(2), 60–68 (2001)
14. Glover, F., Laguna, M.: Tabu Search, vol. 22. Kluwer academic publishers, Boston (1997)
15. Glover, F., Kochenberger, G.A.: Handbook of Metaheuristics. Springer, New York (2003)
16. Grefenstette, J.J., Gopal, R., Rosmaita, B.J., Gucht, D.V.: Genetic algorithms for the traveling salesman problem. In: Proceedings of the 1st international conference on genetic algorithms, pp. 160–168. L. Erlbaum Associates Inc. (1985)
17. Hochbaum, D.S.: Approximation Algorithms for NP-Hard Problems. PWS Publishing Co, Boston (1996)
18. Jati, G.K.: Evolutionary discrete firefly algorithm for travelling salesman problem. In Adaptive and Intelligent Systems, pp. 393–403. Springer, Berlin (2011)
19. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE international conference on neural networks, IEEE 1995, vol. 4, pp. 1942–1948 (1995)
20. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**, 291–307 (1970)
21. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
22. Kochenberger, G.A.: Handbook of Metaheuristics. Springer, New York (2003)
23. Laporte, G.: The traveling salesman problem: an overview of exact and approximate algorithms. Eur. J. Oper. Res. **59**(2), 231–247 (1992)
24. Lawler, E.L., Lenstra, J.K., Kan, A.R., Shmoys, D.B.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Vol. 3. Wiley, Chichester (1985)
25. Lenstra, J.K., Rinnooy, K.A.: Some simple applications of the travelling salesman problem. Oper. Res. Quart. **26**(5), 717–733 (1975)
26. Malek, M., Guruswamy, M., Pandya, M., Owens, H.: Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Ann. Oper. Res. **21**(1), 59–84 (1989)
27. Martin, O., Otto, S.W., Felten, E.W.: Large-step markov chains for the traveling salesman problem. Complex Syst. **5**(3), 299–326 (1991)
28. Melanie, M.: An Introduction to Genetic Algorithms. MIT Press, Massachusetts (1999). (Fifth printing)
29. Ouaarab, A., Ahiod, B., Yang, X.S.: Discrete cuckoo search algorithm for the travelling salesman problem. Neural Comput. Appl. (2013). doi:10.1007/s00521-013-1402-2
30. Papadimitriou, C.H.: Euclidean TSP is NP-complete. Theor. Comput. Sci. **4**, 237–244 (1977)
31. Payne, R.B., Sorenson, M.D.: The Cuckoos, vol. 15. Oxford University Press, Oxford (2005)
32. Potvin, J.Y.: Genetic algorithms for the traveling salesman problem. Ann. Oper. Res. **63**(3), 337–370 (1996)
33. Reinelt, G.: Tsplib a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
34. Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications, vol. 15. Springer, New York (1994)
35. Sahni, S., Gonzalez, T.: P-complete approximation problems. J. ACM (JACM) **23**(3), 555–565 (1976)
36. Shi, X.H., Liang, Y.C., Lee, H.P., Lu, C., Wang, Q.X.: Particle swarm optimization-based algorithms for tsp and generalized tsp. Inf. Process. Lett. **103**(5), 169–176 (2007)
37. Shlesinger, M.F., Zaslavsky, G.M., Frisch, U.: Lévy Flights and Related Topics in physics:(Nice, 27–30 June 1994). Springer, New York (1995)
38. Talbi, E.G.: Metaheuristics: From Design to Implementation, vol. 74. Wiley, Hoboken (2009)

39. Teodorovic, D., Lucic, P., Markovic, G., Orco, M.D.: Bee colony optimization: principles and applications. In: 2006 8th Seminar on Neural Network Applications in Electrical Engineering (NEUREL 2006), IEEE, pp. 151–156 (2006)
40. Teodorovic, D.: Bee colony optimization (BCO). In: Lim, C.P., Jain, L.C., Dehuri, S. (eds.) Innovations in Swarm Intelligence, pp. 39–60. Springer Berlin (2009)
41. Tucker, A.W. Letter to David Shmoys, 17 Feb 1983. [1:3].
42. Wang, K.P., Huang, L., Zhou, C.G., Pang, W.: Particle swarm optimization for traveling salesman problem. In: 2003 International Conference on Machine Learning and Cybernetics, vol. 3, pp. 1583–1585. IEEE (2003)
43. Wong, L.P., Low, M.Y.H., Chong, C.S.: A bee colony optimization algorithm for traveling salesman problem. In: Second Asia International Conference on Modeling and Simulation, 2008. AICMS 08, pp. 818–823. IEEE (2008)
44. Yang, X.S.: Firefly algorithms for multimodal optimization. In: Stochastic Algorithms: Foundations and Applications, pp. 169–178. Springer, Berlin (2009)
45. Yang, X.S., Deb, S., (2009) Cuckoo search via lévy flights. In: World congress on Nature and biologically inspired computing, NaBIC 2009, pp. 210–214. IEEE (2009)
46. Yang, X.S., Deb, S.: Engineering optimisation by cuckoo search. Int. J. Math. Modell. Numer. Optim. **1**(4), 330–343 (2010)
47. Yang, X.S.: Engineering Optimization: An Introdubtion with Metaheuristic Applications. Wiley, Hoboken (2010)
48. Yang, X.S., Gandomi, A.H.: Bat algorithm: a novel approach for global engineering optimization. Eng. Comput. **29**(5), 464–483 (2012)
49. Yang, X.S., Cui, Z.H., Xiao, R.B., Gandomi, A.H., Karamanoglu, M.: Swarm Intelligence and Bio-Inspired Computation: Theory and Applications. Elsevier, Waltham (2013)

Comparative Analysis of the Cuckoo Search Algorithm

Pinar Civicioglu and Erkan Besdok

Abstract Cuckoo Search Algorithm (CS) is a population based, elitist evolutionary search algorithm proposed for the solution of numerical optimization problems. Despite its wide use, the algorithmic process of CS has been scarcely studied in detail. In this chapter, the algorithmic structure of CS and its effective problem solving success have been studied. Fifty benchmark problems were used in the numerical tests performed in order to study the algorithmic behavior of CS. The success of CS in solving benchmark problems was compared with three widely used optimization algorithms (i.e., PSO, DE, and ABC) by means of Kruskal–Wallis statistical test. The search strategy of CS, which utilizes the Lèvy distribution, enables it to analyze the search space in a very successful manner. The statistical results have verified that CS has the superior problem-solving ability as a search strategy.

Keywords Cuckoo search · Stable distributions · Lèvy-walk · Kruskal–Wallis test

1 Introduction

The requirement for the development of new global optimization algorithms which can solve complex numerical optimization problems is in progress. In this chapter, the structural features of the relatively newly generated Cuckoo Search Algorithm (CS) [1–14] and its problem solving ability have been studied. The basic concepts

P. Civicioglu (✉)
College of Aviation, Department of Aircraft Electrics and Electronics,
Erciyes University, Kayseri, Turkey
e-mail: civici@erciyes.edu.tr

E. Besdok
Department of Geomatic Engineering, Erciyes University, Kayseri, Turkey
e-mail: ebesdok@erciyes.edu.tr

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_5,

regarding optimization and optimization terminology have been briefly explained below prior to the commencement of the detailed examination on CS.

Optimization is a considerably important area of research in engineering [1, 2, 15–20]. The purpose of an optimization algorithm is to find out the best values of the parameters of a system under various conditions. The process of finding the global optimum is called *global optimization*. The first step to solve the global optimization problems is to design an objective function suitable for structure of the problem. The objective function establishes the mathematical relation between the object and the parameters of the problem. The optimization problems target, generally, the minimization of an objective-function.

Optimization encompasses both maximization and minimization problems. Any maximization problem can be converted into a minimization problem by taking the negative of the objective function, and vice versa. It is desired that an optimization algorithm is robust, can reach the *global minimum* value of the problem rapidly, has very limited number of control parameters, a smaller computational-cost and easy to apply to different problem models. Generally, since the size of search space increases as the dimension of problem increases, it becomes difficult to find out the global optimums of such problems though the classical techniques. If the objective function in an optimization problem is nonlinear, and if it is not possible to calculate the differential of the objective function, the heuristic search techniques are used to find the global optimum of the problem [1, 2, 20–23].

The most commonly used population-based heuristic search techniques are the swarm-intelligence based optimization algorithms [1, 20, 22–25] and the genetic-evolution based optimization algorithms [16, 17, 26–28]. Among the population-based heuristic optimization techniques, there are the Particle Swarm Optimization (PSO) [2, 20, 22, 23, 29, 30], Differential Evolution (DE) [16, 17, 27, 28], Artificial Bee Colony Optimization (ABC) [24, 25] and Ant Colony Optimization [31] algorithms. The swarm-intelligence based optimization algorithms are generally based on the simplified mathematical models of various rather complex social behaviors of the living creatures. The optimization problem solution abilities of the swarm-based optimization algorithms have developed considerably for the reason that each solution shares information with the other solutions one way or another.

The genetic-evolution based optimization algorithms [32–35] use various mathematical models developed with inspiration from the basic hypotheses of the genetic science (such as *reproduction*, *mutation/morphogenesis*, *recombination*, and *selection*). Among the genetic-evolution based optimization algorithms, the most studied algorithms in the literature are Genetic Algorithm, Genetic Programming, Evolutionary Programming, Evolution Strategy, and Neuro-Evolution [22, 23]. Contrary to the classical optimization techniques, the population-based optimization algorithms do not guarantee finding the optimum parameter values of the problem. But, unlike the classical optimization algorithms, the population-based optimization algorithms are sufficiently flexible to solve different types of problems.

Generally, while trying to transform the parameters vectors into parameters vectors that can generate better objective-function value, the heuristic search algorithms use a *greedy selection* criterion. The *greedy selection* criterion is based on preferring

the one that provides better objective function value between two solutions; in the genetic algorithms, generally, the *greedy selection* criterion is used while selecting the next population elements. Therefore, solution of an optimization problem with genetic algorithms always has the risk of being trapped in any local solution and not being able to reach the global solution of the problem. The DE that is a deterministic genetic algorithm has strategy enabling to avoid the local minimums while searching for the global optimum; in the DE, the selection probabilities of the parent chromosomes selected randomly from the current population to be used to generate a new solution are equal. The PSO manages to avoid the local minimums considerably using numerous particles moving in the search-space. The ABC applies a method resembling the strategy of DE to avoid the local solutions of the problem. The size of population value is rather effective in robustness of the evolutionary optimization algorithm. When the small population dimensions are selected, the algorithms can reach only the local solutions in general. When the initial populations that are larger than necessary are selected, on the other hand, it becomes difficult for the optimization algorithm to reach the global optimum of the problem, or such algorithms remain rather slow in reaching the global optimum. Therefore, the optimum population dimension selection is considerably important for the population-based algorithms.

The population-based heuristic optimization algorithms can be trapped in any local solution of the problem as the diversity of population decreases in the advancing iteration steps. In literature, various approaches are introduced, which are based on the use of dynamic population to prevent the rapid decrease of the diversity of population value in the advancing iteration steps. If the randomly created new elements are added to the current population at the beginning of each iteration, the fast decrease in the population diversity can be prevented partially, but in such a case, it becomes difficult for the population-based optimization algorithms to reach the global optimum. Therefore, the operators used during the calculations to shift the values of the population members exceeding the search space limits due to any reason whatsoever among the search space limits have direct effects on the success of the optimization algorithm.

A population-based heuristic optimization algorithm must have the *global exploration* and *local exploitation* abilities. The *exploration* is the ability of an optimization algorithm to use the whole search-space effectively. The *exploitation*, on the other hand, is the ability of the optimization algorithm to search for the best solution around a new solution it has acquired. The heuristic optimization techniques acquire the new solutions they need to avoid the local minimums in the first iterations generally with their exploration ability. As the iterations advance, the effect of the *exploitation* process on the solutions generated by the algorithm increases. The success of an optimization algorithm is significantly dependent on its *exploration* and *exploitation* abilities and the natural balance between these abilities.

The population-based optimization algorithms use a three-stage process while improving the current population elements to solve an optimization problem: *self-adaptation*, *cooperation*, and *competition*. In the *self-adaptation* stage, each element of the population is improved one by one. In the *cooperation* stage, information

exchange is made between the newly developed solutions. In the *competition* stage, the population elements to be used in the next iteration are determined. The *self-adaptation*, *cooperation*, and *competition* strategies of the heuristic optimization algorithms are based on mathematical methods inspired by simplified models of various complex behaviors of the social creatures.

Although the heuristic optimization methods are rather successful is solution of the optimization problems, there is no single heuristic optimization method that can solve all the different types of optimization problems. Therefore, the need for development of heuristic optimization algorithms that can solve the different types of problems is still continuing.

The PSO is inspired by the simplified mathematical models of the collective behaviors of the living creatures. The DE shows similarity with the genetic algorithms that follow the basic evolutional rules. The ABC, on the other hand, is based on a simplified mathematical model of rather complex social behaviors of the honeybees during the search for nectar. The PSO and DE have been used in solution of numerous different problems in the scientific literature. The studies conducted using ABC that is a relatively new global optimization technique are also rather diversified. The PSO, DE, and ABC have been used commonly in optimization of many different types of problems. Therefore, in order to compare the success of the CS in solving the test functions with the methods in the literature, the PSO, DE, and ABC have been selected. Generally, the success of the PSO, DE, and ABC in solving the global optimization problems varies depending on the *structure of the problem*, *size of the problem*, *initial values of the control parameters of algorithms*, the structure and size of the *initial population*, and the *total number of cycles*.

CS is inspired by the offsprings generation strategy of the cuckoo birds. CS is a population based, elitist evolutionary search algorithm. The offsprings generation strategy of CS effectively improves its problem solving ability. This is a result of CS' not excessively requiring specific programming routines which radically effect the calculating period (i.e., CS code includes very limited number of if-statement codes). Consequently, its simple, but very efficient structure enables CS to be easily coded in many programming languages.

This chapter is organized as follows. In Sect. 2, Comparison Algorithms have been explained. In Sect. 3, Structured Analysis of Cuckoo Search Algorithm is given. In Sects. 4 and 5, Experiments and Conclusions have been given, respectively.

2 Comparison Algorithms

In this section, the general structures of the DE, PSO and ABC have been explained.

2.1 Differential Evolution (DE)

The DE is a random-search algorithm with a parallel structure [16, 27, 28]. DE is a non-gradient-based, evolutionary computation algorithm. Its mathematical structure is very simple. It can be adapted to many problem types rather easily. For such reasons, DE is one of the most frequently optimization techniques in the literature. The basic difference of DE from the genetic algorithm is the structure of the mutation operator. Numerous different mutation strategies can be used in DE. The most frequently used mutation strategy in the literature is the mutation strategy called 'DE/rand/1/bin' [16].

In this chapter, the 'DE/rand/1/bin' mutation strategy has been used in the experiments made with the DE. In the DE, while *ND*, D, and x_i indicates the population size, the problem size, and the chromosomes of the population, respectively. The mutant vector is calculated with the Eq. 1;

$$m_i = x_{r_1} + F(x_{r_2} - x_{r_3}) \,|\, i \neq r_1 \neq r_2 \neq r_3 \tag{1}$$

The scale factor F is a real number in the range of [0 1]. The scale factor F controls the *evolution rate* of the population. After acquiring the mutant vector m_i in the DE, the following crossover operator is applied;

$$u_i = \begin{cases} m_{i,j} \ rand[0\ 1] \leq CR \\ x_{i,j} \ \text{else} \end{cases} \tag{2}$$

where $0 \leq j \leq (D-1)$. $CR \in [0\,1]$ is the crossover probability value. Using the u_i values and x_i chromosome generated by the crossover operator in the DE, the chromosomes to be in the next population are calculated according to the following equation:

$$x_{i+1} = \begin{cases} u_i \ if \ f(u_i) < f(x_i) \\ x_i \ \text{else} \end{cases} \tag{3}$$

where $f(x)$ indicates the objective function value. After acquiring the chromosomes of the next population, the calculations are continued until the predefined stopping criteria are met. The success in searching the global minimum with the DE is rather sensitive to values of the DE control parameters (i.e., *ND*, F, and *CR*), the *size of population* value, and the *maximum cycle* value.

For more detailed information on DE, please refer to the studies given in [16, 22, 23, 27, 28].

2.2 Particle Swarm Optimization (PSO)

The PSO is a stochastic, multi-agent parallel search algorithm [2, 20]. The PSO that is a population-based, heuristic, evolutionary optimization technique is based on the mathematical modeling of various collective behaviors of the living creatures that

display complex social behaviors. In the PSO, each population element is called a particle, and the particles correspond to random solutions in the search-space. In the PSO, the movements of a particle in the search-space are modeled as linear combination of the best global solutions discovered by all particles until that moment and the best local solution discovered by the related particle itself. As the *i*th particle in a D-dimensional search-space, X_i is shown as follows:

$$X_i = [x_{i,1}, \ldots, x_{i,D}] \tag{4}$$

Each particle has a memory in which it can store the best position and speed it has previously had. Let the best position vector the *i*th particle has previously had be shown with $P_i = [p_{i,1}, \ldots, p_{i,D}]$ and the best speed vector with $V_i = [v_{i,1}, \ldots, v_{i,D}]$.

In this case, the PSO is defined with the following Eqs. 5 and 6;

$$v_{i,d} = \omega v_{i,d} + c_1 r_1 (p_{i,d} - x_{i,d}) + c_2 r_2 (p_{g,d} - x_{i,d}) \tag{5}$$

$$x_{i,d} = x_{i,d} + v_{i,d} \tag{6}$$

where acceleration constants c_1, c_2 and inertia weight ω are predefined by the user and r_1, r_2 are the uniformly generated random numbers in the range of [0 1]. The PSO's success in finding the global optimum depends extremely on the initial values of the control parameters (c_1, c_2, ω) of PSO, the size of population value, and the iteration number.

For more detailed information on PSO, please refer to the study given in [20, 22, 23].

2.3 Artificial Bee Colony (ABC)

The ABC is a population-based numeric optimization algorithm [24, 25]. The ABC is based on the simplified mathematical models of the food searching behaviors of the bee-swarms. The ABC gives successful results in training of the artificial neural networks, IIR filter design, and data-clustering applications. In the ABC, any random solution of the problem corresponds to a *source of nectar*. There is one *employed bee* assigned to each nectar source. The number of employed bees equals to the total number of food sources (i.e. the *size of population* value). The employed bee of a nectar source that has run out of nectar turns into a scout bee again. The amount of nectar in a nectar source is expressed with the objective function value of the related nectar source. Therefore, the ABC targets to locate the nectar source that has the maximum amount of nectar. In the first step of the ABC, random nectar sources are generated. The search space must be considered as the environs of the hive containing the nectar sources. The random generation of nectar sources is made in compliance with Eq. 7;

$$x_{i,j} = x_j^{\min} + rand(0, 1)\ (x_j^{\max} - x_j^{\min}) \tag{7}$$

where $i = 1, 2, 3, ..., SN$, $j = 1, 2, 3, ..., D$ and SN is the number of nectar sources. D indicates the number of parameters to be optimized. $x_j^{\min}$ and $x_j^{\max}$ are the lower and upper limits respectively, which are allowed for the search space of the jth parameter. While starting the calculations, value of the *failure* variable in which the number of failures to develop a nectar source is hidden made $failure_i = 0$.

Following the generation of initial nectar resources, the ABC starts to search for solution of the numeric optimization problem using the employed bee, outlooker bee, and scout-bee tools. The employed bee tries to develop the nectar source to which it is assigned using the other nectar sources as well. If the employed bee finds a better nectar source, it memorizes the new nectar source to use it instead of the old one. In the ABC, this process is modeled in Eq. 8;

$$v_{i,j} = x_{i,j} + \varphi_{i,j}(x_{i,j} - x_{k,j}) \tag{8}$$

where $\varphi_{i,j}$ is a random number generated in the range of $[-1 \quad 1]$. $x_{i,j}$ and $x_{k,j}$ indicate the jth parameters of the ith and kth nectar sources respectively. If the v_i value has a better objective function value than the x_i value, the x_i value is updated as $x_i := v_i$ and the *failure* variable becomes $failure_i = 0$. If the v_i value does not have a better objective function value than x_i, the employed bee continues to go to the x_i source, and since the x_i solution cannot be developed, the $failure_i$ value that is the development meter related to the nectar source x_i increases by one unit. Using the objective function value of all nectar sources, the probability values, p_i, to be used by the outlooker bees are obtained;

$$p_i = \frac{fitness_i}{\sum_{i=1}^{SN} fitness_i} \tag{9}$$

and

$$fitness_i = \begin{cases} \frac{1}{1+f_i} & f_i \geq 0 \\ 1 + |f_i| & f_i < 0 \end{cases} \tag{10}$$

As the $fitness_i$ value given in the equation Eq. 10 increases, the number of employed bees that will select this region of nectar source will increase. The ABC selects the nectar sources to be visited by the bees using the roulette selection technique used in the genetic algorithms; a random number within the range of $[0 \quad 1]$ is generated for each nectar source, if the p_i value is higher than the generated random number, the outlooker bees search for new nectar sources to develop the nectar source x_i using the Eqs. 8–10. If an x_i source has a $failure_i$ value higher than a certain threshold value, that x_i source is left, and the employed bee assigned hereto goes to a random nectar source generated newly.

The success of the ABC in finding the global optimum is sensitive to the control parameters of the algorithm (employed-bee or onlooker-bee number and the limit value), the population size, and the maximum cycle value. For more detailed information on ABC, please refer to the study given in [24, 25].

3 Structural Analysis of Cuckoo Search Algorithm (CS)

The structure of CS is analogically based on the simplified mathematical model of the offspring generation behavior of the Cuckoo-Birds [1]. The Cuckoo-Birds do not incubate in order to generate offsprings. Alternatively, they lay their eggs in the nests of other incubating birds. In this way, the cuckoos push the care and feeding of their offsprings off on a different type of bird. This is a *brood parasitic* type of behavior. The brood parasitic behavior enables the cuckoo to lay more eggs which relatively facilitates the feeding and reproducing of the Cuckoo. However, the Cuckoo has to search on which nest it may lay its eggs.

It is possible to model the motion model of a living thing in nature which changes its location in order to search for a nest or food by using analytic random-walk strategies. The random-walk strategies that are generally the most frequently used are the *Brownian-walk* and *Lèvy-walk* strategies [36–38]. In fact, the *Brownian-walk* is a special case of *Lèvy-walk*. In *Brownian-walk* motion, the step lengths (i.e., s) can be scaled by the first and second moment of the probability density distribution of the step lengths (i.e., $P(s)$). Conceptually, the step lengths in *Lèvy-walk* motion do not have a characteristic scale. In other words, the probability density distribution of the step lengths and its second moment possess the self-affine characteristic (Eq.11);

$$P(\lambda s) \sim \lambda^{-\mu} P(s) \;\mid\; 1 < \mu \leq 3 \tag{11}$$

Figure 1 shows certain paths with equal number of segments which are randomly generated in 2D space by using different random-walk strategies.

In practical applications, the probability density distribution $P(s)$ defined in Eq. (11) is calculated by the observed s values. If it is assumed that the s values have a statistical second-moment, the motion models where the Gaussian, Poisson and other distributions are used can be utilized in order to explain the related random-walk strategy. In this case, the random-walk is named as a Brownian-motion model. Some experimental random-walk observations have shown that the s values did not always require a statistical second-moment. This can be expressed by using the Lèvy distribution given in Eq. (12);

$$P(s) \sim s_j^{-\mu} \;\mid\; 1 < \mu \leq 3 \tag{12}$$

Here, $P(s)$ defines the gaussan Brownian-motion for $\mu \geq 3$. Due to the difficulties in obtaining a mathematical model, a standardizable probability distribution has not been defined for $\mu \leq 1$. Therefore, the Lèvy distribution given in Eq. (12), which is a stable-distribution model is widely used in the solution of many different problems.

A special case of of inverse-gamma distribution, the Lèvy distribution, is a continuous probability distribution generated for non-negative random variables. The Lèvy distribution is a type of stable distribution which can be expressed by analytic probability density functions such as the standard distribution and cauchy

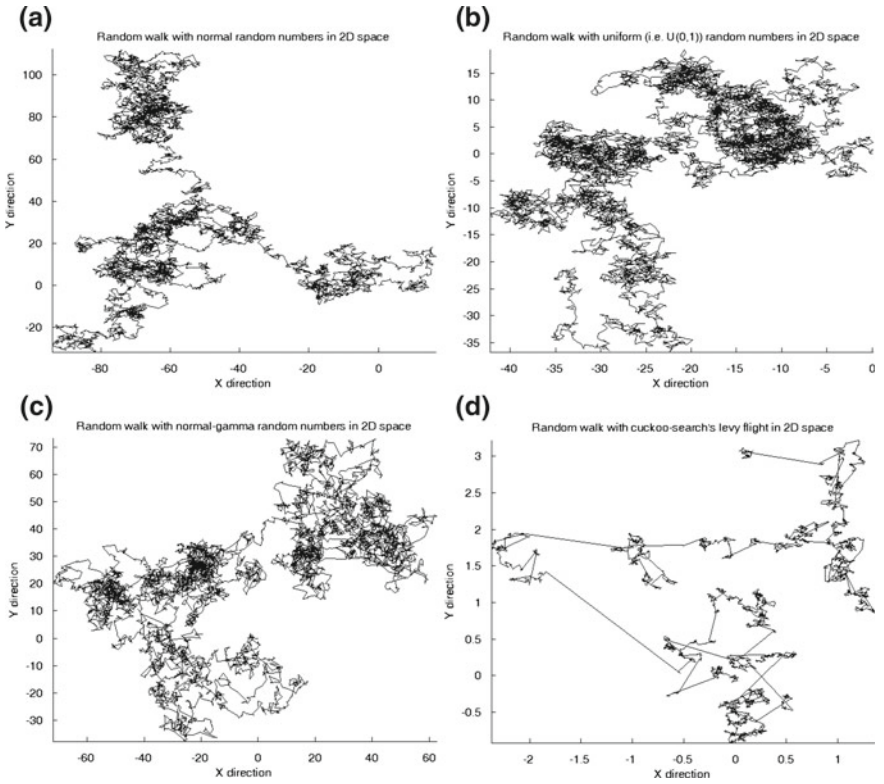

Fig. 1 Examination of the effect of the scale-factor value achieved by different methods (i.e., F) on random-walk in 2D space; **a** $F \sim N(0, 1)$, **b** $F \sim U(0, 1)$, **c** $F \sim \Gamma(\alpha, 1)|\alpha$: *shape factor*, **d** $F \sim (0.01 \cdot L\grave{e}vy(\lambda))$

distribution. Stable distributions are not generally expressed by an analytic probability density function.

α-stable distributions (i.e., $S(\cdot)$) are generally expressed by using four parameters; $S(\alpha, \beta, \gamma, \delta)$. The first parameter, $\alpha \in (0\ 2]$ is named as the characteristic exponent and defines the tail of the distribution. $\beta \in [-1\ 1]$ shows the skewness value. $\beta > 0$ is named as the right-skewed distribution and $\beta < 0$ is named as the left-skewed distribution. $\gamma > 0$ and $\delta \in R$ show the scale and location parameters, respectively. In this case, the Gaussian distribution is defined as $N(\mu, \sigma^2) = S(2, \beta, \frac{\sigma}{\sqrt{2}}, \mu)$. If it is a Cauchy distribution with γ scale and δ location parameters, it can be defined with $S(1, 0, \gamma, \delta)$. In a Lèvy distribution with γ scale and δ location parameters, it is defined with $S(0.5, 1, \gamma, \delta)$.

The *Lèvy-walk* strategy allows more for instant jumpings with big amplitude in search space than *Brownian-walk*. This is a considerably significant feature to avoid the local minimums. Provided that it is in accordance with the selected μ value, the probability of the returning of a living thing in search of nest or food to a location it

has previously visited is less in *Lèvy-walk* than the *Brownian-walk*. In other words, a living thing using the *Lèvy-walk* strategy discovers more new locations than a living thing using the *Brownian-walk* strategy and can find more resources.

CS uses the McCulloch's algorithm in order to generate a random number in α-stable distribution. In this way, a *hypothetic* cuckoo in search of the most convenient nest to lay its eggs successfully simulates the random-walk based search strategically in an analogical way.

The general structure of CS is considerably similar to the general algorithmic structure used for Evolutionary Algorithms and consists of two basic seach stages. The generalized structure of CS is given in Algorithm 1.

Algorithm 1: Generalized Structure of Cuckoo Search Algorithm.

```
Initialization
while stopping conditions are met do
    First Strategy
        Selection
        Mutation
        Update
    end
    Second Strategy
        Selection
        Mutation
        Random-Crossover
        Update
    end
end
```

The pseudo-code of CS given in Algorithm 2 provides more detailed information on its functioning. When Algorithm 2 is examined, it can be seen that CS consists of two basic search strategies. Both search strategies are based on improved random-walk models. The general model of random walk is given in Eq. (13);

$$X_{t+1} = X_t + F \cdot \delta x \mid X = [x_1, x_2, x_3, ..., x_D] \tag{13}$$

where F and δx denote the *scale factor* and *direction-matrix*, respectively. The first search-strategy of CS predicates on evolving all nests towards the nest that provides the best solution; i.e., the first search-strategy of CS is *elitist*. This strategy provides rapid access to a better solution that may be situated between a nest and the nest that provides the best solution. In this strategy, the *scale factor* (i.e., F) that controls the amplitude of δx is a random number generated by the Lèvy distribution. In this strategy, the algorithm suggested by Mantegna (Eq.(14)) has been used in order to generate F values;

$$F = s \cdot \kappa \mid \kappa \sim N(0,1), \;\; s = 0.01 \cdot \frac{u}{|v|^{\frac{1}{\beta}}} \tag{14}$$

where $u \sim N(0, \sigma_u^2)$, $v \sim N(0, \sigma_v^2)$, $\sigma_v = 1$ and $\sigma_u = \left[\frac{\Gamma(1+\beta) \cdot \sin\left(\pi \cdot \frac{\beta}{2}\right)}{\Gamma\left[\frac{1+\beta}{2}\right] \cdot \beta \cdot 2^{\frac{\beta-1}{2}}} \right]^{\frac{1}{\beta}}$.

Algorithm 2: Detailed Pseudo Code of the CS.

Initialization
  Set p_a
  Set Objective function, $f(nest_i) \mid i = 1,2,3,...,nestsize.$
  $nest_{i,j} \sim U(low_j, up_j) \mid j = 1,2,3,...,dim$
end
while *Stopping conditions are met* **do**
  First Strategy
    Find the best nest; $nest_{best}$
    Get a nest; *nest*
    `// Generation of scaled-step size value (i.e., F) by using Lèvy flights`
1   $F = 0.01 \cdot \frac{u}{|v|^{\frac{1}{\beta}}} \cdot n | 1 \leq \beta \leq 2$ and $n, v \sim N(0,1)$, and $u \sim N(0, \sigma_u^2)$
2   where $\sigma_u = \left(\frac{\Gamma(1+\beta)\cdot\sin(\pi\cdot\beta/2)}{\Gamma\left(\frac{1+\beta}{2}\right)\cdot\beta\cdot 2^{\frac{\beta-1}{2}}} \right)^{\frac{1}{\beta}}$ and Γ is gamma function.
    `// Generation of a Cuckoo randomly (i.e., trial)`
3   $trial = nest + F \cdot (nest_{best} - nest)$
4   *Boundary control of trial*
5   $ind \leftarrow f(trial)_{ind} < f(nest) | ind \in \{1,...,nestsize\}$
6   $nest_{ind} = trial_{ind}$ and $f(nest)_{ind} = f(trial)_{ind}$
  end
  Second Strategy
    `// Discover worse nests; wnest`
7   **for** *i=1 to nestsize* **do**
8   **for** *j=1 to dim* **do**
9   **if** $(K_1 > p_a | K_1 \sim U(0,1), 0 < p_a < 1)$ **then**
10   wnest(i,j)=1
11   **else**
12   wnest(i,j)=0
13   **end**
14   **end**
15 **end**
    `// Generation of step size value (i.e., F)`
16 $F = K_2 | K_2 \sim U(0,1)$
    `// Generation of a Cuckoo randomly (i.e., trial)`
17 $trial = nest + F \cdot wnest \circ (nest_{permuted} - nest_{permuted}) | \circ$ *is hadamart operator*
18 *Boundary control of trial*
19 $ind \leftarrow f(trial)_{ind} < f(nest) | ind \in \{1,...,nestsize\}$
20 $nest_{ind} := trial_{ind}$ and $f(nest)_{ind} := f(trial)_{ind}$
  end
end
`// Postprocessing of results`

The second search strategy of CS forces every nest to evolve towards a different nest; i.e., the second search-strategy of CS is an analytically *non-elitist* random search. This strategy explores whether a nest that provides a better solution exists between two nests in a considerably fast manner. In this strategy, the F values controlling the amplitude of δx are random numbers generated by using uniform distribution. It is also decided which eggs within the nest shall evolve at the end of a random self-crossover process. The adaptation of CS according to the solution of various engineering problems is considerably easy as its structure can be easily analyzed. The search strategies of CS enable it to establish an effective balance between its global and local search abilities.

The relatively more compact structure of CS is shown in Algorithm 3.

Algorithm 3: Pseudo Code of the CS.

Initialization
// objective function f(x), x=$\{x_1,x_2,x_3,...,x_D\}$
// Genarate initial population of n host nests x_i, (i=1,2,3,...,n)
1 **while** *stopping conditions are met* **do**

- Get a Cuckoo randomly by *Lèvy* flights
- Evaluate its quality/fitness F_i
- Choose a nest among n (say j) randomly
- If $F_i \geq F_j$ than replace j by the new solution
- A fraction(pa) of worse nests are abandoned and new ones are built
- Keep the best solutions(or nests with quality solutions)
- Rank the solutions and find the current best

2 **end**
// Postprocess results and visualization

4 Experiments

This section presents in detail the tests and benchmark problems, statistical analysis, arithmetic precision and control parameters and stopping conditions used for the optimization algorithms in the tests, along with the statistical results.

4.1 Control Parameters of the Comparison Algorithms

Control parameters of the DE (i.e., DE/rnd/1/bin [1]), PSO and ABC used as the comparison algorithms are given in Table 1.

4.2 Statistical Tests

In order to determine which of the related algorithms were statistically better in the solution of a benchmark problem, pair-wise statistical tests have been performed [39]. Prior to the statistical tests, it was examined whether the related global minimum values complied with the standard distribution or not by means of the Anderson-Darling normality tests and it has been determined that the related values generally

Table 1 Control parameters of the comparison algorithms

Algorithm	Control Parameters.			
DE	$F_{inital} = 0.5$	$CR_{initial} = 0.90$	$\tau_1 = 0.1$	$\tau_2 = 0.1$
PSO	$C_1 = 1.80$	$C_2 = 1.80$	$\omega = 0.5 + (1 - rand)$	
ABC	limit $= N \cdot D$	Size of Employed-Bee = (Size of Colony)/2		

did not comply with the standard distribution. Therefore, Kruskal-Wallis (K-W) test has been chosen as the test tool for the pair-wise statistical test. The statistical significance value for the statistical tests has been specified as $\alpha = 0.05$ and the bonferroni correction method has been used in order to correct the *p-values* obtained by the the statistical tests. The statistical tests performed show that the pair-wise compared data H_0 have equal median distributions.

4.3 Test Functions

The test functions set to be used in the tests must include different types of test functions in order to find out the success of an optimization algorithm in searching the *global optimum* [1, 21–25]. The literature does not include any optimization algorithm that can solve all test functions of different types. In assessment of the success of an optimization algorithm, it is rather important to find out which types of test functions the optimization algorithm solves more successfully. While interpreting the test results, it is necessary to have knowledge about the general features and structures of the test functions used in the tests. Therefore, this section gives information on several structural features of various test functions. A multimodal test function has more than one local optimum solution. In order to test the ability of algorithms to avoid the local minimums, the multimodal test functions are preferred.

In order not to be trapped in the local minimums, an algorithm must have quite developed global exploration ability. Generally, as the size of a test function increases, it becomes more difficult to reach the global optimum value of the test function. In several test functions such as the Perm, Kowalik, and Schaffer functions, the global minimum value is rather close to the local minimum value. In order to reach the global optimum values of these kinds of test functions, the optimization algorithm must have strong local exploitation operators. The global minimum values of several test functions are positioned in a very narrow region of the search space (like Easom, Michalewicz (Dim $= 10$), Powell functions). Therefore, solution of these kinds of functions is very difficult.

Since the Foxholes function has numerous local minimums, it is highly possible to be trapped in one of the local solutions while searching for the global optimum of this function. A κ-dimensional function is categorized as the separable function if it can be expressed with the κ units of single variable functions, or otherwise, as the non-separable function. It is much more difficult to find the global optimums of the non-separable functions than the separable functions. Since the local optimums of the Fletcher-Powell and Langerman functions that are non-symmetric functions are *uniformly distributed* in the *search space*, it is difficult to reach the global optimums of these functions. For $n \geq 30$, the Griewank function transforms from a multimodal function type to a monomodal function and finding its solutions becomes more difficult. The functions that have planar surfaces do not provide information on the search direction. Therefore, it is very difficult to find the global optimums of the functions (such as Stepint, Matyas, and Powersum functions) as well.

Table 2 The benchmark problems used in tests (Type: M: Multimodal, N: Non-Separable, U: Unimodal, S: Seperable, Dim: Dimension, Low, Up: Limits of search space)

Fnc#	Names	Type	Dim	Low	Up	Fnc#	Names	Type	Dim	Low	Up
1	Foxholes	MS	2	−65.536	65.536	26	Michalewics	MS	2	0	π
2	Goldstein-price	MN	2	−2	2	27	Michalewics	MS	5	0	π
3	Penalized	MN	30	−50	50	28	Michalewics	MS	10	0	π
4	Penalized2	MN	30	−50	50	29	Perm	MN	4	−4	4
5	Ackley	MN	30	−32	32	30	Powell	UN	24	−4	5
6	Beale	UN	5	−4.5	4.5	31	Powersum	MN	4	0	4
7	Bohachecsky1	MS	2	−100	100	32	Quartic	US	30	−1.28	1.28
8	Bohachecsky2	MN	2	−100	100	33	Rastrigin	MS	30	−5.12	5.12
9	Bohachecsky3	MN	2	−100	100	34	Rosenbrock	UN	30	−30	30
10	Booth	MS	2	−10	10	35	Schaffer	MN	2	−100	100
11	Branin	MS	2	−5	10	36	Schwefel	MS	30	−500	500
12	Colville	UN	4	−10	10	37	Schwefel_1_2	UN	30	−100	100
13	Dixon-Price	UN	30	−10	10	38	Schwefel_2_22	UN	30	−10	10
14	Easom	UN	2	−100	100	39	Shekel10	MN	4	0	10
15	Fletcher	MN	2	$-\pi$	π	40	Shekel5	MN	4	0	10
16	Fletcher	MN	5	$-\pi$	π	41	Shekel7	MN	4	0	10
17	Fletcher	MN	10	$-\pi$	π	42	Shubert	MN	2	−10	10
18	Griewank	MN	30	−600	600	43	Camelback	MN	2	−5	5
19	Hartman3	MN	3	0	1	44	Sphere2	US	30	−100	100
20	Hartman6	MN	6	0	1	45	Step2	US	30	−100	100
21	Kowalik	MN	4	−5	5	46	Stepint	US	5	−5.12	5.12
22	Langermann	MN	2	0	10	47	Sumsquares	US	30	−10	10
23	Langermann	MN	5	0	10	48	Trid	UN	6	−36	36
24	Langermann	MN	10	0	10	49	Trid	UN	10	−100	100
25	Matyas	UN	2	−10	10	50	Zakharov	UN	10	−5	10

In this chapter, the Quartic test function has been used to examine the success of the optimization algorithms in solving the noisy test functions. It is also very difficult to solve the Penalized test function composed of the combination of different periods of the sinus-based functions. If an optimization algorithm is unable to monitor the direction changes in the functions that have narrow and sinuous valleys, as is the case in the Beale, Colville, and Rosenbrock test functions, it will have difficulty in solving the other test problems of the same type. Generally, if the polynomial degree of the test function increases, it becomes more difficult for them to reach the global optimum (Goldstein-Price, Trid). In the Tests, the widely used 50 benchmark problems have been used [22–24].

Several features of the benchmark problems used in the Tests are given in Table 2.

4.4 Algorithmic Precision

It is sufficient for the arithmetic precision level to be 10^{-16} for the solution of many practical numerical problems. Therefore, the arithmetic precision level has been taken as 10^{-16} regarding the tests performed in this chapter.

4.5 Statistical Results of Tests

In this chapter, the related benchmark problems have been solved by 30 trials by using DE, PSO, ABC and CS. A different initial population has been used in each trial. The mean (i.e., Mean), standard deviation of Mean (i.e., Std), best-solution (i.e., Best), and runtime in seconds (i.e., Runtime) values of the solutions achieved for any and every benchmark problem are given in Tables 3, 4, 5 and 6.

Since the solutions obtained did not generally comply with the standard distribution, the K–W test has been used in order to find the algorithm that statistically better solved any and every benchmark problem. The algorithms that statistically better solve the related benchmark problem according to the K–W test are given in Table 7.

The number of benchmark problems for which CS provides statistically better solutions (i.e, '+'), the number of benchmark problems where the solutions achieved by CS are statistically identical with the related comparison algorithm (i.e, ' =') and the number of benchmark problems where the related comparison algorithms provides statistically better solutions than CS (i.e, '−') are given in the last row of Table 7 in the following format: ' + / = /−'. When the ' + / = /−' values are examined, it can be seen that CS is generally more successful when compared with DE, PSO and ABC. DE's problem solving ability is close to CS; however, CS is generally much faster than DE, as it is seen in Tables 3, 4, 5 and 6. The problem solving success of PSO and ABC substantially resembles one another.

5 Conclusions

In this chapter, the algorithmic process and problem solving ability of CS has been studied. CS has two search strategies; the first strategy is equipped with the elitist search ability whereas the second strategy is equipped with the random search ability. Since the first search strategy controls the amplitude of the *direction-matrix* by using Lèvy distribution based random numbers, it is able to analyze the search space between a nest and the best-nest in a very fast and effective manner. The second search strategy is based on an improved random search strategy and forces the nests to evolve towards one another. When the results obtained from the tests performed were statistically studied, it's been seen that the search strategies could analyze the search space very efficiently when used together. It has been seen that the structure

Table 3 The simple statistical values of the results acquired in the Tests

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
1	Mean	0.9980038377944496	1.3316029264876299	0.9980038377944496	0.9980038377944496
	Std	0.0000000000000000	0.9455237994690697	0.0000000000000001	0.0000000000000000
	Best	0.9980038377944496	0.9980038377944496	0.9980038377944496	0.9980038377944496
	Runtime	66.536	72.527	64.976	51.141
2	Mean	2.9999999999999196	2.9999999999999205	3.0000000465423020	2.9999999999999196
	Std	0.0000000000000020	0.0000000000000013	0.0000002350442161	0.0000000000000009
	Best	2.9999999999999192	2.9999999999999205	2.9999999999999205	2.9999999999999192
	Runtime	13.257	17.892	16.624	19.269
3	Mean	0.0500386842578376	0.1278728062391630	0.0000000000000004	0.0000000000000000
	Std	0.0946840601194795	0.2772792346028402	0.0000000000000001	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000003	0.0000000000000000
	Runtime	159.147	139.555	84.416	65.630
4	Mean	0.1102007525177103	0.0043949463343535	0.0000000000000004	0.0000000000000000
	Std	0.3158351992949586	0.0054747064090173	0.0000000000000001	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000003	0.0000000000000000
	Runtime	223.905	126.506	113.937	79.332
5	Mean	0.4152004039640245	1.5214322973725012	0.0000000000000340	0.0000000000000044
	Std	0.6124958251049220	0.6617570384662601	0.0000000000000035	0.0000000000000000
	Best	0.0000000000000044	0.0000000000000080	0.0000000000000293	0.0000000000000044
	Runtime	82.520	63.039	23.293	41.791

(continued)

Table 3 (continued)

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
6	Mean	0.0000000000000000	0.0000000041922968	0.0000000000000028	0.0000000000000000
	Std	0.0000000000000000	0.0000000139615552	0.0000000000000030	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000005	0.0000000000000000
	Runtime	15.391	32.409	22.367	4.254
7	Mean	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Std	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Runtime	13.903	16.956	1.832	3.828
8	Mean	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Std	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Runtime	13.846	17.039	1.804	3.853
9	Mean	0.0000000000000000	0.0000000000000000	0.0000000000000006	0.0000000000000000
	Std	0.0000000000000000	0.0000000000000000	0.0000000000000003	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000001	0.0000000000000000
	Runtime	13.781	17.136	21.713	3.864
10	Mean	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Std	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Runtime	13.822	17.072	22.395	3.758

(continued)

Table 3 (continued)

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
11	Mean	0.3978873577297382	0.3978873577297382	0.3978873577297382	0.3978873577297382
	Std	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Best	0.3978873577297382	0.3978873577297382	0.3978873577297382	0.3978873577297382
	Runtime	12.982	17.049	10.941	13.538
12	Mean	0.6650611877014533	0.0000000000000000	0.0715675060725970	0.0000000000000000
	Std	1.4200613758851144	0.0000000000000000	0.0579425013417103	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0013425253994745	0.0000000000000000
	Runtime	56.849	44.065	21.487	8.197
13	Mean	6.2961671726817778	0.6666666666666752	0.0000000000000038	0.2308771778828254
	Std	28.3483110247009580	0.0000000000000022	0.0000000000000012	0.3144393814225666
	Best	0.0000054423627161	0.6666666666666722	0.0000000000000021	0.0000000002548662
	Runtime	121.817	167.094	37.604	144.849
14	Mean	−1.0000000000000000	−1.0000000000000000	−1.0000000000000000	−1.0000000000000000
	Std	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Best	−1.0000000000000000	−1.0000000000000000	−1.0000000000000000	−1.0000000000000000
	Runtime	13.078	16.633	13.629	14.040
15	Mean	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Std	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Runtime	27.273	27.859	40.030	7.368

Table 4 Table 3 (continued)

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
16	Mean	0.0000000000000000	48.7465164446927300	0.0218688498331872	0.0000000000000000
	Std	0.0000000000000000	88.8658510972990570	0.0418409568792831	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000016	0.0000000000000000
	Runtime	29.050	95.352	44.572	45.046
17	Mean	318.8955239565633500	918.9518492782851800	11.0681496253547970	0.0318104464725031
	Std	1216.2409310687658000	1652.4810858411431000	9.8810950146557062	0.1623759402353626
	Best	0.0000000000000000	0.0000000000000000	0.3274654777056863	0.0000000000000000
	Runtime	245.294	271.222	43.329	215.143
18	Mean	0.0341098610045316	0.0068943694819713	0.0000000000000000	0.0000000000000000
	Std	0.0599461900320241	0.0080565201649587	0.0000000000000001	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Runtime	93.905	73.895	19.073	26.068
19	Mean	−3.8627821478207531	−3.8627821478207531	−3.8627821478207531	−3.8627821478207531
	Std	0.0000000000000027	0.0000000000000027	0.0000000000000024	0.0000000000000027
	Best	−3.8627821478207558	−3.8627821478207558	−3.8627821478207558	−3.8627821478207558
	Runtime	13.684	19.280	12.613	14.686
20	Mean	−3.2982165473202611	−3.3180320675402468	−3.3219951715842440	−3.3219951715842440
	Std	0.0483702518391573	0.0217068148263721	0.0000000000000014	0.0000000000000013
	Best	−3.3219951715842431	−3.3219951715842431	−3.3219951715842426	−3.3219951715842431
	Runtime	16.836	26.209	13.562	16.624

(continued)

Table 4 (continued)

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
21	Mean	0.0003578568116377	0.0003074859878056	0.0004414866359626	0.0003074859878056
	Std	0.0000994760026221	0.0000000000000000	0.0000568392289725	0.0000000000000000
	Best	0.0003074859878056	0.0003074859878056	0.0003230956007045	0.0003074859878056
	Runtime	64.182	84.471	20.255	31.681
22	Mean	−1.0809384421344381	−1.0809384421344381	−1.0809384421344381	−1.0809384421344381
	Std	0.0000000000000005	0.0000000000000006	0.0000000000000008	0.0000000000000005
	Best	−1.0809384421344377	−1.0809384421344377	−1.0809384421344375	−1.0809384421344377
	Runtime	29.440	27.372	27.546	40.445
23	Mean	−1.4802659159074119	−1.3891992200744647	−1.4999990070800762	−1.4999992233524957
	Std	0.1080837762185712	0.2257194403158633	0.0000008440502079	0.0000000000000009
	Best	−1.4999992233524948	−1.4999992233524948	−1.4999992233524946	−1.4999992233524948
	Runtime	29.055	33.809	37.986	40.721
24	Mean	−1.0803438401124403	−0.9166206788680230	−0.8406348096500682	−1.2511850186414166
	Std	0.4182197082975204	0.3917752367440502	0.2000966365984315	0.3334749587976915
	Best	−1.5000000000003775	−1.5000000000003775	−1.4999926800631387	−1.5000000000003775
	Runtime	33.237	110.798	38.470	93.591
25	Mean	0.0000000000000000	0.0000000000000000	0.0000000000000004	0.0000000000000000
	Std	0.0000000000000000	0.0000000000000000	0.0000000000000003	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000001	0.0000000000000000
	Runtime	17.267	25.358	19.689	3.826

(continued)

Table 4 (continued)

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
26	Mean	−1.8210436836776813	−1.8210436836776813	−1.8210436836776813	−1.8210436836776813
	Std	0.0000000000000009	0.0000000000000009	0.0000000000000009	0.0000000000000009
	Best	−1.8210436836776822	−1.8210436836776822	−1.8210436836776822	−1.8210436836776822
	Runtime	16.977	19.154	17.228	17.030
27	Mean	−4.6802510126075614	−4.6565646397053939	−4.6934684519571128	−4.6934684519571128
	Std	0.0306286262511556	0.0557021530063238	0.0000000000000009	0.0000000000000009
	Best	−4.6934684519571128	−4.6934684519571128	−4.6934684519571128	−4.6934684519571128
	Runtime	24.398	38.651	17.663	25.927
28	Mean	−9.5570815083481193	−8.9717330307549314	−9.6601517156413479	−9.6601517156413479
	Std	0.0846395763792892	0.4927013165009223	0.0000000000000008	0.0000000000000000
	Best	−9.6601517156413497	−9.5777818097208236	−9.6601517156413497	−9.6601517156413479
	Runtime	36.466	144.093	27.051	81.394
29	Mean	0.0775709724361173	0.0119687224560441	0.0838440014038032	0.0000797898440265
	Std	0.1395948617599293	0.0385628598040034	0.0778327303965192	0.0001650496286214
	Best	0.0047968153181906	0.0000044608370213	0.0129834451730589	0.0000000000000000
	Runtime	355.898	359.039	60.216	351.339
30	Mean	0.0000142170357927	0.0000130718912008	0.0002604330013462	0.0000000160437924
	Std	0.0000090808423976	0.0000014288348929	0.0000394921919294	0.0000000105899101
	Best	0.0000005517627966	0.0000095067504097	0.0001682411286088	0.0000000044314919
	Runtime	281.437	567.704	215.722	255.702

Table 5 Table 4 (continued)

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
31	Mean	0.0006372343018424	0.0001254882834238	0.0077905311094958	0.0000031691171181
	Std	0.0017035052280704	0.0001503556280087	0.0062425841086448	0.0000085280644195
	Best	0.0000000925033022	0.0000000156460198	0.0003958766023752	0.0000000000039897
	Runtime	278.417	250.248	34.665	161.744
32	Mean	0.0013731616016240	0.0003548345513179	0.0250163252527030	0.0013585782131698
	Std	0.0008828072208019	0.0001410817500914	0.0077209314806873	0.0009418799367612
	Best	0.0004937236463060	0.0001014332605364	0.0094647580732654	0.0000384178687965
	Runtime	120.578	290.669	34.982	52.367
33	Mean	18.5881360081950720	25.6367602258675550	0.0000000000000000	0.0000000000000000
	Std	4.4411802594305696	8.2943512684216660	0.0000000000000000	0.0000000000000000
	Best	8.9546317787740577	12.9344677422128600	0.0000000000000000	0.0000000000000000
	Runtime	100.909	76.083	4.090	32.431
34	Mean	43.7705466083793910	2.6757043114269696	0.2856833465904132	0.0000000000000000
	Std	74.8808425376813460	12.3490058210003680	0.6247370987465166	0.0000000000000000
	Best	0.0259391054886982	0.0042535368984501	0.0004266049929880	0.0000000000000000
	Runtime	127.729	559.966	35.865	83.362
35	Mean	0.0006477273251676	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Std	0.0024650053428137	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Runtime	13.843	18.163	7.861	9.978

(continued)

Table 5 (continued)

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
36	Mean	−11423.6870113832680000	−7684.6104757783769000	−12569.4866181730160000	−12569.4866181730160000
	Std	357.6086310237069500	745.3954005014177300	0.0000000000022659	0.0000000000018501
	Best	−11977.2949451008230000	−8912.8855854978192000	−12569.4866181730140000	−12569.4866181730140000
	Runtime	78.789	307.427	19.225	37.787
37	Mean	0.0000000000000000	0.0000000000000000	14.5668734126948150	0.0000000000000000
	Std	0.0000000000000000	0.0000000000000000	8.7128443012950338	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	4.0427699323673369	0.0000000000000000
	Runtime	137.883	543.180	111.841	88.186
38	Mean	0.0179428067552174	0.0000000000000000	0.0000000000000005	0.0000000000000000
	Std	0.0859236379042153	0.0000000000000000	0.0000000000000001	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000003	0.0000000000000000
	Runtime	35.487	163.188	20.588	15.865
39	Mean	−10.3130437162426120	−10.1061873621653040	−10.5364098166920480	−10.5364098166920500
	Std	1.2234265179812154	1.6679113661236413	0.0000000000000023	0.0000000000000018
	Best	−10.5364098166920480	−10.5364098166920500	−10.5364098166920480	−10.5364098166920480
	Runtime	14.149	31.018	16.015	17.704
40	Mean	−10.1531996790582240	−9.5373938082045466	−10.1531996790582240	−10.1531996790582240
	Std	0.0000000000000072	1.9062127067994237	0.0000000000000055	0.0000000000000072
	Best	−10.1531996790582310	−10.1531996790582310	−10.1531996790582310	−10.1531996790582310
	Runtime	13.740	25.237	11.958	17.542

(continued)

Table 5 (continued)

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
41	Mean	−10.2257649420933840	−10.4029405668186640	−10.4029405668186640	−10.4029405668186640
	Std	0.9704308630210545	0.0000000000000018	0.0000000000000006	0.0000000000000017
	Best	−10.4029405668186660	−10.4029405668186660	−10.4029405668186640	−10.4029405668186660
	Runtime	14.435	21.237	14.911	20.630
42	Mean	−186.7309088310239500	−186.7309073569884400	−186.7309088310239500	−186.7309088310239500
	Std	0.0000000000000366	0.0000046401472660	0.0000000000000236	0.0000000000000167
	Best	−186.7309088310239800	−186.7309088310239200	−186.7309088310239500	−186.7309088310239800
	Runtime	13.933	19.770	13.342	26.681
43	Mean	−1.0316284534898770	−1.0316284534898770	−1.0316284534898770	−1.0316284534898770
	Std	0.0000000000000005	0.0000000000000005	0.0000000000000005	0.0000000000000005
	Best	−1.0316284534898774	−1.0316284534898774	−1.0316284534898774	−1.0316284534898774
	Runtime	13.372	16.754	11.309	13.833
44	Mean	0.0218469674398909	0.0000000000000000	0.0000000000000004	0.0000000000000000
	Std	0.0808568141606462	0.0000000000000000	0.0000000000000001	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000003	0.0000000000000000
	Runtime	35.519	159.904	21.924	16.334
45	Mean	0.0000000000000000	2.2999999999999998	0.0000000000000000	0.0000000000000000
	Std	0.0000000000000000	1.8597367258983657	0.0000000000000000	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Runtime	15.028	57.276	1.782	13.821

Table 6 Table 5 (continued)

Fnc #	Statistics	*PSO*	*DE*	*ABC*	*CS*
46	Mean	0.0000000000000000	0.1333333333333333	0.0000000000000000	0.0000000000000000
	Std	0.0000000000000000	0.3457459036417604	0.0000000000000000	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Runtime	13.178	20.381	1.700	25.306
47	Mean	0.1395094548769303	0.0000000000000000	0.0000000000000005	0.0000000000000000
	Std	0.5704849211672392	0.0000000000000000	0.0000000000000000	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000003	0.0000000000000000
	Runtime	44.036	564.178	24.172	19.030
48	Mean	−50.0000000000002130	−50.0000000000002060	−49.9999999999996590	−50.0000000000002060
	Std	0.0000000000000407	0.0000000000000361	0.0000000000001408	0.0000000000000361
	Best	−50.0000000000002270	−50.0000000000001710	−50.0000000000000570	−50.0000000000001710
	Runtime	15.308	24.626	22.480	17.829
49	Mean	−209.9986720395207800	−210.0000000000014500	−209.9999999999470500	−210.0000000000026700
	Std	0.0043626506917265	0.0000000000009434	0.0000000000138503	0.0000000000002308
	Best	−210.0000000000036400	−210.0000000000027300	−209.9999999999690800	−210.0000000000027300
	Runtime	83.617	48.580	36.639	23.751
50	Mean	0.0000004465299315	0.0000000000000000	0.0000000402380424	0.0000000000000000
	Std	0.0000022016806583	0.0000000000000000	0.0000002203520334	0.0000000000000000
	Best	0.0000000000000000	0.0000000000000000	0.0000000000000210	0.0000000000000000
	Runtime	48.668	86.369	86.449	9.693

Table 7 Determining the algorithm that statistically provides the best solution for each benchmark problem used in Tests by utilizing Kruskal–Wallis Test ($\alpha = 0.05$)

Fnc #	CS vs. DE	CS vs. PSO	CS vs. ABC
1	CS & DE	CS	CS
2	CS	CS	CS
3	CS	CS	CS
4	CS	CS	CS
5	CS	CS	CS
6	CS & DE	CS	CS
7	CS & DE	CS & PSO	CS & ABC
8	CS & DE	CS & PSO	CS & ABC
9	CS & DE	CS & PSO	CS
10	CS & DE	CS & PSO	CS & ABC
11	CS & DE	CS & PSO	CS & ABC
12	CS	CS & PSO	CS
13	CS	CS	ABC
14	CS & DE	CS & PSO	CS & ABC
15	CS & DE	CS & PSO	CS & ABC
16	CS & DE	CS	CS
17	CS	CS	CS
18	CS	CS	CS
19	CS & DE	CS & PSO	CS
20	DE	CS & PSO	CS & ABC
21	CS	CS & PSO	CS
22	CS & DE	CS	CS
23	CS & DE	CS	CS
24	CS & DE	CS	CS
25	CS & DE	CS & PSO	CS
26	CS & DE	CS & PSO	CS & ABC
27	CS & DE	CS	CS & ABC
28	CS	CS	ABC
29	CS	CS	CS
30	CS	CS	CS
31	CS	CS	CS
32	CS & DE	PSO	CS
33	CS	CS	CS & ABC
34	CS	CS	CS
35	CS & DE	CS & PSO	CS & ABC
36	CS	CS	CS
37	CS & DE	CS & PSO	CS
38	CS	CS & PSO	CS
39	CS & DE	PSO	CS
40	CS & DE	CS & PSO	CS
41	CS & DE	PSO	CS
42	CS	CS	CS
43	CS & DE	CS & PSO	CS & ABC
44	CS	CS & PSO	CS

(continued)

Table 7 (continued)

Fnc #	CS vs. DE	CS vs. PSO	CS vs. ABC
45	CS & DE	CS	CS & ABC
46	CS & DE	CS	CS & ABC
47	CS	CS & PSO	CS
48	DE	CS & PSO	CS
49	CS & DE	CS	CS
50	CS & DE	CS & PSO	CS
$+/=/-$	20/28/2	25/22/3	34/14/2

of CS which included a very few numbers of *if-statements* in terms of software development technique had enabled it to process so quickly. However, particularly the Lèvy flight (i.e., *Lèvy walk*) used in the first strategy prominently reduces the search time of CS. CS is statistically more successful than DE, PSO and ABC in terms of problem solving ability in accordance with the results obtained from the tests.

Acknowledgments The studies in this chapter have been supported within the scope of the scientific research project of 110Y309 supported by TUBITAK.

References

1. Yang, X.S., Deb, S.: Cuckoo search via Lèvy flights. World Congress on Nature and Biologically Inspired Computing'NaBIC-2009, Coimbatore, India, vol. 4, pp. 210–214 (2009)
2. Civicioglu, P., Besdok, E.: A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artif. Intell. Rev. **39**, 315–346 (2013)
3. Dhivya, M., Sundarambal, M.: Cuckoo Search for data gathering in wireless sensor networks. Int. J. Mob. Commun. **9**, 642–656 (2011)
4. Layeb, A.: A novel quantum inspired cuckoo search for knapsack problems. Int. J. Bio-Insp. Comput. **3**, 297–305 (2011)
5. Walton, S., Hassan, O., Morgan, K., Brown, M.R.: Modified cuckoo search: a new gradient free optimisation algorithm. Chaos. Soliton. Fract. **44**, 710–718 (2011)
6. Abdul Rani, K.N., Abd Malek, M.F., Siew-Chin, N.: Nature-inspired cuckoo search algorithm for side lobe suppression in a symmetric linear antenna array. Radioengineering **21**, 865–874 (2012)
7. Durgun, I., Yildiz, A.R.: Structural design optimization of vehicle components using Cuckoo Search Algorithm. Mater. Test. **54**, 185–188 (2012)
8. Gherboudj, A., Layeb, A., Chikhi, S.: Solving 0–1 knapsack problems by a discrete binary version of cuckoo search algorithm. Int. J. Bio-Insp. Comput. **4**, 229–236 (2012)
9. Marichelvam, M.K.: An improved hybrid cuckoo search (IHCS) metaheuristics algorithm for permutation flow shop scheduling problems. Int. J. Bio-Insp. Comput. **4**, 200–205 (2012)
10. Moravej, Z., Akhlaghi, A.: A new approach for DG allocation in distribution network with time variable loads using cuckoo search. Int. Rev. Electr. Eng-I. **7**, 4027–4034 (2012)

11. Natarajan, A., Subramanian, S., Premalatha, K.: A comparative study of cuckoo search and bat algorithm for Bloom filter optimisation in spam filtering. Int. J. Bio-Insp. Comput. **4**, 89–99 (2012)
12. Srivastava, P.R., Sravya, C., Ashima, S., Kamisetti, S., Lakshmi, M.: Test sequence optimisation: an intelligent approach via cuckoo search. Int. J. Bio-Insp. Comput. **4**, 139–148 (2012)
13. Srivastava, P.R., Varshney, A., Nama, P., Yang, X.-S.: Software test effort estimation: a model based on cuckoo search. Int. J. Bio-Insp. Comput. **4**, 278–285 (2012)
14. Burnwal, S., Deb, S.: Scheduling optimization of flexible manufacturing system using cuckoo search-based approach. Int. J. Adv. Manuf. Tech. **64**, 951–959 (2013)
15. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng. Comput. **29**, 17–35 (2013)
16. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Global. Optim. **11**, 341–359 (1997)
17. Qin, A.K., Suganthan, P.N.: Self-adaptive differential evolution algorithm for numerical optimization. IEEE. C. Evol. Computat. **1–3**, 1785–1791 (2005)
18. Igel, C., Hansen, N., Roth, S.: Covariance matrix adaptation for multi-objective optimization. Evol. Comput. **15**, 1–28 (2007)
19. Tsoulos, I.G., Stavrakoudis, A.: Enhancing PSO methods for global optimization. Appl. Math. Comput. **216**(10), 2988–3001 (2010)
20. Clerc, M., Kennedy, J.: The particle swarm—explosion, stability, and convergence in a multidimensional complex space. IEEE Trans. Evol. Comput. **6**, 58–73 (2002)
21. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: GSA: a gravitational search algorithm. Inform. Sci. **13**, 2232–2248 (2009)
22. Civicioglu, P.: Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm. Comput. Geosci. **46**, 229–247 (2012)
23. Civicioglu, P.: Backtracking search optimization algorithm for numerical optimization problems. Appl. Math. Comput. **219**, 8121–8144 (2013)
24. Karaboga, D., Akay, B.: A comparative study of artificial bee colony algorithm. Appl. Math. Comput. **214**, 108–132 (2009)
25. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Global. Optim. **39**, 459–471 (2007)
26. Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans. Evol. Comput. **13**, 398–417 (2009)
27. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V.: Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Trans. Evol. Comput. **10**, 646–657 (2006)
28. Tasgetiren, M.F., Suganthan, P.N., Pan, Q.K.: An ensemble of discrete differential evolution algorithms for solving the generalized traveling salesman problem. Appl. Math. Comput. **215**, 3356–3368 (2010)
29. Liang, J.J., Qin, A.K., Suganthan, P.N., Baskar, S.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans. Evol. Comput. **10**, 281–295 (2006)
30. He, Q., Wang, L.: An effective co-evolutionary particle swarm optimization for constrained engineering design problems. Eng. Appl. Artif. Intel. **20**, 89–99 (2007)
31. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. B. **26**, 29–41 (1996)
32. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**, 182–197 (2002)
33. Ishibuch, H., Yoshida, T., Murata, T.: Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE. Trans. Evol. Comput. **7**, 204–223 (2003)
34. Kishore, J.K., Patnaik, L.M., Mani, V., Agrawal, V.K.: Application of genetic programming for multicategory pattern classification. IEEE Trans. Evol. Comput. **4**, 242–258 (2000)

35. de Carvalho, M.G., Laender, A.H.F., Goncalves, M.A., et al.: A genetic programming approach to record deduplication. IEEE Trans. Knowl. Data. Eng. **24**, 399–412 (2012)
36. Liang, Y., Chen, W.: A survey on computing lèvy stable distributions and a new MATLAB toolbox. Signal. Process. **93**, 242–251 (2013)
37. Hu, Y., Zhang, J., Di, Z., Huan, D.: Toward a general understanding of the scaling laws in human and animal mobility. Europ. Phys. Lett. **96**, 38006–p1-p6 (2011).
38. Humphries, N.E., Weimerskirch, H., Queiroza, N., Southall, E.J., Sims, D.W.: Foraging success of biological lèvy flights recorded in situ. PNAS **109**, 7169–7174 (2012)
39. Derrac, J., Garcia, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol. Comput. **1**, 3–18 (2011)

Cuckoo Search and Firefly Algorithm Applied to Multilevel Image Thresholding

Ivona Brajevic and Milan Tuba

Abstract Multilevel image thresholding is a technique widely used in image processing, most often for segmentation. Exhaustive search is computationally prohibitively expensive since the number of possible thresholds to be examined grows exponentially with the number of desirable thresholds. Swarm intelligence metaheuristics have been used successfully for such hard optimization problems. In this chapter we investigate performance of two relatively new swarm intelligence algorithms, cuckoo search and firefly algorithm, applied to multilevel image thresholding. Particle swarm optimization and differential evolution algorithms have also been implemented for comparison. Two different objective functions, Kapur's maximum entropy thresholding function and multi Otsu between-class variance, were used on standard benchmark images with known optima from exhaustive search (up to five threshold points). Results show that both, cuckoo search and firefly algorithm, exhibit superior performance and robustness.

Keywords Swarm intelligence · Nature inspired algorithms · Optimization metaheuristics · Cuckoo search · Firefly algorithm · Image processing · Multilevel image thresholding

1 Introduction

Image segmentation is a process of dividing an image into its constituent regions or objects, which are visually distinct and uniform with respect to some property, such as gray level, color or texture. The aim of segmentation is to simplify the representation

I. Brajevic (✉)
University of Belgrade, Studentski trg 16, 11000 Belgrade, Serbia
e-mail: ivona.brajevic@googlemail.com

M. Tuba
Megatrend University Belgrade, Bulevar umetnosti 29, 11070 N. Belgrade, Serbia
e-mail: tuba@ieee.org

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_6,

of an image into the set of segments that are more appropriate for further analysis. Some of the practical applications include computer vision [14], industrial quality control [25], medical imaging [1, 6, 13], etc. Since the problem of digital image segmentation is an important research field, many segmentation algorithms have been proposed in the literature.

Thresholding is one of the most widely used approaches to image segmentation because of its intuitive properties and simplicity of implementation. It tries to identify and extract a target from its background on the basis of the distribution of gray levels or texture in image objects. If the image is divided into two classes, such as the background and the object of interest, it is called bi-level thresholding. To split an image into more than two classes, bi-level thresholding is extended to multilevel thresholding.

Computing the thresholds can be performed by parametric or nonparametric methods [50]. Parametric methods assume that gray level distribution of each class obeys a given distribution and finds the optimal threshold by estimating the parameters of the distribution using the given histogram. This leads to a nonlinear estimation problem which is computationally expensive. The nonparametric methods are based on a search for the thresholds by optimizing some criteria. They have been proven to be more robust and accurate compared to the parametric ones.

Among various thresholding criteria which have been proposed in the literature [32], maximum entropy and maximum between-class variance are the most used ones since entropy is widely used optimization criterion [18, 37]. The entropy-based criterion aims to maximize the sum of entropies for each class [21]. Using maximum entropy as an optimality criterion for image thresholding was first proposed by Pun. Later, Kapur found some flaws in Puns derivations and presented corrected and improved version. There are variations of these criteria, such as information-theoretic criterion based on Renyi's entropy [28] or a unified method proposed in [38]. In addition, besides Kapur's entropy, some other entropy measures exist in the literature, such as minimum cross entropy or a measure of fuzzy entropy. Sezgin and Sankur concluded that Kapur's entropy method is better performing thresholding algorithm compared to the other entropy-based methods in the case of nondestructive testing images [32]. Another important criterion is between-class variance defined by Otsu, which aims at maximizing the separability of the classes measured by the sum of between-class variances [26]. Otsu's method gives satisfactory results when the numbers of pixels in each class are close to each other. For bi-level thresholding, these criteria facilitated efficient algorithms, while for optimal multilevel thresholding, computational complexity of the existing conventional algorithms grows exponentially with the number of thresholds.

Considering the computational inefficiency of the traditional exhaustive methods, employing metaheuristics to search for the optimal thresholds have attracted a lot of researchers. A metaheuristic is a general algorithmic framework which can be used for different optimization problems with relatively few modifications to adapt it to a particular problem. These algorithms are able to find very high-quality suboptimal solutions for hard optimization problems in a reasonable amount of time [46, 47]. Therefore, some metaheuristic algorithms have been adopted to search for the

multilevel thresholds. The most successful approaches include nature inspired algorithms [43] with its subclass of swarm intelligence metaheuristics. There is no universally best algorithm [42], so many different metaheuristics have been developed for different classes of problems, combinatorial and continuous, with additions for constrained optimization [9]. These metaheuristics include differential evolution (DE), particle swarm optimization (PSO), honey bee mating optimization (HBMO), bacterial foraging (BF) etc. Usually, initial version introduces algorithm that mimics certain natural phenomenon, while later versions include modifications and hybridizations to improve performance for some classes of problems. Examples are ant colony optimization (ACO) [7] with numerous improvements and applications [19, 20, 35], artificial bee colony (ABC) algorithm [22] with modifications [3, 4], human seeker optimization (HSO) [5, 36], etc.

DE based approach to multilevel thresholding using minimum cross-entropy criterion is proposed in [29]. Yin adopted PSO algorithm to search for the thresholds using the minimum cross-entropy criterion [51]. Horng applied the honey bee mating optimization (HBMO) and ABC to search for the thresholds using the maximum entropy criterion [15, 16]. Akay provided the comparative study of the PSO and ABC for multilevel thresholding using Kapur's and Otsu's criteria [2]. Sathya and Kayalvizhi employed bacterial foraging (BF) and its modified version to select multilevel thresholds using maximum entropy and between-class variance criteria [30, 31]. The adaptation and comparison of six metaheuristic algorithms to multilevel thresholding using between-class variance criterion were presented in [12]. The experimental results have shown that the PSO and DE outperformed the ant colony optimization (ACO), simulated annealing (SA), tabu search (TS) and genetic algorithm (GA) based approaches.

This paper aims to adopt two recently proposed swarm intelligence algorithms, cuckoo search (CS) and firefly algorithm (FA) to search for the optimal multilevel thresholds using Kapur and Otsu criteria. The DE and PSO algorithms were also implemented for purpose of comparison with CS and FA. The exhaustive search method was conducted for deriving the optimal solutions for comparison with the results generated by these four algorithms. The rest of the paper is organized as follows. Section 2 formulates the thresholding problem and presents Kapur and Otsu objective functions. Sections 3 and 4 present CS and FA adopted to search for the optimal multilevel thresholds respectively. Comparative results of the implemented CS, FA, DE and PSO algorithms are presented in Sect. 5. Our conclusions are provided in Sect. 6.

2 Multilevel Thresholding Problem Formulation

Image thresholding technique includes bi-level thresholding and multilevel thresholding. The main objective of bi-level thresholding is to determine one threshold which separates pixels into two groups. One group, G_0, includes those pixels with gray levels below t, i.e. object points, while the other group, G_1, includes the rest,

i.e. background points. Bi-level thresholding can be defined by:

$$G_0 = \{(x, y) \in I \mid 0 \leq f(x, y) \leq t - 1\}$$
$$G_1 = \{(x, y) \in I \mid t \leq f(x, y) \leq L - 1\} \quad (1)$$

The main objective of multilevel thresholding is to determine more than one threshold value which divide pixels into several groups. Let there be L gray levels in a given image I and these gray levels are in the range $0, 1, \ldots, L - 1$. Multilevel thresholding can be defined by:

$$G_0 = \{(x, y) \in I \mid 0 \leq f(x, y) \leq t_1 - 1\}$$
$$G_1 = \{(x, y) \in I \mid t_1 \leq f(x, y) \leq t_2 - 1\}$$
$$G_2 = \{(x, y) \in I \mid t_2 \leq f(x, y) \leq t_3 - 1\}, \ldots$$
$$G_k = \{(x, y) \in I \mid t_k \leq f(x, y) \leq L - 1\} \quad (2)$$

where $f(x, y)$ is the gray level of the point (x, y), t_i $(i = 1, \ldots, k)$ is the ith threshold value, and k is the number of thresholds.

Selecting optimal threshold for bi-level thresholding is not computationally expensive. On the other hand, selecting more than few optimal threshold values requires high computational cost, since computational complexity of the deterministic algorithms grows exponentially with the number of thresholds. The optimal thresholding methods usually search for the thresholds by optimizing some criterion functions. These criterion functions are defined from images and they use the selected thresholds as parameters. In this study two popular thresholding methods are used namely entropy criterion (Kapur's) method and between-class variance (Otsu's) method.

2.1 Entropy Criterion Method

Entropy criterion method been proposed by Kapur in order to perform bi-level thresholding [21]. It considers the image foreground and background as two different signal sources, and when the sum of the two class entropies reaches its maximum, the image is said to be optimally thresholded. Hence, the aim is to find the optimal threshold yielding the maximum entropy.

The entropy of a discrete source is obtained from the probability distribution, where p_i is the probability of the system being in possible state i [27]. Kapur's objective function is determined from the histogram of the image, denoted by h_i, $i = 0, 1, \ldots, L-1$, where h_i represents the number of pixels having the gray level i. The normalized histogram at level i is:

$$P_i = \frac{h_i}{\sum_{j=0}^{L-1} h_j} \tag{3}$$

The aim is to maximize the objective function:

$$f(t) = H_0 + H_1 \tag{4}$$

where

$$\begin{aligned} H_0 &= -\sum_{i=0}^{t-1} \frac{P_i}{w_0} \ln \frac{P_i}{w_0}, & w_0 &= \sum_{i=0}^{t-1} P_i \\ H_1 &= -\sum_{i=t}^{L-1} \frac{P_i}{w_1} \ln \frac{P_i}{w_1}, & w_1 &= \sum_{i=t}^{L-1} P_i \end{aligned} \tag{5}$$

Kapur's method can be extended to multilevel thresholding. In this case, it can be configured as a k-dimensional optimization problem for selection of k optimal thresholds $[t_1, t_2, \ldots, t_k]$. The goal is to maximize the objective function:

$$f(t_1, t_2, \ldots, t_k) = H_0 + H_1 + H_2 + \cdots + H_k \tag{6}$$

where

$$\begin{aligned} H_0 &= -\sum_{i=0}^{t_1-1} \frac{P_i}{w_0} \ln \frac{P_i}{w_0}, & w_0 &= \sum_{i=0}^{t_1-1} P_i \\ H_1 &= -\sum_{i=t_1}^{t_2-1} \frac{P_i}{w_1} \ln \frac{P_i}{w_1}, & w_1 &= \sum_{i=t_1}^{t_2-1} P_i \\ H_2 &= -\sum_{i=t_2}^{t_3-1} \frac{P_i}{w_2} \ln \frac{P_i}{w_2}, & w_2 &= \sum_{i=t_2}^{t_3-1} P_i, \ldots \\ H_k &= -\sum_{i=t_k}^{L-1} \frac{P_i}{w_k} \ln \frac{P_i}{w_k}, & w_k &= \sum_{i=t_k}^{L-1} P_i \end{aligned} \tag{7}$$

2.2 Between-class Variance Method

Between-class variance was introduced by Otsu and it is one of the most referenced thresholding methods proposed for bi-level thresholding [26]. The algorithm assumes that the image to be thresholded contains two classes of pixels or bi-modal histogram (e.g. foreground and background), then calculates the optimum threshold separat-

ing those two classes so that their combined between-class variance is maximal. Between-class variance is defined as the sum of sigma functions of each class. The aim is to maximize the objective function:

$$f(t) = \sigma_0 + \sigma_1 \tag{8}$$

where

$$\sigma_0 = w_0(\mu_0 - \mu_t)^2, \qquad \mu_0 = \sum_{i=0}^{t-1} \frac{iP_i}{w_0}$$

$$\sigma_1 = w_1(\mu_1 - \mu_t)^2, \qquad \mu_1 = \sum_{i=t}^{L-1} \frac{iP_i}{w_1}$$

where $\mu_t = \sum_{i=0}^{L-1} iP_i$ is the total mean intensity of the original image.

Otsu function can be easily extended to multimodal cases. For instance, for k classes, the aim is to maximize the objective function:

$$f(t_1, t_2, \ldots, t_k) = \sigma_0 + \sigma_1 + \cdots + \sigma_k \tag{9}$$

where

$$\begin{aligned}
\sigma_0 &= w_0(\mu_0 - \mu_t)^2, & \mu_0 &= \sum_{i=0}^{t_1-1} \frac{iP_i}{w_0} \\
\sigma_1 &= w_1(\mu_1 - \mu_t)^2, & \mu_1 &= \sum_{i=t_1}^{t_2-1} \frac{iP_i}{w_1} \\
\sigma_2 &= w_2(\mu_2 - \mu_t)^2, & \mu_2 &= \sum_{i=t_2}^{t_3-1} \frac{iP_i}{w_2}, \ldots \\
\sigma_k &= w_k(\mu_k - \mu_t)^2, & \mu_k &= \sum_{i=t_k}^{L-1} \frac{iP_i}{w_k}
\end{aligned} \tag{10}$$

3 Proposed CS Approach to Multilevel Thresholding

The cuckoo search (CS), swarm intelligence algorithm introduced by Yang and Deb [39], is based on the brood parasitism of some cuckoo species that lay their eggs in the nests of other host birds. Furthermore, CS algorithm is enhanced by the so-called Lévy flight used instead of simple isotropic random walk. It was successfully applied to a number of very different problems like structural engineering optimization

[10, 44], test effort estimation for testing the software [33], planar graph coloring problem [52], permutation flow shop scheduling [24], modified version for bound-constrained continuous optimization [34], etc.

In the CS algorithm, a nest represents a solution, while a cuckoo egg represents a new solution. The goal is to use the new and potentially better solutions to replace worse solutions in the population. In order to implement the CS algorithm, the following three idealized rules are used:

1. Each cuckoo can lay only one egg at a time and chose random nest for laying their eggs.
2. Greedy selection process is applied, so only the eggs with highest quality are passed to the next generation.
3. Available host nests number is fixed. Host bird discovers cuckoo egg with probability p_a from [0, 1]. If cuckoo egg is discovered by the host, the host bird can either throw the egg away or abandon the nest, and build a completely new nest. This last assumption can be approximated by the fraction p_a of the n nests that are replaced with new random solutions.

In the basic form of the CS algorithm each nest contains one egg. This algorithm can be extended to more complicated cases in which each nest contains multiple eggs, i.e. set of solutions. The CS algorithm employs fewer control parameters compared to the other population-based metaheuristic algorithms in the literature. In fact, there is essentially only a single parameter in the CS (the probability of discovering cuckoo eggs), apart from the population size and maximum iteration number which are common control parameters for all nature inspired algorithms. Since tuning the control parameters of an algorithm might be very difficult, this is an important advantage [2].

The proposed CS algorithm tries to obtain k-dimensional vector $[t_1, t_2, \ldots, t_k]$ which maximizes the objective function which is described by Eq. (6) in the case of Kapur's criterion or by Eq. (9) in the case of Otsu's criterion. The objective function is also used as the fitness function for the proposed algorithm. The details of the developed CS approach to multilevel threshlding are introduced as follows.

Step 1. (Generate the initial population of solutions)
The CS algorithm generates a randomly distributed initial population of N solutions (nests) ($i = 1, 2, \ldots, N$) with k dimensions denoted by matrix T:

$$T = [t_1, t_2, \ldots, t_N], \qquad t_i = [t_{i1}, t_{i2}, \ldots, t_{ik}] \tag{11}$$

where t_{ij} is the jth component value that is restricted into $[0, \ldots, L-1]$ and the $t_{ij} < t_{i,j+1}$ for all j. The objective function values for all solutions t_i are evaluated and variable $cycle$ is set to 1.

Step 2. (Calculate the new population)
Before starting iterative search process, the CS algorithm detects the most successful solution as t_{best} solution. Also, at this point, step scale factor is being calculated by:

$$\varphi = \left(\frac{\Gamma(1+\beta) \cdot \sin\left(\pi \cdot \frac{\beta}{2}\right)}{\Gamma\left(1+\frac{\beta}{2}\right) \cdot \beta \cdot 2^{\frac{\beta-1}{2}}} \right)^{\frac{1}{\beta}} \tag{12}$$

where β denotes Lévy distribution parameter and Γ denotes gamma function. For the step size according to Lévy distribution, the recommended parameter value $\beta = 1.5$ is used. Then, each solution (nest) t_i from the search population T produces a new solution v_i and tests the objective function value of the new solution. A new solution v_i is calculated by:

$$v_{ij} = t_{ij} + 0.01 \cdot \left(\frac{r_1 \cdot \varphi}{r_2} \right)^{\frac{1}{\beta}} \cdot (t_{ij} - t_{best,j}) \cdot r_3 \tag{13}$$

where r_1, r_2 and r_3 are three normally distributed random numbers and ($j = 1, 2, \ldots, k$). At each computation step, the CS algorithm controls the boundary conditions of the created new solution. Hence, when the value of a variable overflows the allowed search space limits, then the value of the related variable is updated with the value of the closer limit value to the related variable. If the objective function value of the new one (v_i) is higher than that of the previous one (t_i), memorize the new solution and forget the old one. Otherwise, keep the old solution.

Step 3. (Record the best solution)
Memorize the best solution so far (t_{best}), i.e. the solution vector with the highest objective function value.

Step 4. (Fraction p_a of worse nests are abandoned and new nests are being built)
For each solution t_i apply the crossover operator in the search population by:

$$v_{ij} = \begin{cases} t_{ij} + rand_1 \cdot (t_{permute1,j} - t_{permute2,j}), & \text{if } rand_2 < p_a \\ t_{ij}, & \text{otherwise} \end{cases} \tag{14}$$

where $rand_1$ and $rand_2$ are uniform random numbers in range [0, 2] and [0, 1] respectively, $permute_1$ and $permute_2$ are different rows permutation functions applied to nests matrix. As mentioned above, $rand_1$ is a uniform random number between 0 and 2, which is different from the basic version of the CS, where this number is in the range [0, 1]. By increasing the range in which the variable $rand_1$ can take the value, the exploration of the CS is increased. In our proposed CS it was found that this small modification can improve the solution quality and speed convergence. Also at this point, the CS controls the boundary conditions at each computation step.

Step 5. (Record the best solution)
Memorize the best solution so far (t_{best}), and increase the variable $cycle$ by one.

Step 6. (Check the termination criterion)
If the $cycle$ is equal to the maximum number of iterations then finish the algorithm, else go to Step 2.

4 Proposed FA Approach to Multilevel Threshlding

The FA proposed by Yang [41] is a swarm intelligence algorithm inspired by the flashing behavior of fireflies. The fundamental function of such flashes is the communication between fireflies, more precisely attracting mating partners or potential preys. According to this, in the FA brighter firefly attracts its partners regardless of their sex, which makes the search space being explored more efficiently. Hence, the major components of the FA are the attractiveness of the firefly and the movement towards the attractive firefly. For simplicity, it can be assumed the attractiveness of a firefly is determined by its brightness. It is also assumed that for any couple of flashing fireflies, the less bright one will move towards the brighter one. Also, the brightness of a firefly can be formulated in such a way that it is associated with the objective function to be optimized, which makes it possible to formulate new optimization techniques. For a maximization problem, brightness is usually proportional to the value of the objective function. FA has been tested on standard benchmark problems but also was adjusted for multiobjective continuous optimization [49], enhanced with recently popular chaotic maps [11], or used for many specific hard optimization problems: economic dispatch [48], non-linear engineering design problems [45], stock market price forecasting [23], the vector quantization for digital image compression [17], etc.

In the proposed FA, fireflies form a population of threshold values. The threshold values produced by the firefly i is noted $t_i = [t_{i1}, t_{i2}, \ldots, t_{ik}], \quad i = 1, 2, \ldots, N$. All fireflies of the population are handled in the solution search space with the goal that knowledge is collectively shared among fireflies to guide the search to the location in the search space which can maximize the objective function. The objective function is used as the fitness function for the proposed algorithm, and it is described by Eq. (6) in the case of Kapur's criterion or by Eq. (9) in the case of Otsu's criterion. The main steps of the developed FA approach to multilevel threshlding are introduced as follows.

Step 1. (Generate the initial population of solutions)
The FA starts by randomly generating population of N solutions with k dimensions denoted by matrix T, like for the CS algorithm. Also, in the matrix T each threshold value $t_{ij}, \ j = 1, 2, \ldots, k$ produced by the firefly i is restricted into $[0, 1, \ldots, L-1]$ and the $t_{ij} < t_{i,j+1}$ for all j. After the generation of initial population, the objective function values for all solutions t_i are calculated and variable $cycle$ is set to 1.

Step 2. (Calculate the new population)
Before starting iterative search process, the FA algorithm ranks the solutions by their objective function values. Then, each solution of the new population is created from the appropriate solution t_i in the following way:

For each solution t_i, algorithm checks every solution $t_j, \ j = 1, 2, \ldots, i$, iteratively, starting from $j = 1$. If solution t_j has higher objective function value than t_i (t_j is brighter than t_i), the threshold values $t_{ik}, k = 1, 2, \ldots, k$, are changed by:

$$t_{ik} = t_{ik} + \beta_0 \cdot e^{-\gamma \cdot r_{ij}^2} \cdot (t_{ik} - t_{jk}) + \alpha \cdot S_k \cdot \left(rand_{ik} - \frac{1}{2} \right) \tag{15}$$

where the second term is due to the attraction and the third term is randomization term.

In the second term of Eq. (15), r_{ij} is the distance between firefly i and firefly j, while β_0 and γ are predetermined algorithm parameters: maximum attractiveness value and absorption coefficient, respectively. Any monotonically decreasing function of the distance r_{ij} to the chosen firefly j can be selected as firefly attractiveness, since the attractiveness can decrease when the distance between fireflies is increased. As we can see from the Eq. (15), in the FA the selected function which describes firefly attractiveness is the exponential function:

$$\beta = \beta_0 \cdot e^{-\gamma \cdot r_{ij}^2} \tag{16}$$

Distance r_{ij} between fireflies i and j is obtained by Cartesian distance by:

$$r_{ij} = \sqrt{\sum_{m=1}^{K} (t_{i,m} - t_{j,m})^2} \tag{17}$$

Control parameter β_0 describes attractiveness when two fireflies are found at the same point of search space and in general, $\beta_0 \in [0, 1]$ should be used. Control parameter γ determines the variation of attractiveness with increasing distance from communicated firefly. Its value is crucially important in determining the speed of the convergence and how the FA algorithm behaves. In most applications, it typically varies from 0.01 to 100. In our proposed FA, $\beta_0 = 1$ and $\gamma = 1$ are used for all simulations.

In the third term of Eq. (15), $\alpha \in [0, 1]$ is randomization parameter, S_j are the scaling parameters and $rand_{ij}$ is random number uniformly distributed between 0 and 1. In our proposed approach, as well as in [40], we found that it is beneficial to replace α by $\alpha \cdot S_j$, where the scaling parameters S_j in the k dimensions are determined by the actual scales of the problem of interest. Hence, they are calculated by:

$$S_j = u_j - l_j \tag{18}$$

where $j = 1, 2, \ldots, k$, u_j and l_j are the lower and upper bound of the parameter t_{ij}.

From the implementation point of view, it is important to mention that whenever the values of the solution t_i are changed, the FA controls the boundary conditions of the created solution and memorizes the new objective function value instead of the old one. Last solution provided by Eq. (15) is the final solution of the new population which will be transfered in the next cycle of the FA.

Step 3. (Reduce the randomization parameter)
In our proposed firefly algorithm, as in the version of FA proposed to solve structural optimization problems [8], it was found that the solution quality can be enhanced by reducing the randomization parameter α with a geometric progression reduction scheme which can be described by:

$$\alpha = \alpha_0 \cdot \gamma^t \tag{19}$$

where $0 < \gamma < 1$ is the reduction factor of randomization and t is current iteration number. In our proposed approach we used $\gamma = 0.9$ and the initial value of randomization parameter $\alpha_0 = 0.5$. In addition, in the proposed FA it was found that the value of parameter α less than 0.01 do not affect the optimization results. Hence, we reduce the value of parameter from 0.5 to 0.01 in the first few iterations of the FA by Eq. (19), while the fixed value $\alpha = 0.01$ is utilized in the remaining iterations.

Step 4. (Record the best solution)
Memorize the best solution so far t_{best}, and increase the variable *cycle* by one.

Step 5. (Check the termination criterion)
If the *cycle* is equal to the maximum number of iterations then finish the algorithm, else go to Step 2.

5 Experimental Study

In this study the CS and FA were compared against two other standard population-based metaheuristic techniques, PSO and DE. The tests were done on six standard images where the optimal multilevel threshold values were searched for. Two different fitness criteria were used, Kapur's entropy and between-class variance. For the both thresholding criteria, the exhaustive search method was conducted first to derive the optimal solutions for comparison with the results generated by the PSO, DE, CS and FA based methods.

We have implemented all of the algorithms in Java programming language on a PC with Intel(R) Core(TM) i7-3770K 4.2 GHz processor with 16 GB of RAM and Windows 8 × 64 Pro operating system. The PSO and DE algorithms have been used in their basic versions, while the CS and FA have been implemented with the small modifications which are described in previous two sections respectively. Experiments were carried out on the six standard test images namely Barbara, Living room, Boats, Goldhill, Lake and Aerial with 256 gray levels. All the images are size (512 × 512), except the Aerial image which is size (256 × 256). These original images are shown in Fig. 1. Each image has a unique gray level histogram which is shown in Fig. 2.

The number of thresholds k explored in the experiments were 2–5. Since metaheuristic algorithms have stochastic characteristics, each experiment was repeated 50 times for each image and for each k value. The run of each algorithm was terminated when the fitness value of the best solution $f(t*)$ reached the optimal value of the

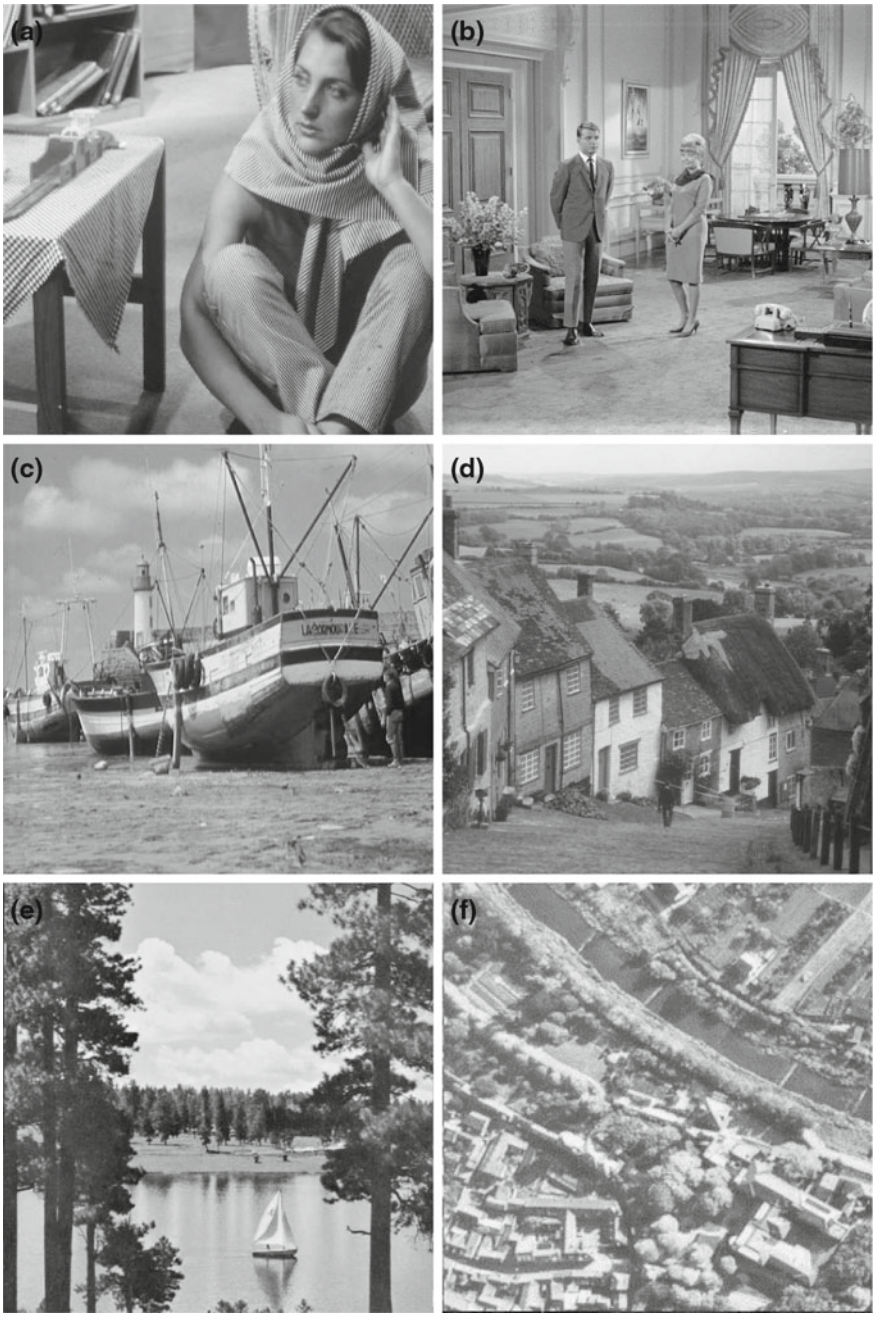

Fig. 1 Test images: **a** Barbara, **b** Living room, **c** Boats, **d** Goldhill, **e** Lake, **f** Aerial

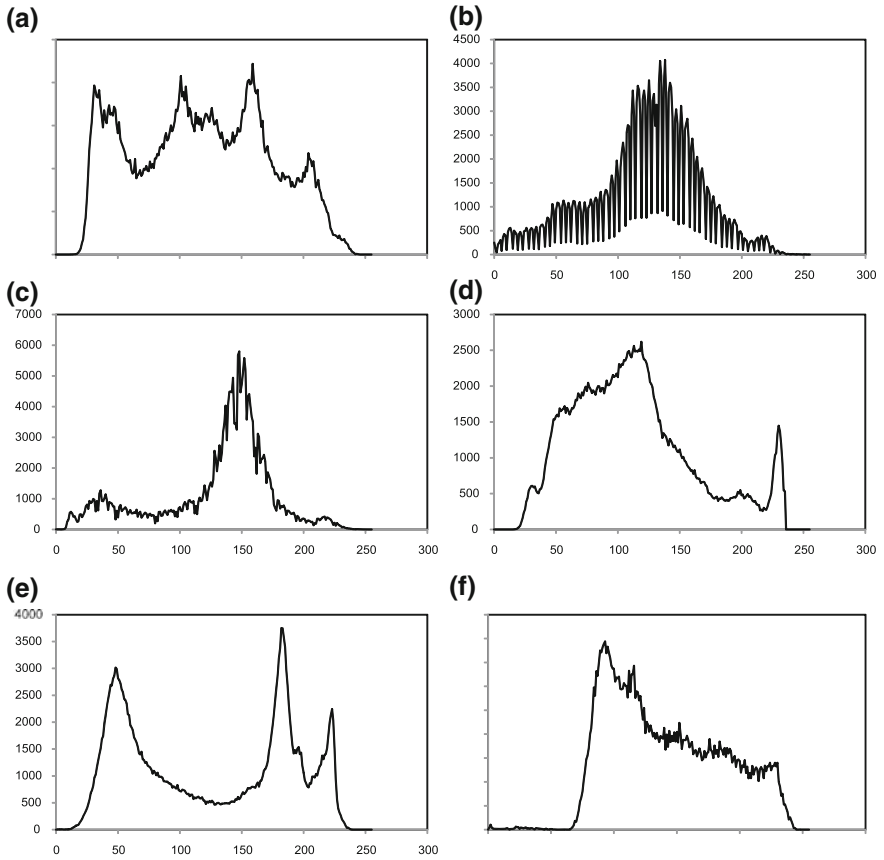

Fig. 2 Gray-level histograms of test images: **a** Barbara, **b** Living room, **c** Boats, **d** Goldhill, **e** Lake, **f** Aerial

known objective function f_{opt} from exhaustive search, i.e. $|f(t*) - f_{opt}| \leq \varepsilon = 10^{-9}$, where ε is a threshold value which fixes the accuracy of the measurement. Hence, the stopping condition for all algorithms is based on the value of the fitness and not of the maximum iteration number. The optimal thresholds, the corresponding optimal objective function values and the processing time provided by the exhaustive search for Kapur's and Otsu's criterion are presented in Tables 1 and 2 respectively.

5.1 Parameter Settings

For fair comparison, the same size of the population of 40 and the maximum iteration number of 2000 are used for all algorithms. Besides these common control

Table 1 Thresholds, objective function values and time processing provided by the exhaustive search for Kapur's method

	k	Kapur		
		Threshold values	Objective function	Time (ms)
Barbara	2	96, 168	12.668336540	28
	3	76, 127, 178	15.747087798	1812
	4	60, 99, 141, 185	18.556786861	122087
	5	58, 95, 133, 172, 210	21.245645311	5647294
Living room	2	94, 175	12.405985592	40
	3	47, 103, 175	15.552622213	1949
	4	47, 98, 149, 197	18.471055578	135259
	5	42, 85, 124, 162, 197	21.150302316	5875781
Boats	2	107, 176	12.574798244	32
	3	64, 119, 176	15.820902860	2063
	4	48, 88, 128, 181	18.655733570	126690
	5	48, 88, 128, 174, 202	21.401608305	5840989
Goldhill	2	90, 157	12.546393623	23
	3	78, 131, 177	15.607747002	2097
	4	65, 105, 147, 189	18.414213765	110650
	5	59, 95, 131, 165, 199	21.099138996	5064697
Lake	2	91, 163	12.520359742	31
	3	72, 119, 169	15.566286745	2094
	4	70, 111, 155, 194	18.365636309	119944
	5	64, 99, 133, 167, 199	21.024982760	5506378
Aerial	2	68, 159	12.538208248	30
	3	68, 130, 186	15.751881495	1880
	4	68, 117, 159, 200	18.615899102	118809
	5	68, 108, 141, 174, 207	21.210455499	5338382

parameters, each of tested algorithms has few other control parameters that greatly influence their performance. We have done preliminary testing on these algorithms in order to get good combinations of parameter values.

In the case of the PSO the values of additional control parameters are: inertia weight w is 0.5, minimum velocity v_{min} is -5, maximum velocity v_{max} is 5, cognitive learning factor c_1 and social learning factor c_2 are 2.

For the DE, the following parametric setup is used for all the test images: differential amplification factor $F = 0.9$ and crossover probability constant $Cr = 0.9$. These control parameter values for the PSO and DE algorithms are also commonly used in most of their applications.

In the proposed CS algorithm the probability of discovering a cuckoo egg p_a is set to 0.9. Although the value 0.25 of parameter p_a was suitable for the most of the CS applications, we have found that the performance of the CS for multilevel thresholding is sensitive to this parameter value and that higher values are more suitable.

Table 2 Thresholds, objective function values and time processing provided by the exhaustive search for Otsu's method

	k	Otsu		
		Threshold values	Objective function	Time (ms)
Barbara	2	82, 147	2608.610778507	12
	3	75, 127, 176	2785.163280467	786
	4	66, 106, 142, 182	2856.262131671	50854
	5	57, 88, 118, 148, 184	2890.976609405	2543860
Living room	2	87, 145	1627.909172752	13
	3	76, 123,163	1760.103018395	850
	4	56, 97, 132, 168	1828.864376614	59790
	5	49, 88, 120, 146, 178	1871.990616316	2731950
Boats	2	93, 155	1863.346730649	18
	3	73, 126, 167	1994.536306242	845
	4	65, 114, 147, 179	2059.866280428	54656
	5	51, 90, 126, 152, 183	2092.775965336	2654269
Goldhill	2	94, 161	2069.510202452	12
	3	83, 126, 179	2220.372641501	809
	4	69, 102, 138, 186	2295.380469158	51381
	5	63, 91, 117, 147, 191	2331.156597921	2573412
Lake	2	85, 154	3974.738214185	13
	3	78,140,194	4112.631097687	877
	4	67, 110, 158, 198	4180.886161109	60693
	5	57, 88, 127, 166, 200	4216.943583790	2740200
Aerial	2	125, 178	1808.171050536	11
	3	109, 147, 190	1905.410606582	801
	4	104, 134, 167, 202	1957.017965982	55015
	5	99, 123, 148, 175, 205	1980.656737348	2463969

The parameter values utilized by the FA are the following: the value of parameter γ is 1, the initial value of attractiveness β_0 is 1, the initial value of parameter α is 0.5, i.e. the reduction scheme described by Eq. (19) was followed by reducing the value of parameter α from 0.5 to 0.01.

5.2 Solution Quality Analysis

To analyze the solution quality of the tested four metaheuristic algorithms, the mean and standard deviations for 50 runs have been calculated and are presented in Table 3 for the experiments based on Kapur's entropy, and in Table 4 for the experiments based on the between class variance. These mean values can be compared to the optimal values of the corresponding objective functions found by an exhaustive search (from Tables 1 and 2).

Table 3 Comparison of the mean values and standard deviations obtained from the PSO, DE, CS and FA based on Kapur's entropy criterion for six test images over 50 runs

Algorithm	k	Barbara		Living room		Boats	
		Mean	SD	Mean	SD	Mean	SD
PSO	2	12.668336540	5.33E−15	12.405792709	1.64E−04	12.574798244	1.42E−14
	3	15.747087798	1.42E−14	15.552015642	2.82E−03	15.820679619	8.84E−04
	4	18.549612938	1.94E−02	18.467328310	6.64E−03	18.640100415	3.00E−02
	5	21.241857967	6.71E−03	21.131564234	2.18E−02	21.392020144	4.12E−02
DE	2	12.668336540	5.33E−15	12.405965639	7.89E−05	12.574798244	1.42E−14
	3	15.747087798	1.42E−14	15.552578874	3.03E−04	15.820899807	1.42E−05
	4	18.556749938	1.51E−04	18.470970822	3.16E−04	18.655660844	1.64E−04
	5	21.245566656	2.34E−04	21.149062508	1.79E−03	21.401458219	3.21E−04
CS	2	12.668336540	5.33E−15	12.405985592	5.33E−15	12.574798244	1.42E−14
	3	15.747087798	1.42E−14	15.552622213	1.07E−14	15.820902860	8.88E−15
	4	18.556786861	2.49E−14	18.471055578	2.49E−14	18.655733570	1.07E−14
	5	21.245645311	1.42E−14	21.149400604	1.64E−03	21.401608305	7.11E−15
FA	2	12.668336540	5.33E−15	12.405985592	5.33E−15	12.574798244	1.42E−14
	3	15.747087798	1.42E−14	15.552622213	1.07E−14	15.820902860	8.88E−15
	4	18.556786861	2.49E−14	18.471014902	2.85E−04	18.655723798	4.79E−05
	5	21.245645311	1.42E−14	21.149483979	1.46E−03	21.401583877	7.33E−05
Algorithm	k	Goldhill		Lake		Aerial	
		Mean	SD	Mean	SD	Mean	SD
PSO	2	12.546393623	7.11E−15	12.520359742	5.33E−15	12.538208248	1.78E−15
	3	15.607747002	1.42E−14	15.566286745	1.24E−14	15.751881495	5.33E−15
	4	18.414173744	2.07E−04	18.357505953	2.02E−02	18.615899102	1.78E−14
	5	21.099092699	1.28E−04	21.015922726	4.40E−02	21.192396874	5.44E−02
DE	2	12.546393623	7.11E−15	12.520359742	5.33E−15	12.538208248	1.78E−15
	3	15.607743578	2.40E−05	15.566286745	1.24E−14	15.751881495	5.33E−15
	4	18.414194980	1.01E−04	18.365579671	2.79E−04	18.615769177	6.36E−04
	5	21.098959164	3.76E−04	21.024780111	4.59E−04	21.210084581	1.12E−03
CS	2	12.546393623	7.11E−15	12.520359742	5.33E−15	12.538208248	1.78E−15
	3	15.607747002	1.42E−14	15.566286745	1.24E−14	15.751881495	5.33E−15
	4	18.414197322	6.53E−05	18.365636309	1.78E−14	18.615899102	1.78E−14
	5	21.099125539	6.59E−05	21.024962923	5.95E−05	21.210455499	1.78E−15
FA	2	12.546393623	7.11E−15	12.520359742	5.33E−15	12.538208248	1.78E−15
	3	15.607747002	1.42E−14	15.566286745	1.24E−14	15.751881495	5.33E−15
	4	18.414213765	2.13E−14	18.365636309	1.78E−14	18.615899102	1.78E−14
	5	21.099138996	0.00E−00	21.024982760	0.00E−00	21.210455499	1.78E−15

We can see from Table 3 that for Kapur's criterion, the CS and FA give better results for both, precision (mean fitness) and robustness (small standard deviations) compared to the PSO and DE for almost all cases considering different test images and different numbers of thresholds. Only for some cases when the number of thresholds is small, 2 or 3, their results are equal. When we compare the performance of the CS and FA for Kapur's criterion, we can see from Table 3 that their results are similar with respect to the accuracy and robustness. Their mean values and standard deviations

are the same for each image when the number of thresholds is less than 4, and for the images Barbara and Aerial with $k = 4$ and $k = 5$. For the rest of test cases, there are small differences in their performances. More precisely, in some cases the CS has slightly better results (for example for images Boats when $k = 4$ and $k = 5$, or Living room when $k = 4$, while in some cases the FA is slightly superior (for example for images Goldhill when $k = 4$ and $k = 5$, Living room and Lake when $k = 5$).

From the results given in Table 4, for between class variance criterion, all algorithms perform equally in terms of accuracy and robustness for each image when $k = 2$. For these test cases, all algorithms achieve the mean value equal to the optimum and consequently zero variance. When the number of thresholds is 3, the Otsu-based PSO, CS and FA reach optimum value in each run for each tested image. Further, for these test cases, the Otsu-based DE algorithm performs worse for images Barbara, Boats and Aerial. When the number of thresholds is higher than 3, from the results given in Table 4, we can see that the Otsu-based CS and FA have better mean values and lower standard deviations compared to the Otsu-based PSO and DE for each test image. The only exception is the test image Goldhill when $k = 5$, where the Otsu-based PSO performs slightly better than the Otsu-based CS. In particular, when comparing the Otsu-based FA with respect to the Otsu-based CS, we can see that the FA performs the equally for each image when $k = 4$ and better in the majority of the test images when $k = 5$. Only for the image Boats when $k = 5$, the CS has slightly better results than the FA, while for the image Lake when $k = 5$ both algorithms have the same mean values and standard deviations.

According to the mean and standard deviation values reported in Tables 3 and 4, we can conclude that the proposed CS and FA metaheuristics based on the Kapur's and Otsu's objective function exhibit superior performance compared to the PSO and DE methods. The mean values for the CS and FA are very close to optimal values provided by the exhaustive search in all cases. Also, for most of the cases their standard deviations are very low. That means that proposed algorithms are stable, which is an important characteristic in real-time applications. If we compare the performance of the FA with the performance of the CS, it can be concluded that the FA provides slightly better results considering both thresholding criteria. More precisely, for both thresholding criteria, for each image and each thresholding number, there are 48 test cases. Both algorithms have equal results in 35 out of these 48 cases. For the remaining 13 cases, the FA produces better results in 9 cases (4 cases for Kapur's entropy and 5 cases for between class variance).

5.3 Computational Time Analysis

Since the running time is critical in real-time applications, computational times for the algorithms have been analyzed. Tables 5 and 6 report the mean number of iterations and the average of the CPU time taken by each algorithm to satisfy the stopping condition for Kapur's and Otsu's criteria respectively.

Table 4 Comparison of the mean values and standard deviations obtained from the PSO, DE, CS and FA based on Otsu's criterion for six test images over 50 runs

Algorithm	k	Barbara		Living room		Boats	
		Mean	SD	Mean	SD	Mean	SD
PSO	2	2608.610778507	1.82E−12	1627.909172752	0.00E−00	1863.346730649	0.00E−00
	3	2785.163280467	2.27E−12	1760.103018395	2.27E−13	1994.536306242	1.59E−12
	4	2856.260804034	6.66E−03	1828.864376614	1.59E−12	2059.866220175	4.22E−04
	5	2890.975549258	5.05E−02	1871.984827146	2.29E−02	2092.771150715	8.36E−03
DE	2	2608.610778507	1.82E−12	1627.909172752	0.00E−00	1863.346730649	0.00E−00
	3	2785.162093432	8.31E−03	1760.103018395	2.27E−13	1994.535269293	7.26E−03
	4	2856.261305066	2.80E−03	1828.860328016	1.30E−02	2059.865271461	6.85E−03
	5	2890.971346990	2.05E−02	1871.976701063	2.34E−02	2092.766907541	2.71E−02
CS	2	2608.610778507	1.82E−12	1627.909172752	0.00E−00	1863.346730649	0.00E−00
	3	2785.163280467	2.27E−12	1760.103018395	2.27E−13	1994.536306242	1.59E−12
	4	2856.261511717	2.45E−03	1828.864376614	1.59E−12	2059.866280428	1.36E−12
	5	2890.976540127	4.85E−04	1871.990230213	2.70E−03	2092.775817560	1.03E−03
FA	2	2608.610778507	1.82E−12	1627.909172752	0.00E−00	1863.346730649	0.00E−00
	3	2785.163280467	2.27E−12	1760.103018395	2.27E−13	1994.536306242	1.59E−12
	4	2856.262131671	4.55E−13	1828.864376614	1.59E−12	2059.866280428	1.36E−12
	5	2890.976609405	3.64E−12	1871.990616316	0.00E−00	2092.773515829	3.57E−03

(continued)

Table 4 (continued)

Algorithm	k	Goldhill		Lake		Aerial	
		Mean	SD	Mean	SD	Mean	SD
PSO	2	2069.510202452	4.55E−13	3974.738214185	3.64E−12	1808.171050536	2.27E−13
	3	2220.372641501	1.36E−12	4112.631097687	4.55E−12	1905.410606582	1.14E−12
	4	2295.380095430	1.48E−03	4180.883976390	7.41E−03	1955.085619462	7.65E+00
	5	2331.156479206	3.56E−04	4216.942888298	3.99E−03	1979.170306260	2.51E+00
DE	2	2069.510202452	4.55E−13	3974.738214185	3.64E−12	1808.171050536	2.27E−13
	3	2220.372641501	1.36E−12	4112.631097687	4.55E−12	1905.410561743	3.14E−04
	4	2295.378073095	1.42E−02	4180.884670497	5.62E−03	1957.014977950	1.18E−02
	5	2331.145127974	2.55E−02	4216.936401364	1.47E−02	1980.627553925	1.23E−01
CS	2	2069.510202452	4.55E−13	3974.738214185	3.64E−12	1808.171050536	2.27E−13
	3	2220.372641501	1.36E−12	4112.631097687	4.55E−12	1905.410606582	1.14E−12
	4	2295.380469158	2.27E−12	4180.886161109	0.00E−00	1957.017965982	0.00E−00
	5	2331.155240485	4.76E−03	4216.943583790	9.09E−13	1980.651043072	1.16E−02
FA	2	2069.510202452	4.55E−13	3974.738214185	3.64E−12	1808.171050536	2.27E−13
	3	2220.372641501	1.36E−12	4112.631097687	4.55E−12	1905.410606582	1.14E−12
	4	2295.380469158	2.27E−12	4180.886161109	0.00E−00	1957.017965982	0.00E−00
	5	2331.156597921	2.27E−12	4216.943583790	9.09E−13	1980.656737348	9.09E−13

Table 5 Mean of the CPU times (in milliseconds) and mean of the iteration numbers obtained from the PSO, DE, CS and FA based on Kapur's entropy criterion for six test images over 50 runs

Algorithm	k	Barbara		Living room		Boats	
		Time (ms)	Iteration	Time (ms)	Iteration	Time (ms)	Iteration
PSO	2	2.18	9.22	102.82	1165.14	3.08	11.84
	3	3.30	14.28	21.96	218.56	16.23	136.58
	4	49.00	495.36	77.23	853.62	123.62	1367.86
	5	88.16	1050.8	153.53	1725.22	74.46	814.22
DE	2	3.55	14.52	14.24	134.26	3.37	16.9
	3	6.23	29.30	8.09	70.64	17.66	152.28
	4	24.65	240.68	33.50	322.98	46.99	478.76
	5	47.99	527.96	77.09	801.22	65.43	683.26
CS	2	71.04	194.54	53.30	129.84	53.84	135.58
	3	150.71	420.42	125.48	322.42	128.92	330.22
	4	189.31	518.58	222.48	570.6	170.28	436.02
	5	301.65	786.64	499.42	1303.8	247.15	631.36
FA	2	15.01	11.96	17.02	11.9	16.39	12.32
	3	34.07	29.82	37.24	29.7	36.24	29.24
	4	43.30	38.6	50.25	77.9	54.68	117.8
	5	50.15	44.04	104.74	515.06	76.22	241.94
Algorithm	k	Goldhill		Lake		Aerial	
		Time (ms)	Iteration	Time (ms)	Iteration	Time (ms)	Iteration
PSO	2	2.54	8.84	2.41	8.86	2.62	10.7
	3	3.18	13.54	3.65	14.58	3.24	13.9
	4	10.22	97.86	26.99	295.76	4.03	19.76
	5	23.56	258.5	77.64	893.98	58.54	695.92
DE	2	2.13	16.44	5.03	15.76	3.88	16.96
	3	7.92	69.28	6.26	29.82	6.48	30.94
	4	17.06	163.98	22.16	165.04	14.08	124.46
	5	43.38	486.12	55.10	603.12	56.82	644.22
CS	2	82.29	209.48	70.48	183.36	56.32	150.88
	3	149.68	441.64	138.23	375.66	104.75	292.22
	4	278.51	828.32	220.44	604.58	174.84	482.26
	5	285.32	835.42	336.29	915.92	193.21	532.76
FA	2	13.66	10.08	15.92	12.32	13.86	11.0
	3	32.08	28.66	34.56	29.42	32.83	29.18
	4	42.24	38.6	43.84	37.66	43.18	38.96
	5	47.12	43.54	50.31	43.6	50.46	45.46

The computational times for the exhaustive search method for Kapur's and Otsu's criteria grow exponentially with the number of required thresholds (Tables 1 and 2). Although the computational time needed to derive the optimal thresholds by the exhaustive search for Otsu's criterion is approximately two times shorter than the same for Kapur's criterion, for both criteria it is not acceptable for $k \geq 4$. On the other side, the results from Tables 5 and 6 show that the run times for the CS and FA

Table 6 Mean of the CPU times (in milliseconds) and mean of the iteration numbers obtained from the PSO, DE, CS and FA based on Otsu's criterion for six test images over 50 runs

Algorithm	k	Barbara		Living room		Boats	
		Time (ms)	Iteration	Time (ms)	Iteration	Time (ms)	Iteration
PSO	2	0.84	9.4	0.90	9.6	0.94	8.8
	3	1.22	13.26	1.24	14.68	1.28	14.1
	4	4.84	138.52	1.60	19.18	3.12	56.56
	5	10.02	261.06	8.14	221.64	32.96	1012.3
DE	2	0.77	14.16	1.03	15.84	0.99	16.54
	3	3.08	68.4	2.26	30.4	2.79	69.46
	4	8.26	200.84	11.10	281.02	4.73	121.68
	5	27.36	798.72	24.28	682.8	32.93	953.74
CS	2	35.80	179.0	30.03	223.34	32.74	204.04
	3	51.66	370.88	56.67	371.50	53.74	378.40
	4	100.78	723.22	83.43	578.94	76.98	595.66
	5	114.63	802.80	122.78	836.32	102.84	677.04
FA	2	8.16	12.02	8.72	12.54	7.17	10.64
	3	15.73	28.62	12.67	28.54	16.10	28.7
	4	17.84	37.7	19.16	38.78	18.43	38.2
	5	21.27	43.9	20.15	43.78	43.51	669.52
Algorithm	k	Goldhill		Lake		Aerial	
		Time (ms)	Iteration	Time (ms)	Iteration	Time (ms)	Iteration
PSO	2	0.94	8.98	0.91	10.12	0.81	9.04
	3	2.86	14.3	1.17	13.48	1.29	15.16
	4	5.16	136.32	6.55	175.58	5.18	138.44
	5	7.52	222.92	7.01	179.38	20.21	622.38
DE	2	0.82	15.24	0.75	14.72	0.82	15.8
	3	1.52	28.88	1.90	29.18	3.18	70.02
	4	7.80	200.44	7.43	201.24	6.99	162.28
	5	23.80	684.12	21.55	603.88	18.63	527.08
CS	2	27.88	195.30	38.04	267.66	32.45	254.10
	3	59.98	424.62	49.68	381.36	52.78	395.44
	4	70.92	531.74	85.30	646.44	83.84	593.52
	5	159.69	1115.44	109.90	746.36	209.36	1487.06
FA	2	8.61	12.68	7.28	11.66	7.94	12.18
	3	15.61	27.78	13.38	30.04	15.56	27.7
	4	17.12	37.66	18.49	38.86	18.05	38.58
	5	20.93	44.54	21.27	43.12	20.58	44.68

increase with the number of thresholds, but these run times are much shorter than those for the exhaustive search, and they tend to grow at a linear rate as the problem dimension increases.

By comparing the computational times for the CS with respect to the other three algorithms, we can see that for almost all tested cases, the PSO, DE and FA are more efficient than the CS for both thresholding criteria. In addition, the computation times

of the PSO, DE and FA are not significantly different. It is also evident that, for all images, the computational times for the CS and FA based on the Otsu's function are better than the CS and FA based on the Kapur's function respectively.

From Tables 5 and 6 we can also observe that the proposed FA converges in considerably less iterations compared to the PSO, DE and CS methods. The exception are some cases where the number of thresholds are small ($k \leq 3$) where the FA, PSO and DE algorithms converge in the similar number of iterations.

Generally, it can be concluded that although the computational time requirements for the CS algorithm are greater compared to the FA, both algorithms are scalable, which means that they are suitably efficient and practical in terms of time complexity for high-dimensional problems.

6 Conclusion

The Cuckoo Search (CS) and the Firefly Algorithm (FA) are relatively new swarm intelligence metauristics and they have been proven to be effective methods for solving some hard optimization problems. In this article, their application to the multilevel thresholding problem is investigated. Therefore, the CS and FA are employed to maximize the Kapur's entropy and between-class variance separately to determine multilevel thresholds. Furthermore, we have implemented two other well known metaheuristic algorithms for solving the multilevel thresholding problem, particle swarm optimization (PSO) and differential evolution (DE), for the purpose of comparison with the CS and FA. Tests were done on six standard test images with known optima derived by the exhaustive search method.

The experimental results show that when the number of desired thresholds is two or three, the quality of the segmentation results for all these algorithms is similar. However, for both thresholding criteria, as the number of desired thresholds increases, the superiority of the CS and FA over the PSO and DE becomes more pronounced, both in terms of precision and robustness, with a small advantage of the FA.

The computational time analysis shows that the FA converges in the lowest number of iterations compared to the other three algorithms. Although in the majority of cases the CS algorithm requires somewhat higher computational cost than the FA, PSO and DE, the CPU times for all these algorithms are reasonable and grow at linear rate as the problem dimension increases, as opposed to the exhaustive search where the grow is exponential.

Additional advantage of the FA and CS compared to many other search heuristics is that they are easy to implement, and that both algorithms, especially the CS, have very few control parameters that have to be set.

From this research it can be concluded that the CS and FA can both be efficiently used for solving multilevel thresholding problem due to their simplicity, reliability and robustness. Their segmentation results are promising and encourage further research for applying them to some other complex and real-time image analysis problems.

Acknowledgments This reserach was supported by Ministry of Education and Science of Republic of Serbia, Grant III-44006.

References

1. Adollah, R., Mashor, M.Y., Rosline, H., Harun, N.H.: Multilevel thresholding as a simple segmentation technique in acute leukemia images. J. Med. Imaging Health Inf. **2**(3), 285–288 (2012)
2. Akay, B.: A study on particle swarm optimization and artificial bee colony algorithms for multilevel thresholding. Appl. Soft Comput. **13**(6), 3066–3091 (2013)
3. Bacanin, N., Tuba, M.: Artificial bee colony (ABC) algorithm for constrained optimization improved with genetic operators. Stud. Inf. Control **21**(2), 137–146 (2012)
4. Brajevic, I., Tuba, M.: An upgraded artificial bee colony algorithm (ABC) for constrained optimization problems. J. Intell. Manuf. **24**(4), 729– 740 (2013)
5. Dai, C., Chen, W., Song, Y., Zhu, Y.: Seeker optimization algorithm: a novel stochastic search algorithm for global numerical optimization. J. Syst. Eng. Electron. **21**(2), 300–311 (2010)
6. Dominguez, A.R., Nandi, A.K.: Detection of masses in mammograms via statistically based enhancement, multilevel-thresholding segmentation, and region selection. Comput. Med. Imaging Graph. **32**(4), 304–315 (2008)
7. Dorigo, M., Gambardella, L.M.: Ant colonies for the travelling salesman problem. Biosystems **43**(2), 73–81 (1997)
8. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Mixed variable structural optimization using firefly algorithm. Comput. Struct. **89**(23–24), 2325–2336 (2011)
9. Gandomi, A.H., Yang, X.S.: Evolutionary boundary constraint handling scheme. Neural Comput. Appl. **21**(6, SI), 1449–1462 (2012)
10. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng. Comput. **29**(1), 17–35 (2013)
11. Gandomi, A.H., Yang, X.S., Talatahari, S., Alavi, A.H.: Firefly algorithm with chaos. Commun. Nonlinear Sci. Numer. Simul. **18**(1), 89–98 (2013)
12. Hammouche, K., Diaf, M., Siarry, P.: A comparative study of various meta-heuristic techniques applied to the multilevel thresholding problem. Eng. Appl. Artif. Intell. **23**(5), 676–688 (2010)
13. Harrabi, R., Ben Braiek, E.: Color image segmentation using multi-level thresholding approach and data fusion techniques: application in the breast cancer cells images. EURASIP J. Image Video Process. (2012)
14. Heikkonen, J., Mantynen, N.: A computer vision approach to digit recognition on pulp bales. Pattern Recogn. Lett. **17**(4), 413–419 (1996) (International Conference on Engineering Applications of Neural Networks (EANN 95), Otaniemi, Finland, 21–23 August 1995)
15. Horng, M.H.: Multilevel minimum cross entropy threshold selection based on the honey bee mating optimization. Expert Syst. Appl. **37**(6), 4580–4592 (2010)
16. Horng, M.H.: Multilevel thresholding selection based on the artificial bee colony algorithm for image segmentation. Expert Syst. Appl. **38**(11), 13,785–13,791 (2011)
17. Horng, M.H.: Vector quantization using the firefly algorithm for image compression. Expert Syst. Appl. **39**(1), 1078–1091 (2012)
18. Jaynes, E.T.: Information theory and statistical mechanics. Phys. Rev. Ser. II **106**(4), 620–630 (1957)
19. Jovanovic, R., Tuba, M.: An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem. Appl. Soft Comput. **11**(8), 5360–5366 (2011)
20. Jovanovic, R., Tuba, M.: Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem. Comput. Sci. Inf. Syst. (ComSIS) **10**(1), 133–149 (2013)

21. Kapur, E.J.N., Sahoo, P.K., Wong, A.K.C.: A new method for gray-level picture thresholding using the entropy of the histogram. Comput. Vis. Graphics Image Process. **29**(3), 273–285 (1985)
22. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical report-tr06, Engineering Faculty, Computer Engineering Department, Erciyes University (2005)
23. Kazem, A., Sharifi, E., Hussain, F.K., Saberi, M., Hussain, O.K.: Support vector regression with chaos-based firefly algorithm for stock market price forecasting. Appl. Soft Comput. **13**(2), 947–958 (2013)
24. Marichelvam, M.K.: An improved hybrid Cuckoo Search (IHCS) metaheuristics algorithm for permutation flow shop scheduling problems. Int. J. Bio-Inspired Comput. **4**(4, SI), 200–205 (2012)
25. Ng, H.F.: Automatic thresholding for defect detection. Pattern Recogn. Lett. **27**(14), 1644–1649 (2006)
26. Otsu, N.: A threshold selection method for grey level histograms. EEE Trans. Syst. Man Cybern. **9**(1), 62–66 (1979)
27. Portes de Albuquerque, M., Esquef, I.A., Gesualdi Mello, A.R.: Image thresholding using tsallis entropy. Pattern Recogn. Lett. **25**(9), 1059–1065 (2004)
28. Sahoo, P., Wilkins, C., Yeager, J.: Threshold selection using Renyi's entropy. Pattern Recogn. **30**(1), 71–84 (1997)
29. Sarkar, S., Patra, G.R., Das, S.: A differential evolution based approach for multilevel image segmentation using minimum cross entropy thresholding. In: Proceedings of the 2nd International Conference on Swarm, Evolutionary, and Memetic Computing, Part I, pp. 51–58 (2011)
30. Sathya, P.D., Kayalvizhi, R.: Optimal multilevel thresholding using bacterial foraging algorithm. Expert Syst. Appl. **38**(12), 15,549–15,564 (2011)
31. Sathya, P.D., Kayalvizhi, R.: Modified bacterial foraging algorithm based multilevel thresholding for image segmentation. Eng. Appl. Artif. Intell. **24**(4), 595–615 (2011)
32. Sezgin, M., Sankur, B.: Survey over image thresholding techniques and quantitative performance evaluation. J. Electron. Imaging **13**(1), 146–165 (2004)
33. Srivastava, P.R., Varshney, A., Nama, P., Yang, X.S.: Software test effort estimation: a model based on cuckoo search. Int. J. Bio-Inspired Comput. **4**(5), 278–285 (2012)
34. Tuba, M., Subotic, M., Stanarevic, N.: Performance of a modified cuckoo search algorithm for unconstrained optimization problems. WSEAS Trans. Syst. **11**(2), 62–74 (2012)
35. Tuba, M., Jovanovic, R.: Improved ant colony optimization algorithm with pheromone correction strategy for the traveling salesman problem. Int. J. Comput. Commun. Control **8**(3), 477–485 (2013)
36. Tuba, M., Brajevic, I., Jovanovic, R.: Hybrid seeker optimization algorithm for global optimization. Appl. Math. Inf. Sci. **7**(3), 867–875 (2013)
37. Tuba, M.: Asymptotic behavior of the maximum entropy routing in computer networks. Entropy **15**(1), 361–371 (2013)
38. Yan, H.: Unified formulation of a class of optimal image thresholding techniques. Pattern Recogn. **29**(12), 2025–2032 (1996)
39. Yang, X.S., Deb, S.: Cuckoo search via Lévy flights. In: Proceedings of the World Congress on Nature & Biologically Inspired, Computing, pp. 210–214 (2009)
40. Yang, X.S.: Firefly algorithm, Lévy flights and global optimization. In: Bramer, M., Ellis, R., Petridis, M. (eds) Research and Development in Intelligent Systems, vol. XXVI, pp. 209–218. Springer, London (2010)
41. Yang, X.S.: Firefly algorithms for multimodal optimization. In: Watanabe, O., Zeugmann, T. (eds) Stochastic Algorithms: Foundations and Applications, SAGA 2009, Lecture Notes in Computer Sciences, pp. 169–178. Springer, Berlin (2009)
42. Yang, X.S.: Free lunch or no free lunch: that is not just a question? Int. J. Artif. Intell. Tools 21(3, SI) (2012)
43. Yang, X.S.: Nature-Inspired Metaheuristic Algorithms, 2nd edn. Luniver Press, Frome (2010)
44. Yang, X.S., Deb, S.: Engineering optimisation by cuckoo search. Int. J. Math. Model. Numer. Optimisation **1**(4), 330–343 (2010)

45. Yang, X.S.: Firefly algorithm, stochastic test functions and design optimisation. Int. J. Bio-Inspired Comput. **2**(2), 78–84 (2010)
46. Yang, X.S.: Review of meta-heuristics and generalised evolutionary walk algorithm. Int. J. Bio-Inspired Comput. **3**(2), 77–84 (2011)
47. Yang, X.S.: Efficiency analysis of swarm intelligence and randomization techniques. J. Comput. Theor. Nanosci. **9**(2), 189–198 (2012)
48. Yang, X.S., Hosseini, S.S.S., Gandomi, A.H.: Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect. Appl. Soft Comput. **12**(3), 1180–1186 (2012)
49. Yang, X.S.: Multiobjective firefly algorithm for continuous optimization. Eng. Comput. **29**(2), 175–184 (2013)
50. Yen, J.C., Chang, F.J., Chang, S.: A new criterion for automatic multilevel thresholding. IEEE Trans. Image Process. **4**(3), 370–378 (1995)
51. Yin, P.Y.: Multilevel minimum cross entropy threshold selection based on particle swarm optimization. Appl. Math. Comput. **184**(2), 503–513 (2007)
52. Zhou, Y., Zheng, H., Luo, Q., Wu, J.: An improved Cuckoo search algorithm for solving planar graph coloring problem. Appl. Math. Inf. Sci. **7**(2), 785–792 (2013)

A Binary Cuckoo Search and Its Application for Feature Selection

L. A. M. Pereira, D. Rodrigues, T. N. S. Almeida, C. C. O. Ramos, A. N. Souza, X.-S. Yang and J. P. Papa

Abstract In classification problems, it is common to find datasets with a large amount of features, some of theses features may be considered as noisy. In this context, one of the most used strategies to deal with this problem is to perform a feature selection process in order to build a subset of features that can better represents the dataset. As feature selection can be modeled as an optimization problem, several studies have to attempted to use nature-inspired optimization techniques due to their large generalization capabilities. In this chapter, we use the Cuckoo Search (CS) algorithm in the context of feature selection tasks. For this purpose, we present a binary version of the Cuckoo Search, namely BCS, as well as we evaluate it with different transfer functions that map continuous solutions to binary ones. Additionally, the Optimum-Path Forest classifier accuracy is used as the fitness function. We conducted simulations comparing BCS with binary versions of the Bat Algorithm,

L. A. M. Pereira · D. Rodrigues · T. N. S. Almeida · J. P. Papa (✉)
Department of Computing, UNESP—Univ Estadual Paulista, Bauru, SP, Brazil
e-mail: papa@fc.unesp.br

L. A. M. Pereira
e-mail: luis.pereira@fc.unesp.br

D. Rodrigues
e-mail: douglasrodrigues.dr@gmail.com

T. N. S. Almeida
e-mail: almeida.tns@gmail.com

C. C. O. Ramos
Department of Electrical Engineering, University of São Paulo, São Paulo, SP, Brazil
e-mail: caioramos@gmail.com

A. N. Souza
Department of Electrical Engineering, UNESP—Univ Estadual Paulista, Bauru, SP, Brazil
e-mail: andrejau@feb.unesp.br

X.-S. Yang
School of Science and Technology, Middlesex University, Hendon, London, UK
e-mail: xy227@cam.ac.uk

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_7,

Firefly Algorithm and Particle Swarm Optimization. BCS has obtained reasonable results when we consider the compared techniques for feature selection purposes.

Keywords Feature selection · Pattern classification · Meta-heuristic algorithms · Optimum-path forest · Cuckoo search algorithm

1 Introduction

In pattern recognition tasks, many problems are characterized for instances composed of hundreds, thousands or millions of features, as face recognition for instance. As such, working with high dimensional space may demand much computational power and requires long processing time. Therefore, it is often desirable to find a subset that can represent the whole set without losing of information. However, finding this subset is known to be NP-hard and the computational load may become intractable [8].

Support by this previous scenario, several works attempt to model the feature selection as a combinatorial optimization problem, in which the set of features that leads to the best feature space separability is then employed to map the original dataset to a new one. The objective function can be the accuracy of a given classifier or some other criterion that may consider the best trade-off between feature extraction computational burden and effectiveness.

In this fashion, natural inspired optimization techniques have been used to model the feature selection as an optimization problem. The idea is to lead with the search space as a n-cube, where n stands for the number of features. In such case, the optimal (near-optimal) solution is chosen among the 2^n possibilities, and it corresponds to one hypercube's corner. For this purpose, we can employ binary versions of optimization heuristic techniques.

Kennedy and Eberhart [11] proposed a binary version of the well-known Particle Swarm Optimization (PSO) [10] called BPSO, in which the standard PSO algorithm was modified in order to handle binary optimization problems. Further, Firpi and Goodman [5] extended BPSO to the context of feature selection. Rashedi et al. [19] proposed a binary version of the Gravitational Search Algorithm (GSA) [18] called BGSA, which was applied for feature selection by Papa et al. [14]. Ramos et al. [17] presented their version of the Harmony Search (HS) [7] for purpose in the context of theft detection in power distribution system. In addition, Nakamura et al. [13] proposed a binary version of Bat algorithm (BBA) and Banati and Monika [1] introduced Firefly algorithm for feature selection.

Recently, Yang and Deb [24] proposed a new meta-heuristic method for continuous optimization namely Cuckoo Search (CS), which is based on the fascinating reproduction strategy of cuckoo birds. Several species engage the brood parasitism laying their eggs in the nests of others host birds. Such approach has demonstrated to outperform some well-known nature-inspired optimization techniques, such as PSO and Genetic Algorithms. Furthermore, CS has been applied with success in several

distinct engineer applications [6, 9, 25], and several studies have proposed variants of standard CS to handle particular problems, such as discrete optimizations [12, 22].

In artificial intelligence, CS has also aroused interests specially for machine learning applications. Vazquez [23], for instance, employed CS to train spiking neural models. In addition, Valian [3] proposed a improved Cuckoo Search for feedforward neural network training. For this purpose, the authors investigated the CS behavior by setting its parameters dynamically. Bansal et al. [21] use CS to optimize the local minimal convergence of k-means method for clustering tasks.

In this chapter, we present a binary version of the Cuckoo Search (BCS) for feature selection purposes. The main idea is to associate a set of binary coordinates for each solution that denote whether a feature will belong to the final set of features or not. The function to be maximized is the one given by a supervised classifier's accuracy. As the quality of the solution is related with the number of nests, we need to evaluate each one of them by training a classifier with the selected features encoded by the eggs' quality and also to classify an evaluating set. Thus, we need a fast and robust classifier, since we have one instance of it for each nest. As such, we opted to use the Optimum-Path Forest (OPF) classifier [15, 16], which has been demonstrated to be so effective as Support Vector Machines, but faster for training. The experiments have been performed in five public datasets against Bat Algorithm, Firefly Algorithm and Particle Swarm Optimization in order to evaluate the robustness of CS.

The remainder of the chapter is organized as follows. In Sect. 2 we revisit the Optimum-Path Forest theory. Section 3 presents the Cuckoo Search algorithm and its binary version. Section 4 discuss a framework to evaluate feature selection algorithms based on nature-inspired. Some simulations and its results are shown in Sect. 5. Finally, conclusions are stated in Sect. 6.

2 Supervised Classification Through Optimum-Path Forest

The OPF classifier works by modeling the problem of pattern recognition as a graph partition in a given feature space. The nodes are represented by the feature vectors and the edges connect all pairs of them, defining a full connectedness graph. This kind of representation is straightforward, given that the graph does not need to be explicitly represented, allowing us to save memory. The partition of the graph is carried out by a competition process between some key samples (prototypes), which offer optimum paths to the remaining nodes of the graph. Each prototype sample defines its optimum-path tree (OPT), and the collection of all OPTs defines an optimum-path forest, which gives the name to the classifier [15, 16].

The OPF can be seen as a generalization of the well known Dijkstra's algorithm to compute optimum paths from a source node to the remaining ones [2]. The main difference relies on the fact that OPF uses a set of source nodes (prototypes) with any smooth path-cost function [4]. In case of Dijkstra's algorithm, a function that summed the arc-weights along a path was applied. In regard to the supervised OPF

version addressed here, we have used a function that gives the maximum arc-weight along a path, as explained below.

Let $Z = Z_1 \cup Z_2 \cup Z_3$ be a dataset labeled, in which Z_1, Z_2 and Z_3 are, respectively, a training, evaluating and test sets. Let $S \subseteq Z_1$ a set of prototype samples. Essentially, the OPF classifier creates a discrete optimal partition of the feature space such that any sample $s \in Z_2 \cup Z_3$ can be classified according to this partition. This partition is an optimum path forest (OPF) computed in $\Re^n$ by the Image Foresting Transform (IFT) algorithm [4].

The OPF algorithm may be used with any *smooth* path-cost function which can group samples with similar properties [4]. Particularly, we used the path-cost function $f_{\max}$, which is computed as follows:

$$f_{\max}(\langle s \rangle) = \begin{cases} 0 & \text{if } s \in S, \\ +\infty & \text{otherwise} \end{cases}$$
$$f_{\max}(\pi \cdot \langle s, t \rangle) = \max\{f_{\max}(\pi), d(s, t)\}, \quad (1)$$

in which $d(s, t)$ means the distance between samples s and t, and a path π is defined as a sequence of adjacent samples. In such a way, we have that $f_{\max}(\pi)$ computes the maximum distance between adjacent samples in π, when π is not a trivial path.

The OPF algorithm works with a training and a testing phase. In the former step, the competition process begins with the prototypes computation. We are interested into finding the elements that fall on the boundary of the classes with different labels. For that purpose, we can compute a Minimum Spanning Tree (MST) over the original graph and then mark as prototypes the connected elements with different labels. Figure 1b displays the MST with the prototypes at the boundary. After that, we can begin the competition process between prototypes in order to build the optimum-path forest, as displayed in Fig. 1c. The classification phase is conducted by taking a sample from the test set (black triangle in Fig. 1d) and connecting it to all training samples. The distance to all training nodes are computed and used to weight the edges. Finally, each training node offers to the test sample a cost given by a path-cost function [maximum arc-weight along a path—Eq. (1)], and the training node that has offered the minimum path-cost will conquer the test sample. This procedure is shown in Fig. 1e.

3 Cuckoo Search

3.1 Standard Cuckoo Search

The parasite behavior of some cuckoo species is extremely intriguing. These birds can lay down their eggs in a host nests, and mimic external characteristics of host eggs such as color and spots. In case of this strategy is unsuccessful, the host can throw the cuckoo's egg away, or simply abandon its nest, making a new one in another

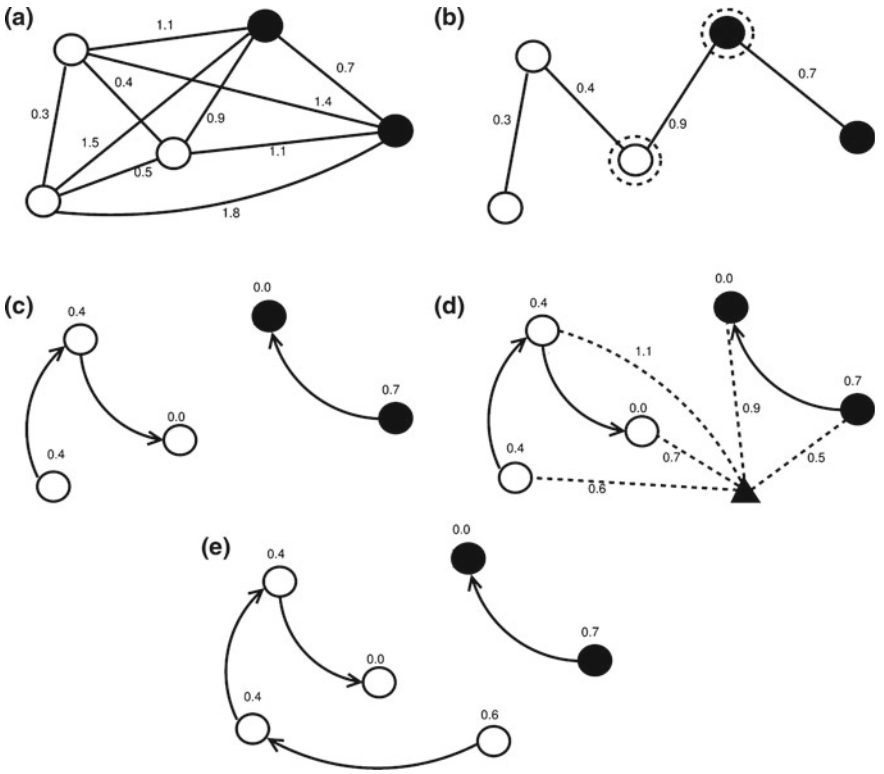

Fig. 1 OPF pipeline: **a** complete graph, **b** MST and prototypes bounded, **c** optimum-path forest generated at the final of training step, **d** classification process and **e** the triangle sample is associated to the white circle class. The values above the nodes are their costs after training, and the values above the edges stand for the distance between their corresponding nodes

place. Based on this context, Yang and Deb [24] have developed a novel evolutionary optimization algorithm named as Cuckoo Search (CS), and they have summarized CS using three rules, as follows:

1. Each cuckoo choose a nest randomly to lays eggs.
2. The number of available host nests is fixed, and nests with high quality of eggs will carry over to the next generations.
3. In case of a host bird discovered the cuckoo egg, it can throw the egg away or abandon the nest, and build a completely new nest. There is a fixed number of host nests, and the probability that an egg laid by a cuckoo is discovered by the host bird is $p_a \in [0, 1]$.

CS performs a balanced combination of a local random walk and the global explorative random walk, controlled by a switching parameter $p_a \in [0, 1]$. The local random walk can be written as

$$x_i^j(t) = x_i^j(t-1) + \alpha \cdot s \oplus H(p_a - \varepsilon) \oplus (x_{k'}^j(t-1) - x_{k''}^j(t-1)), \quad (2)$$

where $x_{k'}^j$ and $x_{k''}^j$ are two different solutions selected by random permutation, and and x_i^j stands for the jth egg at nest i, $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, d$. $H(\cdot)$ is a Heaviside function, ε is a random number drawn from a uniform distribution, and s is the step size.

The global random walk is carried out using Lévy flights as follows:

$$x_i^j(t) = x_i^j(t-1) + \alpha \cdot L(s, \lambda), \quad (3)$$

where

$$L(s, \lambda) = \frac{\lambda \cdot \Gamma(\lambda) \cdot \sin(\lambda)}{\pi} \cdot \frac{1}{s^{1+\lambda}}, \quad s \gg s_0 > 0 \quad (4)$$

The Lévy flights employ a random step length which is drawn from a Lévy distribution. Therefore, the CS algorithm is more efficient in exploring the search space as its step length is much longer in the long run. The parameter $\alpha > 0$ is the step size scaling factor, which should be related to the scales of the problem of interest. Yang and Deb [26] claim that $\alpha = O(S/10)$ can be used in most cases, where S denotes the scale of the problem of interest, while $\alpha = O(S/100)$ can be more effective and avoid flying too far.

3.2 Binary Cuckoo Search for Feature Selection

In standard CS, the solutions are updated in the search space towards continuous-valued positions. Unlike, in the BCS for feature selection [20], the search space is modelled as a n-dimensional boolean lattice, in which the solutions are updated across the corners of a hypercube. In addition, as the problem is to select or not a given feature, a solution binary vector is employed, where 1 corresponds whether a feature will be selected to compose the new dataset and 0 otherwise. In order to build this binary vector, we have employ the Eq. 6, which can provide only binary values in the boolean lattice restricting the new solutions to only binary values:

$$S(x_i^j(t)) = \frac{1}{1 + e^{-x_i^j(t)}} \quad (5)$$

$$x_i^j(t+1) = \begin{cases} 1 & \text{if } S(x_i^j(t)) > \sigma, \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Algorithm 1: BCS-Feature Selection Algorithm

input : Labeled training set Z_1 and evaluating set Z_2, loss parameter p, α value, number of nests n, dimension d, number of iterations T, c_1 and c_2 values.
output : Global best position $\widehat{g}$.
auxiliaries: Fitness vector f with size m and variables acc, $maxfit$, $globalfit$ and $maxindex$.

for *each nest* n_i $(\forall i = 1, \ldots, m)$ **do**
  for *each dimension* j $(\forall j = 1, \ldots, d)$ **do**
    $x_i^j(0) \leftarrow$ Random$\{0, 1\}$;
  end
  $f_i \leftarrow -\infty$;
end
$globalfit \leftarrow -\infty$;
for *each iteration* t $(t = 1, \ldots, T)$ **do**
  for *each nest* n_i $(\forall i = 1, \ldots, m)$ **do**
    Create Z_1' and Z_2' from Z_1 and Z_2, respectively, such that both contains only features in n_i in which $x_i^j(t) \neq 0, \forall j = 1, \ldots, d$;
    Train OPF over Z_1', evaluate its over Z_2' and stores the accuracy in acc;
    if $(acc > f_i)$ **then**
      $f_i \leftarrow acc$;
      for *each dimension* j $(\forall j = 1, \ldots, d)$ **do**
        $\widehat{x}_i^j \leftarrow x_i^j(t)$;
      end
    end
  end
  $[maxfit, maxindex] \leftarrow max(f)$;
  if $(maxfit > globalfit)$ **then**
    $globalfit \leftarrow maxfit$;
    for *each dimension* j $(\forall j = 1, \ldots, d)$ **do**
      $\widehat{g}^j \leftarrow x_{maxindex}^j(t)$;
    end
  end
  for *each nest* n_i $(\forall i = 1, \ldots, m)$ **do**
    for *each dimension* j $(\forall j = 1, \ldots, d)$ **do**
      Select the worst nests with $p_a \in [0, 1]$ and replace them for new solutions;
    end
  end
  for *each nest* n_i $(\forall i = 1, \ldots, m)$ **do**
    for *each dimension* j $(\forall j = 1, \ldots, d)$ **do**
      $x_i^j(t) \leftarrow x_i^j(t-1) + \alpha \oplus$ Lévy(λ);
      if $(\sigma < \frac{1}{1+e^{x_i^j(t)}})$ **then**
        $x_i^j(t) \leftarrow 1$;
        else
          $x_i^j(t) \leftarrow 0$;
        end
      end
    end
  end
end

in which $\sigma \sim U(0, 1)$ and $x_i^j(t)$ denotes the new egg's value at time step t. Algorithm 1 presents the proposed BCS algorithm for feature selection using the OPF classifier as the objective function.

The algorithm starts with the first loop in Lines 1−4, which initialize each nest with a vector of binary values (Line 3), and Lines 8−42 stand to the main algorithm loop. To evaluate each solution, is necessary to build new training Z_1' and and evaluating Z_2' sets. To fulfill this purpose, each sample $s_i \in Z_1$ and $t_i \in Z_2$ is multiplied by a binary solution vector, i.e., $s_i \times n_i \rightarrow Z_1'$ and $t_i \times n_i \rightarrow Z_2'$. Then, Z_1' can be used to generate an OPF training model, which is evaluated using Z_2'. The the classification rate f_i is associated with the nest n_i, and then each nest is evaluated in order to update its fitness value (Lines 12−13).

Lines 14−15 find out and store in $\widehat{g}^j$ the best nest with the best so far vector solutions. The loop in the Lines 26−30 is responsible to replace the nests with the worst solutions using the probability p, generating new nests randomly as described in [25]. Finally, Lines 31−41 update the binary vector for each nest restricting the generated solutions via Lévy flights [See Eqs. (2–4)] and the sigmoid function (Eq. 6).

4 Methodology

We now describe the proposed methodology to evaluate the performance of feature selection techniques discussed in previous sections (Figure 2 depicts a pipeline to clarify this procedure). Firstly, we randomly partitioned the dataset into N folds, i.e., $Z = F_1 \cup F_2 \cup \cdots \cup F_N$. Note that each fold should be large enough to contain representative samples of the problem. Further, for each fold, we train a given instance of the OPF classifier over a subset of this fold, $Z_i^1 \in F_i$, and an evaluation set

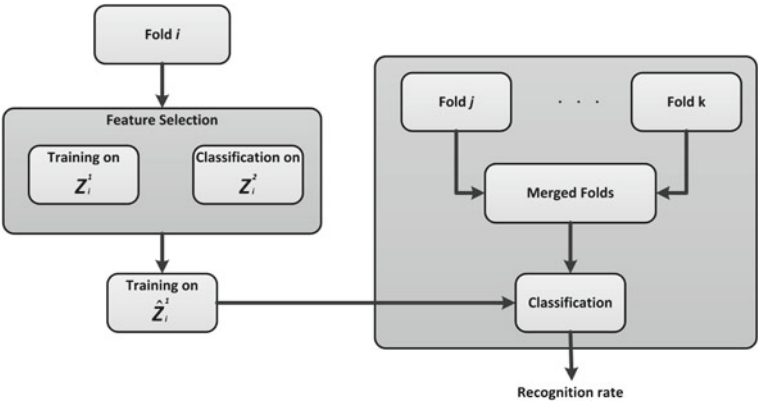

Fig. 2 Flowchart of the proposed methodology

$Z_i^2 \leftarrow F_i \backslash Z_i^1$ is then classified in order to compute a fitness function which will guide a stochastic optimization algorithm to select the most representative set of features. Each member of the population in the meta-heuristic algorithm is associated with a string of bits denoting the presence or absence of a feature. Thus, for each member, we construct a classifier from the training set with only the selected features and compute a fitness function by means of classifying Z_i^2. As long as the procedure converges, i.e, all generations of a population were computed, the agent (bat, firefly, mass, harmony, particle) with the highest fitness value encodes a solution with the best compacted set of features.

Furthermore, we build a classification model using the training set and the selected features, and we also evaluate the quality of the solution computing an effectiveness over the remaining folds, $F_j \in Z \backslash F_i$. Algorithm 2 details the methodology for comparing feature selection techniques.

Algorithm 2: Feature Selection Evaluation

input : A dataset Z, number of folds N, number of agents A, number of iterations I, and a percentage for the training set Z_i^1.
output : A predictive performance for each methods defined by a λ function.
auxiliaries: A bitmap vector V of selected features, and a final training and test sets, $\widehat{Z}^1$, $\widehat{Z}^2$.

```
for each fold F ∈ Z do
    Z^1 ← random set of |Z_1| × |F| samples from F;
    Z^2 ← F\Z^1;
    for each technique T do
        V ← find a minimal subset of features using T, Z^1, Z^2, and the parameters A, I;
        Ẑ^1 ← Z^1\V;
        Create a classifier instance from Ẑ^1;
        for each fold F' ∈ Z\F do
            Ẑ^2 ← F'\V;
            Classify Ẑ^2;
            Compute the predictive performance on Ẑ^2;
        end
    end
end
```

Figure 2 displays the above procedure. As aforementioned, the feature selection is carried on over the fold i, which is partitioned in a training Z_i^1 and an evaluating set Z_i^2. The idea is to represent a possible subset of features as a string of bits, which encodes each agent's position in the search space. Thus, for each agent, we model the dataset using its string of bits, and an OPF classifier is trained over the new Z_i^1 and its effectiveness using this subset of features is assessed over Z_i^2. This recognition rate is then used as the fitness function to guide each agent to new positions until we reach the convergence criterion. The agent with the best fitness function is then

employed to build $\widehat{Z}_i^1$, which is used for OPF training. The final accuracy using the selected subset of features is computed over the remaining folds (red rectangle in Fig. 2). This procedure is repeated over all folds for mean accuracy computation.

5 Simulation, Results and Discussion

In this section, we evaluate the robustness of BCS to accomplish the feature selection task, comparing it with the binary versions of Bat Algorithm (BA), Firefly Algorithm (FA) and Particle Swarm Optimization (PSO). We applied the methodology presented in Sect. 4 to obtain a better quality estimation of each solution. More precisely, we defined $k = 10$ for a cross-validation scheme which implied in ten rounds of feature selection for each method, being the quality of solution evaluated from the remaining nine folds. The performance of those techniques were evaluate over the four public datasets,[1] which the main characteristics are presented in Table 1. In addition, regarding the fitness function and the final classification rates, we used an accuracy measure proposed by Papa et al. [16], which considers the fact that classes may have different concentrations in the dataset. This information can avoid a strong estimation bias towards the majority class in high class imbalance datasets.

We also evaluate how the techniques work with continuous optimization for feature selection purposes, using a sigmoid (Eq. 7) and hyperbolic tangent (Eq. 7) function to map the continuous values to binary ones, respectively:

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{7}$$

and

$$g(x) = |\tanh(x)| . \tag{8}$$

Table 2 presents the parameters used for each evolutionary-based techniques. It is important to clarify that, for all techniques, we assumed a model with a population size of 10 agents and 100 generations to reach a solution.

Table 1 Description of the datasets used for feature selection

Dataset	# samples	# features	# classes
Diabetes	768	8	2
DNA	2,000	180	3
Heart	270	13	2
Mushrooms	8,124	112	2

[1] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Table 2 Parameters setting of optimization techniques

Technique	Parameters
BA	$\alpha = 0.9, \gamma = 0.9$
CS	$\alpha = 1$, $p_a = 0.35, \lambda = 1.5$
FA	$\alpha = 0.8, \beta = 1, \gamma = 0.1$
PSO	$c_1 = 2.0, c_2 = 2.0, w = 0.7$

Figure 3 displays the accuracy performance of all techniques, as well as the number of selected feature over the four datasets. We can see the optimization techniques presented quite similar performances in both case: binary and continuous optimizations. If we observe only the graph of accuracy rates (Fig. 3a, c, e and g), we can infer that feature selection did not improve the classification rates significantly, excepting for the Heart dataset. However, there were considerable feature reductions, specially employing the binary and the continuous optimizations with the sigmoid function.

For Diabetes and Heart datasets, BCS selected the best subset of feature that maximized the OPF accuracy rates (selecting in average six out twelve and five out eight features respectively). The binary PSO was the best on DNA in which it selects the lowest number of feature, around 47 %, and maximized the accuracy rate. For the Mushrooms dataset, BA, FA and PSO performed similarly and two percent better than BCS, even if it has the lowest number of selected features.

In regard to continuous optimization, we can observe the hyperbolic tangent did not work well to transfer the continuous values to binary ones. With this function, the techniques had some difficult to reduced the number of features. This behave may due to the threshold value that those function provide. Unlike, the sigmoid function worked well, providing good feature reductions. However, we can infer that binary and continuous optimization with sigmoid function did not present difference indeed.

6 Conclusions

In this chapter, we discuss the feature selection task as an optimization problem. The main purpose is to evaluate the robustness of Cuckoo Search algorithm to accomplish this task. A binary version of Cuckoo Search was presented and compared against with three other nature-inspired optimization techniques. We provided simulations and analysis over four public datasets, employing a cross-validation strategy to verify how the techniques work for feature selection purposes. The results demonstrated that Cuckoo Search has good capabilities to lead with this kind of problem, being the best one on two out of four datasets, and also similarly to other techniques on the remaining datasets.

As the reader may observe, the binary cuckoo version keeps the parameters α and p_a fixed during all iterations. However, these parameters have an important whole regarding to fine-tunning and convergence rates of the standard Cuckoo Search, as stated by Valian [3]. For the future works, it might be interesting to investigate

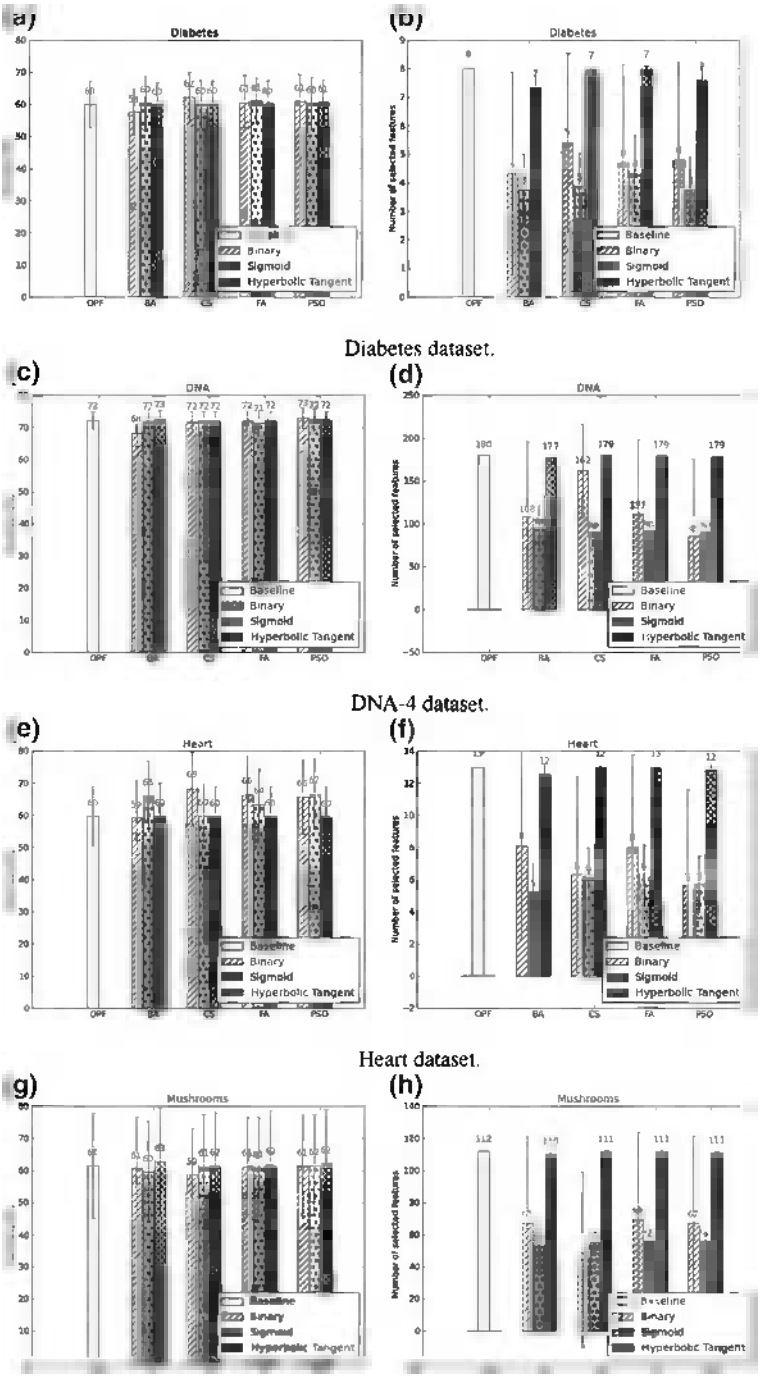

Fig. 3 Experimental results using different transfer functions for each swarm-based optimization technique. **(a)-(b)** Diabetes dataset, **(c)-(d)** DNA dataset, **(e)-(f)** Heart dataset and **(g)-(h)** Mushrooms dataset

how much the binary Cuckoo Search is sensitive to the aforementioned parameters. Further, we should consider to set the parameters dynamically to improve the performance of Binary Cuckoo Search.

References

1. Banati, H., Bajaj, M.: Fire fly based feature selection approach. Int. J. Comput. Sci. Issues **8**(4), 473–480 (2011)
2. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**, 269–271 (1959)
3. Valian, E., Mohanna, S., Tavakoli, S.: On the mean accuracy of statistical pattern recognizers. Int. J. Artif. Intell. Appl. **2**(3), 36–43 (2011)
4. Falcão, A., Stolfi, J., Lotufo, R.: The image foresting transform theory, algorithms, and applications. IEEE Trans. Pattern Anal. Mach. Intell. **26**(1), 19–29 (2004)
5. Firpi, H.A., Goodman, E.: Swarmed feature selection. Proceedings of the 33rd Applied Imagery Pattern Recognition Workshop, pp. 112–118. IEEE Computer Society, Washington, DC, USA (2004)
6. Gandomi, A., Yang, X.S., Alavi, A.: Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Engineering with Computers **29**(1), 17–35 (2013)
7. Geem, Z.W.: Music-Inspired Harmony Search Algorithm: Theory and Applications, 1st edn. Springer Publishing Company, Berlin (2009)
8. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
9. Kaveh, A., Bakhshpoori, T.: Optimum design of steel frames using cuckoo search algorithm with lvy flights. The Structural Design of Tall and Special Buildings pp. n/a–n/a (2011)
10. Kennedy, J., Eberhart, R.: Swarm Intelligence. Morgan Kaufmann, San Francisco (2001)
11. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 5, pp. 4104–4108 (1997)
12. Layeb, A.: A novel quantum inspired cuckoo search for knapsack problems. Int. J. Bio-Inspired Comput. **3**(5), 297–305 (2011)
13. Nakamura, R.Y.M., Pereira, C.R., Papa, J.P., Falcão, A.: Optimum-path forest pruning parameter estimation through harmony search. In: Proceedings of the 24th SIBGRAPI Conference on Graphics, Patterns and Images, pp. 181–188. IEEE Computer Society, Washington, DC, USA (2011)
14. Papa, J., Pagnin, A., Schellini, S., Spadotto, A., Guido, R., Ponti, M., Chiachia, G., Falcão, A.: Feature selection through gravitational search algorithm. In: Proceedings of the 36th IEEE International Conference on Acoustics, Speech and, Signal Processing, pp. 2052–2055 (2011)
15. Papa, J.P., Falcão, A.X., Albuquerque, V.H.C., Tavares, J.M.R.S.: Efficient supervised optimum-path forest classification for large datasets. Pattern Recogn. **45**(1), 512–520 (2012)
16. Papa, J.P., Falcão, A.X., Suzuki, C.T.N.: Supervised pattern classification based on optimum-path forest. Int. J. Imaging Syst. Technol. **19**(2), 120–131 (2009)
17. Ramos, C., Souza, A., Chiachia, G., Falcão, A., Papa, J.: A novel algorithm for feature selection using harmony search and its application for non-technical losses detection. Comput. Electr. Eng. **37**(6), 886–894 (2011)
18. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: GSA: A gravitational search algorithm. Inf. Sci. **179**(13), 2232–2248 (2009)
19. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: BGSA: Binary gravitational search algorithm. Nat. Comput. **9**, 727–745 (2010)
20. Rodrigues, D., Pereira, L.A.M., Almeida, T.N.S., Ramos, C.C.O., Souza, A.N., Yang, X.S., Papa, J.P.: BCS: A binary cuckoo search algorithm for feature selection. In: Proceedings of the IEEE International Symposium on Circuits and Systems. Beijing, China (2013)

21. Senthilnath, J., Das, V., Omkar, S., Mani, V.: Clustering using levy flight cuckoo search. In: J.C. Bansal, P. Singh, K. Deep, M. Pant, A. Nagar (eds.) Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012), Advances in Intelligent Systems and Computing, vol. 202, pp. 65–75. Springer India (2013)
22. Tein, L.H., Ramli, R.: Recent advancements of nurse scheduling models and a potential path. In: Proceedings of 6th IMT-GT conference on mathematics, statistics and its applications (ICMSA 2010) (2010)
23. Vazquez, R.: Training spiking neural models using cuckoo search algorithm. In: IEEE Congress on Evolutionary Computation (CEC), pp. 679–686 (2011)
24. Yang, X.S., Deb, S.: Cuckoo search via lévy flights. In: World Congress on Nature Biologically Inspired Computing (NaBIC 2009), pp. 210–214 (2009)
25. Yang, X.S., Deb, S.: Engineering optimisation by cuckoo search. Int. J. Math. Model. Numer. Optimisation **1**, 330–343 (2010)
26. Yang, X.S., Deb, S.: Cuckoo search: recent advances and applications. Neural Comput. Appl. 1–6 (2013)

How to Generate the Input Current for Exciting a Spiking Neural Model Using the Cuckoo Search Algorithm

Roberto A. Vazquez, Guillermo Sandoval and Jose Ambrosio

Abstract Spiking neurons are neural models that try to simulate the behavior of biological neurons. This model generates a response (spikes or spike train) only when the model reaches a specific threshold. This response could be coded into a firing rate and perform a pattern classification task according to the firing rate generated with the input current. However, the input current must be carefully computed to obtain the desired behavior. In this paper, we describe how the Cuckoo Search algorithm can be used to train a spiking neuron and determine the best way to compute the input current for solving a pattern classification task. The accuracy of the methodology is tested using several pattern recognition problems.

Keywords Cuckoo Search Algorithm · Spiking Neural Networks · Pattern recognition

1 Introduction

Artificial neural networks have been broadly applied in pattern recognition, forecasting and control tasks. Despite their power, they still have some drawbacks that limit their applicability in complex pattern recognition problems. In most of the real cases, in order to solve a pattern recognition problem with an acceptable performance, the expert could spend a lot of time on the design and training of the neural networks.

R. A. Vazquez (✉) · G. Sandoval · J. Ambrosio
Intelligent Systems Group, Universidad La Salle, Benjamin Franklin 47 Col. Hipódromo Condesa, 06140 Mexico City, Mexico
e-mail: ravem@lasallistas.org.mx

G. Sandoval
e-mail: guillermo.sandoval@lasallistas.org.mx

J. Ambrosio
e-mail: jose.ambrosio@lasallistas.org.mx

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_8,

Although some meta-heuristics have been applied to eradicate this disadvantage, the necessity of new powerful neural models is imminent.

Spiking neural networks have shown to be a suitable type of artificial neural networks for solving pattern recognition problems. In the last years, they have been applied in a wide range of problems, including brain region modeling [20], auditory processing [2, 21], visual processing [1, 38], robotics [12, 13] and so on. Although they have been widely used by the neuroscientist comunity, their application in pattern recognition is gaining popularity within the computational intelligence comunity.

Several spiking neural models have been proposed in the last years [24] and have been called the 3rd generation of artificial neural networks [30]. However, its application in the field of artificial intelligence is practically new. These models increase the level of realism in a neural simulation and incorporate the concept of time. Taking into account new mechanisms based on the behavior of biological neurons or neurological and cognitive aspects of human brain, artificial neurons could reach their maximum power and up-perform their accuracy in pattern recognition problems [43, 46]. This suggests that spiking neural models are natural candidates to be applied in the field of artificial intelligence, including the solution of pattern classification tasks.

Spiking neural models have been applied in some pattern recognition tasks. During the training phase, they adjust their synaptic weights using several techniques such as the back-propagation algorithm [7, 9]. In [15], the authors described how a genetic algorithm can be used during the learning process of a spiking neural network; however, the model was not applied to perform a pattern recognition problem. In [3], the authors show how a spiking neural network can be trained by a quantum PSO and then applied to a string pattern recognition problem. In [16] the authors trained a spiking neural network using a PSO algorithm, and then it was applied in three pattern recognition problems; however, in each problem, the authors had to design the topology of the network. Although these models have been successfully applied in some pattern recognition problems, some of the major drawbacks occur during the training and design phase.

In [39, 41, 42, 44, 45], the authors show how only one spiking neuron such as Leaky-Integrate-and-Fire and Izhikevich models [23, 25] can be applied to solve different linear and non-linear pattern recognition problems. In general, the methodology described in those algorithms goes as follows: Given a set of input patterns belonging to K classes, (1) each input pattern is transformed into an input current, (2) then the spiking neuron is stimulated during T ms (3) and finally, the firing rate is computed. After adjusting the synaptic weights of the neuron model by means of a bio-inspired algorithm, input patterns belonging to the same class must generate similar firing rate. On the contrary, patterns belonging to other classes must generate firing rates different enough to discriminate among the classes.

In those papers, the authors use bio-inspired algorithms as a learning strategies to adjust the synaptic weights of two different neural models. The obtained results suggest that using only one spiking neuron a pattern recognition task can be solved with an acceptable accuracy. Furthermore, after comparing the accuracy obtained using several bio-inspired algorithms, we observed that the accuracy of these models

slightly varies during a pattern recognition task. This fact means that the accuracy of these models is not related to the learining algorithm. Although the accuracy obtained is highly acceptable, it could be possible to increase their acuracy exploring the behavior of different neural models and the way to compute the input current.

However, the development of new spiking models or new ways to compute the input current to stimulate the model is not a trivial task. Nonetheless, bio-inspired algorithms could be a mechanism of exploration and exploitation to implement new ways to compute the input current or design new spiking neural models.

Several meta-heuristics have been proposed in the recent years. Although they have common features, they also have some features that make them different from each other. Among the most popular, we could mention the particle swarm optimization (PSO), differential evolution (DE) and artificial bee colony (ABC) algorithms [28, 29, 35]. These algorithms have been applied in a wide range of optimization problems and the field of artificial neural networks has been beneficed with these kinds of methods. These methods have been applied not only to the problem of adjusting the synaptic weight of an artificial neural network, but also to design the topology and select the transfer function of the neurons that compose the network [15, 16, 27].

Cuckoo Search algorithm (CS) is a novel meta-heuristic based on the obligate brood parasitic behavior of some cuckoo species in combination with the Lévy flight behavior of some birds and fruit flies. This algorithm has been applied in several optimization problems, outperforming the results achieved by the well-known PSO algorithm [49]. Therefore, these promising results suggest the CS algorithm could be a powerful method to adjust the synaptic weights of an spiking neuron and find the best way to compute its input current.

In this paper, we analyze the advantage of automatically design the way to compute the input current that stimulates a spiking neuron model. It is presented a combination of the methodology described in [39, 41] with the one we propose to automatically design the equation to compute the input current by means of CS. In order to test the accuracy of the proposed methodology, we applied the spiking neuron model to solve some non-linear and two real pattern recognition problems: odor recognition and crop classification.

2 Cuckoo Search Algorithm

Based on the description presented in [41], Cuckoo Search (CS) algorithm is a novel meta-heuristic proposed by Xin-She Yang [49]. This algorithm was inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds. Some host nest can engage in direct conflict. If a host bird discovers the eggs are not their owns, they will either throw these alien eggs away or simply abandon its nest and build a new nest elsewhere. Some other species have evolved in such a way that female parasitic cuckoos are often very specialized in the mimicry

in colour and pattern of the eggs of a few chosen host species. This reduces the probability of their eggs being abandoned and thus increases their reproductivity.

On the other hand, several studies have shown that flight behaviour of many animals and insects have the typical characteristics of the Lévy flights. Taking into account these breeding and flight behaviors, the authors in [49] proposed the CS algorithm.

The CS algorithm follows the next three idealized rules: (1) Each cuckoo lays one egg at a time, and dumps its egg in randomly chosen nest; (2) The best nests with high quality of eggs will carry over to the next generations; (3) The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in [0, 1]$. In this case, the host bird can either throw the egg away or abandon the nest, and build a completely new nest.

Based on these three rules, the basic steps of the Cuckoo Search (CS) can be summarized as the next pseudo code:

1: Generate initial population of N host nest $x_i \forall_i, i = 1, ..., n$
2: **while** $t < MaxGeneration$ or (stop criterion) **do**
3: Get a cuckoo randomly by Lévy flights and evaluate its fitness F_i.
4: Choose randomly a nest j among N.
5: **if** $F_i > F_j$ **then**
6: Replace j by the new solution.
7: **end if**
8: A fraction (p_a) of worse nest are abandoned and new ones are built.
9: Keep the best solutions (or nest with quality solutions).
10: Rank the solutions and find the current best.
11: **end while**

This algorithm uses a balanced combination of a local random walk and the global explorative random walk, controlled by a switching parameter p_a [19]. The local random walk can be written as

$$x_i^{t+1} = x_i^t + \alpha s \otimes H(p_a - \varepsilon) \otimes (x_j^t - x_k^t) \tag{1}$$

where x_j^t and x_k^t are two different solutions selected by random permutation, $H(u)$ is a Heaviside function, ε is a random number drawn from a uniform distribution, and s is the step size. On the other hand, the global random walk is carried out using Lévy flights:

$$x_i^{t+1} = x_i^t + \alpha L(s, \lambda) \tag{2}$$

where

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda) \sin(\pi \lambda / 2)}{\pi} \frac{1}{s^{1+\lambda}}, \; s >> s_0 > 0 \tag{3}$$

Here, $\alpha > 0$ is the step size scaling factor, which should be related to the scales of the problem of interest. In most cases, we can use $\alpha = O(L/10)$, where L is the characteristic scale of the problem of interest, while in some cases, $\alpha = O(L/100)$

can be more effective and avoid the need to fly too far. Equation (3) is essentially the stochastic equation for a random walk. In general, a random walk is a Markov chain whose next status/location only depends on the current location (the first term in Eq. (3)) and the transition probability (the second term).

The Lévy flight has been applied to a diverse range of fields, describing animal foraging patterns, the distribution of human travel and even some aspects of earthquake behaviour [6].

A Lévy distribution is advantageous when target sites (solutions) are sparsely and randomly distributed because the probability of returning to a previously visited site is smaller than for a gaussian distribution [47]. Due to this walk is more efficient in exploring the search space, it could be used to adjust the synaptic weights of a spiking neuron.

Here the steps essentially construct a random walk process with a power-law step-length distribution with a heavy tail. Some of the new solutions should be generated by Lévy walk around the best solution obtained so far, this will speed up the local search. However, a substantial fraction of the new solutions should be generated by far field randomization whose locations should be far enough from the current best solution. This will make sure the system will not be trapped in a local optimum.

3 Spiking Neural Models

Following the section described in [41], A typical spiking neuron can be divided into three functionally distinct parts, called dendrites, soma, and axon. The dendrites play the role of the *input device* that collects signals from other neurons and transmits them to the soma. The soma is the *central processing unit* that performs an important non-linear processing step: if the total input exceeds a certain threshold, then an output signal is generated. The output signal is transmmited by the *output device*, the axon, which delivers the signal to other neurons. The neuronal signals consist of short electrical pulses. The pulses, so-called action potentials or spikes, have an amplitude of about 100 mV and typically a duration of 1–2 ms.

A chain of action potentials emitted by a single neuron is called a spike train (a sequence of stereotyped events, which occur at regular or irregular intervals). Since all spikes of a given neuron look alike, the form of the action potential does not carry any information. Rather, it is the number and the timing of spikes which matter. The action potential is the elementary unit of signal transmission. Action potentials in a spike train are usually well separated. Even with a very strong input, it is impossible to excite a second spike during or immediately after a first one. The minimal distance between two spikes defines the absolute refractory period of the neuron. The absolute refractory period is followed by a phase of relative refractoriness where it is difficult, but not impossible to excite an action potential [17].

Several spiking models have been proposed in the recent years. Models such as the well-know Integrate-and-Fire model, Resonate-and-Fire, Izhikevich model, FitzHugh-Nagumo model, Hodgkin-Huxley model, present different neuro-dynamic properties [24].

Due to its simplicity, we adopted the Izhikevich model which is described with the next equation

$$\begin{aligned} C\dot{v} &= k\,(v - v_r)\,(v - v_t) - u + I \quad \text{if } v \geq v_{peak} \text{ then} \\ \dot{u} &= a\,\{b\,(v - v_r) - u\} \quad v \leftarrow c, u \leftarrow u + d \end{aligned} \tag{4}$$

As the reader can observe, this model has only nine dimensionless parameters. Depending on the values of a and b, it can be an integrator or a resonator. The parameters c and d take into account the action of high-threshold voltage-gated currents activated during the spike, and affect only the after-spike transient behavior. v is the membrane potential, u is the recovery current, C is the membrane capacitance, v_r is the resting membrane potential, and v_t is the instantaneous threshold potential [25].

The parameters k and b can be found when one knows the neuron's rheobase and input resistance. The sign of b determines whether u is an amplifying ($b < 0$) or a resonant ($b > 0$) variable. The recovery time constant is a. The spike cutoff value is v_{peak} , and the voltage reset value is c. The parameter d describes the total amount of outward minus inward currents activated during the spike and affecting the after-spike behavior. A detailed description of the model can be found in [25].

Different choices of the parameters result in various intrinsic firing patterns: RS (*regular spiking*) neurons are the most typical neurons in the cortex; IB (*intrinsically bursting*) neurons fire a stereotypical burst of spikes followed by repetitive single spikes; CH (*chattering*) neurons can fire stereotypical bursts of closely spaced spikes; FS (*fast spiking*) neurons can fire periodic trains of action potentials with extremely high frequency practically without any adaptation (slowing down); and LTS (*low-threshold spiking*) neurons can also fire high-frequency trains of action potentials, but with a noticeable spike frequency adaptation [23].

This kind of models can, in principle, be applied to the same problems as a classic perceptron is. However, as was reported in other papers [39, 41, 42, 44, 45], this model can improve the accuracy of the classic perceptron even in non-linear pattern recognition problems.

It is important to notice that the response of the Izhikevich neuron changes, when the input current signal changes. This current provokes that the neuron generates different firing rates. In terms of the firing rate, the spiking models are able to perform a pattern recognition task.

The firing rate is computed as the number of spikes generated in an interval of duration T divided by T. The neuron is stimulated during T ms with an input signal and fires when its membrane potential reaches a specific value generating an action potential (spike) or a train of spikes.

4 Proposed Methodology

As we described in [41], the authors in [39, 42, 44] proposed a methodology which describes how a spiking neuron can be applied to solve different pattern recognition problems. That methodology is based on the hypothesis that patterns belonging to

the same class generate similar firing rates in the output of the spiking neuron. On the contrary, patterns belonging to other classes generate firing rates different enough to discriminate among the classes.

Following the same approach and according to [41], let $\mathrm{D} = \left\{\mathbf{x}^i, k\right\}_{i=1}^{p}$ be a set of p input patterns where $k = 1, \ldots, K$ is the class to which $\mathbf{x}^i \in \mathbb{R}^n$ belongs. First of all, each input pattern must be transformed into an input current I. It is important to notice that the spiking neuron model is not directly stimulated with the input pattern $\mathbf{x}^i \in \mathbb{R}^n$, but with an injection current I computed from the input pattern. Since synaptic weights of the model are directly connected to the input pattern $\mathbf{x}^i \in \mathbb{R}^n$, the injection current, generated with this input pattern, can be computed as:

$$I = \gamma \cdot \mathbf{x} \cdot \mathbf{w} \tag{5}$$

where $\mathbf{w}^i \in \mathbb{R}^n$ is the set of synaptic weights of the neuron model and $\gamma = 100$ is a gaining factor that helps the neuron fire. This transformation could provoke that more than one input pattern (from the same class) be transformed into the same or similar current, helping the neuron to produce similar firing rates.

After that, the spiking neuron is stimulated with the input current I during T ms and the firing rate is computed.

These three steps are applied to each input pattern. Once the firing rate of each input has been obtained, the average firing rate $\mathbf{AFR} \in \mathbb{R}^K$ of each class must be computed.

The synaptic weights of the model must be adjusted in order to generate the desired behavior in the output of the spiking neuron. This stage corresponds to the learning (training) phase of the spiking neuron. To achieve this goal, the cuckoo search algorithm is used as a learning strategy to train the spiking neural model.

Once the spiking neuron is trained, the firing rates produced by each input pattern are used to determine the class to which an unknown input pattern $\tilde{\mathbf{x}}$ belongs. This is done by the next equation

$$cl = \arg\min_{k=1}^{K} \left(|AFR_k - fr|\right) \tag{6}$$

where fr is the firing rate generated by the neuron model stimulated with the input pattern $\tilde{\mathbf{x}}$.

The synapses of the neuron model $\mathbf{w}$ are adjusted by means of the cuckcoo search algorithm. In order maximize the accuracy of the spiking neuron, the best set of synaptic weights must be found. The following fitness function could be used to find the set of synaptic weights that aids the model to generate the desired behavior:

$$f(\mathbf{w}, D) = 1 - \text{performance}(\mathbf{w}, D) \tag{7}$$

where performance$(\mathbf{w}, D)$ is a function which follows the steps described in above section and computes the classification rate given by the number of patterns correctly

classified divided by the number of tested patterns, **w** are the synapses of the model and D is the set of input patterns.

The above function makes it possible to apply the spiking neuron in different pattern classification tasks.

4.1 Approach for Designing the Input Current Equation

We already described how input current stimulates the spiking neuron to get the firing rate. Equation (5) is a linear combination that computes the input current on the dendritic branches (Ib) in terms of the input pattern features and the synaptic weights.

$$I = Ib_1 + Ib_2 + \cdots + Ib_{n-1} + Ib_n \tag{8}$$

where $Ib_i = x_i \cdot w_i$.

However, each brach could have several dendrites. One way to compute the input current in the dendritic branch composed of several dendrites is defined as:

$$Ib = \left[x_1^{p_1} \cdot x_2^{p_2} \cdot \cdots \cdot x_{n-1}^{p_{n-1}} \cdot x_n^{p_n}\right] \cdot w \tag{9}$$

where x_i represents the voltage in dendrite i, p_i representes the synaptic value that interacts with dendrite i and w is the synaptic value of the branch. This equation could be reduced to

$$Ib = \prod_{j=1}^{n} x_j^{p_{ij}} \cdot w_i \tag{10}$$

where i represents a brach of the neuron, x_j represents the voltage in dendrite j, p_{ij} representes the synaptic value that interacts with dendrite j of brach i and w_i is the synaptic value of the branch i.

Finally, if we consider that input current in the brach could be an inhibitory or exhitatory signal, we could introduce an operator O that forces the current to be inhibitory or exhitatory, resulting in the following equation:

$$Ib = O_i \prod_{j=1}^{n} x_j^{p_{ij}} \cdot w_i \tag{11}$$

where $O \in \{+, -\}$.

Based on this equation, we can compute the input current I that stimulates the spiking neuron model. However, it is necesary to determine the number of branches in the neuron, if the branch emits an exhitatory or inhibitory signal as well as the dendrites of each brach in terms of the input features. This fact leads us to the problem of designing automatically the equation used to compute the input current.

Instead of finding only the values of synaptic weights **w** as was described in [41], the proposed methodology must also find if the branch emits an exhitatory or inhibitory signal, the dendrites of each brach in term of the input features as well as synaptic weights of dendrites and branches.

In order to design the equation to compute the input current we use the following equation:

$$I = \sum_{i=1}^{m} O_i \prod_{j=1}^{n} \left(x_j \cdot s_{ij}\right)^{p_{ij}} \cdot w_i \tag{12}$$

where m represents the number of branches, n the number of features, i represents a brach of the neuron, x_j represents the voltage in dendrite j, p_{ij} representes the synaptic value that interacts with dendrite j of brach i, w_i is the synaptic value of the branch i, $O \in \{+, -\}$ indicates if the signal is exhitatory $(+)$ or inhibitory $(-)$ and $s_{ij} \in \{0, 1\}$ indicates if feature j is associate with a dendrite in the branch i.

Based on this equation, we have to find the best values of matrix $\mathbf{p} \in \mathbb{R}^{m \times n}$, $\mathbf{s} \in \{0, 1\}^{m \times n}$, $\mathbf{w} \in \mathbb{R}^m$ and $\mathbf{O} \in \{+, -\}^m$ using the Cuckoo Search algorithm that maximize the accuracy of the spiking model in terms of Eq. (7). According to this, each nest of the Cuckoo Search algorithm will have a size of $2\,(m \times n) + 2m$ elements.

Suppose that based on a pattern recognition problem we define $m = 3$ and $n = 4$. Then suppose that after applying the Cuckoo Search algorithm to find the equation to compute the input current that maximizes the accuracy of the spiking model, we have the following matrix values:

$$\mathbf{s} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \mathbf{p} = \begin{bmatrix} 2 & 1 & -1 & 2 \\ 1 & 3 & 2 & 1 \\ 0.1 & 2 & -1.5 & 2 \end{bmatrix}, \mathbf{O} = \begin{bmatrix} + \\ - \\ + \end{bmatrix}$$

Finally, by subtituting the corresponding values obtained with the Cuckoo Search Algorithm in Eq. (12), the equation that computes the input current is defined as: $I = \left(x_2^1 x_3^{-1}\right) \cdot w_1 - \left(x_1^1 x_3^2\right) \cdot w_2 + \left(x_2^2 x_4^2\right) \cdot w_3$.

5 Experimental Results

In this section, we present the experimental results obtained with the proposed methodology. The section is divided into three subsection. The first section presents a comparison of the proposed methodology evolving the input current and synaptic weights against evolving only synaptic weights using different bio-inspired algorithms. In the last two subsections, we present preliminar results applying spiking neural models using the proposed methodology for solving a crop classification problem and an odor recognition problem.

5.1 Analysis and Comparisson of Experimental Results

Six groups of experiments were performed in order to evaluate the accuracy of the proposed method when the equation to compute the input current that stimulates the spiking neuron is designed using the cuckoo search algorithm. Each group of experiments describes a pattern recognition problem took from different datasets. Five of them were taken from the UCI machine learning benchmark repository [31]: iris plant, glass, diabetes, liver-bupa and wine datasets. The other one was generated from a real object recognition problem.

The iris plant dataset describes a pattern recognition problem that classifies three different types of iris plant in terms of four features. The wine dataset is composed of three classes, and each input pattern is composed of 13 features. The glass identification problem tries to classify six types of glass in terms of nine features related to their oxide content. In our case, we classified only two types of glasses. The diabetes dataset is composed of two classes, and each input pattern is composed of eight features. The liver dataset is composed of two classes, and each input pattern is composed of six features.

For the case of the real object recognition problem, a dataset was generated from a set of 100 images which contains five different objects whose images are shown in Fig. 1 [46]. Objects were not recognized directly from their images, but by an invariant description of each object. Several images of each object in different positions, rotations and scale changes were used. To each image of each object, a standard thresholder [34] was applied to get its binary version. Small spurious regions were eliminated from each image by means of a size filter [26]. Finally, the seven well-known Hu invariant moments, to translations, rotations and scale changes [22], were computed to build the object recognition dataset.

The parameters for the Izhikevich neuron were set as $C = 100$, $v_r = -60$, $v_t = -40$, $v_{peak} = 35$, $k = 0.7$, $a = 0.03$, $b = -2$, $c = -50$, and $d = 100$. The Euler method was used to solve the differential equation of the model with $dt = 1$. The number of terms used to compute input current I from the input pattern was set to 5 with a duration of $T = 1000$. For the case of the cuckoo search algorithm, $N = 40$, $MAXGEN = 1000$, $p_a = 0.25$.

The classification rate of the model was computed as the number of input patterns correctly classified divided by the total number of tested input patterns. To validate

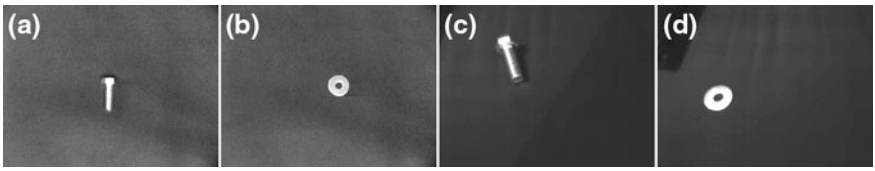

Fig. 1 **a**, **b** Some of the images used to train the spiking neuron using the proposed method. **c**, **d** Some of the images used to test the trained spiking neuron

the accuracy of the proposed method when the spiking neuron is trained with the cuckoo search algorithm, 10 experiments over each dataset were performed. It is important to notice that, for every experiment two subsets were randomly generated from each dataset. The 80 % of the samples compose the training subset, and the remain the testing subset.

From these experiments, we observed that when the equation for computing the input current was included during the learning process, the learning error rapidly decreases at the beginning of the evolutionary learning process. On the contrary, the learning error changes at a slowly rate when a certain number of generations is reached. Furthermore, we observed that, whereas for some problems the achieved learning error was small, for some other problems the learning error was not good enough. In general, the behavior of the learning error using this approach was the same compared against the behavior when input current equation was not included during learning process, see [41]. In Fig. 2 is shown how the learning error evolves through each generation of the cuckoo search algorithm. Particularly, for the case of glass, diabetes and liver problems, the error did not converge to an acceptable value.

Once the equation for computing the input current was designed and the spiking neuron was trained, we proceed to evaluate the accuracy of the neuron using the testing subset. In Table 1, we show the equations designed with the proposed methodology which provide the best results for each dataset. As the reader can observe, each equation is composed of five terms and each term is composed by a combination of some features of the dataset problem. According to the number of times the features

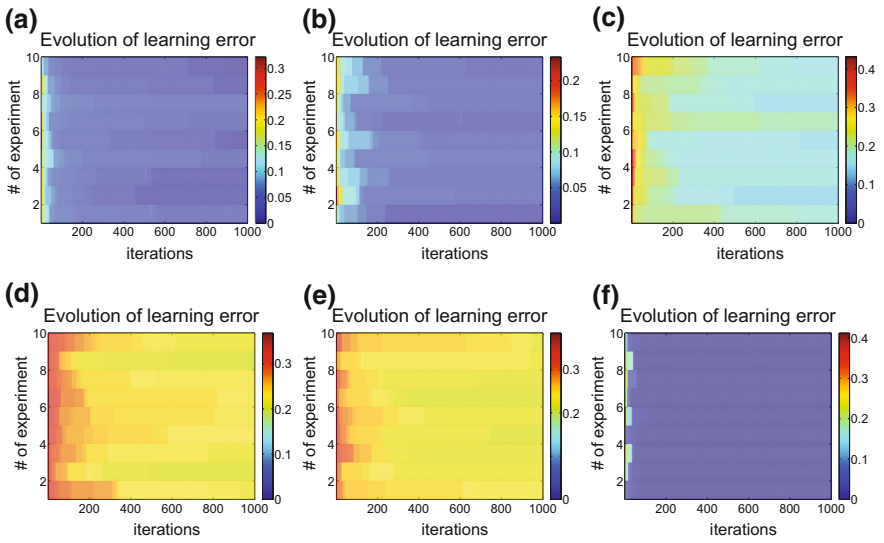

Fig. 2 Evolution of training error achieved by the spiking model during the learning phase using the cuckoo search algorithm. Ten experiments over each dataset are shown **a** Wine dataset. **b** Iris dataset. **c** Glass dataset. **d** Diabetes dataset. **e** Liver dataset. **f** Object recognition dataset

Table 1 Equations generated with the proposed methodology for computing input currernt

Dataset	Equation generated with the proposed methodology
Wine	$I = -[(x_1^{-4.25}x_3^{1.18}x_{10}^{3.95}x_{11}^{4.70})]w_1 + [(x_1^{-2.65}x_5^{-0.10}x_6^{-4.00}x_9^{-1.11}x_{11}^{1.69}x_{13}^{-3.81})]w_2$ $+ [(x_2^{-5.00}x_3^{0.72}x_6^{-0.28}x_7^{-0.39}x_{11}^{1.59})]w_3 + [(x_1^{0.15}x_7^{-4.27}x_{10}^{0.89}x_{12}^{-4.18})]w_4$ $+ [(x_2^{-1.72}x_3^{4.14}x_4^{0.75}x_7^{-5.00}x_8^{4.73})]w_5$
Iris plant	$I = -[(x_2^{23.54}x_4^{46.00})]w_1 - [(x_3^{-0.61}x_4^{-0.87})]w_2 - w_3$ $-[(x_2^{17.14}x_3^{6.56}x_4^{-27.35})]w_4 + [(x_2^{35.60})]w_5$
Glass	$I = +[(x_1^{-1.20}x_6^{2.28}x_7^{0.66})]w_1 - [(x_3^{-4.52}x_4^{4.62}x_5^{-5.00}x_7^{-3.88})]w_2$ $+ [(x_1^{-1.65}x_3^{2.55}x_4^{4.73}x_7^{-0.54})]w_3 + [(x_1^{-4.44}x_3^{-4.14}x_4^{2.52})]w_4 + [(x_1^{-4.57}x_5^{-5.00}x_8^{4.95})]w_5$
Diabetes	$I = -[(x_1^{11.59})]w_1 - [(x_4^{6.55}x_5^{2.10}x_7^{4.89}x_8^{13.99})]w_2 + [(x_2^{-10.17}x_6^{-5.36}x_8^{-0.56})]w_3$ $+ [(x_2^{-0.62})]w_4 + [(x_1^{2.98}x_2^{-2.67})]w_5$
Liver	$I = +[(x_5^{4.24})]w_1 - [(x_2^{-0.44}x_4^{22.40})]w_2 - [(x_1^{-25.07}x_2^{-4.07}x_3^{-27.22}x_4^{4.90}x_5^{10.78})]w_3$ $- [(x_1^{-3.83}x_2^{2.39}x_4^{-22.43}x_6^{2.44})]w_4 + [(x_1^{6.93}x_2^{-4.43}x_5^{1.40}x_6^{4.98})]w_5$
Object rec.	$I = -[(x_2^{1.55}x_4^{4.50}x_6^{-2.05})]w_1 + [(x_1^{0.80}x_2^{1.59})]w_2 + [(x_3^{1.66}x_4^{0.89}x_6^{3.60}x_7^{-0.36})]w_3$ $- [(x_1^{-0.74}x_3^{-0.24}x_7^{-2.13})]w_4 - [(x_2^{3.19}x_5^{4.91}x_7^{-2.49})]w_5$

contribute to the equation we could figured out how relevant they are for solving the problem. Furthermore, the exponent associated to each feature could indicate how much contributes for solving the problem. This information could help us to perform a reduction of the dimentionality in the dataset.

In general, the accuracy of the neuron was highly acceptable for the six datasets, see Fig. 3. For the case of the wine, iris and object recognition, we observed that the best percentage of classification achieved with the cuckoo search algorithm, using the training subset was of 99.3, 99.1 and 100 %, respectively. Furthermore, the best classification performance achieved with the testing subset was 100 % for the three datasets, better than that achieved with the training subset.

On the contrary, for the case of the glass, diabetes and liver dataset we observed that the best percentage of classification achieved, using the training subset was of 85.4, 79.4 and 79.7 %, respectively. This fact was provoked due to during training the proposed method did not achieve an acceptable error. As with the previous datasets, the accuracy of the spiking neuron increased when it is stimulated with patterns took from the testing subset, 82.7, 82.3 and 82.1 %, respectively. The results achieved with the spiking neuron stimulated with the input current designed with the cuckoo search algorithm were acceptable.

In terms of the percentage of classification, we observe that the spiking neuron is able to solve these pattern recognition problems with an acceptable accuracy. This means that using the firing rates generated with each input pattern is possible to perform a discrimination task, in other words, patterns from the same class generate similar firing rates. Besides, patterns belonging to other classes generates different firing rates. This phenomenon can be observed in Figs. 4, 5. Each Figure presents the experimental results of two of the ten experiments done with each dataset; each

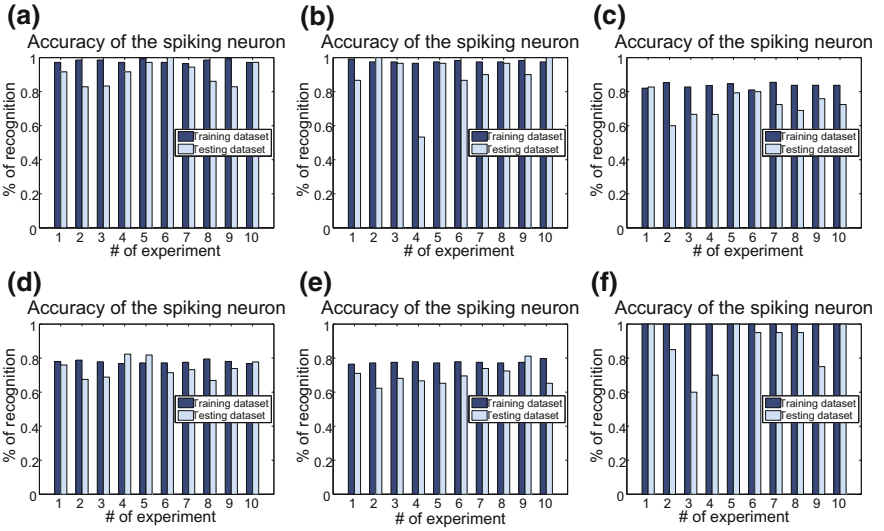

Fig. 3 Percentage of recognition obtained with the spiking neuron using the cuckoo search algorithm. **a** Wine dataset. **b** Iris plant dataset. **c** Glasss dataset. **d** Diabetes dataset. **e** Liver dataset. **f** Object recognition problem

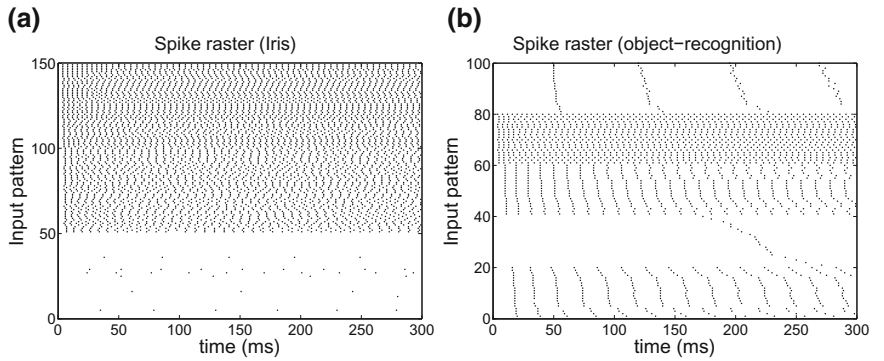

Fig. 4 **a** Spiking time obtained with the iris plant dataset. Three different firing rates which correspond to three different classes can be observed. Notice that different firing rates for the same problem were found in each experiment. **b** Spiking time obtained with the object recognition dataset. Five different firing rates which correspond to five different classes can be observed. Notice that different firing rates for the same problem were found in each experiment

spot in the plot represents a spike generated by the neuron in a time t with a pattern p. The X axis indicates the time that the spiking neuron was stimulated by an input pattern. With the intention of appreciating the firing rates generated by the neuron, we decide to plot the spiking time using different periods of stimulation depending of the dataset. The Y axis represents a label that indicates the number of patterns

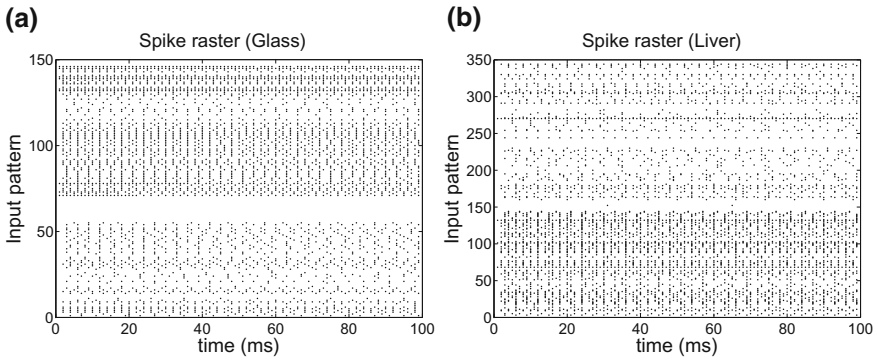

Fig. 5 **a** Spiking time obtained with the glass dataset. Two different firing rates which correspond to two different classes can be observed. Notice that different firing rates for the same problem were found in each experiment. **b** Spiking time obtained with the liver dataset. Two different firing rates which correspond to two different classes can be observed. Notice that different firing rates for the same problem were found in each experiment

used to stimulate the trained spiking neuron. These patterns are ordered by class, in other words, the first set of p patterns belongs to the same class and the second set of p patterns belongs to another class, and so on.

Figure 4 show the spiking time when the neuron is stimulated with patterns from the iris and object recognition datasets. Notice that the spiking time generated with patterns from the same class is almost the same. Therefore, the set of synaptic weights found with the cuckoo search algorithm provokes that the Izhikevich neuron generates similar firing rates when it is stimulated with patterns from the same class. Furthermore, we also observe that different firing rates were generated with patterns that belong to other classes. These different firing rates make possible to apply this methodology in a pattern classification task.

On the other hand, Fig. 5 show the experimental results obtained with the glass and liver datasets, respectively. In this case, the set of synaptic weights found with the cuckoo search algorithm provokes that the Izhikevich neuron does not generate similar firing rates when it is stimulated with patterns from the same class; this fact could cause that some patterns from the same class to be classified in a different class. Furthermore, the Izhikevich neuron does not generate firing rates different enough to discriminate among patterns from different classes. Do not forget that during training phases, the methodology did not converge to a good solution. Nonetheless, the percentage of recognition achieved with the proposed method was in general acceptable.

The average classification rate computed from all experimental results is shown in Table 2. The results obtained with the spiking neuron model using the proposed methodology were compared against the results obtained with the method described in [39, 41, 42, 44, 45]. Although with some datasets the proposed methodology provides better results than when only synaptic weights are evolved and vice verse,

Table 2 Average accuracy provided by the methods using different databases

Dataset	Method using DE [39]		Method using PSO [45]		Method using CS [41]		Proposed method using CS	
	Tr. cr.	Te. cr.	Tr. cr.	Te. cr.	Tr. cr.	Te. cr.	Tr. cr.	Te. cr.
Wine	0.9796	0.8744	0.9782	0.8879	0.9831	0.9078	0.9796	0.9074
Iris plant	0.9933	0.9833	0.9933	0.9700	0.9942	0.9467	0.9775	0.8967
Glass	0.8158	0.7411	0.8178	0.7457	0.8080	0.7646	0.8362	0.7251
Diabetes	0.8038	0.7371	0.7990	0.7619	0.8051	0.7477	0.7777	0.7397
Liver	0.7620	0.6870	0.7591	0.6754	0.7609	0.6536	0.7761	0.6957
Object rec.	1.0000	0.9850	1.0000	0.9950	1.0000	1.0000	1.0000	0.8750

Tr. cr. = Training classification rate, Te. cr. = Testing classification rate

we could not say that one approach is better than the other. However, the input current equation could provide information about the importance of each feature of the dataset problem, which could help to reduce the dimensionality of the dataset.

As was shown in previous works, one spiking neuron can be considered as an alternative way to perform different pattern recognition tasks. Compared against the results obtained with a network of perceptrons, the proposed method provides better results [39]. These preliminary results suggest that evolving the equation for computing the input current does not really help to improve the accuracy of a spiking model; however, the information that the equation provides could help to understand how relevant is each feature for solving the pattern recognition problem, even to reduce the dimensionality of the pattern. We can also conjecture that if we design a neuron model for an specific pattern recognition problem, perhaps we could improve the experimental results obtained in this research. However, that is something that should be proven.

Related to the meta-heuristics, the cuckoo search algorithm can be considered as a learning strategy to adjust the synaptic weights of a third generation neural model. Furthermore, its exploration and explotation capabilities make it possible to automatically design the equation to compute the input current. As the experimental results shown, the spiking neuron trained with this algorithm and using the designed input current equation could perform a pattern classification task with an acceptable accuracy.

5.2 Application of the Proposed Methodology in a Crop Classification Problem

In this subsection, a remote sensing approach to crop classification is presented, which uses information within the visible part of the electromagnetic spectrum. This specific problem has been around approximately since 1972, with studies such as [32] and government initiatives in the United States. It is of high importance for those administrating food necessities, resources required in agricultural activities,

land owners interested in yield estimation and crop rotation, etc., and method for crop classification using remote sensing that could be completely relied on would allow it to be cheaper, to be carried on more frequently and to be more comprehensive.

Images used in remote sensing are very different depending on the sensor used to acquire the information. One of these sensors is the Synthetic Aperture Radar (SAR), which captures dielectric characteristics of objects such as their structure (size, shape and orientation). Other types of images are the ones known as multiespectral and hiperespectral. The first ones normally refer to information captured in the visible electromagnetic spectrum which also include near infrared or infrared bands. Hiperespectral images are similar to multispectral images in the sense of spatial resolution and type of information, with the main difference being the higher band resolution hiperespectral images have, with numbers between 128 and 256 bands, making them a very detailed source of information. Studies working with this kind of images more often try to reduce the number of bands as shown in [18, 37].

One of the biggest disadvantages, of using radar or multiespectral and hiperespectral images is the fact that they are not accessible to everyone, at least not in the sense of one being able to go online in the internet locating a region and appling a classification technique to that region.

In this subsection, an approach which focuses on satellite obtained images of data within the visible electromagnetic spectrum is presented. With the main objective of reducing the requirements imposed on the characteristics of the information used for the classification process. A comparison of the performance of two feature extraction and three classification algorithms is performed. For feature extraction Gray Level Co-Occurrence Matrix (GLCM) [19] and Color Average Features (CAF) are used. While the classifying process is done with a Radial Basis Function Neural Network (RBFNN) [36], a Bayesian classifier [11] and a spiking neuron trained with the proposed methodology.

The images used to test the accuracy of the proposed methodology were obtained from the National Institute of Statistics and Geography (INEGI, according to its acronym in Spanish). This type of image has only three bands, one for the red part of the spectrum, one for the blue and another one for the green. Its quantification is of 8 bits per pixel on each band and the spatial resolution of 1 meter per pixel, with a total size of 5946 pixels width and 7071 pixels height.

The information was captured on December of 2008, over the region on the coordinates with a northwest corner in 222755.0, 2331410.0 and a southeast corner in 228701.0, 2324339.0. It has a scale of 1:40,000 and has been orthorectified with the help of geodetic control points and Digital Elevation Model.

Once obtained the image, we defined a crop classification problem for recognizing two diffent types of crops. To apply the proposed methodology, a computer program was developed to help manually segment the data; a screen capture of this program is shown on Fig. 6. Segmentation was done by visually defining polygons that covered single types of crops.

With this software a total of 72 polygons were defined over the image; each of them belonging to one and only one crop class. Polygon counts for each class were roughly the same, although each polygon had different areas. The classes of these

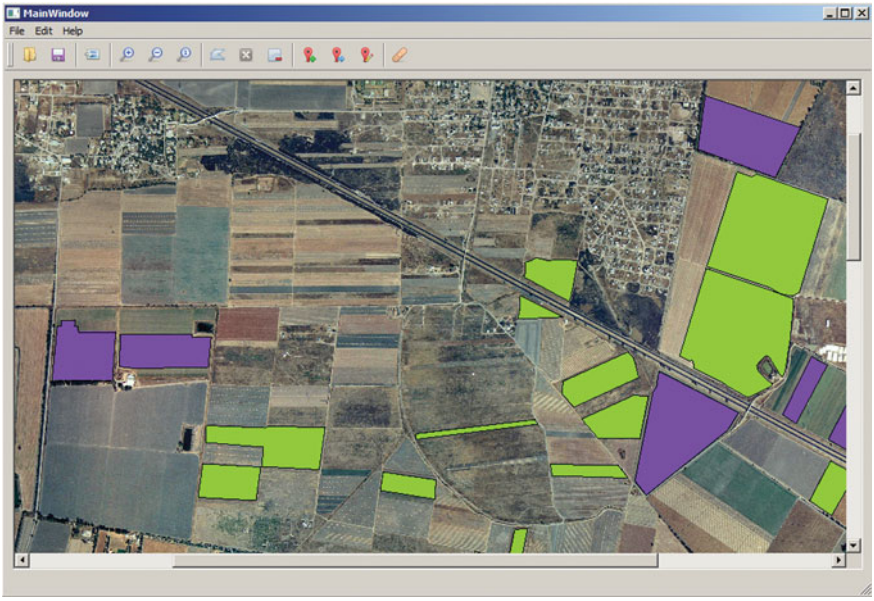

Fig. 6 Image used for crop classification problem

polygons are identified by crop 1 and crop 2. This can be done because given the resolution of the image it is considerable easy to visually identify which fields belong to the same crop. And this helps to generalize the results for any crop besides the ones that are being tested.

With all the polygons identified the next step was to obtain their characteristics using GLCM and CAF over each color band. For the co-occurrence matrix a window sizes of 50×50 pixels were selected in combination with a neighborhood relationship of (1, 1). The reasons for setting the window size was because via experimentation it was found that windows smaller than 5 squared pixels did not offer enough data to characterize a sample, and windows bigger than 50×50 pixels made the number of patterns recovered from the polygons to be very small. After applying the feature extraction technique over the polygons using a window of 50×50 pixels, we built a dataset composed of 620 patterns with 24 features per pattern.

The same window size was used for the CAF. But before obtaining this descriptor three image preprocessing steps were performed. The first was a histogram correction, making it wider so theres a little bit more of contrast; the second one was to apply a sigmoid function to the histogram, which made bright colors brighter and dark colors darker; and finally a Gaussian blur filter was applied with a kernel of 25 pixels, which results in an effect similar to that obtained when applied a Wavelet transformation and keeping the approximation coefficients. After applying the feature extraction technique over the polygons using a window of 50×50 pixels, we built a dataset composed of 620 patterns with 4 features per pattern.

The parameters for the Izhikevich neuron were set as $C = 100$, $v_r = -60$, $v_t = -40$, $v_{peak} = 35$, $k = 0.7$, $a = 0.03$, $b = -2$, $c = -50$, and $d = 100$. The Euler method was used to solve the differential equation of the model with $dt = 1$. The number of terms used to compute input current I from the input pattern was set to 5 with a duration of $T = 1000$. For the case of the cuckoo search algorithm, $N = 40$, $MAXGEN = 1000$, $p_a = 0.25$.

For the case of the RBFNN, we used an hybrid topology composed of three layers. The first layer performs a mean centering and is composed of the same amount of neurons as the size input pattern, second layer is composed of two RBF neurons and finally the output layer is composed of one logarithmic tangential neuron. The centers were initialized using k-means and the standard deviation was set to 0.5. The learning rate parameters for adjusting synaptic weights, centers and standard deviations were set to 0.1, 0.001 and 0.0001, respectively. The RBFNN was trained during 5000 iterations.

The classification rate of the spiking model as well as the bayes classifier and RBFNN was computed as the number of input patterns correctly classified divided by the total number of tested input patterns. Thirty experiments over each dataset were performed to validate the accuracy of the proposed method when the equation for computing the input current that stimulates the spiking neuron is generated with the cuckcoo search algorithm. It is important to notice that, for every experiment two subsets were randomly generated from each dataset. The 50 % of the samples compose the training subset, and the remain the testing subset.

Table 3 present a comparisson of the different classification methods used to perform a crop classification.

After these experiment, it is observed that Co-Occurrence Matrix on windows of 50 pixels, combined with any classifier gives us the best classification percentages. This leads us to believe that although there is important color information contained in the crop windows, it is even more discriminatory the textural information. Concerning to the results obtained with the proposed methodology, we observed that one spiking neuron provides similar results compared with the results obtained with the bayes classifier and better results compared with the results obtained with a RBF neuron.

In this case the equation, designed with the proposed methodology to compute the input current, that best result provides was defined as:

$$I = -[(x_1^{67.40})]w_2 - [(x_1^{-55.38}x_2^{15.20}x_3^{8.02})]w_3 + [(x_2^{32.63}x_3^{-2.59})]w_4 + [(x_2^{-10.47})]w_5$$

Table 3 Average accuracy obtained with the classification method for the crop classification problem

Dataset	Bayes Classifier		RBFNN classifier		Proposed method using CS	
	Tr. cr.	Te. cr.	Tr. cr.	Te. cr.	Tr. cr.	Te. cr.
CAF	1	1	0.8522	0.8341	1	0.9984
GLCM	1	0.9986	1	1	0.9980	0.9974

Tr. cr. = Training classification rate, Te. cr. = Testing classification rate

when the CAF feature extraction method was used, and

$$
\begin{aligned}
I = &+ [(x_6^{-4.9} x_7^{3.8} x_8^{2.6} x_{10}^{2.6} x_{12}^{-0.4} x_{13}^{1.1} x_{14}^{3.9} x_{15}^{-3.6} x_{16}^{-3.8} x_{17}^{4.1} x_{18}^{-2} x_{20}^{2.3} x_{22}^{-2.7} x_{24}^{4.3})] w_1 \\
&- [(x_9^{1.4} x_{11}^{0.9} x_{15}^{1.4} x_{16}^{-5} x_{19}^{3.2} x_{20}^{-3.4} x_{21}^{0.9} x_{23}^{5})] w_2 \\
&- [(x_3^{-5} x_4^{0.3} x_5^{4.7} x_6^{1.7} x_7^{1.09} x_8^{1.1} x_9^{5} x_{12}^{1.09} x_{13}^{4.2} x_{14}^{2.1} x_{15}^{-0.6} x_{16}^{0.7} x_{24}^{-4.1})] w_3 \\
&+ [(x_2^{0.8} x_4^{-1.4} x_7^{-4.6} x_{10}^{4.9} x_{11}^{1.2} x_{12}^{4.9} x_{14}^{4.7} x_{15}^{-3.2} x_{16}^{-1.1} x_{17}^{3.7} x_{18}^{2.1} x_{21}^{3.2} x_{22}^{-4.8} x_{23}^{3.6} x_{24}^{1.1})] w_4 \\
&+ [(x_1^{-1.4} x_2^{1.3} x_3^{-0.3} x_4^{4.7} x_5^{4.9} x_{13}^{-3.8} x_{15}^{0.1} x_{16}^{-0.8} x_{17}^{-4.8} x_{19}^{0.2} x_{20}^{-3.5} x_{21}^{1.3} x_{23}^{-3})] w_5
\end{aligned}
$$

when the GLCM feature extraction method was used.

These results support that the proposed approach can be consider as a pattern recognition method useful in a wide range of applications.

5.3 Application of the Proposed Methodology in a Odor Recognition Problem

In this subsection, we present some preliminar results obtained with the proposed methodology in a problem related to odor recognition. Several works have been developed in the line of biological odor system emulation in different applications such as: edible oil classification, industrial process control, fragrances and cosmetics development, narcotics detection, explosives detection, biometrical identification systems, environment care, medical diagnosis, etc. [5, 33].

The sensors role is crucial for the proper system performance, sensitivity relays on sensors characteristics, various ideas have been proposed to improve sensing capability, like sensor precise functionality analysis, test conditions, and the combination of different sensor types [4, 50].

Another key factor for proper odor detection and classification is the pattern recognition algorithm, in this area several research projects have been developed using neural networks, bionic networks and biological inspired models among others [8, 10, 14, 40].

Although, the embedded systems group at our university has started to design and built an electronic noise prototype, considering sensors capable to obtain relevant data from odor samples, data adjustment and interpretation performed by a microcontroller firmware, and simple and efficient recognition system, we prefered to use a standard dataset to test the accuracy of the proposed methodology.

The dataset used was building in [8] where the authors developed an automated gas delivery experimental setup for extracting volatile compounds at given concentrations from liquids, composed of two pumps, two mass flow controllers (MFCs), one bubbler, a gas chamber and a data acquisition system. This dataset contains patterns obtained from Ethanol or butanol vapors injected into the gas chamber at a flow rate determined by the mass flow controllers. The authors also used sensor arrays composed of five commercial TGS Figaro gas sensors (TGS 2600, 2602, 2610, 2611 and 2620). The potential differences across the sensor resistances were measured using a voltage divider with 2.2 k load resistors while keeping the heating voltage constant to 5 V. Finally, the sensors output voltages were sampled at a rate of 10 Hz and quantized with an 11 bit analog to digital converter to buildt a dataset composed of 124 patterns with 5 features each pattern.

As equal as in previous subsection, the parameters for the Izhikevich neuron were set as $C = 100$, $v_r = -60$, $v_t = -40$, $v_{peak} = 35$, $k = 0.7$, $a = 0.03$, $b = -2$, $c = -50$, and $d = 100$. The Euler method was used to solve the differential equation of the model with $dt = 1$. The number of terms used to compute input current I from the input pattern was set to 5 with a duration of $T = 1000$. For the case of the cuckoo search algorithm, $N = 40$, $MAXGEN = 1000$, $p_a = 0.25$.

For the case of the RBFNN, we used an hybrid topology composed of three layers. The first layer performs a mean centering an is composed of five neurons, seconf layer is composed of two RBF neurons and finally the output layer is composed of one logarithmic tangential neuron. The centers were initialized using k-means and the standard deviation was set to 0.5. The learning rate parameters for adjusting synaptic weights, centers and standard deviations were set to 0.01, 0.0001 and 0.00001, respectively. The RBFNN was trained during 10000 iterations.

The classification rate of the spiking model as well as the RBFNN was computed as the number of input patterns correctly classified divided by the total number of tested input patterns. Ten experiments over each dataset were performed to validate the accuracy of the proposed method when the equation for computing the input current that stimulates the spiking neuron is generated with the cuckcoo search algorithm. It is important to notice that, for every experiment two subsets were randomly generated from each dataset. The 80 % of the samples compose the training subset, and the remain the testing subset.

Table 4 present a comparisson of the proposed methodology against a RBFNN applied to an odor recognition problem. According to the results presented in Table 4, the reader can observe that the accuracy of the spiking neuron is slighly better than the accuracy obtained with the RBFNN. Furthermore, we had to perform several experiments to determine the best values for the RBFNN parameters. It is important to remark that, for the case of the spiking neuron and the proposed methodology, all experiments presented in this chapter were done using the same set of parameters.

Table 4 Average accuracy obtained with the proposed methodology and a RBFNN applied to an odor recognition problem

Dataset	RBFNN classifier		Proposed method using CS	
	Tr. cr.	Te. cr.	Tr. cr.	Te. cr.
Odor dataset	0.9647	0.9309	0.9627	0.9472

Tr. cr. = Training classification rate, Te. cr. = Testing classification rate

In this case the equation, designed with the proposed methodology to compute the input current, that best result provides was defined as:

$$I = +[(x_1^{15.15}x_4^{15.53}x_5^{-59.83})]w_1 - [(x_1^{-10.08}x_4^{-24.31})]w_2 - [(x_1^{23.56}x_2^{-78.41}x_4^{-16.66})]w_3 + [(x_2^{7.46}x_3^{30.91}x_4^{34.35}x_5^{28.13})]w_4 + [(x_2^{-23.37}x_3^{33.97}x_5^{-105.69})]w_5$$

These results obtained with the proposed methodology confirm that spiking neurons can be consider as a pattern recognition technique useful in a wide range of applications, including odor recognition.

6 Conclusions

In this paper, we described how the cuckcoo search algorithm can be applied to design an equation to compute the input current that stimulates a spiking neural network. The experimental results obtained with the spiking neuron model suggest that cuckoo search algorithm is a good alternative to adjust the synaptic weights of the neuron and design the equation to compute the input current. Through the experimental resuls we observed that the spiking neuron combined with this approach, provides acceptable results during the solution of pattern recognition problems.

In general, the behavior achieved with the spiking neuron satisficed that patterns belonging to the same class generate almost the same firing rate in the output of the spiking neuron, and patterns belonging to different classes generate firing rates different enough to discriminate among the different classes. Thank to the cuckoo search algorithm used during the evolutionary learning process, the spiking model behaves like that.

We also observed that the spiking neuron did not behave as good as we desired with the liver dataset and crop classification problem. Nonetheless, the percentage of recognition with the iris, wine, object recognition, diabetes, glass datasets and the ordor recognition problem was highly acceptable.

In addition, a comparison between the proposed methodology and three bio-inspire algoritmhs (cuckoo search, particle swarm optimization and differential evolution) was performed. From all set of experimets we observe that the results obtained with the proposed methodology, evolving the equation to compute the

input current, were slighly better than those obtained with the method described in [39, 41, 42, 44, 45]. Although with some datasets the proposed methodology provides better results than other algorithms, we could not say that one learning strategy is better than the other.

Concerning to the design of the equation to compute the input current, we could remark some advantages: according to the number of times that the features contribute to the equation we could figured out how relevant they are for solving the problem; furthermore, the exponent associated to each feature could indicate how much it contributes for solving the problem. This information could help us to perform a reduction of the dimentionality of the dataset. However, designing the equation to compute the input current do not really help to improve the accuracy of a spiking model applied in a pattern recognition task.

Nowadays, we are exploring the way to develop automatically new spiking neuron models using bio-inspired algorithms. Despite the good results achieved with this methodology, we are developing a new methodology to design netwoks of spiking neurons, evolving at the same time the synaptic weights, topology, and the spiking neuron model.

Acknowledgments The authors would like to thank CONACYT-INEGI and Universidad La Salle for the economical support under grant number 187637 and I-061/12, respectively.

References

1. A chaos synchronization-based dynamic vision model for image segmentation. **5**, (2005)
2. Exploration of rank order coding with spiking neural networks for speech recognition. **4**, (2005)
3. Abdull Hamed, H.N., Kasabov, N., Michlovský, Z., and Shamsuddin, S.M.: String pattern recognition using evolving spiking neural networks and quantum inspired particle swarm optimization. In: Proceedings of the 16th International Conference on Neural Information Processing. Part II, ICONIP '09, LNCS, pp. 611–619. Springer-Verlag, Heidelberg (2009)
4. Balasubramanian, S., Panigrahi, S., Logue, C., Gu, H., Marchello, M.: Neural networks-integrated metal oxide-based artificial olfactory system for meat spoilage identification. J. Food Eng. **91**(1), 91–98 (2009)
5. Baldwin, E.A., Bai, J., Plotto, A., Dea, S.: Electronic noses and tongues. Applications for the food and pharmaceutical industries. Sensors **11**(5), 4744–4766 (2011)
6. Barthelemy, P., Bertolotti, J., Wiersma, D.S.: A levy flight for light. Nature **453**, 495–498 (2008)
7. Belatreche, A., Maguire, L.P., and McGinnity, T.M.: Pattern recognition with spiking neural networks and dynamic synapse. In: International FLINS Conference on Applied Computational Intelligence, pp. 205–210 (2004)
8. Bermak, A., Martinez, D.: A compact 3d vlsi classifier using bagging threshold network ensembles. IEEE Trans. Neural Netw., **14**(5), 1097–1109 (2003)
9. Bohte, S.M., Kok, J.N., Poutre, H.L.: Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing **48**(1–4), 17–37 (2002)
10. Chen, H.T., Ng, K.T., Bermak, A., Law, M., Martinez, D.: Spike latency coding in biologically inspired microelectronic nose. IEEE Trans. Biomed. Circ. Syst., **5**(2), 160–168 (2011)
11. Devroye, L., Györfi, L., Lugosi, G.: A probabilistic theory of pattern recognition. Springer, Heidelberg (1996)

12. Di Paolo, E.: Spike-timing dependent plasticity for evolved robots. Adapt. Behav., **10**, 243–263 (2002)
13. Floreano, D., Zufferey, J.-C., Nicoud, J.-D.: From wheels to wings with evolutionary spiking neurons. Artif. Life, **11**(1–2), 121–138 (2005)
14. Fu, J., Li, G., Qin, Y., Freeman, W.J.: A pattern recognition method for electronic noses based on an olfactory neural network. Sens. Actuators B: Chemical, **125**(2), 489–497 (2007)
15. Garro, B., Sossa, H., and Vazquez, R.: Design of artificial neural networks using a modified particle swarm optimization algorithm. In: International Joint Conference on Neural Networks. IJCNN, pp. 938–945 (2009)
16. Garro, B.A., Sossa, H., and Vázquez, R.A.: Design of artificial neural networks using differential evolution algorithm. In: Proceedings of the 17th International Conference on Neural Information Processing: models and applications, Vol. Part II, pp. 201–208, ICONIP'10, LNCS, Springer-Verlag, Heidelberg (2010)
17. Gerstner, W., Kistler, W.M.: Spiking Neuron Models. Single Neurons, Populations. Cambridge University Press, Plasticity (2002)
18. Gomez-Chova, L., Calpe, J., Camps-Valls, G., Martin, J., Soria, E., Vila, J., Alonso-Chorda, L., Moreno, J.: Feature selection of hyperspectral data through local correlation and sffs for crop classification. In: Geoscience and Remote Sensing Symposium. IGARSS '03. Proceedings. IEEE, International, vol. 1, pp. 555–557 (2003)
19. Haralick, R., Shanmugam, K., and Dinstein, I.: Textural features for image classification. IEEE Trans. Syst. Man Cybern., SMC, **3**(6): 610–621 (1973)
20. Hasselmo, M.E., Bodelón, C., Wyble, B.P.: A proposed function for hippocampal theta rhythm: separate phases of encoding and retrieval enhance reversal of prior learning. Neural Comput. **14**, 793–817, April 2002
21. Hopfield, J.J., Brody, C.D.: What is a moment Cortical sensory integration over a brief interval. Proc. Natl. Acad. Sci., vol. 97 (25), 13919–13924, Dec 2000
22. Hu, M.-K.: Visual pattern recognition by moment invariants. IEEE Trans. Inform. Theory, **8**(2), 179–187, Feb 1962
23. Izhikevich, E.M.: Simple model of spiking neurons. IEEE Trans. Neural Netw., **14**(6), 1569–1572, Nov 2003
24. Izhikevich, E.M.: Which model to use for cortical spiking neurons? IEEE Trans. Neural Netw., **15**(5), 1063–1070, Sept 2004
25. Izhikevich, E.M.: Dynamical systems in neuroscience: the geometry of excitability and bursting. MIT Press, Computational Neurosci. (2007)
26. Jain, R., and Schunck, R.K.B.G.: Machine Vision. McGraw-Hill, New York (1995)
27. Karaboga, D., Akay, B., and Ozturk, C.: Artificial bee colony (abc) optimization algorithm for training feed-forward neural networks. In: Proceedings of the 4th International Conference on Modeling Decisions for Artificial Intelligence, MDAI '07, LNCS, pp. 318–329, Springer-Verlag, Heidelberg (2007)
28. Karaboga, D., and Basturk, B.: Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems. In: IFSA (1)'07, LNCS, pp. 789–798 (2007)
29. Kennedy, J., and Eberhart, R.: Particle swarm optimization. In: Proceedings. IEEE Int. Conf. Neural Netw. (1995) vol. 4, pp. 1942–1948, Aug 2002
30. Maass, W., Graz, T.U.: Networks of spiking neurons: the third generation of neural network models. Neural Netw. **10**, 1659–1671 (1997)
31. Murphy, P., Aha, D.: UCI Repository of machine learning databases. Technical report, University of California, Department of Information and Computer Science, Irvine, CA, USA (1994)
32. Nagy, G., Tolaba, J.: Nonsupervised crop classification through airborne multispectral observations. IBM J. Res. Dev. **16**(2), 138–153 (1972)
33. Oh, E.H., Song, H.S., Park, T.H.: Recent advances in electronic and bioelectronic noses and their biomedical applications. Enzym. Microb. Tech. **48**(67), 427–437 (2011)
34. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Syst. Man Cybern. **9**(1), 62–66, Jan 1979

35. Price, K.V., Storn, R.M., Lampinen, J.A.: Differential evolution a practical approach to global optimization. Nat. Comput. Ser. Springer-Verlag, Berlin (2005)
36. Schwenker, F., Kestler, H.A., Palm, G.: Three learning phases for radial-basis-function networks. Neural Netw. **14**(45), 439–458 (2001)
37. Senthilnath, J., Omkar, S.N., Mani, V., and Karnwal, N.: Hierarchical artificial immune system for crop stage classification. In: Annuals IEEE India Conference (INDICON), pp. 1–4 (2011)
38. Thorpe, S.J., Guyonneau, R., Guilbaud, N., Allegraud, J.-M., and VanRullen R.: Spikenet: real-time visual processing with one spike per neuron. Neurocomputing, 58–60:857–864 (2004). (Comput. Neurosci.: Trends Res. 2004)
39. Vazquez, R.: Izhikevich neuron model and its application in pattern recognition. Aust. J. Intell. Inf. Process. Syst. **11**(1), 35–40 (2010)
40. Vazquez, R.: A computational approach for modeling the biological olfactory system during an odor discrimination task using spiking neuron. BMC Neurosci. **12**(Suppl 1), p. 360 (2011)
41. Vazquez, R.: Training spiking neural models using cuckoo search algorithm. In: IEEE Congress on Evolutionary Computation (CEC), pp. 679–686 (2011)
42. Vazquez, R., and Cacho andn, A.: Integrate and fire neurons and their application in pattern recognition. In: 7th International Conference on Electrical Engineering Computing Science Automatic Control (CCE), pp. 424–428 (2010)
43. Vazquez, R., Sossa, H., Garro, B.: 3d object recognition based on some aspects of the infant vision system and associative memory. Cognitive Comput. **2**, 86–96 (2010)
44. Vázquez R.A.: Pattern recognition using spiking neurons and firing rates. In: Proceedings of the 12th Ibero-American Conference on Advances Artificial Intelligence, IBERAMIA'10, LNAI, pp. 423–432, Springer-Verlag, Heidelberg (2010)
45. Vázquez, R.A., and Garro, B.A.: Training spiking neurons by means of particle swarm optimization. In: Proceedings of the Second International Conference on Advances in Swarm Intelligence, vol. Part I, ICSI'11, pp. 242–249. Springer-Verlag, Heidelberg (2011)
46. Vazquez Espinoza De Los Monteros, R.A., and Sossa Azuela, J.H.: A new associative model with dynamical synapses. Neural Process. Lett., **28**:189–207, Dec 2008
47. Viswanathan, G.M., Buldyrev, S.V., Havlin, S., da Luz, M.G.E., Raposo, E.P., Stanley, H.E.: Optimizing the success of random searches. Nature **401**, 911–914 (1999)
48. Yang, X.-S.: 1-optimization and metaheuristic algorithms in engineering. Metaheuristics in water, geotechnical and transport engineering, pp. 1–23. Elsevier, Oxford (2013)
49. Yang, X.-S., and Deb, S.: Cuckoo search via levy flights. In: World Congress on Nature Biologically Inspired Computing, NaBIC, pp. 210–214 (2009)
50. Yin, Y., Yu, H., Zhang, H.: A feature extraction method based on wavelet packet analysis for discrimination of chinese vinegars using a gas sensors array. Sens. Actuators B: Chemical **134**(2), 1005–1009 (2008)

Multi-Objective Optimization of a Real-World Manufacturing Process Using Cuckoo Search

Anna Syberfeldt

Abstract This chapter describes the application of Cuckoo Search in simulation-based optimization of a real-world manufacturing process. The optimization problem is a combinatorial problem of setting 56 unique decision variables in a way that maximizes utilization of machines and at the same time minimizes tied-up capital. As in most real-world problems, the two optimization objectives are conflicting and improving performance on one of them deteriorates performance of the other. To handle the conflicting objectives, the original Cuckoo Search algorithm is extended based on the concepts of multi-objective Pareto-optimization.

Keywords Cuckoo search · Simulation-based optimization · Multi-objective problem

1 Introduction

Discrete-event simulation combined with optimization, so called simulation-based optimization, is a powerful method to improve real-world systems [1]. While traditional, analytical optimization methods have been unable to cope with the challenges imposed by many simulation-based optimization problems in an efficient way, such as multimodality, non-separability and high dimensionality, so called metaheuristic algorithms have been shown to be applicable to this type of problem [2][3]. Metaheuristic algorithms are powerful stochastic search algorithms with mechanisms inspired from natural science. A metaheuristic algorithm optimizes a problem by iteratively improving one or several candidate solution(s) with regard to a given objective function. The algorithms are not guaranteed to find the optimal solution for the given problem, but are instead computationally fast and make no explicit assumptions about the underlying structure of the function to be optimized. These

A. Syberfeldt (✉)
University of Skövde, PO. 408, SE-54148 Skövde, Sweden
e-mail: anna.syberfeldt@his.se

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_9,

properties make them well suited for simulation-based optimization as the simulation is often time consuming and can be seen as a black-box [4].

There exist plenty of metaheuristic algorithms, among the most well-known including simulated annealing, tabu search, and genetic algorithms. One of the most recently proposed is called Cuckoo Search, which is inspired by the parasitic breeding behavior of cuckoos [5, 6]. Several studies indicate that Cuckoo Search is a powerful algorithm and successful results have been achieved in various applications such as welded beam design, nurse scheduling and wireless sensor networks [7–9]. A review of the literature, however, reveals no applications in manufacturing optimization and the aim of this study is therefore to investigate the algorithm's performance in this domain.

A real-world problem of engine manufacturing at a Volvo factory in Sweden is focus of the study. In short, the problem is about finding the best prioritization of the different engine components being simultaneously processed in a manufacturing line. Basically, this is a combinatorial problem involving the setting of 56 unique decision variables. The prioritization is used to determine which specific component has precedence when two or more components are available at a machine or station. Based on the priorities, it is possible to create a schedule showing which component should be processed in which machine at each point in time. However, finding an efficient processing schedule is not trivial, due to the considerable complexity of the cell in combination with a fluctuating inflow and resource constraints. This fact has raised a need to perform automatic simulation-optimizations and has motivated an evaluation of different algorithms for this purpose. To perform simulation-optimizations, a discrete-event simulation of the model is constructed by simulation experts at Volvo using the SIMUL8 software. The next section of this chapter presents the optimization in further detail.

2 Real-World Manufacturing Optimization

In Volvo's manufacturing line under study, engine components of eleven different types are being processed. The line includes ten different processing stations that can perform single or multiple operations (including for example burring, welding, and assembly). The operations performed at a specific station and the tools used vary depending on the type of component. The high degree of variety in processing of different components, in combination with a fluctuating inflow and several resource constraints, make it virtually impossible to plan the production manually in an efficient way. Automatic optimizations are instead needed, and for this purpose a discrete-event simulation model of the line has been developed by simulation experts at the company using the SIMUL8 software package.[1] The simulation model has a front-end interface developed in Excel, which facilitates the user in entering input parameters to the model without the need to learn the simulation language. Valid-

[1] www.simul8.com

ity tests indicate that the simulation model represents reality well, and the model is generally accepted among operators working in the line.

The optimization problem to be solved using the simulation model consists of constructing a processing schedule for the manufacturing line by setting 56 unique decision variables. These variables basically represent the order of operations for different components and also which component has precedence in case of queues in front of machines. According to the company, a processing schedule should preferably be configured so that a high utilization of all machines is achieved, as these involve expensive investments. The company has also stated that it is important to reduce tied-up capital by minimizing each component's lead time (that is, the time between a component entering and exiting the line). Altogether there are two objectives that must be considered simultaneously in the optimization process: (a) utilization, and (b) tied-up capital.

For high utilization, a large number of components within the line is needed in order to avoid machine starvation. However, a large number of components imply occasional queues in front of some machines, which results in longer lead-times for components and thereby tied-up capital. In relation to each other, the two objectives of maximal utilization and minimal tied-up capital are therefore conflicting. No single optimal solution with respect to both objectives exists, as improving performance on one objective deteriorates performance of the other objective.

One way to handle conflicting objectives is to derive a set of alternative trade-offs, so called Pareto-optimal solutions [10]. Figure 1 illustrates the Pareto concept for a minimisation problem with two objectives f_1 and f_2. In this example, solution *A*-*D* are non-dominated, i.e. Pareto optimal, since for each of these solutions there exist no other solution that is superior in one objective without being worse in another objective. Solution *E* is dominated by *B* and *C* (but not by *A* or *D*, since *E* is better than these two in f_1 and f_2, respectively).

Although it is important to find as many (trade-off) optimal solutions as possible in multi-objective optimisation, the user needs only one solution regardless of the number of objectives [10]. Which of the optimal solutions to choose is up to the user to decide based on previous experiences and qualitative information (for example, ergonomic conditions or set-up of factory workers for the day).

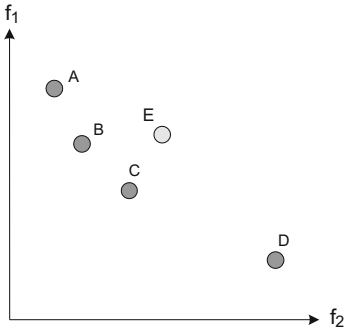

Fig. 1 Illustration of dominance

3 Cuckoo Search

This section presents the optimization of the manufacturing line using the Cuckoo Search algorithm.

3.1 Algorithm Description

In nature, cuckoos lay eggs in other birds' nests and rely on those other birds to rear the cuckoo's offspring. According to the so called "selfish gene theory" proposed by Dawkins in 1989 [11], this parasitic behavior increases the chance of survival of the cuckoo's genes since the cuckoo needs not expend any energy rearing its offspring. Instead, the cuckoos can spend more time on breeding and thereby increasing the population. However, the birds whose nests are invaded have developed counter strategies and increasingly sophisticated ways of detecting the invading eggs.

These behaviors found in nature are utilized in the Cuckoo Search algorithm in order to traverse the search space and find optimal solutions. A set of nests with one "egg" (candidate solution) inside are placed at random locations in the search space. A number of "cuckoos" traverse the search space and records the highest objective values for different encountered candidate solutions. The cuckoos utilize a search pattern called Lévy flight which is encountered in real birds, insects, grazing animals and fish according to Viswanathan et al. [12]. The Lévy flight is characterized by a variable step size punctuated by 90-degree turns, as can be seen in Fig. 2. The large steps occasionally taken make the algorithm suitable for global search. Lévy flights, according to Yang [6] and Gutowski [13], are more efficient for searching than regular random walks or Brownian motions.

Lévy flights are used in the Cuckoo Search algorithm to globally explore the search space, while local random walks are used for exploitation. A switching parameter p_a is used to balance between exploration and exploitation. Eq. 1 describes how the

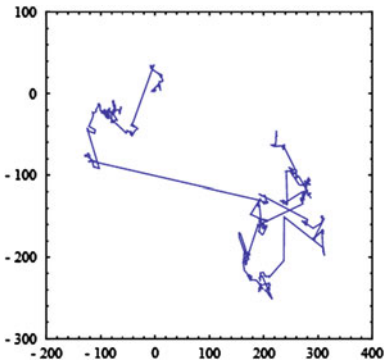

Fig. 2 Example of Lévy flight starting at [0,0]

local walks are performed.

$$x_i^{(t+1)} = x_i^{(t)} + \alpha s \oplus H(p_a - \varepsilon) \oplus \left(x_j^{(t)} - x_k^{(t)}\right) \tag{1}$$

In the equation, x_j and x_k are two different solutions randomly selected, is H a Heaviside function, is a random uniform number, and is the step size. The global random search using the concept of Lévy flights are described in Eq. 2.

$$x_i^{(t+1)} = x_i^{(t)} + \alpha L\acute{e}vy(s, \lambda) \tag{2}$$

In the equation, $\alpha > 0$ represents the step size scaling factor, and this must be fine-tuned for the specific problem at hand. Yang [6] advocates the use of α as $\alpha = O(L/100)$, where L represents the difference between the maximum and minimum valid value of the given problem. In order to implement the Lévy flight, a fast algorithm needed to be used to approximate the Lévy distribution. Leccardi [14] compared three different approaches to generating Lévy distributed values and found that the algorithm published in [14] proved to be the most efficient. Mantegna's algorithm is divided into three steps in order to generate the step length, in Eq. 3 needed for the Lévy flight.

$$s = \frac{u}{|\nu|^{\frac{1}{\beta}}} \tag{3}$$

The parameters u and ν are given by the normal distributions in Eq. 4.

$$u = N(0, \sigma_u^2), \qquad \nu = N(0, \sigma_\nu^2) \tag{4}$$

The variance, σ, is calculated as Eq. 5 with $1 \leq \beta \leq 2$ and where $\Gamma(z)$ is the gamma function.

$$\sigma_u = \left\{ \frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\Gamma[(1+\beta)/2]\beta 2^{(\beta-1)/2}} \right\}^{\frac{1}{\beta}}, \quad \sigma_\nu = 1 \tag{5}$$

Cuckoo Search is a population based, elitist, single-objective optimization algorithm. The pseudo code for the algorithm is presented in Fig. 3 [5]. Note that only two parameters need to be supplied to the algorithm; the discovery rate $p_a \in [0, 1]$ and the size of the population, n. When n is fixed, p_a controls the elitism and the balance of randomization and local search [6]. The fact that the algorithm comes with just two parameters does not only increase the ease of implementation, but also potentially makes it a more general optimization solution to be applied to a wide range of problems.

The real-world problem to be optimized by Cuckoo Search in this study involves the optimization of two objectives. The next section of this chapter describes how the algorithm is adjusted to consider more than one objective.

```
Generate initial population of n host nests
while (t < MaxGeneration or stopCriterionFulfilled)
    Generate a solution by Lévy flights and then evaluate its quality/fitness F_i
    Choose a nest among n (say, j) randomly
    if (F_i > F_j),
        Replace j by the new solution
    end
    Abandon the p_a nests with worst quality and build new ones
    Keep best solutions/nests
    Rank the solutions/nest and find the current best
    Pass the current best to the next generation.
end while
```

Fig. 3 Pseudo code for Cuckoo Search with Lévy flight

3.2 Multi-Objective Extension

In order to convert Cuckoo Search from a single-objective optimization algorithm into multi-objective optimization algorithm, concepts from an existing Pareto-based algorithm is used, namely elitist non-dominated sorting genetic algorithm (NSGA-II). NSGA-II is a widely used multi-objective algorithm that has been recognized for its great performance and is often used for benchmarking [16]. The pseudo-code for NSGA-II can be seen in Fig. 4 and for details about the algorithm the reader is referred to [17].

In the NSGA-II algorithm, the selection of solutions is done from a set R, which is the union of a parent population and an offspring population (both of size N) [18]. Non-dominated sorting is applied to R and the next generation of the population is formed by selecting solutions from one of the fronts at a time. The selection starts with solutions in the best Pareto front, then continues with solutions in the second best front, and so on, until N solutions have been selected. If there are more solutions in the last front than there are remaining to be selected, niching is applied to determine which solutions should be chosen. In other words, the highest ranked solutions located in the least crowded areas are the ones chosen. All the remaining solutions are discarded. The selection procedure is illustrated in Fig. 5 (adopted from [10]).

The concept of non-dominated sorting utilized in the NSGA-II algorithm has been used to create a multi-objective extension of Cuckoo Search (a similar approach is presented in [19]). Besides the non-dominated sorting procedure, other differences between NSGA-II and the new version of the Cuckoo Search algorithm include the use Lévy flights instead of mutation and the abandonment and movement of nests instead of crossover. The pseudo code for the multi-objective version of Cuckoo Search is presented in Fig. 6.

The next section of this chapter presents results from applying the multi-objective version of Cuckoo Search on the real-world manufacturing problem.

```
Initialize Population
Generate N random solutions and insert into Population
for (i = 1 to MaxGenerations) do
    Generate ChildPopulation of size N
    Select Parents from Population
    Create Children from Parents
    Mutate Children
    Combine Population and ChildPopulations into CurrentPopulation with size
     2N
    for each individual in CurrentPopulation do
        Assign rank based on Pareto – Fast non-dominated sort
    end for
    Generate sets of non-dominated vectors along PF known
    Loop (inside) by adding solutions to next generation of Population starting
     from the best front
        until N solutions found and determine crowding distance between
         points on each front
end for
Present results
```

Fig. 4 Pseudo code for NSGA-II

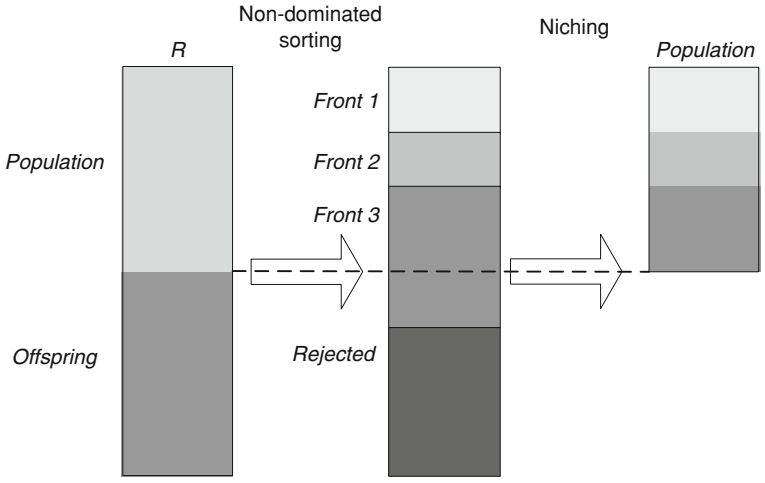

Fig. 5 Non-dominated sorting

```
Initialize Population
Generate N random solutions and insert into Population
for (i = 1 to MaxGenerations) do
    Generate ChildPopulation using Cuckoo Search (pseudo code in Fig. 3)
    Non-dominated-sort
    for each individual in CurrentPopulation do
        Generate sets of non-dominated vectors along PF_known
        Loop (inside) by adding solutions to next generation starting from the
        best front
            until N solutions found and determine crowding distance between
              points on each front
    end for
end for
Present results
```

Fig. 6 Pseudo code for the proposed multi-objective Cuckoo Search

4 Evaluation

4.1 Configuration

As previously mentioned in Sect. 2 of this chapter, the real-world manufacturing problem is basically a combinatorial problem with 56 unique decision variables representing a processing schedule. The proposed multi-objective extension of Cuckoo Search is applied on the problem with the maximum number of simulation evaluations set to 1000 (according to the time budget stated by the company). The population size is set to 20, and the percentage of abandoned nests to 40% (the values have been found by trial-and-error tests). The algorithm uses a swap operator utilizing the Lévy flights to determine the indices to swap. The step size for the Lévy flight is set to $\frac{upperbound}{100}$, where upperbound is the maximum permitted value of each parameter.

For comparison, the NSGA-II algorithm is also applied on the problem. NSGA-II is run with the same number of simulation evaluations as Cuckoo Search and implemented with the same population size. NSGA-II uses swap range mutation with a mutation probability of 10%. Furthermore, the algorithm is implemented with a partially mapped crossover operator and a crossover probability of 90%.

As baseline for the comparison between Cuckoo Search and NSGA-II, a basic scheduling function defined by the company is being used. In this function, a Critical Ratio (CR) value of each component is calculated to determine how well it is on schedule. The CR value is derived by dividing the time to due date (i.e. scheduled completion) by the time expected to finish the component, according to Eq. 6.

$$CR = \begin{cases} due \geq now : \frac{1+due-now}{1+TRPT} \\ due < now : \frac{1}{(1+now-due)*(1+TRPT)} \end{cases} \quad (6)$$

In this equation, *due* is the due date of the component (i.e. deadline), *now* is the current time, and *TRPT* is the theoretical total remaining processing time (the active operation time including set-up times and movements between machines/stations). A component with a CR value of 1.0 is "according to schedule", while it is behind if the value is less than 1.0 and ahead if the value is larger than 1.0. In case of a race condition, the component with the lowest CR value has precedence.

4.2 Integrating the Optimization Algorithm and the Simulation

To run the algorithms along with the SIMUL8 simulation model, a component called "simulation controller" is implemented. The simulation controller, whose purpose is to start the simulation and collect results, is implemented using the application programming interface (API) provided by SIMUL8. This API enables the simulation model to be controlled programmatically. When the simulation controller receives a request (i.e., a set of priority values to be evaluated) from the optimization algorithm, the simulation controller invokes the simulation model with a "Run" command and provides the priority values. When the simulation has been completed (after about two seconds), the simulation controller collects the results and sends them back to the optimization algorithm.

4.3 User Interface

The user interacts with the simulation-optimization through a web page displayed in a web browser. Using a web page as user interface enables access to the platform from any hardware (e.g., PC, mobile phone, tablet, etc.) at any site, as long as a web browser and an internet connection are available. This is especially advantageous in industrial environments, since employees often have limited (or no) possibilities of installing, running, and maintaining software at their work stations.

The content of the web page was developed during an iterative process undertaken in close cooperation between the company and the University's software developers. A screenshot of one part of the resulting web page is presented in Fig. 7 (for integrity reasons, company specific information was deleted before the screenshot was taken). The panel to the left in the screenshot constitutes the menu from which the user accesses various functions in the platform. In the screenshot, the menu alternative "RESULTS" have been chosen. This alternative shows the output from a simulation-optimization run, namely, a production schedule derived from a prioritization of all components to be processed during a specific time period. The aim of the production schedule is to support the operators of the manufacturing cell by specifying which component should be processed in which machine at each point in time.

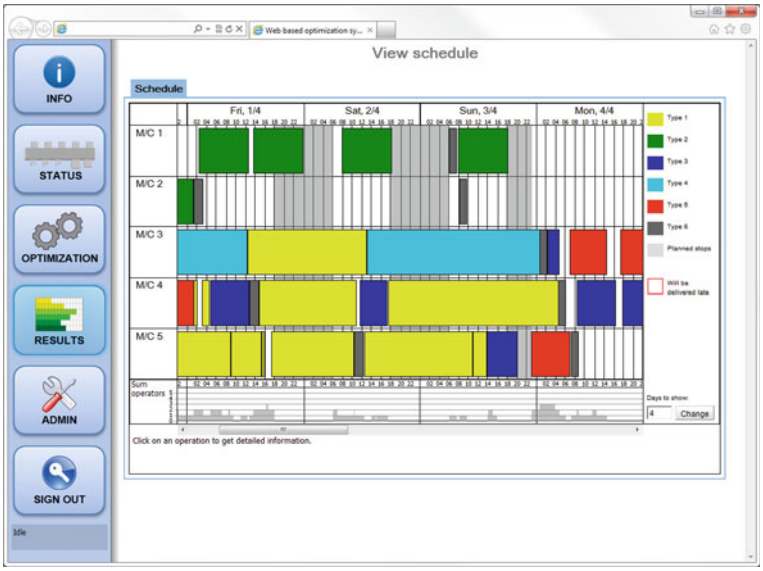

Fig. 7 Screenshot of user interface

4.4 Results

The optimization results achieved by Cuckoo Search and NSGA-II are presented in Table 1 and stated in relation to the CR function. More specifically, the percentage numbers given in the table represent the relative improvement achieved by each algorithm when compared to the result of the CR function. No real numbers can be presented due to company restrictions.

As shown in Table 1, the results clearly indicate that NSGA-II outperforms Cuckoo Search considering both optimization objectives. An analysis of this result is given in the next section of this chapter.

5 Analysis

This section presents an analysis of the results, both from a technical point of view and from a user perspective.

Table 1 Results with CR function as baseline

	Utilization (improvement)	Tied-up capital (improvement)
Cuckoo Search (%)	10	15
NSGA-II (%)	19	23

Average of 20 replications

5.1 Technical Analysis

As mentioned in the introduction, Cuckoo Search has proven to be able to efficiently solve different optimization problems in various domains. Nevertheless the algorithm shows a weak performance on the real-world manufacturing problem of focus for this paper. When going through previous studies on Cuckoo Search and analyzing the nature of the optimization problems solved in these papers, it is clear that continuous problems have been the main target for Cuckoo Search so far. In this study, the problem to be solved is combinatorial and a hypothesis is this fact is the reason behind the results. To investigate the hypothesis, five continuous problems from the well-known ZDT test suit are implemented. A detailed description of the ZDT problems is provided in [20].

Results from applying the multi-objective version of Cuckoo Search on the continuous ZDT problems are presented in Table 3. For comparison, the NSGA-II algorithm has also been applied on the same problems. Two performance measures are being used: convergence and spread (both are to be minimized). These two are commonly used for the ZDT problems and a detailed explanation of their implementation can be found in [10].

As can be seen in Table 2 Cuckoo Search achieves the best results on the ZDT1, ZDT2 and ZDT4 problems. On the ZDT3 problem, it achieves a better convergence than the NSGA-II algorithm, but a worse spread. On the ZDT6 problem, NSGA-II performs the best on both objectives.

The results from the test suit show that Cuckoo Search outperforms NSGA-II on the majority of the ZDT problems, which is probably due to use of the efficient Lévy flight method. These results are in line with results from previous studies found in the literature on applying Cuckoo Search on continuous optimization problems.

To investigate further if Cuckoo Search is better suited for continuous optimization problems, the algorithm is also evaluated on a combinatorial test problem. The problem implemented is the symmetrical traveling salesman problem named berlin52

Table 2 Results from ZDT problems

		Convergence (minimize)	Spread (minimize)
ZDT1	NSGA-II	0.037582	0.613663
	Cuckoo Search	0.002561	0.613206
ZDT2	NSGA-II	0.030753	0.684501
	Cuckoo Search	0.003855	0.649924
ZDT3	NSGA-II	0.001460	0.615773
	Cuckoo Search	0.001149	0.616611
ZDT4	NSGA-II	0.075159	0.700399
	Cuckoo Search	0.000437	0.623540
ZDT6	NSGA-II	0.002848	0.616384
	Cuckoo Search	0.085899	0.675200

Average of 1000 replications

Table 3 Results from combinatorial TSP problem

	Distance (minimize)
Cuckoo Search	21716
NSGA-II	10961

Average of 1000 replications

[21], which was chosen due to the similar number of parameters to the real-world problem. The objective of the problem is to find the shortest route between a number of points, while only going through each point once, and then returning to the starting point. The distance between the points is measured through Euclidean distance. Results from applying Cuckoo Search and NSGA-II on the problem are presented in Table 3. As shown in the table, NSGA-II clearly outperforms the Cuckoo Search algorithm on the combinatorial TSP problem. It should be noted that there exist a specialized Cuckoo Search tailor-made for the travelling salesman problem that has been proven to perform very well [22], but as this version is single-objective it has not been considered in the study.

Although a more extensive evaluation including a larger number of optimization problems are needed for a general statement, the results obtained in this study indicates that Cuckoo Search in the form implemented in this study is suited for continuous problems rather than combinatorial ones. A probable reason for the weak performance on combinatorial problems is that the Lévy flight pattern is not suited to be used as a basis for swap mutation. As previously described, the algorithm uses a swap operator utilizing the Lévy flights to determine the indices to swap.

5.2 User Perspective

The simulation-optimization has also been analyzed by a test group at Volvo Aero. This test group included eight employees who represented all three user groups supported by the platform (operator, shift leader, and manager), as well as logistics engineers at the company. University representatives first demonstrated the simulation-optimization on site at the company, after which it was made available to the test group. They tried the system out for a couple of weeks, during which their feedback was collected and compiled.

After the test period, the evaluation feedback was discussed and analyzed. The feedback highlighted the great potential the simulation-optimization demonstrated with regard to improving processing schedules, especially during periods of heavy workloads. Besides improving the processing schedules, using the platform was also considered a way of significantly reducing the human effort currently associated with creating the schedules. A further advantage raised by the test group was the possibility of easily sharing optimized production schedules in real-time among stakeholders. Since the platform is web-based, it allows all users to easily view

results simultaneously, which stands in contrast to traditional desktop applications where results must be printed or e-mailed.

A negative aspect of the platform raised, more or less, by everyone in the test group was the need to manually specify the current status and configuration of the manufacturing cell before each simulation-optimization. The required data includes shift period, status of components currently under process, list of components entering, availability of fixtures, availability of operators, and scheduled maintenance. The simulation model needs all this data to be able to mimic the operations of the cell as close to reality as possible. Collecting and specifying the data can be time-consuming and should preferably be eliminated so that the optimization can be fully automatic and run frequently, or even constantly. To realize this, the platform can be integrated with the computerized control system of the manufacturing cell. It is essential to strive for such integration, which would probably not be too difficult considering the manufacturing cell has modern, built-in control systems that process the data needed.

6 Conclusions

This paper describes a case study of applying Cuckoo Search, a recently proposed metaheuristic algorithm, in simulation-based optimization of a real-world manufacturing line. The manufacturing line is highly automated and produces engine components of eleven different types. The Cuckoo Search algorithm has previously shown promising results in various problem domains, which motives to evaluate it also on the optimization problem under study in this paper. The optimization problem is a combinatorial problem of setting 56 unique decision variables in a way that maximizes utilization of machines and at the same time minimizes tied-up capital. As in most real-world problems, these two objectives are conflicting and improving performance on one of them deteriorates performance of the other. To handle the conflicting objectives, the original Cuckoo Search algorithm is extended based on the concepts of multi-objective Pareto-optimization. We argue that a multi-objective version of Cuckoo Search is needed not only for this specific study, but for Cuckoo Search to be truly useful in general as most real-world problems involve the optimization of more than one objective.

Optimization of the manufacturing line is performed based on a discrete-event simulation model constructed using the SIMUL8 software. Results from the simulation-based optimization show that the extended Cuckoo Search algorithm is inefficient in comparison with the multi-objective benchmark algorithm NSGA-II. A possible reason might be that the Cuckoo Search algorithm is not suited for combinatorial optimization problems due to that the Lévy flight pattern is not suited to be used as a basis for swap mutation. To investigate this further, the algorithm is applied on five continuous test problems and also one combinatorial test problem. Results from these test problems shows that the Cuckoo Search algorithm outperforms NSGA-II on a majority of the continuous test problems. However, the Cuckoo

Search algorithm performs considerable worse than the NSGA-II algorithm on the combinatorial test problem. An explanation for the weak results on this problem, as well as the real-world problem, might be that the Lévy flight pattern is not suited to be used as a basis for swap mutation. Further evaluations are needed, however, to investigate this circumstance further. It is also recommended to study other possible ways to adapt Cuckoo Search to combinatorial optimization problems possibly suggest an improved version that is able to handle this type of problem effectively.

Another interesting improvement of the Cuckoo Search algorithm would be to consider simulation randomness. To capture the stochastic behavior of most real-world complex systems, simulations contain randomness. Instead of modeling only a deterministic path of how the system evolves in the process of time, a stochastic simulation deals with several possible paths based on random variables in the model. To tackle the problem of randomness of output samples is crucial because the normal path of the algorithm would be severely disturbed if estimates of the objective function come from only one simulation replication. The common technique to handle randomness is to send the algorithm with the average values of output samples obtained from a large number of replications. Although this technique is easy to implement, the large number of replications needed to obtain statistically confident estimates from simulation of a complex system that requires long computation time can easily render the approach to be totally impractical. There are methods that guarantee to choose the "best" among a set of solutions with a certain statistically significance level, which require fewer replications in comparison to the others (e.g. Kim-Nelson ranking and selection method found in [23]). However, combining statistically-meaningful procedures that require relatively light computational burden with metheuristics is still an important topic for further research [24].

Besides technical issues, it is also important to carefully considering user aspects for a successful realization of the optimization. During the Volvo case study, it was clear that the graphical design and layout of the user interface was an important factor in gaining user acceptance of the simulation-optimization. In general, software development more easily focuses on algorithms and technical details, but putting at least as great an effort into the graphical design is recommended. It is especially important to keep usability in mind when developing systems for industrial adoption, since the success of a system is dependent on its acceptance by its users [25].

References

1. April, J., Better, M. Glover, F., Kelly, J.: New advances for marrying simulation and optimization. In: Proceedings of the 2004 Winter Simulation Conference, Washington, DC, pp. 80–86 (2004)
2. Boesel, J., Bowden, R., Glover, F., Kelly, J., Westwig, E.: Future of simulation optimization. In: Proceedings of the 2001 Winter Simulation Conference, Arlington, VA, pp. 1466–1470 (2001)
3. Laguna, M., Marti, R.: Optimization Software Class Libraries. Kluwer Academic Publishers, Boston (2003)

4. Bäck, T., Fogel, D., Michalewicz, Z.: Handbook of Evolutionary Computation. Oxford University Press, Oxford (1997)
5. Yang, X.-S., Deb, S.: Cuckoo Search via Levy Flights. In: Proceedings of World Congress on Nature and Biologically Inspired Computing, India, pp. 210–214 (2009)
6. Yang, X.-S.: Nature-Inspired Metaheuristic Algorithms, 2nd edn. Luniver Press, Beckington (2010)
7. Yang, X.-S., Deb, S.: Engineering optimisation by Cuckoo Search. Int. J. Math. Model. Numer. Optim. **1**(4), 330–343 (2010)
8. Tein, L.H., Ramli, R.: Recent advancements of nurse scheduling models and a potential path. In: Proceedings of 6th IMT-GT Conference on Mathematics, Statistics and its Applications, pp. 395–409 (2010)
9. Dhivya, M., Sundarambal, M., Anand, L.N.: Energy efficient computation of data fusion in wireless sensor networks using Cuckoo based particle approach (CBPA). Int. J. Commun. Netw. Syst. Sci. **4**(4), 249–255 (2011)
10. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms, 2nd edn. Wiley, Chichester (2004)
11. Dawkins, R.: The Selfish Gene, 30th edn. Oxford University Press Inc, New York (1989)
12. Viswanathan, G.M., Buldyrev, S.V., Havlin, S., da Luz, M.G.E., Raposo, E.P., Eugene Stanley, H.: Optimizing the success of random searches. Lett. Nat. **401**, 911–914 (1999)
13. Gutowski, M.: Lévy flights as an underlying mechanism for global optimization algorithms. Optimization **8** (2001)
14. Leccardi, M.: Comparison of three algorithms for Lèvy noise generation. In: Proceedings of Fifth EUROMECH Nonlinear Dynamics Conference, Mini Symposium on Fractional Derivatives and their Applications(2005)
15. Mantegna, R.N.: Fast, accurate algorithm for numerical simulation of Levy stable stochastic processes. Phys. Rev. E **49**, 4677–4683 (1994)
16. Coello Coello, C.A., Lamon, G.B., Van Veldhuizen, D.A.: Evolutionary Algorithms for Solving Multi-objective Problems, 2nd edn. Springer Science Business Media LLC, New York (2007)
17. Deb, K., Pratap, A., Agarwal, S., Meyarivan T.: A fast and Elitist multi-objective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **VI:2**, 182–197 (2002)
18. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multi-objective genetic algorithm NSGA-II. KanGAL Report 2000001, Indian Institute of Technology Kanpur, India (2000)
19. Yang, X.-S., Deb, S.: Multiobjective cuckoo search for design optimization. Comput. Oper. Res. **40**(6), 1616–1624 (2013)
20. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: empirical results. Evol. Comput. **8**(2), 173–195 (2000)
21. Reinelt, G. TSPLIB, 8 August, [Online]. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/(2008)
22. Ouaarab, A., Ahiod, B., Yang, X-S.: Discrete cuckoo search algorithm for the travelling salesman problem. Neural Comput. Appl. (2013)
23. Gosavi, A.: Simulation-based Optimization: Parametric Optimization Techniques and Reinforcement Learning. Kluwer Academic, Boston (2003)
24. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments—a survey. IEEE Trans. Evol. Comput. **9**(3), 303–317 (2005)
25. Rogers, Y., Sharp, H., Preece, J.: Interaction Design: Beyond Human-Computer Interaction, 3rd edn. Wiley, New York (2007)

Solving Reliability Optimization Problems by Cuckoo Search

Ehsan Valian

Abstract A powerful approach to solve engineering optimization problems is the cuckoo search algorithm. It is a developed by Yang and Deb [1, 2]. In this chapter uses CS algorithm, to solve the reliability optimization problem. The reliability optimization problem involves setting reliability objectives for components or subsystems in order to meet the resource consumption constraint, e.g. the total cost. The difficulties facing reliability optimization problem are to maintain feasibility with respect to three nonlinear constraints, namely, cost, weight and volume related constraints. The reliability optimization problems have been studied in the literature for decades, usually using mathematical programming or metaheuristic optimization algorithms. The performance of CS algorithm is tested on five well-known reliability problems and two complex systems. Finally, the results are compared with those given by several well-known methods. Simulation results demonstrate that the optimal solutions obtained by CS, are better than the best solutions obtained by other methods.

Keywords Cuckoo search algorithm · Lévy flight · Reliability optimization problem

1 Introduction

System reliability optimization plays a vital role in real-world applications and has been studied for decades [3–14]. In order to be more competitive in the market, many designers have worked on improving the reliability of manufacturing systems or product components. To attain higher system reliability, two main ways are usually used. The first approach is to increase the reliability of system

E. Valian (✉)
Faculty of Electrical and Computer Engineering,
University of Sistan and Baluchestan, Zahedan, Iran
e-mail: ehsanvalian@gmail.com

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_10,

components. It may improve the system reliability to some degrees; however the required reliability enhancement may be never achievable even though the most currently reliable elements are used. The second approach is to use redundant components in different subsystems. Using this method increased cost, weight and volume. This approach is named reliability redundancy allocation problem (RAP) [15].

Various complex systems come out with the development of industrial engineering. A design engineer often tries to attain the highest reliability for such systems. The reliability problem is subject to several resource constraints such as cost, weight, and volume and can usually be formulated as a nonlinear programming problem.

In order to cope with optimization problems arising in system reliability, important contributions have been made since 1970 [15]. To solve a category of reliability optimization problems with multiple-choice constraints, Chern and Jan [16] developed a 2-phase solution method. They presented a general model that can be stated as the problem of finding the optimum number of redundancies maximizing the system reliability. Prasad and Kuo [15] offered a search method (P and K-Algorithm) based on lexicographic order, and an upper bound on the objective function for solving redundancy allocation problems in coherent systems. The main advantages of the P and K-Algorithm are its simplicity and applicability to a wide range of complex optimization problems arising in system reliability design. A penalty guided artificial immune algorithm to solve mixed-integer reliability design problems was proposed in (Chen [6]). To efficiently find the feasible optimal/near optimal solution, it can search over promising feasible and infeasible regions. Gen and Yun [17] employed a soft computing approach for solving a variety of reliability optimization problems. To prevent the early convergence situation of its solution, this method combined rough search and local search techniques. Moreover, several optimization algorithms based on swarm intelligence, such as particle swarm optimization algorithm [8, 10, 11, 18], genetic algorithm [5, 19–21], evolutionary algorithm [7, 9, 22], ant colony algorithm [23] and harmony search algorithms [12, 13] have been employed to solve reliability problems. Kuo [24] reviewed recent advances in optimal RAP and summarized several techniques.

The cuckoo search algorithm is a new meta-heuristic approach, was developed recently by Yang and Deb [1, 2]. The optimal solutions obtained by Cuckoo Search (CS) are far better than the best solutions obtained by an efficient particle swarm optimiser [2]. The meta-heuristic optimization approach employing penalty guided based CS for the reliability optimization problems are proposed in this chapter. The CS algorithm is described in Sect. 3.

The chapter is organized as follows. In Sect. 2, the procedure of CS algorithm is briefly presented. In Sect. 3, four reliability optimization problems and a large-scale reliability optimization problem as well as two complex systems are introduced. In Sect. 4, a large number of numerical experiments are carried out to test the performance and effectiveness of the CS algorithm in solving complex reliability optimization problems. In addition, the chapter ends with some conclusions in Sect. 5.

2 Cuckoo Search Algorithm

In order to describe the Cuckoo Search more clearly, let us briefly review the interesting breed behavior of certain cuckoo species. Then, the basic ideas and steps of the CS algorithm will be outlined.

2.1 Cuckoo Breeding Behavior

Cuckoo search is an optimization algorithm developed by Xin-She Yang and Suash Deb in 2009 [1]. It was inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds (of other species). Some host birds can engage direct conflict with the intruding cuckoos. For example, if a host bird discovers the eggs are not their own, it will either throw these alien eggs away or simply abandon its nest and build a new nest elsewhere. Some cuckoo species such as the New World brood-parasitic Tapera have evolved in such a way that female parasitic cuckoos are often very specialized in the mimicry in colors and pattern of the eggs of a few chosen host species .This reduces the probability of their eggs being abandoned and thus increases their reproductively [25].

Furthermore, the timing of egg-laying of some species is also amazing. Parasitic cuckoos often choose a nest where the host bird just laid its own eggs. In general, the cuckoo eggs hatch slightly earlier than their host eggs. Once the first cuckoo chick is hatched, the first instinct action it will take is to evict the host eggs by blindly propelling the eggs out of the nest, which increases the cuckoo chick's share of food provided by its host bird [25]. Studies also show that a cuckoo chick can also mimic the call of host chicks to gain access to more feeding opportunity.

Cuckoo search idealized such breeding behavior, and thus can be applied for various optimization problems. In addition, Yang and Deb [1, 2] discovered that the random-walk style search is better performed by Lévy Flights rather than simple random walk.

2.2 Cuckoo Search

Each egg in a nest represents a solution, and a cuckoo egg represents a new solution. The aim is to use the new and potentially better solutions (cuckoos) to replace a not-so-good solution in the nests. In the simplest form, each nest has one egg. The algorithm can be extended to more complicated cases in which each nest has multiple eggs representing a set of solutions. CS is based on three idealized rules:

- Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest;
- The best nests with high quality of eggs (solutions) will carry over to the next generations;

- The number of available host nests is fixed, and a host can discover an alien egg with probability

$p_a \in [0, 1]$. In this case, the host bird can either throw the egg away or abandon the nest so as to build a completely new nest in a new location [1].

For simplicity, this last assumption can be approximated by a fraction (P_a) of the n nests being replaced by new nests (with new random solutions at new locations). For a maximization problem, the quality or fitness of a solution can simply be proportional to the objective function. Other forms of fitness can be defined in a similar way to the fitness function in genetic algorithms [1].

Based on these three rules, the basic steps of the CS can be summarized as the pseudo code, as follows [1]:

begin
Objective function $f(x), x = (x_1, ..., x_d)^T$
Generate initial population of n host nests $x_i (i = 1, 2, ..., n)$
While (t <MaxGeneration) or (stop criterion)
Get a cuckoo randomly by Lévy *flights evaluate its quality /fitness* F_i
Choose a nest among n (say.j) randomly
if $(F_i > F_j)$,
replace j by the new solution;
end if
A fraction (p_a) *of worse nests are abandoned and new ones are built;*
Keep the best solutions (or nests with quality solutions);
Rank the solutions and find the current best
end while
Postprocess results and visualization
end

When generating new solutions x(t+1) for, say cuckoo i, a Lévy flight is performed

$$x_i (t + 1) = x_i (t) + \alpha \oplus \text{Lévy}\ (\lambda) \tag{1}$$

where $\alpha > 0$ is the step size, which should be related to the scale of the problem of interest. In most cases, $\alpha = O(1)$ could be used. The above equation is essentially the stochastic equation for random walk. The product $\oplus$ means entry-wise multiplications. This entry-wise product is similar to those used in PSO, but here the random walk via Lévy flight is more efficient in exploring the search space as its step length is much longer in the long run [1]. A Lévy flight is a random walk in which the step-lengths are distributed according to a heavy-tailed probability distribution. Specifically, the distribution used is a power law of the form:

$$\text{Lévy u} = t^{-\lambda}, 1 < \lambda \leq 3 \tag{2}$$

which has an infinite variance. Here the consecutive jumps/steps of a cuckoo essentially form a random walk process which obeys a power-law step- length distribution

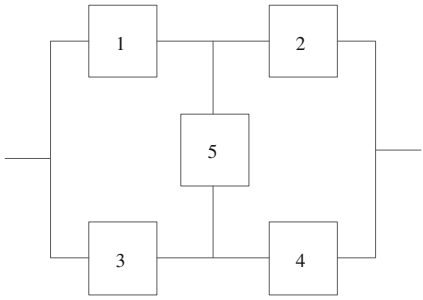

Fig. 1 The schematic diagram of complex (bridge) system

with a heavy tail. It is worth pointing out that, in the real world, if a cuckoo's egg is very similar to a host's eggs, then this cuckoo's egg is less likely to be discovered, thus the fitness should be related to the difference in solutions. Therefore, it is a good idea to do a random walk in a biased way with some random step sizes [1].

3 Case Studies: Reliability Optimization Problems

To evaluate the performance of the CS algorithm on reliability optimization problems, five case studies are considered. They are a complex (bridge) system, a series system, a series–parallel system, an overspeed protection system and large-scale systems as well as two complex systems (10-unit and 15-unit systems).

3.1 Case Study 1: A Complex (Bridge) System

This case study [3, 4, 6], and [10–14] is a nonlinear mixed-integer programming problem for a complex (bridge) system with five subsystems, as shown in Fig. 1.

The problem is formulated as follows:

$$\begin{aligned} Max \quad f(r,n) &= R_1R_2 + R_3R_4 + R_1R_4R_5 + R_2R_3R_5 \\ &\quad - R_1R_2R_3R_4 - R_1R_2R_3R_5 - R_1R_2R_4R_5 \\ &\quad - R_1R_3R_4R_5 - R_2R_3R_4R_5 + 2R_1R_2R_3R_4R_5 \end{aligned}$$

$s.t.$

$$g_1(r,n) = \sum_{i=1}^{m} w_i v_i^2 n_i^2 - V \leq 0$$

$$g_2(r,n) = \sum_{i=1}^{m} \alpha_i \left(-\frac{1000}{\ln(r_i)}\right)^{\beta_i} \left[n_i + \exp(0.25n_i)\right] - C \leq 0$$

Table 1 Data used in complex (bridge) and series systems

i	$10^5\alpha_i$	β_i	$w_i v_i^2$	w_i	V	C	W
1	2.330	1.5	1	7			
2	1.450	1.5	2	8			
3	0.541	1.5	3	8	110	175	200
4	8.050	1.5	4	6			
5	1.950	1.5	2	9			

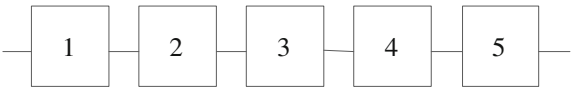

Fig. 2 The schematic diagram of a series system

$$g_3(r,n) = \sum_{i=1}^{m} w_i n_i \exp(0.25 n_i) - W \le 0$$
$$0 \le r_i \le 1, \quad n_i \in Z^+, \quad 1 \le i \le m$$

where m is the number of subsystems in the system. n_i, r_i, and $q_i = 1 - r_i$ are the number of components, the reliability of each component, and the failure probability of each component in subsystem i, respectively. $R_i(n_i) = (1 - q_i)^{n_i}$ is the reliability of subsystem$i$, whilst $f\,(r,n)$ is the system reliability. For each component in subsystem i, w_i, v_i, and c_i are the weight, volume, and cost, respectively. V, C, and W refer to the upper limit on the sum of the subsystems' products of volume and weight, the upper limit on the cost of the system, and the upper limit on the weight of the system, respectively. The parameters α_i and β_i are physical features of system components. Constraint $g_1(r,n)$ is a combination of weight, redundancy allocation, and volume. $g_2(r,n)$ and $g_3(r,n)$ are constraints on the cost and weight. The input parameters of the complex (bridge) system are given in Table 1.

3.2 Case study 2: A Series System

This case study [3, 4, 6, 11] and [14] is a nonlinear mixed-integer programming problem for a series system with five subsystems, as shown in Fig. 2.

The problem formulation is as follows:

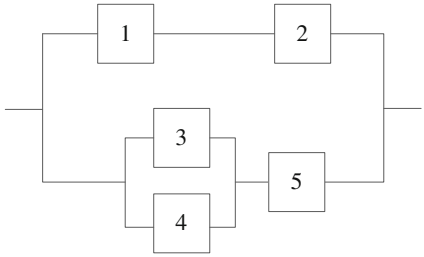

Fig. 3 The schematic diagram of a series–parallel system

$$Max \quad f(r,n) = \prod_{i=1}^{m} R_i(n_i)$$

$$s.t.$$

$$g_1(r,n) = \sum_{i=1}^{m} w_i v_i^2 n_i^2 - V \leq 0$$

$$g_2(r,n) = \sum_{i=1}^{m} \alpha_i \left(-\frac{1000}{\ln(r_i)}\right)^{\beta_i} \left[n_i + \exp(0.25 n_i)\right] - C \leq 0$$

$$g_3(r,n) = \sum_{i=1}^{m} w_i n_i \exp(0.25 n_i) - W \leq 0$$

$$0 \leq r_i \leq 1, \quad n_i \in Z^+, \quad 1 \leq i \leq m$$

This case study has three nonlinear constraints, which are the same as those of the 1st case study. Also, the input parameters of the series system are the same as those of the complex (bridge) system.

3.3 Case study 3: A Series-parallel System

This case study [3, 4, 6, 11], and [14] is a nonlinear mixed-integer programming problem for a series–parallel system with five subsystems, as shown in Fig. 3.

The reliability function reported by (Chen 2006, Hsieh 1998) was $f(r,n) = 1 - (1 - R_1R_2)(1 - (1 - R_3)(1 - R_4)R_5)$, in fact it is wrong (Wu *et al.* [11]), and the right problem formulation is as follows:

Table 2 Data used in series–parallel system

i	$10^5\alpha_i$	β_i	$w_i v_i^2$	w_i	V	C	W
1	2.500	1.5	2	3.5			
2	1.450	1.5	4	4			
3	0.541	1.5	5	4	180	175	100
4	0.541	1.5	8	3			
5	2.100	1.5	4	4.5			

$$
\begin{aligned}
&Max \quad f(r,n) = 1 - (1 - R_1R_2)(1 - (R_3 + R_4 - R_3R_4)R_5)\\
&s.t.\\
&\qquad g_1(r,n) = \sum_{i=1}^{m} w_i v_i^2 n_i^2 - V \le 0\\
&\qquad g_2(r,n) = \sum_{i=1}^{m} \alpha_i \left(-\frac{1000}{\ln(r_i)}\right)^{\beta_i} \left[n_i + \exp(0.25n_i)\right] - C \le 0\\
&\qquad g_3(r,n) = \sum_{i=1}^{m} w_i n_i \exp(0.25n_i) - W \le 0\\
&\qquad 0 \le r_i \le 1, \quad n_i \in Z^+, \quad 1 \le i \le m
\end{aligned}
$$

This case study has the same nonlinear constraints as those of the 1st case study, but it has different input parameters, given in Table 2.

3.4 Case study 4: An Overspeed System for a Gas Turbine

Overspeed detection is continuously provided by the electrical and mechanical systems. When an overspeed occurs, it is necessary to cut off the fuel supply. For this purpose, 4 control valves ($V1 - V4$) must be closed. The control system is modeled as a 4-stage series system, as shown in Fig. 4. The objective is to determine an optimal level of r_i & n_i at each stage i so that the system reliability is maximized.

This case study [6, 10–14], and [19] is formulated as follows:

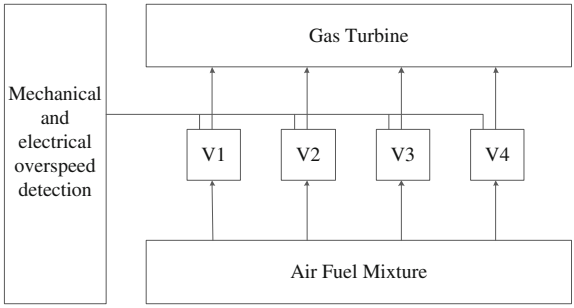

Fig. 4 The schematic diagram of an overspeed system for a gas turbine

$$Max \quad f(r,n) = \prod_{i=1}^{m} [1-(1-r_i)^{n_i}]$$

$s.t.$

$$g_1(r,n) = \sum_{i=1}^{m} v_i n_i^2 - V \leq 0$$

$$g_2(r,n) = \sum_{i=1}^{m} C(r_i)\left[n_i + \exp(0.25n_i)\right] - C \leq 0$$

$$g_3(r,n) = \sum_{i=1}^{m} w_i n_i \exp(0.25n_i) - W \leq 0$$

$$0.5 \leq r_i \leq 1-10^{-6}, \quad r_i \in R^+, \quad 1 \leq n_i \leq 10, \quad n_i \in Z^+$$

where r_i, n_i, v_i, and w_i refer to the reliability of each component, the number of redundant components, the product of weight and volume per element, and the weight of each component, all in stage i. The term $\exp(0.25n_i)$ accounts for the interconnecting hardware. The cost of each component with reliability r_i at subsystem i is given by $\mathrm{C(r_i)} = \alpha_\mathrm{i}(-\frac{\mathrm{T}}{\ln(\mathrm{r_i})})^{\beta_\mathrm{i}}$. Parameters α_i and β_i are constants representing the physical characteristics of each component at stage i. T is the operating time during which the component must not fail. The input parameters defining the overspeed protection system are given in Table 3.

Table 3 Data used in overspeed protection system

i	$10^5\alpha_i$	β_i	v_i	w_i	V	C	W	T
1	1.0	1.5	1	6				
2	2.3	1.5	2	6	250	400	500	1000 h
3	0.3	1.5	3	8				
4	2.3	1.5	2	7				

3.5 Case study 5: Large-Scale System Reliability Problem

Recently, Prasad and Kuo [15] developed P and K-Algorithm for solving large-scale system reliability optimization design problem. Gen and Yun [17] developed a soft computing approach to solve the same problem. The considered mathematical formulations are as follows:

$$Max \quad f(r,n) = \prod_{j=1}^{n} [1-(1-r_j)^{x_j}]$$

$$s.t.$$

$$g_1(r,n) = \sum_{j=1}^{n} \alpha_j x_j^2 - \left(1+\frac{\theta}{100}\right) \sum_{j=1}^{n} \alpha_j l_j^2 \le 0$$

$$g_2(r,n) = \sum_{j=1}^{n} \beta_j \exp\left(\frac{x_j}{2}\right) - \left(1+\frac{\theta}{100}\right) \sum_{j=1}^{n} \beta_j \exp(\frac{l_j}{2}) \le 0$$

$$g_3(r,n) = \sum_{j=1}^{n} \gamma_j x_j - \left(1+\frac{\theta}{100}\right) \sum_{j=1}^{n} \gamma_j l_j \le 0$$

$$g_4(r,n) = \sum_{j=1}^{n} \sqrt{\delta_j} x_j - \left(1+\frac{\theta}{100}\right) \sum_{j=1}^{n} \sqrt{\delta_j} l_j \le 0$$

$$1 \le x_j \le 10, \quad j = 1,2,\ldots,n.$$

Table 4 shows the design data of reliability systems for case study 5. The component reliabilities are generated from the uniform distribution in [0.95, 1.0]. The coefficients $\alpha_j, \beta_j, \gamma_j$ and δ_j are generated from uniform distributions in [6, 10] , [1, 5] , [11, 20] and [21, 40], respectively. Here, l_j represents the lower bound of x_j .The tolerance error θ, which implies 33% of the minimum requirement of each resource (l_j), is available for optimization. For the reliability system with n subsystems, the average minimum resource requirements $\sum_{j=1}^{n} g_{ij}(l_j)$, $(i = 1, ..., 4)$ and the average values of which is represented by $b_i = (1 + \frac{\theta}{100}) \sum_{j=1}^{n} g_{ij}(l_j)$, $(i = 1, ..., 4)$. In this way, we set the available system resources for reliability system with 36, 38, 40, 42 and 50 subsystems, respectively as shown in Table 5.

The best solutions obtained by the above two approaches [15, 17] are reported in Table 6. Here, VTV represents the variables which get value 2 in optimum conditions and the other variables are equal to 1.

Table 4 Data used in Large-scale system reliability problem

j	$1-r_j$	α_j	β_j	γ_j	δ_j	j	$1-r_j$	α_j	β_j	γ_j	δ_j
1	0.005	8	4	13	26	26	0.029	8	1	18	35
2	0.026	10	4	16	32	27	0.022	8	3	16	32
3	0.035	10	4	12	23	28	0.017	9	3	15	29
4	0.029	6	3	12	24	29	0.002	10	1	18	35
5	0.032	7	1	13	26	30	0.031	9	2	19	37
6	0.003	10	4	16	31	31	0.021	7	5	15	28
7	0.020	9	2	19	38	32	0.023	9	5	11	22
8	0.018	9	3	15	29	33	0.030	6	3	15	29
9	0.004	7	4	12	23	34	0.026	7	3	14	27
10	0.038	6	4	16	31	35	0.009	6	5	15	29
11	0.028	6	5	14	28	36	0.019	10	5	17	33
12	0.021	10	3	15	30	37	0.005	9	5	19	37
13	0.039	9	1	17	34	38	0.019	10	5	11	22
14	0.013	10	4	20	39	39	0.002	6	2	17	34
15	0.038	7	4	14	28	40	0.015	8	3	17	33
16	0.037	10	2	13	25	41	0.023	10	5	17	33
17	0.021	10	1	15	29	42	0.040	8	3	18	35
18	0.023	8	3	19	38	43	0.012	8	1	18	35
19	0.027	10	5	18	36	44	0.026	6	4	19	38
20	0.028	7	4	13	26	45	0.038	6	4	13	26
21	0.030	6	2	15	30	46	0.015	8	1	19	37
22	0.027	6	2	12	24	47	0.036	7	4	14	28
23	0.018	7	2	20	40	48	0.032	10	2	19	37
24	0.013	8	5	19	38	49	0.038	8	3	15	30
25	0.006	9	5	15	39	50	0.013	10	2	11	22

Table 5 Available system resources for each system in case study 5

n	i	1	2	3	4
36	b_i	391	257	738	1454
38	b_i	416	278	778	1532
40	b_i	435	289	823	1621
42	b_i	458	306	870	1712
50	b_i	543	352	1040	2048

3.6 Case study 6: 10-Unit System Reliability Problem with Different Combinations of Parameters

For a 10-Unit structure (Fig. 5) the problem is defined as (Agarwal and Vikas [30]):

Table 6 The optimization results of case study 5 with different dimensions

n	VTV in optimum	$f(x)$
36	{5,10,15,21,33}	0.519976
38	{10,13,15,21,33}	0.510989
40	{5,10,13,15,33}	0.503292
42	{4,10,11,15,21,33}	0.479664
50	{4,10,15,21,33,45,47}	0.405390

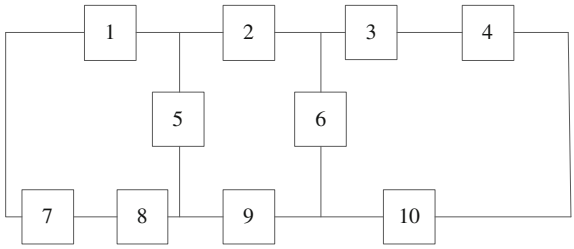

Fig. 5 The schematic diagram of 10-unit system

$$
\begin{aligned}
Max\ f(x) = {} & R_1R_2R_3R_4 + R_1R_2R_6R_{10}(Q_3 + R_3Q_4) \\
& + R_1R_5R_9R_{10}(Q_2 + R_2R_3Q_6 + R_2R_3Q_4Q_6) \\
& + R_7R_8R_9R_{10}(Q_1 + R_1Q_2Q_5 + R_1R_2Q_3Q_5Q_6 + R_1R_2Q_4Q_5Q_6) \\
& + R_2R_3R_4R_5R_7R_8Q_1(Q_9 + R_9Q_{10}) \\
& + Q_1R_3R_4R_6R_7R_8R_9Q_{10}(Q_2 + R_2Q_5) \\
& + R_1Q_2R_3R_4R_6R_7R_8R_9Q_{10} + R_1Q_2R_3R_4R_5R_6R_9Q_{10}(Q_7 + R_7Q_8) \\
& + Q_1R_2R_5R_6R_7R_8Q_9R_{10}(Q_3 + R_3Q_4)
\end{aligned}
$$

$s.t.$

$$
\begin{aligned}
& g_y(r,n) = \sum_{i=1}^{10} c_{yi}x_i \le b_y \quad y = 1, 2, \ldots, m \\
& x_i \in Z^+, \quad i = 1, 2, \ldots, 10.
\end{aligned}
$$

Where $R_i(x_i) = 1-(1-r_i)^{x_i}$ and $Q_i = 1 - R_i$ are the reliability, unreliability of subsystem i, respectively. The coefficients c_{yi} and r_i are generated from uniform distributions in [0, 100] and [0.6, 0.85], respectively. The m refer to number of constrain. The case study 6 is varied by taking $m = 1, 5$ and $b_y = rand(1.5, 3.5) * \sum_{i=1}^{10} c_{yi}$. This results in 2 sets of problems. The expressions for $f(x)$in this case study was obtained using the recursive disjoint product method given by Abraham [31].

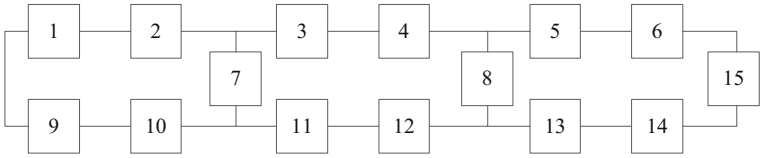

Fig. 6 The schematic diagram of 15-unit system

3.7 Case study 7: 15-Unit System Reliability Problem with Different Combinations of Parameters

For a 15-Unit structure (Fig. 6) the problem is defined as (Agarwal and Vikas [30]):

$$
\begin{aligned}
Max\ f(x) = {} & R_1R_2R_3R_4 + R_9R_{10}R_{11}R_{12}R_{13}R_{14}R_{15} \\
& \times (Q_1 + R_1Q_2 + R_1R_2Q_3 + R_1R_2R_3Q_4 + R_1R_2R_3R_4Q_5 \\
& + R_1R_2R_3R_4R_5Q_6) \\
& + R_3R_4R_5R_6R_7R_9R_{10}(Q_{11} + R_{11}Q_{12} + R_{11}R_{12}Q_{13} \\
& + R_{11}R_{12}R_{13}Q_{14} + R_{11}R_{12}R_{13}R_{14}Q_{15}) \\
& \times (Q_1 + R_1Q_2) + ((Q_1 + R_1Q_2)(Q_3 + R_3Q_4 + R_3R_4Q_7) \\
& + R_1R_2Q_7(Q_3 + R_3Q_4)) \\
& \times (Q_{13} + R_{13}Q_{14} + R_{13}R_{14}Q_{15})R_5R_6R_8R_9R_{10}R_{11}R_{12} \\
& + R_1R_2R_5R_6R_7R_8R_{11}R_{12} \\
& \times (R_9R_{10} + Q_9 + R_9Q_{10})(Q_3 + R_3Q_4) \\
& \times (Q_{13} + R_{13}Q_{14} + R_{13}R_{14}Q_{15}) + (Q_5 + R_5Q_6) \\
& \times ((Q_7 + R_7Q_{11} + R_7R_{11}Q_{12})(Q_9 + R_9Q_{10}) \\
& + R_9R_{10}(Q_{11} + R_{11}Q_{12})) \\
& \times R_1R_2R_3R_4R_8R_{13}R_{14}R_{15} \\
& + R_1R_2R_7R_{11}R_{12}R_{13}R_{14}R_{15}(Q_9 + R_9Q_{10}) \\
& \times (Q_3 + R_3Q_4 + R_3R_4Q_5 + R_3R_4R_5Q_6) \\
& + R_3R_4R_7R_8R_9R_{10}R_{13}R_{14}R_{15}(Q_1 + R_1Q_2) \\
& \times (Q_{11} + R_{11}Q_{12})(Q_5 + R_5Q_6)
\end{aligned}
$$

$$
s.t.
$$

$$
g_y(r,n) = \sum_{i=1}^{15} c_{yi}x_i \le b_y \quad y = 1, 2, \ldots, m
$$

$$
x_i \in Z^+, \quad i = 1, 2, \ldots, 15.
$$

Table 7 Data used in 10 unit and 15-unit system reliability problem

i	r	c_1	c_2	c_3	c_4	c_5
1	0.6796	33.2468	35.6054	13.7848	44.1345	10.9891
2	0.7329	27.5668	44.9520	96.7365	25.9855	68.0713
3	0.6688	13.3800	28.6889	85.8783	19.2621	1.0164
4	0.6102	0.4710	0.4922	63.0815	12.1687	29.4809
5	0.7911	51.2555	39.6833	78.5364	23.9668	59.5441
6	0.8140	82.9415	59.2294	11.8123	28.9889	46.5904
7	0.8088	51.8804	78.4996	97.1872	47.8387	49.6226
8	0.7142	77.9446	86.6633	45.0850	25.0545	59.2594
9	0.8487	26.8835	7.8195	3.6722	76.9923	87.4070
10	0.7901	85.8722	27.7460	55.3950	53.3007	55.3175
11	0.6972	41.8733	90.4377	75.7999	95.0057	54.1269
12	0.6262	61.6181	58.0131	98.5166	97.9127	59.1341
13	0.6314	90.0418	77.8206	60.6308	37.2226	40.9427
14	0.6941	75.5947	36.4524	70.4654	96.9179	40.2141
15	0.6010	88.5974	61.0591	18.8802	42.1222	80.0045
b	–	3.2150	3.4710	3.3247	2.6236	3.4288

Where $R_i(x_i)$, Q_i, c_{yi} and r_i are similar to those used in 10-unit system. The m refer to number of constrain. The problem is varied by taking $m = 1, 5$ and $b_y = rand(1.5, 3.5) * \sum_{i=1}^{15} c_{yi}$. This results in 2 sets of problems. The expressions for $f(x)$ in case study 7 was obtained using the recursive disjoint product method given by Abraham [31]. The random data used in case study 6 and 7 for $m = 1, 5$ as shown in Table 7.

4 Experimental Results, Analysis and Discussion

The parameters of the CS algorithm used for reliability optimization problems are shown in Table 8.

Table 9 includes the numerical experiments results of fifty independent runs of the CS algorithm and all experiments are performed in MATLAB. The PC used is an INTEL32, X2, 3.0GHz having 4GB of memory. Moreover, execution times for all case-studies have been mentioned in Table 9 for 50 iterations.

Here, 'SD' represents the standard deviation based on fifty converged objective function values whilst 'Median' represents the average value of the objective function. 'NFOS' represents the number of feasible optimal solutions found in fifty runs.

Tables 10, 11, 12, 13 compare the best results obtained in this chapter for four case studies with those provided by other methods reported in the literatures [3, 4, 6, 10–13, 19] and [27–29]. Clearly, the solutions obtained by the CS algorithm are better than those given by the other methods. An improvement index is required to measure the improvement of the best solutions found by the CS algorithm in comparison with

Table 8 Parameter for CS

Example	Population size	Number of iterations	Probability p_a	α
Case study1	10	1000	0.25	1
Case study 2	10	1000	0.25	1
Case study 3	10	1000	0.25	1
Case study 4	10	1000	0.25	1
Case study 5-(36 DIM)	10	2000	0.25	1
Case study 5-(38 DIM)	10	2000	0.25	1
Case study 5-(40 DIM)	10	2500	0.25	1
Case study 5-(42 DIM)	10	2500	0.25	1
Case study 5-(50 DIM)	10	3000	0.25	1
Case study 6-(10×1)	10	750	0.25	1
Case study 6-(10×5)	10	750	0.25	1
Case study 7-(15×1)	10	750	0.25	1
Case study 7-(15×5)	10	750	0.25	1

Table 9 Simulation results after fifty runs

Example	Best	Worst	Median	SD	NFOS	Execution time for 50-iteration (s)
Case study1	0.99988964	0.99968396	0.99986946	3.2703e-05	50	71
Case study 2	0.93168239	0.89905558	0.92903027	4.5025e-03	50	38
Case study 3	0.99997665	0.99984484	0.99996550	1.5389e-05	50	40
Case study 4	0.99995468	0.99991922	0.99995336	4.5576e-06	50	40
Case study 5-(36 DIM)	0.51997597	0.49949243	0.51718476	4.2094e-03	50	115
Case study 5-(38 DIM)	0.51098860	0.49467807	0.50801105	2.9530e-03	50	132
Case study 5-(40 DIM)	0.50599242	0.49604469	0.50184162	2.0457e-03	50	162
Case study 5-(42 DIM)	0.47966355	0.46659835	0.47421700	2.7735e-03	50	167
Case study 5-(50 DIM)	0.40695474	0.39205805	0.40347689	3.2395e-03	50	233
Case study 6-(10×1)	0.98335697	0.98218267	0.98279202	5.5595e-04	50	31
Case study 6-(10×5)	0.67189992	0.67181316	0.67189992	3.3645e-16	50	36
Case study 7-(15×1)	0.97276197	0.97024537	0.97223452	3.6879e-04	50	43
Case study 7-(15×5)	0.95825279	0.95594309	0.95797782	2.2199e-04	50	45

those given by the other methods. This index, which has been called maximum possible improvement [15], is as follows:

$$\text{MPI}(\%) = \left(\frac{f_{CS} - f_{\text{other}}}{1 - f_{\text{other}}} \right) \tag{3}$$

where f_{CS} represents the best system reliability obtained by the CS algorithm, while f_{other} denotes the best system reliability obtained by any other method. It should

Table 10 Case Study 1: Bridge (complex) system

Parameter	Hikita [3]	Hsieh [4]	Chen [6]	Coelho [10]	Zou [12]	Wu [11]	CS
f(r,n)	0.9997894	0.99987916	0.99988921	0.99988957	0.99988962	0.99988963	0.99988964
n_1	3	3	3	3	3	3	3
n_2	3	3	3	3	3	3	3
n_3	2	3	3	2	2	2	2
n_4	3	3	3	4	4	4	4
n_5	2	1	1	1	1	1	1
r_1	0.814483	0.814090	0.812485	0.826678	0.82883148	0.82868361	0.82809404
r_2	0.821383	0.864614	0.867661	0.857172	0.85836789	0.85802567	0.85800449
r_3	0.896151	0.890291	0.861221	0.914629	0.91334996	0.91364616	0.91416292
r_4	0.713091	0.701190	0.713852	0.648918	0.64779451	0.64803407	0.64790779
r_5	0.814091	0.734731	0.756699	0.715290	0.70178737	0.70227595	0.70456598
MPI	47.5973	8.672625	0.388122	0.063389	0.018120	0.009060	–
(%)	41						

be noted that in high reliability applications, it is often difficult to obtain even very small improvements in reliability.

For the complex (bridge) system, as mentioned in Table 10, the best results reported by Hikita [3], Hsieh [4], Chen [6], Coelho [10], Zou [12] and Wu [11], are 0.9997894, 0.99987916, 0.99988921, 0.99988957, 0.99988962 and 0.99988963, respectively. The CS result is 47.597341 %, 8.672625 %, 0.388122 %, 0.063389 %, 0.018120 %, and 0.009060 %, better than aforementioned algorithms in terms of improvement indices, respectively.

For the series system, Table 11 shows that the best results reported by Kuo [27], Xu [28], Hikita [3], Hsieh [4], Chen [6] and Wu [11], are 0.92975, 0.931677, 0.931363, 0.931578, 0.931678, and 0.931680 respectively. The result given by the CS is better

Table 11 Case study 2: Series system

Parameter	Kuo [27]	Xu [28]	Hikita [3]	Hsieh [4]	Chen [6]	Wu [11]	CS
f(r,n)	0.92975	0.931677	0.931363	0.931578	0.931678	0.931680	0.931682
n_1	3	3	3	3	3	3	3
n_2	2	2	2	2	2	2	2
n_3	2	2	2	2	2	2	2
n_4	3	3	3	3	3	3	3
n_5	2	3	3	3	3	3	3
r_1	0.77960	0.77939	0.777143	0.779427	0.779266	0.78037307	0.77941694
r_2	0.80065	0.87183	0.867514	0.869482	0.872513	0.87178343	0.87183328
r_3	0.90227	0.90288	0.896696	0.902674	0.902634	0.90240890	0.90288508
r_4	0.71044	0.71139	0.717739	0.714038	0.710648	0.71147356	0.71139387
r_5	0.85947	0.78779	0.793889	0.786896	0.788406	0.78738760	0.78780371
MPI (%)	2.750748	0.007904	0.465347	0.152583	0.006440	0.003513	–

Table 12 Case study 3: Series-parallel system

Parameter	Hikita [3]	Hsieh [4]	Chen [6]	Wu [11]	CS
f(r,n)	0.99996875	0.99997418	0.99997658	0.99997664	0.99997665
n_1	3	2	2	2	2
n_2	3	2	2	2	2
n_3	1	2	2	2	2
n_4	2	2	2	2	2
n_5	3	4	4	4	4
r_1	0.838193	0.785452	0.812485	0.81918526	0.81992709
r_2	0.855065	0.842998	0.843155	0.84366421	0.84526766
r_3	0.878859	0.885333	0.897385	0.89472992	0.89549155
r_4	0.911402	0.917958	0.894516	0.89537628	0.89544069
r_5	0.850355	0.870318	0.870590	0.86912724	0.86831878
MPI (%)	25.280000	9.566228	0.298890	0.042808	–

Table 13 Case study 4: Over-speed system

Parameter	Dhingra [29]	Yokota [19]	Chen [6]	Coelho [9]	Zou [12]	Wu [11]	CS
f(r,n)	0.99961	0.999468	0.999942	0.999953	0.99995467	0.99995467	0.99995468
n_1	6	3	5	5	5	5	5
n_2	6	6	5	6	6	6	5
n_3	3	3	5	4	4	4	4
n_4	5	5	5	5	5	5	6
r_1	0.81604	0.965593	0.903800	0.902231	0.90186194	0.90163164	0.90161460
r_2	0.80309	0.760592	0.874992	0.856325	0.84968407	0.84997020	0.88822337
r_3	0.98364	0.972646	0.919898	0.948145	0.94842696	0.94821828	0.94814103
r_4	0.80373	0.804660	0.890609	0.883156	0.88800590	0.88812885	0.84992090
MPI (%)	88.461523	91.541402	22.413812	4.255389	0.012186	0.000910	–

than six mentioned results. The improvement indices are 2.750748 %, 0.007904 %, 0.465347 %, 0.152583 %, 0.006440 %, and 0.003513 %, respectively.

A same investigation has been done for series-parallel system. The CS algorithm results in 25.280000 %, 9.566228 %, 0.298890 %, and 0.042808 % improvement compare with the best results reported by Hikita [3], Hsieh [4], Chen [6] and Wu [11] (Table 12).

Considering an overspeed system for a gas turbine, Table 13 shows that the best results reported by Dhingra [29], Yokota [19], Chen [6], Coelho [10], Zou [12] and Wu [11], are 0.99961, 0.999468, 0.999942, 0.999953, 0.99995467, and 0.99995467 respectively. The result provided by the CS is better than the above five results. The improvement indices are 88.461538 %, 91.541353 %, 22.413793 %, 4.255319 %, 0.012186 %, and 0.000910 % respectively.

For case study 5 the results of CS algorithm are shown in Table 14.In addition, Table 15 indicates the comparison between CS and Wu algorithms [11] for case study 5. It can be seen in Table 15 that the best results of CS algorithm and Wu [11]

Table 14 The best results of case study 5 with different dimensions by CS algorithm

n	VTV in optimum	*f(x)*
36	{5,10,15,21,33}	0.51997597
38	{10,13,15,21,33}	0.51098860
40	{4,10,11,21,22,33}	0.50599242
42	{4,10,11,15,21,33}	0.47966355
50	{4,10,15,21,33,42,45}	0.40695474

Table 15 Comparison results for the Large-scale system reliability problem

Example	Algorithm	Best	Worst	Median	SD
Case Stydy5-(36 DIM)	IPSO(Wu 2010) [11]	**0.51997597**	0.49705368	0.50874873	5.8624e-03
	CS	**0.51997597**	**0.49949243**	**0.51718476**	**4.2094e-03**
Case Stydy5-(38 DIM)	IPSO(Wu 2010) [11]	**0.51098860**	0.48707780	0.50557752	5.3420e-03
	CS	**0.51098860**	**0.49467807**	**0.50801105**	**2.9530e-03**
Case Stydy5-(40 DIM)	IPSO(Wu 2010) [11]	**0.50599242**	0.48016779	0.49772492	5.4393e-03
	CS	**0.50599242**	**0.49604469**	**0.50184162**	**2.0457e-03**
Case Stydy5-(42 DIM)	IPSO(Wu 2010) [11]	**0.47966355**	0.45788422	0.47090324	4.6902e-03
	CS	**0.47966355**	**0.46659835**	**0.47421700**	**2.7735e-03**
Case Stydy5-(50 DIM)	IPSO(Wu 2010) [11]	0.40657143	0.36682410	0.39338827	6.5878e-03
	CS	**0.40695474**	**0.39205805**	**0.40347689**	**3.2395e-03**

Table 16 Comparison results for the 10-nuit and 15-nuit system reliability problem

Example	Algorithm	Best	Worst	Median	SD
Case Stydy6-(10×1)	GA	0.57473724	0.48518883	0.52702874	1.1723e-01
	ABC	0.95403857	0.52635359	0.80094474	9.9360e-02
	HS	0.88752850	0.88752850	0.60847752	1.9472e-01
	NGHS	0.98201395	0.94484918	0.97011678	9.4964e-03
	CS	**0.98335697**	**0.98218267**	**0.98279202**	**5.5595e-04**
Case Stydy6-(10×5)	GA	0.40859193	0.24604448	0.34072963	9.9898e-02
	ABC	0.63631535	0.27879103	0.42419274	8.8019e-02
	HS	0.50157124	0.12456904	0.26027637	9.3011e-02
	NGHS	0.64820235	0.44171963	0.57267935	4.4999e-02
	CS	**0.67189992**	**0.67181316**	**0.67189992**	**3.3645e-16**
Case Stydy7-(15×1)	GA	0.82700343	0.31578718	0.58041963	1.1514e-01
	ABC	0.95260704	0.93872370	0.94819588	3.4596e-03
	HS	0.97235030	0.96904606	0.97141480	7.0892e-04
	NGHS	0.97235030	0.96718809	0.97089223	9.9082e-04
	CS	**0.97276197**	**0.97024537**	**0.97223452**	**3.6879e-04**
Case Stydy7-(15×5)	GA	0.82381093	0.36520795	0.61764965	1.3423e-01
	ABC	0.95141221	0.93911154	0.94695160	2.9954e-03
	HS	**0.95825279**	0.95364976	0.95685535	1.0764e-03
	NGHS	0.95810719	0.95055900	0.95594432	1.8335e-03
	CS	**0.95825279**	**0.95594309**	**0.95797782**	**2.2199e-04**

for n = 36, 38, 40, 42 are same but for n = 50 the CS provides better results. Also the worst, median and SD results given by the CS are better than those given by Wu for n = 36, 38, 40, 42 and 50 in case study 5.

For case study 6 and 7, 10-unit and 15-Unit system reliability problem, Table 16 shows the results given by Genetic Algorithm (GA) (Holland [32]), Artificial Bee Colony (ABC) [33], Harmony Search (HS) [34], Novel Global Harmony Search (NGHS) [12] and CS. The parameters of GA, ABC, HS and NGHS algorithms are as follows:

- GA: Chromosome = 50, Crossover = 0.9, Mutation = 0.3,Generations = 2000
- ABC: Colony = 20, Limit = 100,Iterations = 1000
- HS: HMS = 5, HMCR = 0.9,PAR = 0.3, BW = 0.01, Number Of Harmony = 5000
- NGHS: HMS = 5,p_m = 0.05, Number Of Harmony = 5000

The result provided by the CS in this two case study 6 and 7 for $m = 1, 5$ is better than the all above algorithm results.

According to the above comparisons, it can be concluded that the CS algorithm outperforms the other methods in literature to find best solutions for the given reliability optimization problems.

5 Conclusion

In this chapter, the Cuckoo Search by Lévy flights has been used to solve five well-known reliability optimization problems and two complex systems. In these optimization problems, both the redundancy (number of redundant components) and the corresponding reliability of each component in each subsystem under multiple constraints are considered to be decided simultaneously. Simulation results showed that the CS algorithm led to best solutions in comparison with other methods The studies show that CS algorithm can be powerful and robust for problems with large search space and difficult-to-satisfy constraints.

References

1. Yang, X. S., Deb, S.: Cuckoo search via Lévy flights. In: Proc. World Congr. Nat. Biol. Ins. Comput. (NaBIC 2009, India), 210–214.(2009)
2. Yang, X.S., Deb, S.: Engineering optimisation by cuckoo search. Int. J. Mathe. Mod. Num. Optim. **1**(4), 330–343 (2010)
3. Hikita, M., Nakagawa, H., Harihisa, H.: Reliability optimization of systems by a surrogate constraints algorithm. IEEE Trans. Reliab. **41**(3), 473–480 (1992)
4. Hsieh, Y.C., Chen, T.C., Bricker, D.L.: Genetic algorithm for reliability design problems. Microelectron. Reliab. **38**(10), 1599–1605 (1998)

5. Gen, M., Kim, J.R.: GA-based reliability design: State-of-the-art survey. Comput. Indust. Eng. **37**(1–2), 151–155 (1999)
6. Chen, T.C.: IAs based approach for reliability redundancy allocation problems. Appl. Math. and Comput. **182**(2), 1556–1567 (2006)
7. Salazar, D., Rocco, C.M., Galvn, B.J.: Optimization of constrained multipleobjective reliability problems using evolutionary algorithms. Reliab. Eng. and Sys. Saf. **91**(9), 1057–1070 (2006)
8. Yin, P.Y., Yu, S.S., Wang, P.P., Wang, Y.T.: Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization. J. Sys. Soft. **80**(5), 724–735 (2007)
9. Ramirez-Marquez, J.E.: Port-of-entry safety via the reliability optimization of container inspection strategy through an evolutionary approach. Reliab. Eng. Sys. Saf. **93**(11), 1698–1709 (2008)
10. Coelho, L.S.: An efficient particle swarm approach for mixed-integer programming in reliability-redundancy optimization applications. Reliab. Eng. Sys. Saf. **94**(4), 830–837 (2009)
11. Wu, P., Gao, L., Zou, D., Li, S.: An improved particle swarm optimization algorithm for reliability problems. ISA Trans. **50**(1), 71–81 (2010)
12. Zou, D., Gao, L., Wu, J., Li, S., Li, Y.: A novel global harmony search algorithm for reliability problems. Comput. Ind. Eng. **58**(2), 307–316 (2010)
13. Zou, D., Gao, L., Li, S., Wu, J.: An effective global harmony search algorithm for reliability problems. Expert Sys. Appl. **38**(4), 4642–4648 (2011)
14. Kanagaraj, G., Jawahar, N.: Simultaneous allocation of reliability & redundancy using minimum total cost of ownership approach. J. comput. Appl. Res. Mech. Eng. **1**(1), 1–16 (2011)
15. Prasad, V.R., Kuo, W.: An annotated overview of system-reliability optimization. IEEE Trans. Reliab. **49**(2), 176–187 (2000)
16. Chern, M.S., Jan, R.H.: Reliability optimization problems with multiple constraints. IEEE Trans. Reliab. **35**(4), 431–436 (1986)
17. Gen, M., Yun, Y.S.: Soft computing approach for reliability optimization: State-of-the-art survey. Reliab. Eng. Sys. Saf. **91**(9), 1008–1026 (2006)
18. Elegbede, C.: Structural reliability assessment based on particles swarm optimization. Struct. Saf. **27**(2), 171–186 (2005)
19. Yokota, T., Gen, M., Li, H.H.: Genetic algorithm for nonlinear mixed-integer programming problems and its application. Comput. Ind. Eng. **30**(4), 905–917 (1996)
20. Marseguerra, M., Zio, E., Podofillini, L.: Optimal reliability/availability of uncertain systems via multi-objective genetic algorithms. IEEE Trans. Reliab. **53**(3), 424–434 (2004)
21. Painton, L., Campbell, J.: Genetic algorithms in optimization of system reliability. IEEE Trans. Reliab. **44**(2), 172–178 (1995)
22. Aponte, D.E.S., Sanseverino, C.M.R.: Solving advanced multi-objective robust designs by means of multiple objective evolutionary algorithms (MOEA): A reliability application. Reliab. Eng. Sys. Saf. **92**(6), 697–706 (2007)
23. Meziane, R., Massim, Y., Zeblah, A., Ghoraf, A., Rahli, R.: Reliability optimization using ant colony algorithm under performance and cost constraints. Electr. Power Sys. Res. **76**(1–3), 1–8 (2005)
24. Kuo, W.: Recent advances in optimal reliability allocation. IEEE Trans. Sys., Man, and Cyber.-Part A: Sys. Hum. **37**(2), 143–156 (2007)
25. Payne, R. B. et al.: The Cuckoos. Oxford University (2005)
26. Prasad, V.R., Kuo, W.: Reliability optimization of coherent systems. IEEE Trans. Reliab. **49**(3), 323–330 (2000)
27. Kuo, W., Hwang, C.L., Tillman, F.A.: A note on heuristic methods in optimal system reliability. IEEE Trans. Reliab. **27**(5), 320–324 (1978)
28. Xu, Z., Kuo, W., Lin, H.H.: Optimization limits in improving system reliability. IEEE Trans. Reliab. **39**(1), 51–60 (1990)
29. Dhingra, A.K.: Optimal apportionment of reliability & redundancy in series systems under multiple objectives. IEEE Trans. Reliab. **41**(4), 576–582 (1992)

30. Agrwal, M., Vikas, K.S.: Ant colony approach to constrained redundancy optimization in binary systems. Appl. Math. Model. **34**, 992–1003 (2010)
31. Abraham, J.A.: An improved algorithm for network reliability. IEEE Trans. Reliab. **28**, 58–61 (1979)
32. Holland, J.H.: Adaption in natural and artificial systems. University of Michigan Press, Ann Arbor, MI (1975)
33. Karaboga, D.: An idea based on honeybee swarm for numerical optimization. Erciyes University, Turkey (2005)
34. Lee, K.S., Geem, Z.W.: A new meta-heuristic algorithm for continuous engineering optimization, harmony search theory and practice. Comp. Meth. Appl. Mech. Eng. **194**, 3902–3933 (2005)

Hybridization of Cuckoo Search and Firefly Algorithms for Selecting the Optimal Solution in Semantic Web Service Composition

Ioan Salomie, Viorica Rozina Chifu and Cristina Bianca Pop

Abstract This chapter investigates how the Cuckoo Search and Firefly Algorithm can be hybridized for performance improvement in the context of selecting the optimal or near-optimal solution in semantic Web service composition. Cuckoo Search and Firefly Algorithm are hybridized with genetic, reinforcement learning and tabu principles to achieve a proper exploration and exploitation of the search process. The hybrid algorithms are applied on an enhanced planning graph which models the service composition search space for a given user request. The problem of finding the optimal solution encoded in the enhanced planning graph can be reduced to identifying a configuration of semantic Web services, out of a very large set of possible configurations, which maximizes a fitness function which considers semantics and *QoS* attributes as selection criteria. To analyze the benefits of hybridization we have comparatively evaluated the classical Cuckoo Search and Firefly Algorithms versus the proposed hybridized algorithms.

Keywords Cuckoo search algorithm · Firefly algorithm · hybrid nature-inspired algorithm · Optimal or near-optimal Web service composition

I. Salomie (✉) · V. R. Chifu (✉) · C. B. Pop (✉)
Computer Science Department, Technical University of Cluj-Napoca, 26-28 G. Baritiu street, Cluj-Napoca, Romania
e-mail: Viorica.Chifu@cs.utcluj.ro

I. Salomie
e-mail: Ioan.Salomie@cs.utcluj.ro

C. B. Pop
e-mail: Cristina.Pop@cs.utcluj.ro

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_11,

1 Introduction

Service Oriented Computing is a software paradigm that has emerged due to the necessity of ensuring Business-to-Business Collaboration over the Internet. Web services represent the building blocks of Service Oriented Computing as they provide a self-describing and platform-independent solution for exposing on the Internet the functionalities offered by different business entities. Web services are involved in composition processes when complex user requests need to be satisfied. An important step in automatic Web service composition is the selection of the optimal composition solution that best satisfies all the non-functional constraints specified in the business requests. The selection of the optimal composition solution can be viewed as a combinatorial optimization problem because it implies the search for the optimal configuration of Web services out of a large number of allowed combinations which best satisfies multiple objectives that can be even conflicting. Such optimization problems can be solved using heuristic methods that are capable of providing high quality solutions in a short time and without processing the entire search space. Nature-inspired metaheuristics represent a class of heuristic methods inspired by natural phenomenas or behaviours that have ensured the survivability of animals, birds, insects over thousands of years by enabling them to find perfect solutions to almost all their problems (e.g. food search, mating, breeding, etc.) [29]. A new trend in developing nature-inspired metaheuristics is to combine the algorithmic components from various metaheuristics aiming to improve the performance of the original metaheuristics in solving hard optimization problems.

This chapter addresses the problem of hybridizing two nature-inspired metaheuristics, Cuckoo Search [28] and Firefly Algorithm [27], in the context of selecting the optimal solution in semantic Web service composition. To hybridize the nature-inspired metaheuristics we have followed the next steps: (1) formally define the problem to be solved as an optimization problem, (2) develop a hybrid nature-inspired model which defines the core and hybrid components representing the main building blocks of the hybrid nature-inspired selection algorithms, (3) develop the hybrid nature-inspired algorithms, and (4) evaluate the hybrid nature-inspired selection algorithms by identifying the optimal values of their adjustable parameters.

The chapter is structured as follows. Section 2 introduces the theoretical background. Section 3 reviews the state of the art, Sect. 4 presents the steps for developing hybrid nature-inspired techniques for selecting the optimal solution in Web service composition. Sections 5–8 present the Hybrid Cuckoo Search-based and Hybrid Firefly-based techniques for selecting the optimal solution in semantic Web service composition. Section 9 presents the paper's conclusions.

2 Background

This section presents the theoretical background required for developing the Hybrid Cuckoo Search-based and Hybrid Firefly-based techniques.

2.1 Nature-Inspired Metaheuristics

Nature-inspired metaheuristics, relying on concepts and search strategies inspired from nature, have emerged as a promising type of stochastic algorithms used for efficiently solving optimization problems. By the number of solutions processed within an algorithm iteration, nature-inspired metaheuristics can be classified into *population-based* and *trajectory-based*. Population-based metaheuristics (e.g. genetic algorithms, ant algorithms, particle swarm optimization) process simultaneously a population of solutions thus following the evolution of a set of solutions in the search space, while trajectory-based algorithms (e.g. tabu search, simulated annealing) process only a single solution thus following a trajectory in the search space [3]. By the usage of a search history during the search process, nature-inspired metaheuristics can be classified as: (1) *memory usage-based metaheuristics*—rely on the information about past moves and decisions stored in a short or long term memory which is used to take future decisions in the search process, and (2) *memory-less-based metaheuristics*—rely on the information of the current state which is used to take future decisions in the search process [3].

In what follows, we present the nature-inspired metaheuristics proposed in the research literature that are relevant for this chapter.

The Cuckoo Search Algorithm. The Cuckoo Search algorithm [28] is inspired by the cuckoo's reproduction behavior which consists of laying eggs in the nests of other birds. In the Cuckoo Search algorithm, cuckoos are abstracted as agents having associated a solution (i.e. the egg) of the optimization problem that they try to place in a solution container (i.e. the host bird's nest). The Cuckoo Search algorithm has an initialization stage and an iterative stage. In the initialization stage, each container is initialized with a solution of the optimization problem. In the iterative stage, the following steps are performed until a stopping condition is satisfied [28]:

Step 1. An agent is randomly selected and a new solution x^{t+1} is generated for the agent at the moment t in time by performing a random walk:

$$x^{t+1} = x^t + \alpha L(s, \lambda) \tag{1}$$

where α is a step-size-scaling factor, s is the step size, and L refers to a random walk defined as [30]:

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda) \sin(\pi \lambda / 2)}{\pi} * \frac{1}{s^{1+\lambda}} \tag{2}$$

Step 2. A container where the agent may lay its solution is randomly chosen.
Step 3. The agent's solution will replace the container's solution, if the solution associated to the container has a lower (in case of maximization problems) or higher (in case of minimization problems) fitness than the fitness of the agent's solution.
Step 4. Some of the containers containing the worst candidate solutions are destroyed (similar to the abandon of the nest by the host bird) and replaced with new ones containing randomly generated solutions.
Step 5. The containers with the best solutions are kept.
Step 6. The solutions are ranked and the current best solution is found.

The Firefly Algorithm. The Firefly algorithm [27] is inspired by the behavior of fireflies which search for a mating partner by emitting a flashing light. The algorithm relies on a set of agents, each agent having an associated solution, which are attracted to one another based on their brightness (the solution fitness). The Firefly Algorithm consists of an initialization stage followed by an iterative stage. In the initialization stage, each agent, part of the population of agents, is initialized with a solution of the optimization problem. Then, in the iterative stage, the following steps are performed until a stopping condition is fullfilled [27]:

Step 1. The light intensity of each agent i located in position x_i is compared with the light intensity of the other agents in the population: if the light intensity of the agent i is lower than the light intensity of another agent j located in position x_j, then the agent i will be attracted by the agent j towards it will move as follows:

$$x_i = x_i + \beta_0 * e^{-\gamma * r_{ij}^2} * (x_j - x_i) + \alpha * \varepsilon_i \tag{3}$$

where r_{ij} is the distance between the two agents computed based on their positions, β_0 is attractiveness between two agents for which the distance $r_{ij} = 0$, α is a randomization parameter, and ε_i is a vector of random numbers.
Step 2. All agents are ranked based on their solution's fitness value.
Step 3. The current global best is updated (if it is the case).

2.2 Hybridization of Nature-Inspired Metaheuristics

Hybrid metaheuristics combine algorithmic components from various optimization algorithms aiming to improve the performance of the original metaheuristics in solving optimization problems [4]. Hybridizing a metaheuristic does not guarantee that it will work well for all optimization problems [4]. Hybridization by combining components from other metaheuristics, and hybridization by cooperative search are considered as two main hybridization techniques [3]. The hybridization of metaheuristics with components from other metaheuristics combines trajectory-based metaheuristics with population-based metaheuristics. This approach is motivated by the fact

that population-based metaheuristics can explore the promissing search space areas which can be then exploited to quickly find the best solutions using trajectory-based metaheuristics [4]. The hybridization of metaheuristics by cooperative search relies on the cooperation between multiple search algorithms that are executed in parallel.

2.3 Semantic Web Service Composition Flow

The Service Oriented Architecture has emerged as a Service Oriented Computing-based architecture for developing distributed applications using Web services [19]. A Web service is a software component exposing various functionalities which can be published, located and invoked on the Internet. The Semantic Web technologies led to the definition of semantic Web services which facilitate the automation of the services' lifecycle phases (e.g. discovery, selection, composition, execution). There are situations in which semantic Web services need to be composed in order to fulfill a complex business request which can not be addressed by a single atomic service. The semantic Web service composition flow is shown in Fig. 1. A prerequisite of automatic composition is the existence of a set of Web services semantically annotated and published in a Universal Description Discovery and Integration (UDDI) registry. Thus, a domain ontology describing the functional and non-functional features of Web services must be used to semantically annotate the WSDL syntactic descriptions of the Web services. Having annotated the set of Web services, the next step is to publish them in a UDDI registry. The existence of a large number of semantic Web services in the UDDI registry makes the discovery process inefficient, unless services are grouped according to specific criteria. Incorporating service clustering capabilities in UDDI registries is a solution to this problem as it aims to gather into clusters the services sharing common semantic features.

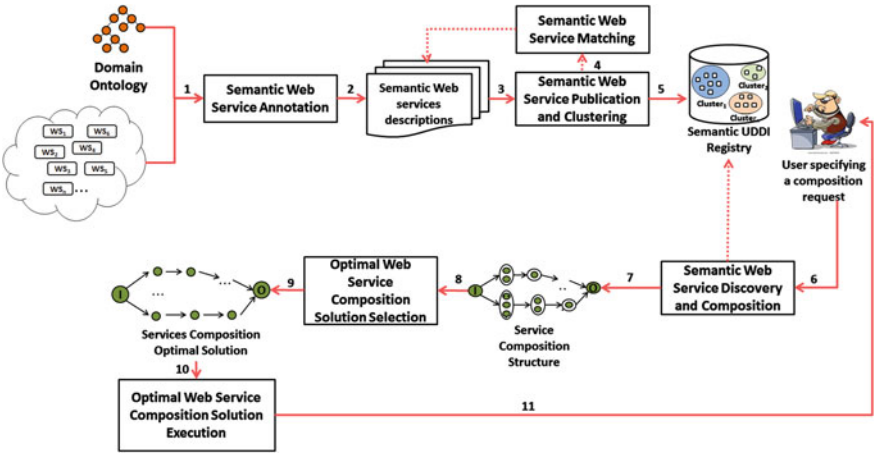

Fig. 1 Semantic web service composition flow (the *arrow* numbers indicate the processing flow)

The chaining of Web services in a composition can be identified based on the semantic matching between the output and the input parameters of the discovered services. Most of the composition methods focus on the functional aspects, without considering the non-functional ones such as Quality of Service (*QoS*) attributes. The non-functional aspects are considered in the selection process which aims to identify the optimal or a near-optimal composition solution. The existence of a large number of services with similar functionalities can lead to compositions containing multiple alternative solutions. In this case appropriate methods for selecting the optimal solution that best satisfies the composition requirements need to be used. After the optimal composition is identified, the underlying semantic Web services are executed.

3 Literature Review

This section presents the state of the art in non-hybrid and hybrid nature-inspired approaches for selecting the optimal solution in Web service composition.

3.1 Non-hybrid Nature-Inspired Techniques

A non-hybrid nature-inspired technique for selecting the optimal Web service composition is based on adapting a general purpose-nature-inspired metaheuristic. In the context of selecting the optimal Web service composition, the following nature-inspired metaheuristics have been mostly applied: Genetic Algorithms, Ant Colony Optimization, Clonal Selection. In what follows, we present a survey of non-hybrid nature-inspired techniques used for identifying the optimal Web service composition. Most of these approaches apply the non-hybrid nature-inspired technique on an abstract workflow in which each task has a set of concrete services associated.

Genetic-based techniques for selecting the optimal Web service composition have been proposed in [2, 5, 10, 11, 14, 20–22] (see Table 1). In the approaches presented in Table 1, a genetic individual is mapped on a composition solution encoded using discrete representations (e.g. integer, binary). These genetic-based techniques start from an initial population that is randomly generated. The main differences between these approaches are related to how the individuals' evolution is achieved (i.e. what selection, crossover and mutation operators are applied) and to how the fitness function is defined. Most of the approaches use roulette-based selection operators, one point/two points crossover operators, and random mutation operators.

In [15, 23, 31], Ant Colony Optimization-inspired techniques for selecting the optimal Web service composition have been presented (see Table 2). In these approaches, each ant builds a composition solution in each algorithm iteration by starting from the graph origin and by probabilistically choosing candidate services to be added to its partial solution. The probability of choosing a candidate service

Table 1 Comparative analysis of the state of art genetic-inspired approaches for selecting the optimal solution in Web service composition

Reference	Concepts encoding	Fitness function	Update strategies
[5]	Solution: integer array	*QoS*, penalty, constraint-based	Roulette-based and elitism selection operator, two point crossover, random mutation
[10]	Solution: n-tuple	*QoS*, penalty-based	Roulette-based selection, uniform and hybrid crossover, random mutation
[21]	Solution: binary	*QoS*-based	Roulette-based selection, one point crossover, random mutation
[22]	Solution: binary	*QoS*-based	Elitism, two-point crossover, random mutation
[14]	Solution: integer array	*QoS*, semantic, penalty, constraint-based	Elitism, multipoint crossover, random mutation
[20]	Solution: hybrid encoding	*QoS*-based	Roulette-based selection, one point crossover, random mutation
[2]	Solution: hybrid encoding	*QoS*, penalty, customer satisfaction-based	Not specified
[11]	Solution: hybrid encoding	*QoS*-based	One point crossover, random mutation

depends on the pheromone level associated to the edge in the abstract workflow connecting the current service to the candidate service and on heuristic information. In [23, 31], the pheromone reflects the *QoS* attributes of the candidate service, while in [15] the pheromone is a numerical value equal for each edge connecting two services in the graph of services. One main difference between these approaches consists in the way the pheromone update is performed. In [23, 31], the update is a two-step process which consists of decreasing the pheromone level associated to each edge by means of an evaporation strategy and of increasing the pheromone level associated to each edge part of promissing solutions. In [15], the update is performed in a single step and depends on a pheromone evaporation rate, the current pheromone level and a chaos operator which aims to improve the convergence speed of the algorithm towards the optimal solution.

Immune-inspired techniques for selecting the optimal Web service composition have been proposed in [25, 26] (see Table 3). In these approaches, the immune concepts (antigen, antibody, affinity, cloning, somatic hypermutation) are mapped on the concepts from the problem of selecting the optimal service composition solution as follows: the antigen is represented as a fitness function, the antibody is represented as a candidate composition solution encoded using discrete representations, the affinity between an antigen and an antibody is represented by the value of the

Table 2 Comparative analysis of the state of art ant colony optimization-inspired approaches for selecting the optimal solution in Web service composition

Reference	Concepts encoding	Fitness function	Update strategies
[31]	Solution: graph path Pheromone: k-tuple	*QoS*-based	Pheromone evaporation and pheromone increase based on *QoS* attribute values Probabilistic choice
[23]	Solution: graph path Pheromone: numerical value	*QoS*-based	Pheromone evaporation and pheromone increase based on *QoS* attribute values Probabilistic choice
[15]	Solution: graph path Pheromone: numerical value	*QoS*-based	Pheromone evaporation and pheromone increase based on *QoS* attribute values Probabilistic choice

fitness function for the associated composition solution, cloning consists of replicating a composition solution, and somatic hypermutation consists of applying a set of updating strategies (e.g. mutation, crossover) on a composition solution to improve its affinity. Besides the affinity between the antigen and an antibody, [25, 26] also model the affinity between two antibodies as a Hamming distance between the associated composition solutions. In [26], the composition solutions having a low concentration in the population have the highest chances to be updated using the genetic crossover and mutation operators. In [25], the proposed algorithm applies two immune operations, one aiming to generate a diverse population of antibodies (i.e. compositin solutions) based on the affinity between antibodies and the antibodies concentration, and one aiming to update the antibodies population using a clonal selection-inspired approach. This approach also relies on a mutation operator which modifies the solution elements that alter its fitness.

3.2 Hybrid Nature-Inspired Techniques

A hybrid method combining Particle Swarm Optimization (PSO) [12] with Simmulated Annealing is proposed in [8] for selecting the optimal or a near-optimal service composition solution based on *QoS* attributes. Authors model service composition using an abstract workflow on which concrete services are mapped. A composition solution is considered as the position of a particle in PSO, while velocity is used to modify a composition solution. To avoid the problem of premature stagnation in a local optimal solution, a Simmulated Annealing-based strategy is introduced which produces new composition solutions by randomly perturbing an initial solution.

Another hybrid method based on PSO is presented in [24] which introduces a non-uniform mutation strategy that aims to modify the global best optimal composition solution for ensuring the exploration of new areas of the search space. In addition,

Table 3 Comparative analysis of the state of art immune-inspired approaches for selecting the optimal solution in web service composition

Reference	Concepts encoding	Fitness function	Update strategies
[26]	Solution: binary string	*QoS* and penalty-based	One point crossover, mutation, affinity between antibodies, antibodies concentration
[25]	Solution: binary string	*QoS*-based	Mutation, affinity between antibodies, antibodies concentration

to improve the convergence speed the authors use an adaptive weight strategy for adjusting the particle velocity. A local best first strategy is used to replace the services having a low *QoS* score from a composition solution with others having a higher *QoS* score.

In [13], authors model the selection of the optimal semantic Web service composition solution as a *QoS* constraint satisfaction problem which they solve using a hybrid algorithm that combines tabu seach with simulated annealing. The Web service composition model is obtained by mapping concrete services on the tasks of an abstract composition. Tabu search is used to generate a neighbour composition plan by replacing some of the services from a given plan. The service replacement strategy implies performing the following steps for each unsatisfied constraint: (1) sort the services that violate a constraint in descending order, (2) replace some of the top services with other services that have the highest evaluation scores, (3) record the service replacements in a tabu list to avoid stagnation in a local optimum [13]. To establish whether a neighbour plan should be accepted or not, a simulated annealing-based strategy is applied which takes into account the plan's degree of constraint violation.

In [16], authors combine genetic algorithms with Ant Colony Optimization to select the optimal composition. In this approach, a Web service composition solution is encoded as a set of binary strings, where a string represents the identification number of a concrete service selected to implement the associated abstract task. Initially, the Ant Colony Optimization is used to generate the initial population of individuals that will be submitted to the genetic algorithm. The genetic algorithm iteratively modifies the population of individuals by applying crossover and mutation operators until a stopping condition is fullfilled.

In [1], a genetic algorithm is hybridized with Tabu Search for performance improvements in the context of selecting the optimal composition solution. In this approach, a composition solution is encoded as an array of integer values, each value indicating the index of the concrete service selected for implementing the associated abstract task. The two-point crossover and the random mutation operators are used to evolve composition solutions from one generation to another. A penalty-based fitness function is used to evaluate a composition solution in terms of *QoS* attributes and constraint satisfaction. The concept of tabu list is used from Tabu Search to store the composition solution from each iteration having the best fitness after the crossover and mutation operators are applied upon a set of solutions. Each solution

stored in the tabu list has an aspiration time associated which decreases with each iteration until it becomes 0. When this happens, the associated solution is deleted from the tabu list and may replace similar solutions from the current population of solutions.

4 The Steps for Developing Hybrid Nature-Inspired Techniques for Selecting the Optimal Web Service Composition Solution

This section presents the steps for developing hybrid nature-inspired techniques for selecting the optimal or a near-optimal solution in Web service composition.

Step 1—Problem Definition. A first step in designing a hybrid nature-inspired selection technique is to choose a proper problem formalization to eliminate the processing overhead. The problem of selecting the optimal solution in Web service composition can be formulated as an optimization problem, as its main objective is to find the appropriate configuration of services such that the optimal or a near-optimal solution with respect to a given fitness function is identified in short time while fullfilling a set of constraints. Formally, the problem of selecting the optimal solution in Web service composition can be defined as:

$$\Pi = (S, F, C) \tag{4}$$

where S is the service compositions search space, F is the set of fitness functions that evaluate the quality of a composition solution sol, and C is a set of constraints.

The set of fitness functions, F, is formally defined as:

$$F = \{F_1(sol), F_2(sol), \ldots, F_n(sol)\} \tag{5}$$

where $F_i(sol) : S \rightarrow \Re$ is an objective function that must be minimized/maximized, subject to the set of constraints C. An objective function may consider the minimization/maximization of a *QoS* attribute in the case of *QoS*-aware Web service composition, or the maximization of the semantic matching between the services involved in a composition solution in the case of semantic Web service composition. If the objectives are normalized in the same interval, then the set of objective functions can be aggregated into a single objective function.

The set of constraints, C, is formally defined as follows:

$$C = [C_1, C_2, \ldots, C_m] \tag{6}$$

Constraints may be related to an atomic service or to the entire composition [2]. The constraints related to an atomic service can be classified as: (*i*) *conflicting constraints* (e.g. if one service is selected then other subsequent services can not be selected for composition [20]), (*ii*) *dependency constraints* (e.g. if one service owned

by a service provider is chosen in the composition solution then in the next steps other services belonging to the same provider must be used [6, 20]), or (*iii*) *data constraints* (e.g. constraints related to the data used by a service [2]). The constraints related to the composition include *composition QoS constraints* (e.g. the global *QoS* value of the composition should be in a specified range) or *semantic constraints* (e.g. the semantic matching score of a composition should be in a specified range).

The goal of an optimization problem is to find either the global optimal service composition solution or a near optimal service composition solution.

Step 2—Developing a Hybridization Model. This step for developing a hybridization model for selecting the optimal service composition can be decomposed into the following four sub-steps: (1) identify the nature-inspired metaheuristics suitable for solving the selection problem, (2) analyze the chosen metaheuristics to identify how they can be improved in terms of execution time and solution quality through hybridization with concepts/strategies from other metaheuristics, and (3) map the metaheuristics' concepts to the concepts of the selection problem domain.

Step 3—Developing a Hybrid Selection Algorithm. This step implies adapting and enhancing an existing nature-inspired metaheuristic to the problem of selecting the optimal Web service composition solution by injecting the components of the hybrid nature-inspired model in the existing nature-inspired metaheuristic.

Step 4—Performance Evaluation. The convergence of a metaheuristic algorithm towards the optimal solution is influenced by the values of its adjustable parameters. The proper choice of the values for the adjustable parameters ensures a good balance between exploration and exploitation. To identify the optimal values of the adjustable parameters the following two steps are addressed: (1) an exhaustive search of the composition model is performed to identify the optimal composition solution's score which will be further used to fine tune the values of the adjustable parameters, and (2) different strategies for controlling the values of the adjustable parameters are applied (e.g. trial-and-error, algorithmic parameter tuning) [7].

The following sections present how these steps are applied to develop the Hybrid Cuckoo Search-based and Hybrid Firefly-based techniques.

5 Formal Definition

This section presents how we have instantiated the elements of the formal definition (see Eq. (4)) of the Web service composition selection problem presented in the first step of the method introduced in Sect. 4 of this chapter. In our case, the service composition search space is modeled as an enhanced planning graph defined as a set of layers:

$$EPG = \{(SC_0, PC_0), \ldots, (SC_n, PC_n)\} \tag{7}$$

where SC_i is a set of service clusters and PC_i is a set of parameters clusters from layer i of the EPG. A service cluster groups services with similar functionalities, while a parameter cluster groups similar input and output service parameters. In our

approach, two services/parameters are similar if the degree of match between their semantic descriptions is higher than a user-defined threshold. The EPG construction is triggered by a composition request issued by a user, specified as a set of ontology concepts semantically describing the provided inputs and requested outputs. The concepts annotating the provided inputs are organized in clusters in the set PC_0 of the first EPG layer. The set SC_0 of service clusters of the first layer is empty. At each step, the EPG is extended with a new layer which contains: (1) a set of service clusters having their inputs provided by the services from the previous layers, and (2) a set of parameters clusters obtained by extending the previous parameters clusters with the output parameters of the services from the current layer. The construction of the EPG ends when the user provided outputs are found in the set of parameters clusters from the current layer, or when the EPG can not be extended with new layers. A feasible solution encoded in the EPG is composed of a set of services providing one service from each cluster of each EPG layer.

To evaluate a composition solution, we define a fitness function QF which aggregates a set of objectives, normalized in the same interval of values: the QoS attributes of the services involved in a composition solution as well as the semantic quality of the connections between these services. The QF equation is defined as:

$$QF(sol) = \frac{w_{QoS} * QoS(sol) + w_{Sem} * Sem(sol)}{(w_{QoS} + w_{Sem}) * |sol|} \tag{8}$$

where: (i) $QoS(sol)$ [18] is the QoS score of the composition solution sol, (ii) $Sem(sol)$ [18] is the semantic quality score of the composition solution sol, and (iii) w_{QoS} and w_{Sem} are the weights corresponding to user preferences related to the relevance of QoS and semantic quality.

In our approach, the only constraint for the problem of selecting the optimal Web service composition is to build feasible composition solutions.

6 Hybridization Model

This section addresses the second step of the method presented in Sect. 3 of this chapter by describing the components used to hybridize the Cuckoo Search and the Firefly algorithms (see Fig. 2). The two metaheuristics are hybridized with the same components to better evaluate the hybridization impact on each of them.

6.1 Core Components

The core components of the hybridization model are the Cuckoo Search-based component and the Firefly-based component. Each of these core components will be

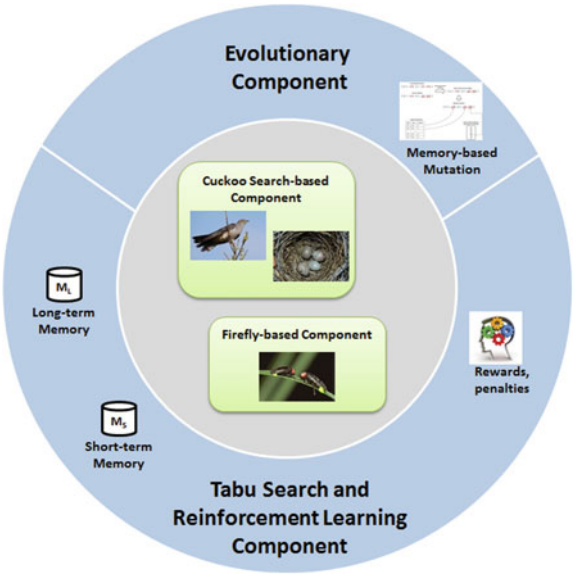

Fig. 2 Hybridization Model Components

hybridized with the components presented in the next sub-section resulting in two hybrid nature-inspired selection algorithms.

The Cuckoo Search-based component is defined by mapping the concepts from the Cuckoo Search Algorithm to the concepts of selecting the optimal Web service composition solution as: (i) a cuckoo becomes an agent, (ii) the egg becomes a Web service composition solution, (iii) the nest becomes a container of Web service composition solutions, (iv) the forrest in which cuckoos live becomes an EPG structure.

The Firefly-based component is defined by mapping the concepts from the Firefly Algorithm to the concepts of selecting the optimal Web service composition solution as: (i) a firefly becomes an agent, (ii) the position of a firefly becomes a Web service composition solution, (iii) the brightness of a firefly becomes the quality of a solution evaluated with a fitness function, (iv) the attractiveness between two fireflies becomes the similarity between two composition solutions, (v) the movement of a firefly is mapped to a modification of the firefly's current composition solution, (vi) the environment in which fireflies fly is mapped to an EPG structure.

6.2 Hybridization Components

This section presents the components that will be used to hybridize the core components introduced in the previous sub-section.

Tabu Search and Reinforcement Learning Component. The tabu search and reinforcement learning component improves the search capabilities of the agents,

used by the core components, by means of long-term and short-term memory structures borrowed from tabu search [9]. The long-term memory (see Eq. (9)) contains the history of service replacements and the associated rewards and penalties and is consulted each time a new Web service composition solution is improved:

$$M_L = \{m_l \mid , m_l = (s_i, s_j, rlScore)\} \tag{9}$$

where s_i is the service that has been replaced by s_j and *rlScore* is the reinforcement learning score used for recording rewards and penalties for the tuple (s_i, s_j). The concepts of rewards and penalties are borrowed from the reinforcement learning technique. The *rlScore* value of a service replacement is updated each time the specified replacement is used to modify a solution. If the replacement improves the quality of a solution then the *rlScore* is increased (a reward is granted), otherwise the *rlScore* is decreased (a penalty is granted). The short-term memory structure contains the set of solutions that are set as tabu and can not be used in the process of generating new composition solutions in the current iteration. In the Hybrid Cuckoo Search-based Model, the short-term memory stores containers in which the agents can not lay any solution, while in the Hybrid Firefly-based Model the short-term memory stores the agents that can not participate in the mating process.

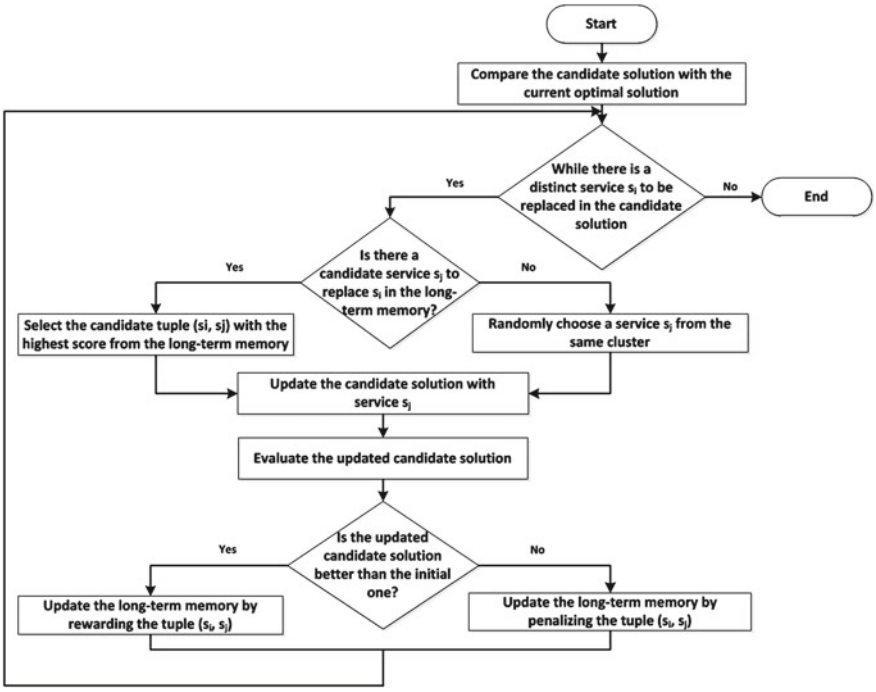

Fig. 3 The processing steps executed when applying the memory-based mutation operator

Evolutionary Component. The evolutionary component introduces a memory-based mutation operator which is applied upon the current optimal solution sol_{opt}, and upon another solution $sol_{current}$ which is submitted to an update process. By applying the memory-based mutation operator, the following steps are executed for each service s_i belonging to $sol_{current}$ and which is not part of sol_{opt} (see Fig. 3):

- The long-term memory is searched for a triple (s_i, s_j, *rlScore*) having the highest *rlScore*. If such a triple is found then s_i is replaced by service s_j in $sol_{current}$, otherwise s_i is replaced with a service s_k randomly chosen from the same cluster.
- The updated $sol_{current}$ is evaluated using the fitness function in Eq. (8) and if its quality is improved (compared to the initial solution) then rewards are granted to the pair (s_i, s_j)/(s_i, s_k), otherwise penalties are granted to this pair.

7 Hybrid Selection Algorithms

This section presents the algorithms obtained by hybridizing the Cuckoo Search and Firefly algorithms with the components of the Hybridization Model discussed in Sect. 6 of this chapter. The algorithms take as inputs an EPG structure resulted from the Web service composition process and the weights w_{QoS} and w_{Sem} showing the relevance of a solution's *QoS* compared to its semantic quality. Additionally, each algorithm takes as inputs some specific parameters which will be further discussed. The algorithms return the optimal or a near-optimal composition solution. Each selection algorithm consists of an initialization stage followed by an iterative stages.

7.1 The Hybrid Cuckoo Search-Based Algorithm

For the Hybrid Cuckoo Search-based Algorithm (see Algorithm 1) we have considered four specific parameters: the number *nmbConts* of containers, the number *nmbAgs* of agents, the number *nmbRep* of containers that will be destroyed, and the maximum number of iterations *maxIt*. In the initialization stage, each container is populated with a randomly generated composition solution (line 4), and the current optimal solution is identified (line 5). The initial solutions generated in this stage are further improved in the iterative stage which is executed until the maximum number of iterations is reached. Each algorithm iteration executes the following sequence of actions: (1) Generate and lay solutions, (2) Replace worst solutions, and (3) Reset memories. Each of these actions is further discussed below.

Generate and Lay Solutions. Each agent in the set of agents randomly selects a container *cont* that is not tabu. Consequently, the selected container is set as tabu. Then, by applying the memory-based mutation operator upon the current optimal solution and *cont*'s solution, a new solution is generated. The new solution will replace the solution of a randomly selected container if its *QF* score is better and the

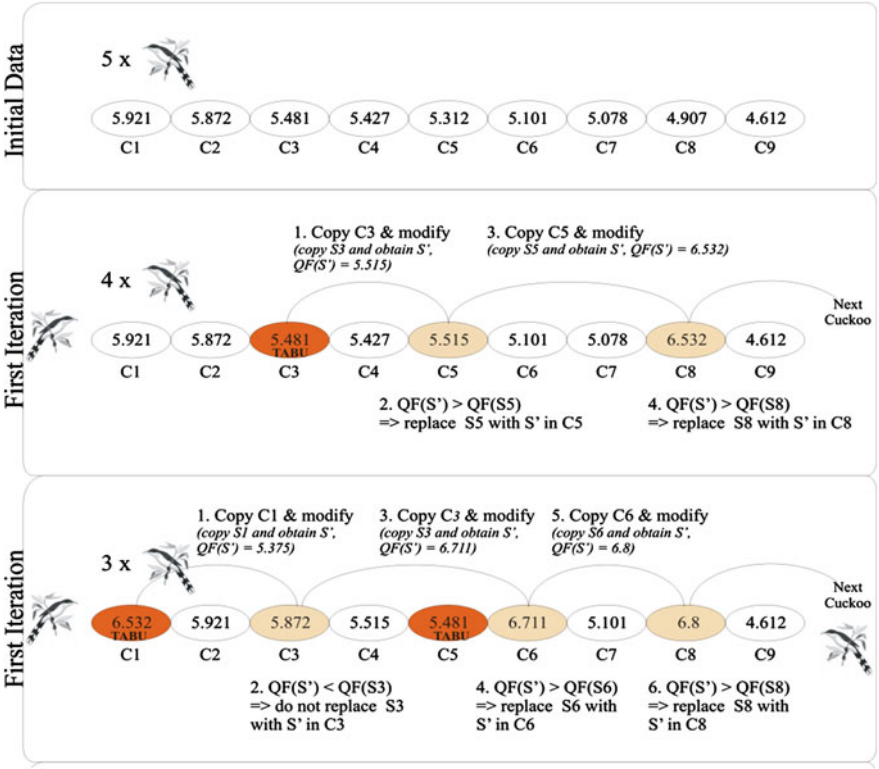

Fig. 4 Example of tracing step 1 for two agents in an algorithm iteration

pairs of services that improve the quality of the new solution will be rewarded. If the new solution is not better than the container's solution, then the pairs of services that negatively affect the quality of the new solution will be penalized. Additionally, the current optimal solution is updated if it is the case. The process of applying the memory-based mutation operator is repeated until all containers are processed. Figure 4 presents the tracing of this step for two agents during an algorithm iteration. In this figure, a population of five agents and eight solution containers are considered. The first cuckoo performs the following operations: (1) Randomly selects the $C3$ container out of containers $\{C1, C2, C3, C4, C5, C6, C7, C8\}$; (2) Sets $C3$ as tabu; (3) Generates a new composition solution S' by applying the memory-based mutation upon $C3$'s solution and the current optimal solution (corresponding to $C1$); (4) Randomly selects $C5$ out of containers $\{C4, C5, C6, C7, C8\}$; (5) Replaces $C5$'s solution with S' since the fitness of S' is higher than the fitness of the $C5$'s solution; (6) Generates a new composition solution S' by applying the memory-based mutation upon $C5$'s solution and the current optimal solution (corresponding to $C1$); (7) Randomly selects $C8$ out of containers $\{C6, C7, C8\}$; (8) Replaces $C8$'s solution with S' since the fitness of S' is higher than the fitness of the $C8$'s solution.

After the first cuckoo finishes the containers' processing, all containers are sorted in descending order based on their solutions' fitness. Then, the second cuckoo performs the following operations: (1) Randomly selects container $C1$ out of containers $\{C1, C2, C3, C4, C6, C7, C8\}$ that are not tabu ($C5$ is tabu); (2) Sets $C1$ as tabu; (3) Generates a new composition solution S' by applying the memory-based mutation upon $C1$'s solution and the current optimal solution (corresponding to $C1$); (4) Randomly selects $C3$ out of containers $\{C2, C3, C4, C6, C7, C8\}$; (5) Does not replace $C3$'s solution with S' since the fitness of S' is lower than the fitness of the $C3$'s solution; (6) Generates a new composition solution S' by applying the memory-based mutation upon $C3$'s solution and the current optimal solution (corresponding to $C1$); (7) Randomly selects $C6$ out of containers $\{C4, C6, C7, C8\}$; (8) Replaces $C6$'s solution with S' since the fitness of S' is higher than the fitness of the $C6$'s solution; (9) Randomly selects $C8$ out of containers $\{C7, C8\}$; (10) Replaces $C8$'s solution with S' since the fitness of S' is higher than the fitness of the $C8$'s solution.

Replace worst solutions. A number *nmbRep* of the containers having the worst solutions are initialized with new randomly generated solutions.

Reset memories. The short-term and long-term memories are emptied.

7.2 The Hybrid Firefly Search-Based Algorithm

For the Hybrid Firefly Search-based Algorithm (see Algorithm 2) we have considered three specific parameters: (i) the number *nmbAgs* of agents used to search for the best composition, (ii) the threshold Δ_c used in crossover-based operations, and (iii) the maximum number of iterations *maxIt*. In the initialization stage, each agent is associated to a randomly generated composition solution (lines 5–7), and the current optimal solution is identified (line 8). The initial solutions generated in this stage are further improved in the iterative stage which is executed until the maximum number of iterations is reached.

In an algorithm iteration, each agent i compares its solution sol_i with the solutions of the other agents that are not tabu. If the score of sol_i is lower than the score of a solution sol_j associated to an agent j, then:

- The distance r between sol_i and sol_j is computed as the difference between the scores of the two solutions (line 15).
- The crossover operator is applied upon sol_i and sol_j in a number of points established based on the value of the distance r (line 16). If $r \geq \Delta_c$, then a lower number of modifications are required as the attractiveness decreases once the distance between agents increases. If $r < \Delta_c$, then a higher number of modifications are required. As a result of applying the crossover operator, two new solutions will be obtained, the one having the highest score (according to the QF function) replacing the agent's i solution having the smallest score.
- The memory-based mutation operator presented in Sect. 6 of this chapter is applied to agent's i solution, sol_i, resulted after applying the crossover operator (line 18).

Algorithm 1 Cuckoo_Web_Service_Selection

```
1  Input: EPG, w_QoS, w_Sem, nmbConts, nmbCucks, nmbRep, maxIt
2  Output: sol_opt
3  Comments: Conts - the set of containers
4  begin
5    Conts = Initialize_Containers(EPG, nmbConts)
6    sol_opt = Get_Best_Solution(Conts)
7    nmbIt = 0
8    M_S = M_L = ∅
9    while (nmbIt < maxIt) do
10     foreach cuckoo in Cuckoos do
11       cont = Select_Container(Conts, M_S)
12       M_S = Update_Short_Term_Memory(M_S, cont)
13       while (More_Containers_to__be_Processed()) do
14         sol_c = Cuck_Generate_Sol(cont, sol_opt, M_L)
15         cont = Cuck_Select_Cont(Conts)
16         sol_old = Get_Cont_Solution(cont)
17         sol_new = Cuckoo_Lay_Sol(sol_c, cont)
18         if (sol_new! = sol_old) then
19           M_L = Reward(sol_new, sol_old, M_L)
20           if (QF(sol_new) > QF(sol_opt)) then
21             sol_opt = sol_new
22           end if
23         else M_L = Penalize(sol_c, sol_old, M_L)
24         end if
25       end while
26       Conts = Sort_Descending(Conts)
27     end foreach
28     Conts = Replace(Conts, nmbRep)
29     Conts = Sort_Descending(Conts)
30     M_S = Reset_Short_Term_Memory(M_S)
31     M_L = Reset_Long_Term_Memory(M_L)
32     nmbIt = nmbIt + 1
33   end while
34   return sol_opt
35 end
```

- If the new solution obtained by applying the memory-based mutation operator improves the score of the initial solution (the solution before the memory-based mutation operator is applied), then rewards are granted, otherwise penalties are granted. In addition, the current optimal solution is updated.
- The agent j is set as tabu if a randomly generated number is greater or equal to the threshold τ.

8 Performance Evaluation

This section addresses the fourth step of the method introduced in Sect. 4 of this chapter refering to the performance evaluation of the proposed hybrid nature-inspired

Algorithm 2 Firefly_Web_Service_Selection

```
1  Input: EPG, w_QoS, w_Sem, nmbAgs, Δ_t, maxIt, τ
2  Output: fSol_best
3  begin
4    FSOL = ∅
5    for i = 1 to nmbAgs do
6      FSOL = FSOL ∪ Gen_Random_Solution(EPG)
7    end for
8    sol_opt = Get_Best_Firefly(FSOL)
9    M_L = ∅
10   nmbIt = 0
11   while(nmbIt < maxIt) do
12       for i = 1 to nmbAgs do
13           for j = 1 to nmbAgs do
14               if (QF(FSOL[i]) < QF(FSOL[j])) then
15                   Δ = Compute_Distance(FSOL[i], FSOL[j])
16                   FSOL[i] = Crossover(FSOL[i], FSOL[j], Δ, Δ_c)
17                   sol_aux = FSOL[i]
18                   FSOL[i] = Memory_based_Mutation(FSOL[i], sol_opt, M_L)
19                   if (QF(sol_aux) < QF(FSOL[i])) then
20                       M_L = Reward(FSOL[i], sol_aux, M_L)
21                       if (QF(FSOL[i]) > QF(sol_opt)) then sol_opt = FSOL[i]
22                   else M_L = Penalize(FSOL[i], sol_aux, M_L)
23                   end if
24               end if
25               FSOL = Set_Tabu_Probabilistically(FSOL[j], τ)
26           end for
27       end for
28       nmbIt = nmbIt + 1
29   end while
30   return sol_opt
31 end
```

algorithms. To evaluate the performance of the Hybrid Cuckoo Search-based and Hybrid Firefly-based algorithms we have performed experiments on a set of scenarios from the trip planning domain involving EPG structures of different complexities (see Table 4).

In Table 4, for each scenario are specified: (i) the scenario code, (ii) the EPG configuration with its layers, the clusters of each layer (the number of clusters is given by the cardinality of each set associated to a layer) and the services per cluster (the number of services is given by the value of each element in a set associated to a layer), (iii) the search space complexity in terms of total number of possible solutions encoded in the EPG structure (the number has been obtained by counting the number of solutions generated in an exhaustive search procedure), (iv) the global optimal fitness value identified by performing an exhaustive search, (v) the execution time in which the optimal solution has been found when performing the exhaustive search.

8.1 Setting the Optimal Values of the Adjustable Parameters

To set the optimal values of the adjustable parameters of the proposed algorithms, a trial-and-error strategy was adopted. In what follows, we present fragments of the best experimental results obtained while tuning the values of the adjustable parameters of both algorithms for scenarios S, M and L. The experiments have been performed on a computer having an Intel Core i5 processor with two processor cores and a frequency of 2.66 GHz.

8.1.1 Setting the Optimal Values of the Adjustable Parameters for the Hybrid Cuckoo Search-Based Algorithm

The adjustable parameters of the Hybrid Cuckoo Search-based Algorithm are the following: *nmbConts*—the number of containers, *nmbAgs*—the number of agents,

Table 4 EPG configurations and other additional information

Scenario code	EPG configuration	Search space complexity	Global optimal fitness	Execution time (min:s)
S	Layer 1: {4 5 6} Layer 2: {6 4 6} Layer 3: {4 6 5}	2073600	6.456	3:8
M	Layer 1: {3 5 4 6} Layer 2: {6 4 6 5} Layer 3: {4 6}	6220800	7.482	15:47
L	Layer 1: {6 5 3 3} Layer 2: {4 6 4 3} Layer 3: {6 5 6}	13996800	8.024	56:18

Table 5 Experimental results for scenario S (ordered descendingly by the fitness value)

nmbConts	nmbAgs (%)	nmbRep	Fitness	Time (s)	Deviation
10	5	25	6.456	0.42	0
15	5	25	6.456	0.58	0
20	9	25	6.456	1.06	0
20	15	25	6.456	1.54	0
15	10	25	6.45	0.72	0.06
20	6	25	6.45	0.74	0.06
15	**7**	**25**	**6.45**	**0.97**	**0.06**
10	3	20	6.444	0.32	0.012
10	3	25	6.434	0.32	0.013
10	5	20	6.429	0.42	0.027

Table 6 Experimental results for scenario M (ordered descendingly by the fitness value)

nmbConts	nmbAgs	nmbRep (%)	Fitness	Time (s)	Deviation
20	6	25	7.48	1.08	0.002
20	15	25	7.48	1.33	0.002
15	**7**	**25**	**7.478**	**1.13**	**0.004**
15	5	25	7.47	0.828	0.012
20	9	25	7.47	1.52	0.012
10	5	25	7.46	0.69	0.022
15	10	25	7.46	1.44	0.022
10	3	25	7.45	0.44	0.032
10	3	20	7.44	0.44	0.033
10	5	20	7.44	0.69	0.033

Table 7 Experimental results for scenario L (ordered descendingly by the fitness value)

nmbConts	nmbAgs	nmbRep (%)	Fitness	Time (s)	Deviation
15	**7**	**25**	**8.02**	**1.5**	**0.004**
20	6	25	8.02	1.51	0.004
20	9	25	8.02	1.92	0.004
15	10	25	8.02	2.01	0.004
20	15	25	8.02	3.3	0.004
15	5	25	8.01	1.18	0.005
10	3	25	8	0.63	0.024
10	3	20	8	0.63	0.024
10	5	25	8	0.9	0.024
10	5	20	7.99	0.95	0.025

and *nmbRep*—the percentage of containers that will be destroyed. We have varied the values of *nmbConts* and *nmbRep* as suggested in [28]: $nmbConts \in \{5, 10, 15, 20, 50, 100, 150, 250, 500\}$, $nmbRep \in \{0, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.4, 0.5\}$. Tables 5, 6, 7 present a fragment of the best experimental results obtained while varying the values of the algorithm's adjustable parameters for scenarios S, M, and L (each table row represents an average value of the results obtained while running the algorithm for 100 times). In these tables, *deviation* refers to the fitness difference between the global optimum obtained through exhaustive search and the optimum solution provided by the hybrid algorithm. By analyzing the experimental results it can be observed that the values of *nmbConts* and *nmbRep* recommended in [28] ($nmbConts = 15$, and $nmbRep = 25\,\%$) can be considered as good values for the considered scenarios. Regarding the number of agents, a value of 7 provides good results in all the considered scenarios. The table rows showing the results obtained for the configuration $nmbConts = 15$, $nmbRep = 25\,\%$, and $nmbAgs = 7$ of the adjustable parameters are highlighted in bold in Tables 5, 6, 7.

8.1.2 Setting the Optimal Values of the Adjustable Parameters for the Hybrid Firefly-Based Algorithm

The adjustable parameters of the Hybrid Firefly-based Algorithm are the following: *nmbAgs*—the number of agents, Δ_c—the threshold used to compute the number of crossover points, and τ—the threshold used to set an agent as tabu. We have varied the values of the adjustable parameters as follows: *nmbAgs* value from 15 to 300, Δ_c value in the range [0, 1], and τ value in the range [0, 1]. For seting the range of values for *nmbAgs* we were guided by the values proposed in [27]. Tables 8, 9, 10 illustrate a fragment of the best experimental results obtained while varying the values of the algorithm's adjustable parameters for scenarios S, M, and L (each table row represents an average value of the results obtained while running the algorithm for 100 times). By analyzing the experimental results it can be observed that for $nmbAgs = 65$, $\Delta_c = 0.4$, and $\tau = 0.9$, the algorithm provides good results for all scenarios. The results obtained for this configuration are highlighted as a bold row in Tables 8, 9, 10.

Table 8 Experimental results for scenario S (ordered descendingly by the fitness value)

nmbAgs	Δ_c	τ	Fitness	Time (s)	Deviation
80	0.4	0.9	6.302	0.319	0.154
65	**0.4**	**0.9**	**6.264**	**0.256**	**0.192**
300	0.5	0.5	6.254	0.382	0.202
200	0.5	0.5	6.245	0.272	0.211
100	0.5	0.6	6.242	0.171	0.214
80	0.4	0.9	6.239	0.332	0.217
300	0.5	0.5	6.228	0.372	0.228
130	0.5	0.6	6.226	0.208	0.23
160	0.5	0.6	6.196	0.249	0.26
60	0.3	0.9	6.212	0.27	0.244

Table 9 Experimental results for scenario M (ordered descendingly by the fitness value)

nmbAgs	Δ_c	τ	Fitness	Time (s)	Deviation
60	0.3	0.9	7.367	0.389	0.115
100	0.5	0.6	7.323	0.268	0.159
80	0.4	0.9	7.309	0.518	0.173
65	**0.4**	**0.9**	**7.302**	**0.440**	**0.18**
200	0.5	0.5	7.302	0.413	0.18
300	0.5	0.5	7.289	0.561	0.193
300	0.5	0.5	7.282	0.566	0.2
80	0.4	0.9	7.277	0.516	0.205
160	0.5	0.6	7.275	0. 377	0.207
130	0.5	0.6	7.236	0.318	0.246

Table 10 Experimental results for scenario L (ordered descendingly by the fitness value)

nmbAgs	Δ_c	τ	Fitness	Time (s)	Deviation
65	**0.4**	**0.9**	**7.93**	**0.66**	**0.094**
300	0.5	0.5	7.896	0.879	0.128
60	0.3	0.9	7.895	0.638	0.129
200	0.5	0.5	7.894	0.632	0.13
80	0.4	0.9	7.88	0.808	0.144
130	0.5	0.6	7.879	0.494	0.145
160	0.5	0.6	7.879	0.583	0.145
300	0.5	0.5	7.878	0.870	0.146
100	0.5	0.6	7.875	0.395	0.149
80	0.4	0.9	7.871	0.8	0.153

8.2 Evaluations of Hybridization

This section presents the individual contribution of each hybridization component, part of the hybridization model presented in Sect. 6 of this chapter, to uncover the strength of the proposed Hybrid Cuckoo Search-based and Hybrid Firefly-based algorithms.

Evaluation of Cuckoo Search Hybridization. We have implemented, evaluated and compared the following algorithm versions:

- CS—the classical Cuckoo Search algorithm proposed in [29].
- CSU—a version of the CS algorithm where more agents lay solutions in more than one container in each iteration.
- CSUG—a version of the CSU algorithm where only the evolutionary component is used (without considering the tabu search and reinforcement learning component). Thus, each candidate solution is compared with the optimal one and the services that are different in the candidate solution are randomly changed with other services from the same cluster without consulting the memory structures.
- CSUGTR—a version of the CSU algorithm where the evolutionary component is used together with the tabu search and reinforcement learning component.

In Tables 11, 12, 13 we illustrate the experimental results obtained by the algorithm versions for S, M, and L scenarios (the optimal configurations of adjustable parameters have been considered for all algorithm versions).

Table 11 Hybrid Cuckoo search-based algorithm—results (scenario S)

Algorithm	Fitness	Time (s)	Deviation
CS	5.46	0.06	0.996
CSU	5.87	0.17	0.586
CSUG	6.13	0.2	0.326
CSUGTR	6.45	0.97	0.06

Table 12 Hybrid Cuckoo search-based algorithm—results (scenario M)

Algorithm	Fitness	Time (s)	Deviation
CS	6.47	0.07	1.012
CSU	6.9	0.28	0.582
CSUG	7.1	0.3	0.382
CSUGTR	7.478	1.13	0.004

Table 13 Hybrid Cuckoo search-based algorithm—results (scenario L)

Algorithm	Fitness	Time (s)	Deviation
CS	7.248	0.08	0.776
CSU	7.578	0.44	0.446
CSUG	7.817	0.46	0.207
CSUGTR	8.02	1.5	0.004

Each row in Tables 11, 12, 13 represents an average obtained for 100 runs of an algorithm. By analyzing the experimental results it can be noticed that the hybridization of Cuckoo Search has brought a significant improvement in the fitness of the identified optimal solution with a time penalty resulted from the increased number of operations introduced through hybridization.

Evaluation of Firefly Algorithm Hybridization. We have implemented, evaluated and compared the following algorithm versions:

- FS—the classical Firefly algorithm proposed in [27].
- FSG—a version of the FS algorithm where only the evolutionary component is used (without considering the tabu search and reinforcement learning component).
- FSGTR—a version of the FSG algorithm where the evolutionary component is used together with the tabu search and reinforcement learning component.

In Tables 14, 15, 16 we illustrate the experimental results obtained by the FS, FSG, and FSGTR algorithms for S, M, and L scenarios (the optimal configurations

Table 14 Hybrid firefly-based algorithm—results (scenario S)

Algorithm	Fitness	Time (s)	Deviation
FS	6.243	3.924	0.213
FSG	5.946	3.765	0.51
FSGTR	6.264	0.256	0.192

Table 15 Hybrid firefly-based algorithm—results (scenario M)

Algorithm	Fitness	Time (s)	Deviation
FS	7.032	1.181	0.45
FSG	7.037	5.809	0.391
FSGTR	7.302	0.44	0.18

Table 16 Hybrid firefly-based algorithm—results (scenario L)

Algorithm	Fitness	Time (s)	Deviation
FS	7.593	1.843	0.431
FSG	7.702	9.309	0.322
FSGTR	7.93	0.66	0.094

of adjustable parameters have been considered for all algorithm versions). Each table row represents an average obtained for 100 runs of an algorithm.

By analyzing the experimental results it can be noticed that the hybridization of the Firefly Algorithm has brought an improvement in the fitness of the identified optimal solution with a significant reduction of the execution time.

9 Conclusions

In this chapter we have presented the Hybrid Cuckoo Search-based and the Hybrid Firefly-based algorithms for selecting the optimal solution in semantic Web service composition. The proposed algorithms combine principles from population-based meta-heuristics with principles from trajectory-based meta-heuristics and reinforcement learning to optimize the search process in terms of execution time and fitness value reflecting the *QoS* and semantic quality of a solution. The selection algorithms are applied on a service composition search space encoded as an enhanced planning graph which is dynamically generated for each user request. To justify the need for hybridization we have comparatively analyzed the experimental results provided by the proposed selection algorithms with the ones provided by the Cuckoo Search and Firefly algorithms. Experimental results prove that: (1) by hybridizing the Cuckoo Search algorithm, a significant improvement in the fitness of the identified optimal solution is obtained with an execution time penalty, and (2) by hybridizing the Firefly Algorithm, an improvement in the fitness of the identified optimal solution is obtained with a significant reduction of the execution time.

References

1. Bahadori, S., Kafi, S., Far, K.Z., Khayyambashi, M.R.: Optimal Web service composition using hybrid GA-TABU search. J. Theor. Appl. Inf. Technol. **9**(1), 10–15 (2009)
2. Batouche, B., Naudet, Y., Guinand, F.: Semantic web services composition optimized by multi-objective evolutionary algorithms. In: Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services, pp. 180–185 (2010)
3. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. **35**(3), 268–308 (2003)

4. Blum, C., Puchinger, J., Raidl, G.R., Roli, A.: Hybrid metaheuristics in combinatorial optimization: a survey. Appl. Soft Comput. J. **11**(6), 4135–4151 (2011)
5. Canfora, G., Penta, M., Di Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Proceedings of the 2005 Conference on Genetic and, Evolutionary Computation, pp. 1069–1075 (2005)
6. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: A framework for QoS-aware binding and re-binding of composite web services. J. Syst. Softw. **81**(10), 1754–1769 (2008)
7. Crepinsek, M., Liu, S., Mernik, M.: Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput. Surv. **45**(3), pp. 35 (2013)
8. Fan, X., Fang, X.: On optimal decision for QoS-aware composite service selection. Inf. Technol. J. **9**(6), 1207–1211 (2010)
9. Glover, F., Laguna, M.: Tabu search. Kluwer Academic Publishers, Norwell, MA, USA (1997)
10. Jaeger, M.C., Muhl G.: QoS-based selection of services: the implementation of a genetic algorithm. In: Proceedings of the 2007 ITG-GI Conference on Communication in Distributed Systems, pp. 1–12 (2007)
11. Jiang, H., Yang, X., Yin, K., Zhang, S., Cristoforo, J.A.: Multi-path QoS-aware web service composition using variable length chromosome genetic algorithm. Inf. Technol. J. **10**, 113–119 (2011)
12. Kennedy, J., Eberhart, R.C.: Particle swarm optimization, pp. 1942–1948. In: Proceedings of IEEE International Conference on Neural Networks (1995)
13. Ko, J.M., Kim, C.O., Kwon, I.H.: Quality-of-service oriented web service composition algorithm and planning architecture. J. Syst. Softw. **81**(11), 2079–2090 (2008)
14. Lecue, F.: Optimizing QoS-aware semantic web service composition. In: Proceedings of the 8th International Semantic Web Conference, pp. 375–391 (2009)
15. Li, W., Yan-xiang, H.: Web service composition algorithm based on Global QoS optimizing with MOCACO. In: Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing, Lecture Notes in Computer Science 6082/2010, pp. 218–224 (2010)
16. Liu, H., Zhong, F., Ouyang, B., Wu, J.: An approach for QoS-aware web service composition based on improved genetic algorithm. In: Proceedings of the 2010 International Conference on Web Information Systems and Mining, pp. 123–128 (2010)
17. Ming, C., Zhen-wu, W.: An approach for web services composition based on QoS and discrete particle swarm optimization. In: Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed, Computing, pp. 37–41 (2007)
18. Pop, C.B., Chifu, V.R., Salomie, I., Dinsoreanu, M.: Immune-inspired method for selecting the optimal solution in web service composition. In: Resource Discovery, Lecture Notes in Computer Science vol. 6162, pp. 1–17 (2010)
19. Salomie, I., Cioara, T., Anghel, I., Salomie, T.: Distributed computing and systems. Albastra Publishing House, Cluj-Napoca, Romania (2008)
20. Tang, M., Ai, L.: A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition. In: Proceedings of the 2010 World Congress on, Computational Intelligence, pp. 1–8 (2010)
21. Vanrompay, Y., Rigole, P., Berbers, Y.: Genetic algorithm-based optimization of service composition and deployment. In: Proceedings of the 3rd International Workshop on Services Integration in Pervasive, Environments, pp. 13–18 (2008)
22. Wang, J., Hou, Y.: Optimal web service selection based on multi-objective genetic algorithm. In: Proceedings of the International Symposium on Computational Intelligence and Design, pp. 553–556 (2008)
23. Wang, X.L., Jing, Z., Yang, H.: Service selection constraint model and optimization algorithm for web service composition. Inf. Technol. J. **10**, 1024–1030 (2011)
24. Wang, W., Sun, Q., Zhao, X., Yang, F.: An improved particle swarm optimization algorithm for QoS-aware web service selection in service oriented communication. Int. J. Comput. Intell. Syst. **3**(1), 18–30 (2010)

25. Xu, J., Reiff-Marganiec, S.: Towards heuristic web services composition using immune algorithm. In: Proceedings of the International Conference on Web Services, pp. 238–245 (2008)
26. Yan, G., Jun, N., Bin, Z., Lei, Y., Qiang, G., Yu, D.: Immune algorithm for selecting optimum services in web services composition. Wuhan Univ. J. Nat. Sci. **11**, 221–225 (2006)
27. Yang, X.-S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press, Frome, United Kingdom (2008)
28. Yang, X.S., Deb, S.: Cuckoo Search via Levy flights. In: Proceedings of the World Congress on Nature and Biologically Inspired, Computing, pp. 210–214 (2009)
29. Yang, X.-S.: Engineering Optimization: An Introduction with Metaheuristic Applications. Wiley, Hoboken, USA (2010)
30. Yang, X.S., Cui, Z., Xiao, R., Gandomi, A.H., Karamanoglu M.: Swarm Intelligence and Bio-inspired Computation: Theory and Applications. Elsevier, Amsterdam, The Netherlands (2013)
31. Zhang, W., Chang, C.K., Feng T., Jiang, H.: QoS-based dynamic web service composition with ant colony optimization. In: Proceedings of the 34th Annual Computer Software and Applications Conference, pp. 493–502 (2010)

Geometric Firefly Algorithms on Graphical Processing Units

A. V. Husselmann and K. A. Hawick

Abstract Geometric unification of Evolutionary Algorithms (EAs) has resulted in an expanding set of algorithms which are search space invariant. This is important since search spaces are not always parametric. Of particular interest are combinatorial spaces such as those of programs that are searchable by parametric optimisers, providing they have been specially adapted in this way. This typically involves redefining concepts of distance, crossover and mutation operators. We present an informally modified Geometric Firefly Algorithm for searching expression tree space, and accelerate the computation using Graphical Processing Units. We also evaluate algorithm efficiency against a geometric version of the Genetic Programming algorithm with tournament selection. We present some rendering techniques for visualising the program problem space and therefore to aid in characterising algorithm behaviour.

Keywords CUDA · Visualisation · Combinatorial optimisation · GPU · Geometric

1 Introduction

Evolutionary algorithms continue to provide an important means of tackling optimisation problems. A group of such algorithms related to the Firefly approach [1] are particularly powerful, especially when accelerated using modern parallel processing hardware such as graphical processing units (GPUs).

Geometric Evolutionary Algorithms are recent additions to the global optimiser family introduced by Moraglio in 2007 [2]. The methodology introduced by Moraglio has already been used to derive the geometric counterparts of Differential Evolution for Hamming space [3], and program space [4], as well as a Geometric Particle Swarm

A. V. Husselmann · K. A. Hawick (✉)
Department of Computer Science, Massey University, Auckland, New Zealand
e-mail: k.a.hawick@massey.ac.nz

A. V. Husselmann
e-mail: a.v.husselmann@massey.ac.nz

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_12,

Optimiser (GPSO) [5]. This unification among EAs has resulted from a rigorous mathematical generalisation by Moraglio, who created a framework providing a method by which an EA can be generalised to arbitrary search spaces, provided that a space-specific algorithm is derived from the generalisation to the desired search space [2].

Unlike the formal derivations of Moraglio and colleagues in these papers, we follow an informal path to obtain an effective combinatorial Firefly Algorithm for k-expression program space. This algorithm could then be a very suitable candidate for generalisation to other search spaces, particularly Hamming space, as this is the metric space we use to denote distances between candidates. The formal derivation of a Geometric Firefly Algorithm to Hamming space is out scope in this chapter. Our analysis of the algorithm covers parallelisation over Graphical Processing Units (GPUs) of the new algorithm, and also an analysis of how the exponential decay affects convergence rates.

Among the Geometric Evolutionary Algorithms already proposed, Togelius et al. in 2008 introduced Particle Swarm Programming (PSP) [6] as the amalgamation of the Geometric Particle Swarm Optimiser (GPSO) and program space in the spirit of Genetic Programming. The authors in that paper [6] note that an initial proposal of this nature is unlikely to outperform the majority of algorithms already highly specialised within parametric and combinatorial optimisation; Poli et al. [7] also note that it is too early to be certain whether Geometric unification is a good approach. We further the work within this area by examining combinatorial optimisers as opposed to parametric ones.

We use several pieces of apparatus to accomplish the goal of a suitable Firefly Algorithm (FA) for use in conjunction with GPU hardware and expression trees. Before discussing these, we provide a brief overview of population-based combinatorial optimisation in Sect. 2. We omit an extensive overview of the Firefly Algorithm, and instead refer the reader to Yang's original work on the Firefly Algorithm [1]. As for the apparatus we use: firstly, we require an appropriate representation for the candidate expression trees in Hamming space, which we discuss in Sect. 3. We also require parallel considerations for the canonical Firefly Algorithm, which we discuss in Sect. 4 along with relevant background information on scientific GPU programming.

The original Firefly Algorithm combines contributions from the entire population for one candidate, therefore, in order to facilitate a fair fitness-based contribution from all candidates in crossover (hence an n-parent "global crossover"), we also require additional apparatus which we discuss in Sect. 5. Finally, we draw together Sects. 3 and 4 and present our parallel, expression-tree Firefly Algorithm in Sect. 5. In Sect. 6 we discuss how we evaluate the efficacy of this new algorithm and we consider some areas for further work in Sect. 9 prior to offering some conclusions in Sect. 10.

2 Evolutionary Population-Based Combinatorial Optimisation

The biological metaphor of evolution has been used extensively throughout the Evolutionary Algorithms (EAs) literature. John Holland's initial work towards the Genetic Algorithm [8] has resulted in a distillation of the core process in natural adaptation: selection, crossover and mutation. These three phases are often known as genetic "operators", and collectively drive a population of individuals, known as "candidates" (sometimes "chromosomes") towards better solutions. These candidates are composed of binary numbers or real numbers. Koza specialised this algorithm to one which operates on the search space of programs, and invented Genetic Programming (GP) [9]. These candidates would be composed of a series of terminal symbols and function symbols, traditionally in a tree structure. The genetic operators as used in the literature by Holland and Koza are described below. Candidates are usually trees or linear sets of instructions made up of abstract symbols sometimes known as codons.

The typical lifecycle of GP is to first initialise a random set of individuals and compute their fitness using some objective function. This function could be as simple as a polynomial (Symbolic Regression [10]) or program-space search with a stochastic evaluation function (Santa Fe Ant Trail [11]). Then, the genetic operators are run on the population, generating a new set of individuals which are evaluated, and then again passed to the genetic operators. This process then repeats.

The Genetic Algorithm [8] and Genetic Programming [9] algorithms are essentially sophisticated search algorithms. They are complex in the sense that stochastic influences among candidates compound continuously and bias each other toward the global optimum.

Augusto and Barbosa state that there are only two criteria to use GP, which also make them highly applicable in many circumstances [12]. The first is that the solution to the problem can be represented in the form of a program, and the second is there exists a metric which can evaluate two arbitrary candidates, and indicate which is the better candidate solution. Keeping these two criteria in mind, it is easy to see why GP has gained such great research interest.

Crossover

The **crossover operator** deals with the genetic recombination of two candidates. In the traditional representation of abstract syntax trees (ASTs), this is done by performing a subtree swap on the two candidate trees on a random point, named the "crossover site". This attempts to construct two new candidates which can then be used in the subsequent computations. This is analogous to the biological process of genetic crossover.

Usually the actual methodology of this operator is dependent on the representation of the candidates. For example, the representation of Linear Genetic Programming

(LGP) [13] is a finite sequence of instructions which can be executed serially. Performing crossover between two of these candidates usually involves selecting a point, and then exchanging the two sequences of instructions about that point with the corresponding sequences of the other candidate. The result is a set of two candidates from the input set of two candidates, which are different.

Selection

The **selection operator** is used to select two candidates to be given to the crossover operator as inputs. There are several ways of accomplishing this, the traditional method being Roulette wheel selection, where the probability of selecting a candidate is based on its fitness. Tournament selection is another method, where n tournaments are held, where two (or more) candidates are chosen in a uniform random fashion, and then the candidate with the lowest fitness is discarded.

Mutation

The **mutation operator** is the simplest, in which candidates are perturbed in solution space, effectively introducing diversity. This serves as a method for exploring solution space thoroughly. Without this operator, the algorithm is only able to cause a drift, and it is unlikely the global optimum will ever be discovered.

These operators can only be usefully applied however, if the problem in general and the program solution in particular can be cast into an appropriate genetic form.

3 Candidate Representation Using *Karva*

One of the central pragmatic problems in using genetic programming is finding an appropriate form of representation of the problem so that GP operators can be applied. *Karva* is a language introduced by Ferreira in her work proposing the Gene Expression Programming (GEP) algorithm [14, 15]. Karva provides a solution to the problematic issue with Genetic Programming algorithms and other combinatorial optimisers of representing the candidate solutions in genetic operable form. This often dictates which genetic operators can actually be applied across the population, or at least, their efficacy in producing a steady flow of diversity and fitness increase. The original Genetic Programming algorithm by John Koza [9] operated on the space of LISP-style S-expressions, where these expressions denote abstract syntactic trees which are often decision trees or mathematical expressions.

Alternative representations have been proposed in the past, such as Linear Genetic Programming (LGP) [13], where candidates are stored as linear sequences of

instructions which are executed sequentially. Research into better representations are on-going, since LGP also suffers from the Halting Problem [16].

The advantage behind the representation language of GEP is its ability to effortlessly handle any-arity functions, variable length, and introns. Expressions based on Karva in GEP are known as k-expressions, and are constituted by a set of codons in a specific order.

These can be visualised as follows:

```
0123456789012345 67
Q-+/-abaa+a-bbacda
```

The symbols `Q`, `+`, `-` and `/` are known as functions (where `Q` is the square root function), and the symbols `a`, `b`, `c` and `d` (arbitrary constants) are known as terminals. The line of digits above the chromosome is simply an indexing convenience. The sequence of symbols shown is a genotype, meaning that is must first be interpreted before being executed. The interpretation of this (or "phenotype") is shown in Fig. 1. It is worth noting that neither of the symbols `c` or `d` appear in the phenotype. The interpretation is constructed simply by placing the first symbol at the root of the tree, and then filling arguments of the tree level by level, left to right advancing through the sequence, symbol by symbol.

Though these candidates would be kept at a constant length in the population, the size of the trees depend on the interpretation and the head length, position and number of function symbols. The chromosome must be divided into head and tail sections, which have their length governed by the equation of Ferreira [15] shown in Eq. (1).

$$t = h(n_{max} - 1) + 1 \tag{1}$$

Here, t is the length of the tail section of the expression tree, and h is the length of the head section, and n_{max} is the largest arity possible. Essentially this equation prevents the expression tree not having enough terminals to satisfy the phenotype. Only terminal symbols are allowed in the tail section.

While there are several crossover operators proposed in Ferreira's work of 2006 [15], we provide an example of the simplest for clarity. Consider the following k-expression:

```
01234567890123456
*-+*-abaa/abbacda
```

When this expression is recombined with the following, with a crossover site at index 5:

```
01234567890123456
*-++baa/bbabbacda
```

Then the two candidates generated are shown below:

```
01234567890123456
*-++babaa/abbacda
*-+*-aa/bbabbacda
```

Interpretations of these candidates are shown in Figs. 2 and 3. It is noteworthy that the structures in these figures differ greatly.

The other types of operators that Ferreira propose [14] are: replication, mutation, insertion sequence transposition, root insertion sequence transposition, gene transposition, 1-point recombination, 2-point recombination, and gene recombination. We focus on 1-point recombination, in the spirit of the GA and GP algorithms. We do however use 1-point mutation, which simply exchanges one symbol for another. Depending on whether this symbol is in the tail or head section, appropriate exchanges are done so that the head and tail sections are still legal.

The term "Memetic Algorithms" (MAs) was introduced in 1989 by Moscato [17] who noted that MAs are not new but have actually been in use for some time.

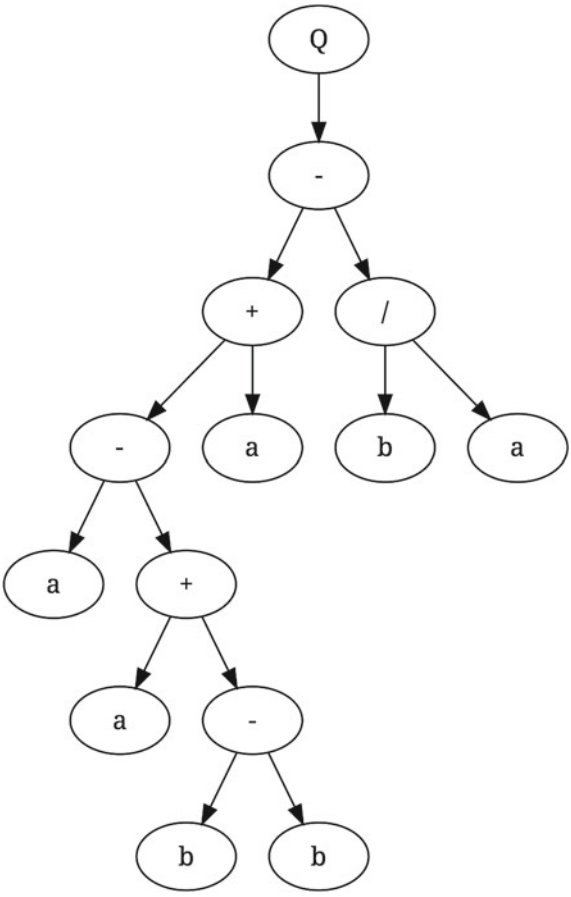

Fig. 1 The abstract syntax tree representing the *karva*-expression `Q-+/-abaa+a-bbacda`

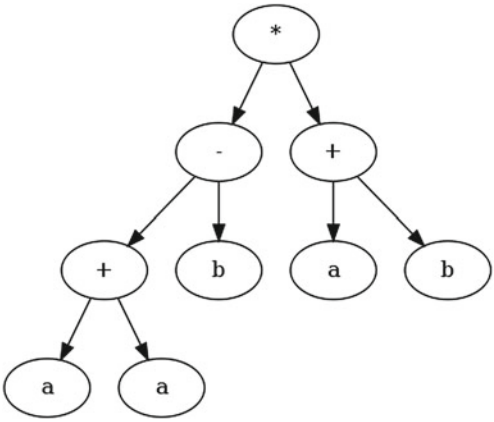

Fig. 2 The abstract syntax tree generated from the genotype given by the *k*-expression `*-++babaa/abbacda`

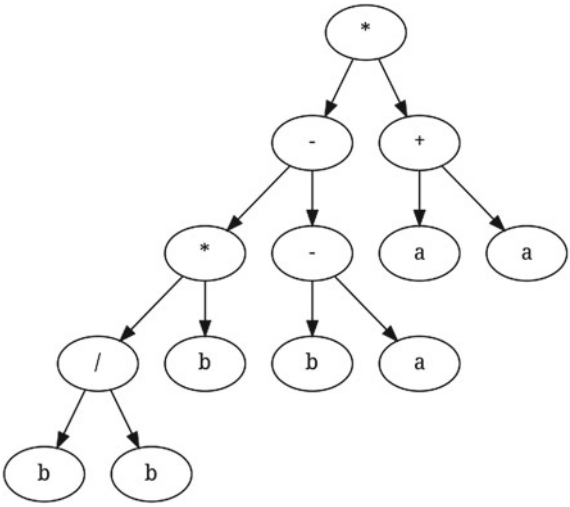

Fig. 3 The abstract syntax tree generated from the genotype given by the *k*-expression `*-+*-aa/bbabbacda`

The name is used in recent research as a term alluding to hybridisation of heuristic algorithms, especially Evolutionary Algorithms [18]. In some ways, the work we present here can be classified as a Memetic Algorithm due to our use of several apparatus introduced by several authors on the subject of metaheuristics.

All the techniques discussed above are potentially demanding of computational time and resources.

4 GPU-Parallel Evolutionary Algorithms

Many optimisation problems have combinatorially large spaces. Parallel processing cannot control the size of these spaces and we still require good, clever algorithms to manage the combinatorial explosion. However, if a good algorithm can also make use of parallelism it can lead to significantly reduced search times. The data parallelism offered by Graphical Processing Units (GPUs) is particularly attractive, both economically and in terms of pragmatic programming reasons.

GPUs have gained increasing interest in the scientific community over the last few years [19]. Before the recent advances of CUDA, researchers were resigned to use shader languages such as Cg for harnessing the power of the GPU [20]. Much of this interest in GPUs is economical, since the peak computing power of a well-engineered implementation on a modern GPU is competitive with some parallel grid computers of far greater cost. Commodity-priced hardware such as these are therefore very attractive as a solution for accelerating scientific applications as they are readily available.

The search for high performance algorithms with regards to Evolutionary Algorithms such as popular metaheuristics and so-called "swarm-intelligence" [1, 21–23] has resulted in excessive use of GPUs, sometimes ignoring more elegant and faster solutions on CPUs [24]. Chitty showed that using loop unrolling, 2D stacks, SSE instructions and cache optimisations that a proper multi-core GP implementation can also achieve a much greater performance improvement over the canonical GP [24].

Representation of GP candidates on GPUs has been a prominent issue. Early attempts used texture memory to store data for GP [20], while others resorted to techniques such as Reverse Polish notation for symbolic regression problems [25]. In this article, we exploit the natural storage characteristics of *Karva* to improve execution efficiency.

Some impressive results have been obtained in the past from specialised Evolutionary Algorithms (EAs) [26–29]. Cano et al. report an impressive 834X speedup against the CPU, and a 212X speedup to a 4-threaded implementation on CPU. There is convincing rationale for pressing forward in the search of high performance algorithms in EA. Schulz et al. [30] assert that combinatorial optimisation on GPUs are still very much an art and under studied. They advocate for a focus on efficiency of implementation and also more complex optimisers as opposed to the relatively simple discrete optimisers on GPU that have been published.

In this article we use Nvidia's Compute Unified Device Architecture (CUDA) software [31] for accelerating our algorithms. Using CUDA is not a new approach for scientific computing and particularly EA, as there have been several EAs proposed in the literature exploiting the nature of GPUs and CUDA specifically [26, 32–34], including even CUDA distributed across several machines in a cluster [35] as well as Genetic Algorithms [36] using a very elegant STL-like library known as Thrust [37]. Our algorithm involves combining multiple GPU parallelisation techniques together however and we describe it below.

The typical lifecycle of a GPU-based algorithm involves copying some data onto the device, executing a sequence of GPU-specific code named a "kernel" and copying the resulting data back to the host. These memory copies across the PCI bus incur an excessive time penalty. However, the benefits gained from using the GPU often outweighs these penalties. CUDA-based algorithms follow this form and exposes these features to developers with syntax extensions to the C language.

Parallelisation of EAs depend on the extent to which they are delegated to multi-threaded hardware. Fitness evaluation phases are particularly well-suited to parallelisation, as all the computations can be done independently. Although difficult and not always effective, it is possible to parallelise the rest of the algorithm in question. However, having mitigated the greatest computational expense of fitness evaluation, it often does not improve performance as greatly. Moreover, the evaluation phase varies extensively in cycles necessary, and may also involve noise, local minima or simply be stochastic.

Algorithm 1 Characteristic CUDA-based genetic algorithm.

```
allocate & initialise space for n candidates

while termination criteria not met do
  copy candidates to device
  execute CUDA kernel
  copy back to host

  if end-of-generation then then
    apply genetic operators to candidates
    replace old candidates with new ones
  end if
  visualise the result
end while
```

The algorithm shown in Algorithm 1 is the typical process that a parallel genetic algorithm would follow. Simple variations of this process involves steady-state EAs [38], Memetic Algorithms [17, 18], and Evolutionary Strategies [39].

In order to prepare for implementing a parallel Firefly Algorithm (FA) to operate on the space of syntax trees in the form of *karva*-expressions, we must consider two additional changes to the process in Algorithm 1. FA operates by having all candidates be aware of one another. Implementing a crossover operator in these circumstances with Karva involves multi-parent recombination. As a result, and which will be made clearer later, we must also consider the use of Roulette selection, which does not parallelise extremely well in general.

Eiben et al. in their work of 1994 [40] and Eiben's work of 1997 [41] introduce a method known as Gene Scanning for multi-parent recombination. Moscato and colleagues also mention multi-parent recombination as a promising area of future study [18]. Eiben introduced several multi-parent crossover operators for binary chromosome Genetic Algorithms. These include uniform scanning, occurrence-based scan-

ning and fitness-based scanning among others. We make a trivial modification to Eiben's fitness-based scanning operator to operate on k-expressions and the symbols they use. The reason we use fitness-based scanning is to ensure that crossover is biased according to fitness values of each candidate. This allows us to steer clear of adding additional operators[1] in order to ensure the algorithm converges.

The fitness-based scanning operator simply selects genes to copy into the candidate under examination from the set of corresponding genes from all the other candidates. Selecting the gene for a particular index of the candidate involves choosing a gene with a probability based on the fitness of the candidate from which the gene is taken. More detail on this can be sought from the work of Eiben et al. in 1994 [40]. In order to accomplish a fitness-based scanning crossover operator, we are forced to use Roulette selection. This involves keeping track of a running total of scaled probabilities, to which a uniform random deviate can be applied to properly select a certain gene. Since this involves ultimately performing $\mathcal{O}(N^2)$ operations per frame, we must attempt to mitigate this, as the memory access pattern and memory bank that would need to be used for this would cause a dramatic drop in performance, just as it would in a sequential implementation; however, it is important to effectively accept and circumvent the restrictions the GPU architecture imposes on the developer in order to take full advantage of the computing power available on these devices. One method which is popular in the literature for improving performance in this case is named tiling, where global memory is essentially paged into CUDA blocks and these pages or "tiles" are shared amongst threads at high speed.

Algorithm 2 Parallel Genetic Programming algorithm operating on k-expressions with fitness-based roulette gene scanning.

```
allocate & initialise space for n candidates on host and GPU

while termination criteria not met do
  copy candidates to device
  CUDA: compute k-expression execution map
  CUDA: evaluate fitness of k-expressions with 50 input sets
  copy back to host

  Report averages

  CUDA: apply gene scanning recombination to candidates with P(crossover)
  CUDA: apply mutation to candidates with P(mutation)
  CUDA: replace old candidates with new ones
end while
```

Further to the issue above, another problem which we must mitigate is that of executing the candidate syntax trees in order to evaluate their efficacy in solving the problem at hand. Typically, these trees would be executed recursively in a sequential CPU implementation. On GPUs and CUDA-enabled devices, recursion is possible,

[1] Effectively turning our algorithm into a Memetic Algorithm [18].

and while stack frames are similar sizes, threads do not have local storage big enough, and stack overflows are almost inevitable. This can be solved by allocating global memory and managing thread stacks explicitly. This, however, introduces an excessive memory overhead, as global memory is extremely slow to access. Therefore, we have implemented a non-recursive tree evaluator which maintains a very small local stack. Since we operate on algebraic expression trees, we aggressively reduce this stack at every opportunity. This ensures that we never run out of memory.

Since k-expressions are the genotypes of their respective phenotypes, we must first interpret the expressions to obtain an executable tree. Since these trees are constructed level by level, we must interpret the expressions by computing the indexes of the arguments of each node or symbol in the tree. This simple process returns an array which gives the index of the first (and second) arguments for each symbol. For brevity, we omit a thorough discussion of this. More detail on this can be found in the authors' work of [11].

At this point, we have essentially implemented a Genetic Programming algorithm operating on *karva*-expressions using Roulette selection with a fitness-based gene scanning crossover operator; this is shown in Algorithm 2. We now present the modifications to the algorithm above to enable the GPU-parallel expression tree Firefly Algorithm.

5 GPU-Parallel Expression-Tree FA

Having presented an algorithm in Sect. 4 suitable for extension by the Firefly Algorithm, we must alleviate another two issues. First, we must decide on a metric space for computing distances between candidates. For simplicity, we use Hamming distances between candidates to represent the structural differences between candidate genotypes.

For convenience, the original update formula for two fireflies i and j in Eq. 2.

$$x_i \leftarrow x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha(d) \tag{2}$$

Having introduced the all-to-all selection mechanism in Sect. 4, we have therefore already incorporated the ability to update each candidate with information available from every other candidate. This now works in our favour, as each candidate must be compared against each other candidate in the original FA. We further incorporate the observed fitness values by exponential decay as follows. Before the recombination by roulette selection occurs, we "observe" the fitness values of all candidates in the context of the candidate the current thread is concerned with. Since we use tiling for paging expression trees from global memory, we use the same technique for paging fitness scores from global memory. Therefore, when a score tile or "page" is loaded from memory, we immediately modify it by the following formula:

$$s_{b'} = s_b e^{-\gamma H(a,b)^2} \tag{3}$$

In this equation, $s_{b'}$ is the updated score of candidate b, s_b is the previous score of candidate b; $H(a, b)$ is the Hamming distance between candidate a and b. Finally, γ defines the shape of the decay curve, which we refer to as the decay parameter.

With these modifications in place, we have now have a fully operational GPU-parallel Firefly Algorithm for expression trees using k-expressions and fitness-based gene scanning for recombination. Pseudocode at the centre of the recombination kernel is shown as,

Listing 12.1 Parallel Firefly update Kernel

```
update_fireflies(scores,programs,newprograms,params) {
  const uint index = <global thread index>
  extern __shared__ unsigned char progtile[];
  __shared__ float scoretile[32];
  load_my_program(myprog);
  float newscoresum = 0.0f;

  // sum all observed scores for each thread
  for (int i=0; i < agent_count/tile_size; ++i) {
    scoretile[threadIdx.x] = observe_fitness(myprog,
        programs + (i*tile_size + threadIdx.x)*maxlen,
        scores[i*tile_size + threadIdx.x],params);
    __syncthreads();
    for (int i=0; i < tile_size; ++i)
      newscoresum += scoretile[i];
    __syncthreads();
  }

  float symbolrand[progsize];
  for (int i=0; i < progsize; ++i)
    symbolrand[i]=<rand between 0 and 1> * newscoresum;

  for (int i=0; i < agent_count/tile_size; ++i) {
    globalindex = (i * tile_size + threadIdx.x)*maxlen;
    for (int j=0; j < maxlen; ++j)
      progtile[threadIdx.x*maxlen+j]=programs[globalindex+j];

    scoretile[threadIdx.x] = observe_fitness(myprog,
        programs + (i*tile_size + threadIdx.x)*maxlen,
        scores[i*tile_size + threadIdx.x], params);
    __syncthreads();
    if (<rand between 0 and 1> < pcrossover)
      for (int j=0; j < tile_size; ++j) {
        running_score += observe_fitness(myprog,
                programs + (i*tile_size + j)*maxlen,
                scoretile[j], params);
        for (int k=0; k < maxlen; ++k) {
          if (symbolrand[k] < running_score) {
            if (scoretile[j] > myscore)
              myprog[k] = progtile[j*maxlen + k];
```

```
            symbolrand[k] = scoresum+1.0f;
          }
        }
      }
    __syncthreads();
  }

  if (<rand between 0 and 1> < pmutate) {
    int pt = <rand int between 0 and maxlen>;
    if (pt < program_head_length)
      myprog[pt] = <rand int between 0 and 4 inc>;
    else
      myprog[pt] = 0; // the only terminal (x)
  }
  for (int i = 0; i < maxlen; ++i)
    newprograms[index * maxlen + i] = myprog[i];
}
```

6 Experimental Methodology

Symbolic Regression is a long standing test base for Genetic Programming algorithms, and serves as a good indication of algorithm effectiveness [9]. Expression tree induction for a polynomial function such as the sextic function is deceptively difficult:

$$f(x) = x^6 - 2x^4 + x^2 \tag{4}$$

In our experiments, we aim to obtain a suitable expression tree for computing $f(x)$, given x, and a candidate expression tree.

We evaluate the effectiveness of one candidate by evaluating the expression tree given 50 separate input/output pairs. The output value is compared to the expected, and if the absolute difference is less than 0.01 then this is counted as 1.0 and added to the fitness of the candidate. Essentially this arrangement allows for some noise in the fitness function, since a particular expression tree may be completely incorrect, and still obtain a non-zero fitness. Therefore, the maximum fitness value is 50 for one test.

We allow chromosome sizes to extend to 64, but head lengths to extend only to 24. The function symbols allowed are $+, -, *, \%$ and the only terminal symbol is x. The search space size is therefore comprised of 4^{24} unique chromosomes, since in this case there is only one terminal symbol. At this point we could only speculate about the shape of the fitness landscape, but we anticipate that the entire spectrum of fitness is reachable within this space.

We compare the algorithm against a CUDA-based Genetic Programming implementation with tournament selection, but still implemented over k-expressions.

The algorithms were configured with 1024 agents, mutation probability of 0.08 and crossover probability of 0.99. For the Tournament-selection GP, however, we selected a crossover probability of 0.8 and mutation probability of 0.1.

7 Selected Results

There are a number of outcomes that can be used to evaluate the efficacy of the approach. In this section we present some generated programs as well as plots of fitness convergence and some visual representations of solutions and associated spaces.

Figure 4 shows a program generated by the algorithm. It resulted in a full score of 50, and when interpreted and reduced, results in $x^6 + 2x^4 - x^2$. This differs in a subtle way from the objective function, in that the $+$ and $-$ operators are swapped. This may well have resulted due to the relatively large error allowed for a successful "hit" of 0.01 in absolute error. Reducing this to a smaller margin of 0.001 may yield a more correct result overall (Fig. 5).

Figure 6 presents 100-run averages of standard deviations, lowest and highest population fitness mean, and average population fitness mean. The data plotted in this graph was generated by setting $\gamma = 0.0$ in Eq. 3, effectively allowing a candidate to perceive the full undegraded fitness values of all other candidates during selection and crossover. For this value of γ, the success rate yielded was an unimpressive 11%.

The plot contains some interesting anomalies at generations 420, 530, 680, 760, 800, and 950. The standard deviations appear to climb instantly at these points, and so do the mean, while the lowest and highest remain the same. We believe this is caused by the algorithm escaping local minima by pure chance, since the lowest

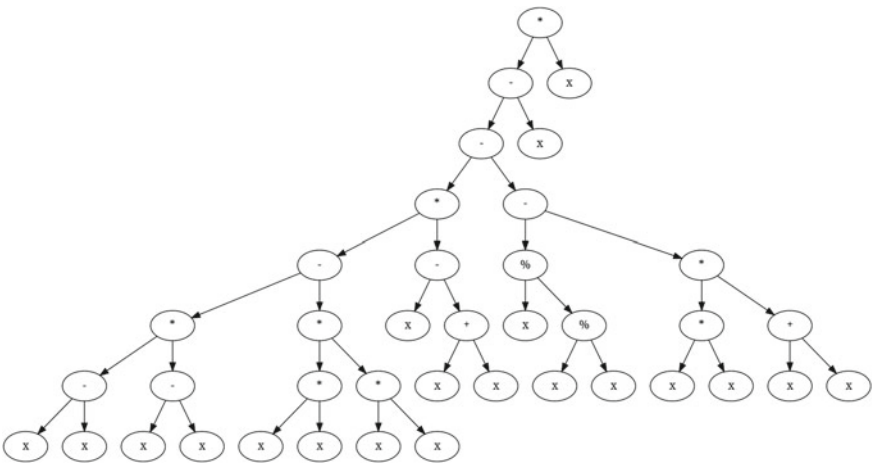

Fig. 4 A sample program generated by the algorithm which yielded a full score of 50

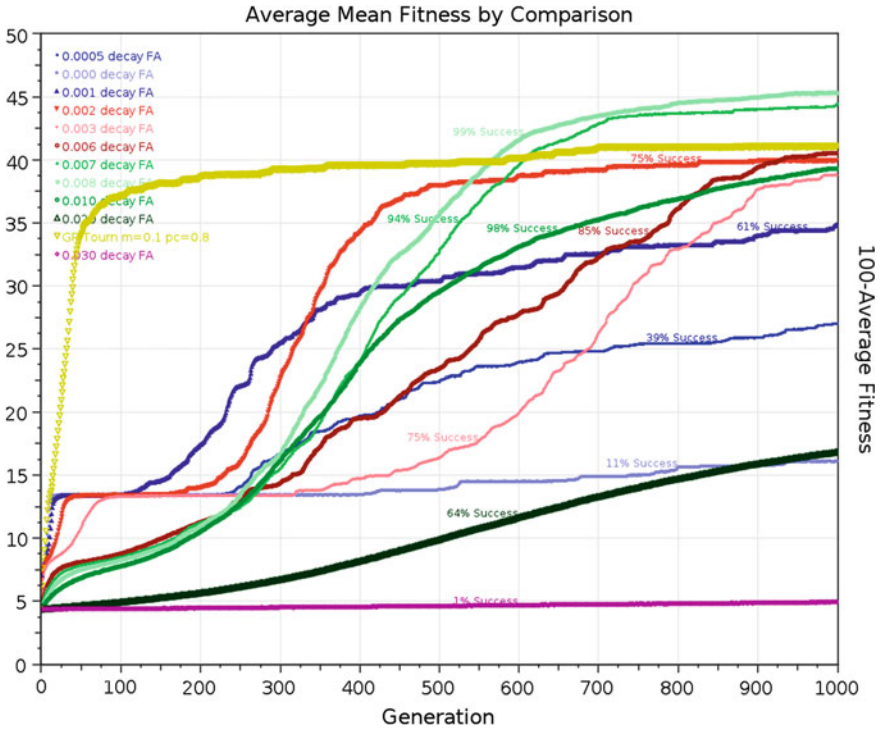

Fig. 5 Comparison plot of different settings of the decay parameter in the algorithm

fitness mean is not affected. Essentially, as in parametric optimisers, this appears to be some kind of premature convergence, where the search would stagnate.

Figure 7 shows 100-run averages of the highest, lowest and mean population fitness by generation, in the same format as Fig. 6. Here, the difference is clear, as the average mean increases steadily. For this value of γ, the success rate was 98 %, as 98 runs succeeded in producing an expression which produced 50 outputs all within 0.01 of the true sextic function.

Figure 5 shows a comparison for different decay values of the algorithm proposed, and also the tournament-selection GP. This graph is somewhat misleading, as the means do not necessarily convey the success of the algorithm and the range of diversity within them by generation. For example, $\gamma = 0.003$ gains a success rate of 75 %, yet $\gamma = 0.02$ which is a similar curve gains a success rate of 98 %. Therefore, we have also provided a 3D plot of success rates, mean fitness and Generation number in Fig. 8. Here, it is more obvious that similar means obtained do not necessarily correspond to a higher success rate.

Decay curves are shown in Fig. 9 for various values of γ. The curve with the highest success rate has been highlighted ($\gamma = 0.008$). It is interesting to note, that the Hamming distance between two candidates can reach to a maximum of 64 in

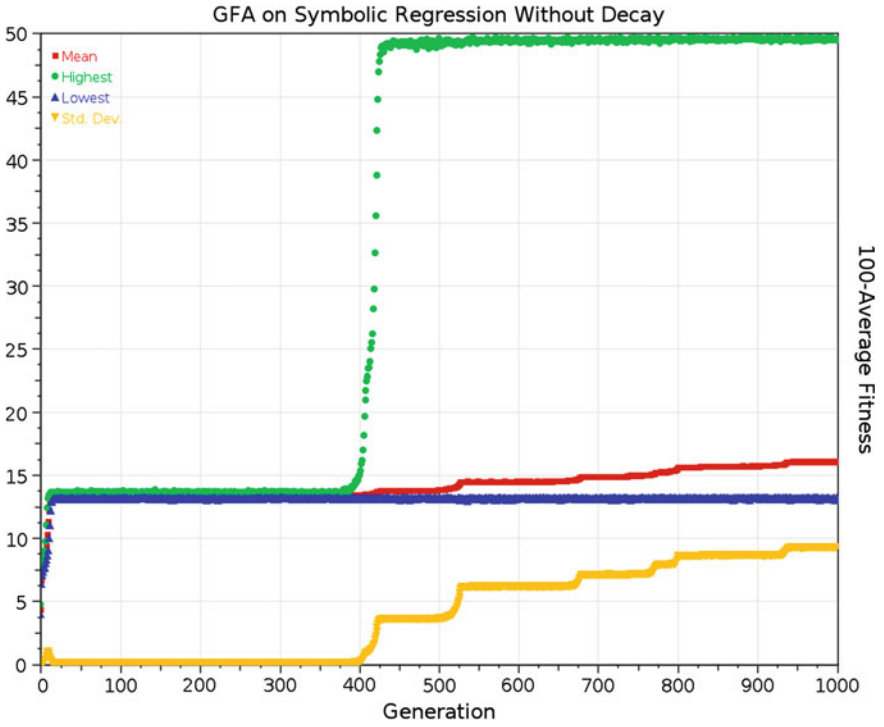

Fig. 6 Fitness plot of the algorithm when observed fitness values from the point of view of one candidate are not decayed at all

this experiment. Even slight changes to the value of γ for this curve makes dramatic changes to the success rate of the algorithm. The effective interaction radius in Hamming space of the algorithm with $\gamma = 0.008$ is reminiscent of appropriate settings of γ in the canonical Firefly Algorithm, where a γ too large would result in stagnation, and a γ too small would suffer from premature convergence. This particular exponential decay appears to maintain an optimal local search neighbourhood around a candidate.

We also present success rates by γ (Decay) value in Fig. 10. At this stage we can only speculate as to the characteristics of this curve, and why it contains certain features. These points are subject to turbulence however, but empirical observation suggests that there is little variability in practice. We believe that this may be related to the head length that we have chosen for candidates.

Figure 11 contains average performance data across the 100 runs of the algorithm. As is expected, initial complexity of the candidates would be high, since the head sections of candidates would be randomly initialised. The result is a decrease in complexity towards generation 500, where average generation evaluation time tends to 12 ms. The average generation compute time is at a near constant 31.8 ms. Our efforts in parallelising this algorithm was worth the effort, as it would generally

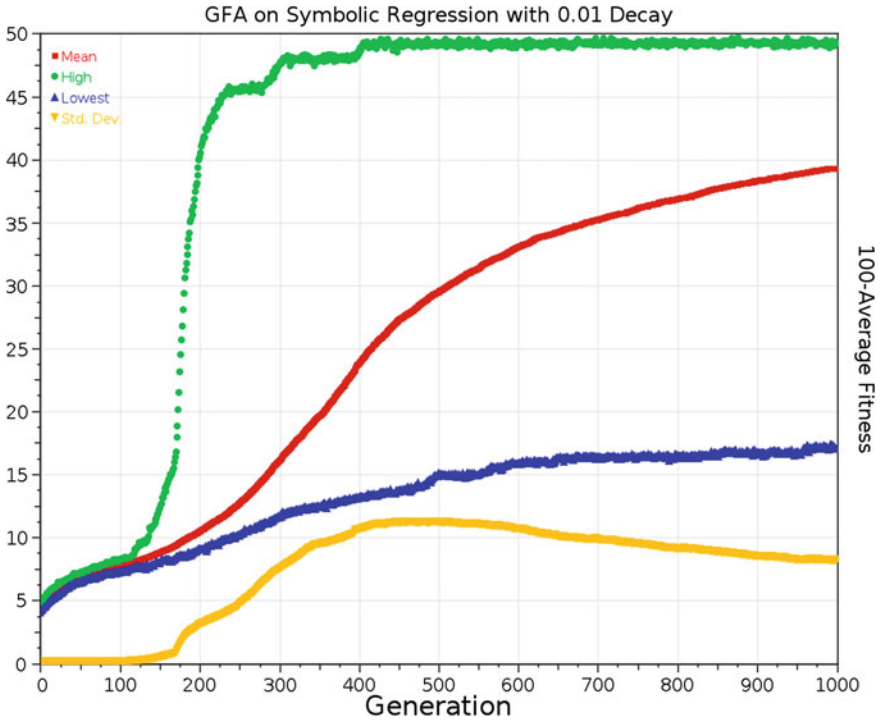

Fig. 7 Fitness plot of the algorithm when the decay parameter is set to 0.01

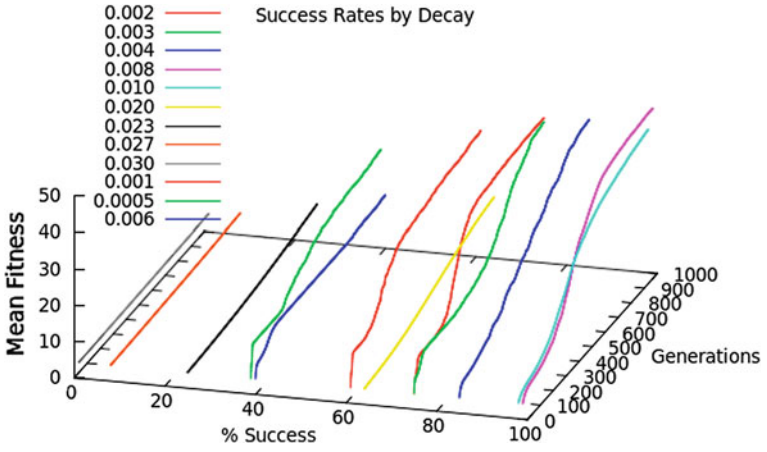

Fig. 8 3D plot of success rates by decay values for each generation over 100 independent runs for different decay values

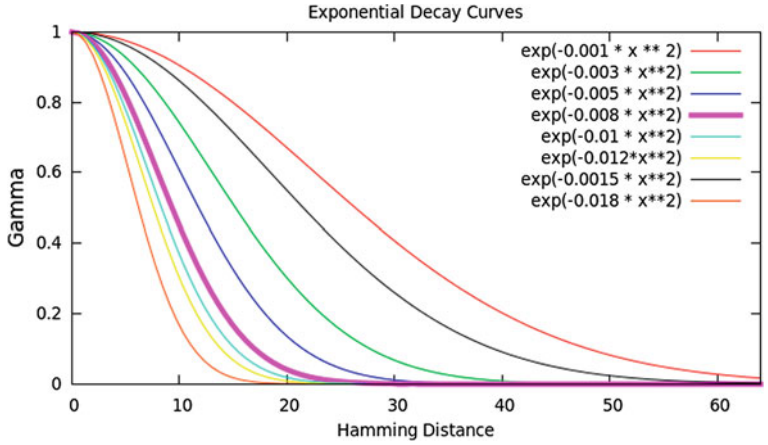

Fig. 9 Sample decay curves produced by various γ values for Hamming distances to candidates observed by the current candidate

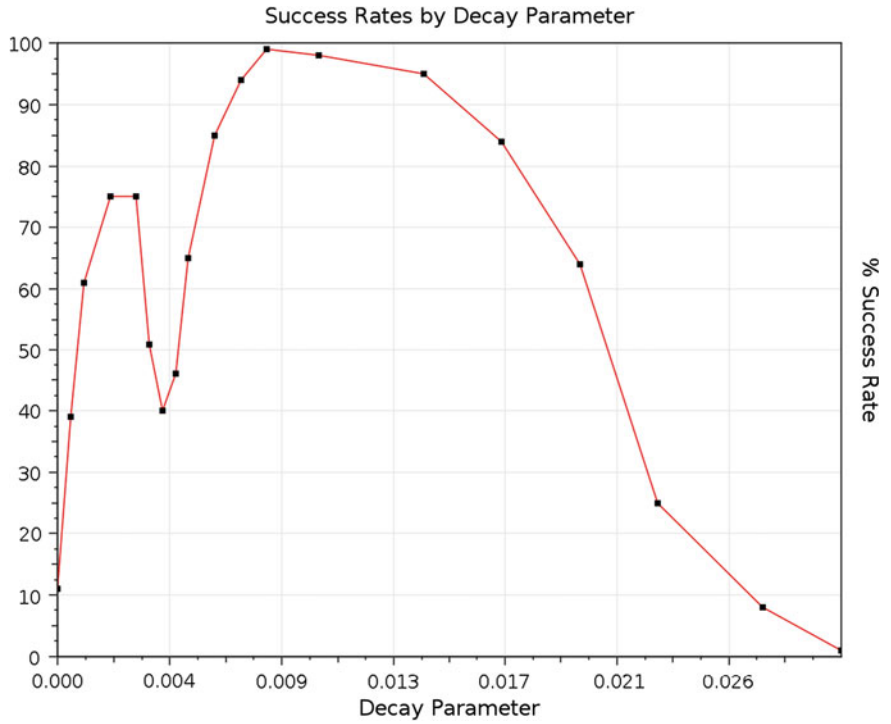

Fig. 10 Success rates over the 100 runs of each decay parameter used

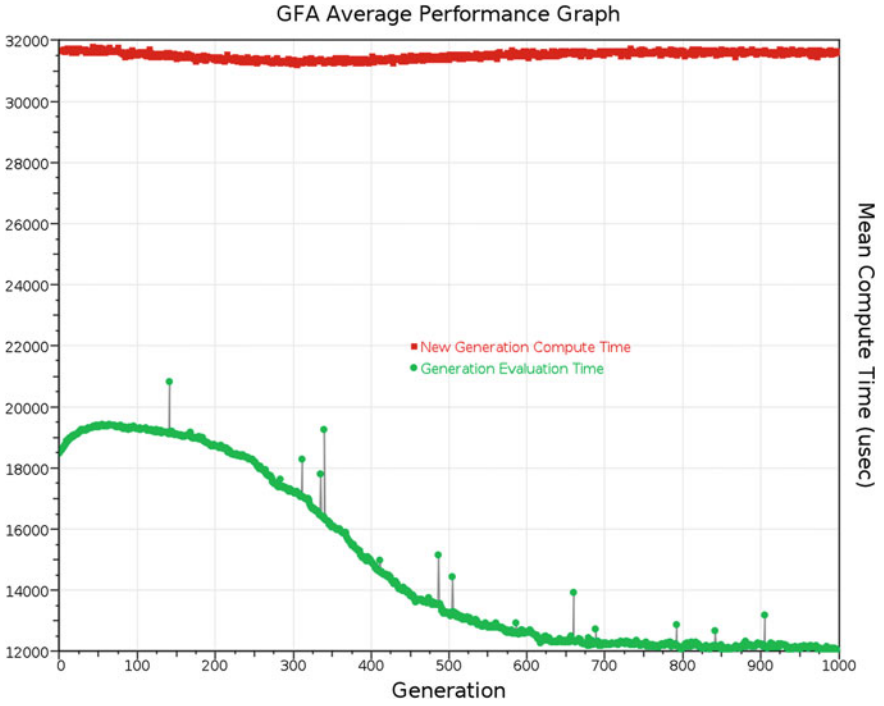

Fig. 11 Average performance data by generation for both generation evaluation time and the time taken to compute a new generation for evaluation

be far more expensive than the objective function to compute, therefore reducing overall compute time greatly. The outliers in the generation evaluation time is likely due to mutation introducing extra complexity and possibly activating dormant intron function symbols.

8 Visualisation

Visualising the results of an algorithm such as this not only assists in validation, but also for fine tuning and often provides valuable insights. We have previously introduced a method by which one may visualise a generation of expression trees by a recursive space division mechanism [42]. This method works well for shallow trees, but as space is divided in a binary fashion, the cells become too small to observe the effects clearly. A visualisation of the final generation from the best run we executed is shown in Fig. 15.

We have also experimented with better methods of visualising an entire generation of expression trees, and the general instantaneous exploration characteristics of one

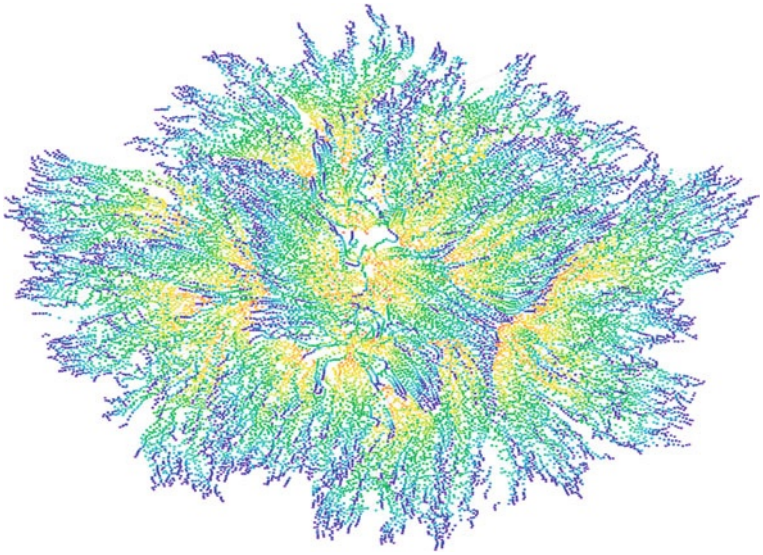

Fig. 12 2D visualisation of the first generation from the run with the best decay value of 0.008

generation. We have computed a tree-based graph of generations 1, 500, and 1000 of the best run. These images are shown in Figs. 12, 13 and 14.

In Fig. 12 a two dimensional visual representation is given for the first generation of a run with best decay value of 0.008. The tree based graph shows a complex graphical structure and we have used artificial colouring to show the space with red at the root and a normal rainbow spectral progression through to blue at the leaves. The flattened rendering shows a tendril-like structure and gives some insights into the space.

Figures 13 and 14 show simpler structures as the algorithm has progressed. They show the 500th and last generation run respectively.

An alternative way to visually render a tree space is shown in Fig. 15. This approach is described more fully in [42]. In summary, for a new expression to be added to the program space visualisation, the space is first divided into n sections vertically, where n is the number of terminal and non-terminal symbols. Each section represents a symbol. The first symbol in the expression determines the section next divided. Suppose this is the third section from the top. This section is then divided into n sections in a horizontal fashion. The next symbol in the expression determines which section will then be divided further, and so forth. Finally, when no symbols remain in the expression, a dot is drawn to indicate the location of the expressions.

It is particularly noteworthy that Fig. 15 is quite sparse, giving an indication of the tree depth steepness and some insights into how the algorithm has homed in on particular expressions.

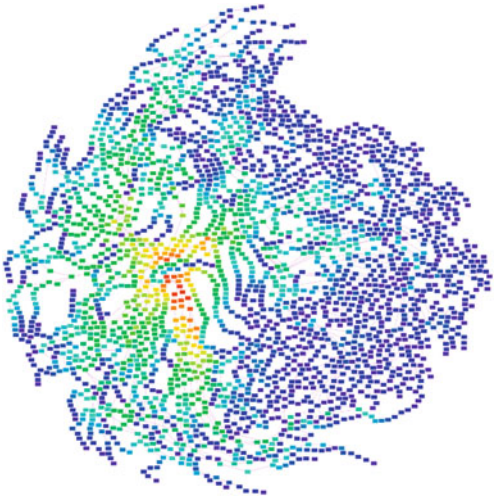

Fig. 13 2D visualisation of the 500th generation from the run with the best decay value of 0.008

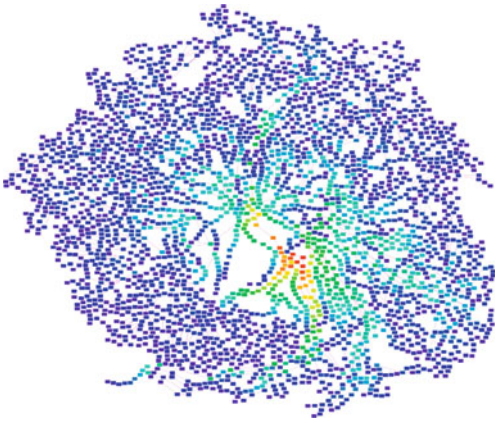

Fig. 14 2D visualisation of the last generation from the run with the best decay value of 0.008

9 Discussion

We have managed to construct a parallel Firefly Algorithm for expression trees using k-expressions and fitness-based gene scanning for recombination that can perform well on graphical processing units. The Karva representation of program solution space proved a good choice for our implementation and supported the operators we needed.

We have found that the Firefly Algorithm's exponential decay of observed fitness values property is particularly good for combinatorial optimisers and this approach appears to warrant further study.

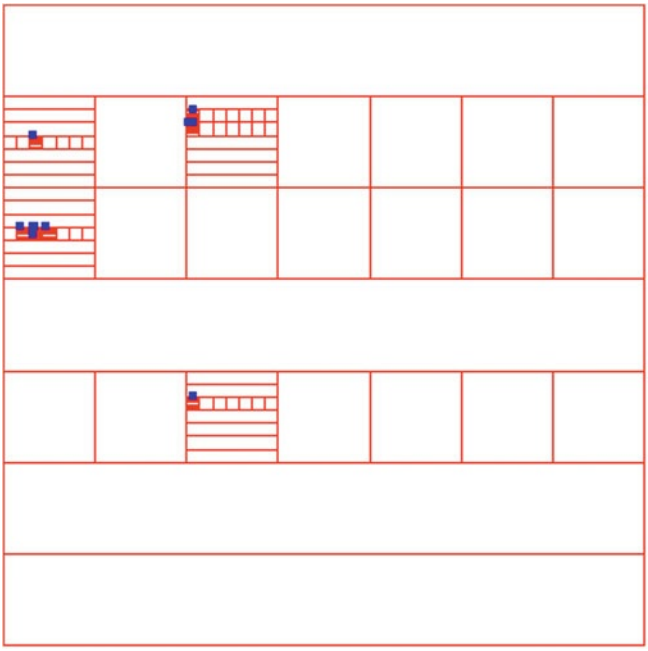

Fig. 15 2D visualisation using recursive space division [42] of the final generation from the best decay value of 0.008

It was necessary to approach the problem of attaining a parallel Firefly algorithm in two stages. Our work emphasises the need for both good algorithms that search the right part of problem space, and in addition those that are parallelisable and which can be readily speeded up to make use of available computational resources. We have focused on the parallelism available from GPU accelerators in this chapter. Furthermore we have largely focused on use of NVidia GPUs that can be programmed using Compute Unified Device Architecture (CUDA) software.

New devices with many processing cores such as multi-cored CPUs or multi-cored conventional processing accelerators from vendors such as Intel or AMD are also potential candidates to help speed up combinatorial search space algorithms. Hybrid hardware solutions that make use of both CPU cores and Accelerator cores [43] will be increasingly relevant in the future. Such combinations offer better memory bandwidth and therefore the ability to mix and interleave both CPU and accelerator computations. This is likely to be useful for the mix of genetic operators and approaches we have described.

A range of application problems that make use of multi-agents to explore problem spaces can make use of the ideas we have discussed in this chapter. One promising area is automatic algorithm discovery [44, 45] and this also suggests some uses for program-space optimisers. Gaining an understanding on the structure of program

spaces is also an open challenge with scope for visualisation [42] and other techniques to guide automated solutions.

10 Conclusions

In conclusion, we have presented a two-part GPU-based algorithm for combinatorial optimisation. The first part concerned the effective representation of individuals in the system, as well as a selection mechanism suitable for extension, crossover operators, mutation, and parallelising this combination to maximise performance. The second part concerned the extension of the algorithm to incorporate the algorithmic structure of the canonical Firefly Algorithm.

A particular problem solved included the type of metric used for judging difference between candidates. Although simple, we chose to use the Hamming distance for this. This allowed us to quantify the search space and more easily introduce exponentially degraded fitness values. There may be scope for other less simple metrics to be employed in this sort of algorithm. We also made use of roulette selection in order to use fitness values to bias the search further.

The results are promising, however, warrant further research, especially in order to characterise the behaviour of the algorithm when optimising different objective functions, as well as different system sizes and parameter combinations. We have also presented several examples of visualising the results, and experimented with a different method as well using graphs. Visualisation of these algorithms offer very valuable insights into system dynamics, especially in light of the stochastic nature of Evolutionary Algorithms. The visualisations in this case are useful for judging convergence and spatial behaviour.

There is also scope for future work to investigate the behaviour of this algorithm in the context of other application problems. It may be feasible to build up an experiential library of typical visualisation snapshots that will help guide investigations of new algorithms.

In summary, we have shown that geometric algorithms based on the Firefly family of algorithms can be implemented using GPU parallelism and that there is great promise in this general approach to evolutionary algorithm design, visualisation and computational performance.

References

1. Yang, X.S.: Firefly algorithms for multimodal optimization. In: Stochastic algorithms: Foundations and applications. SAGA, Sapporo (2009)
2. Moraglio, A.: Towards a geometric unification of evolutionary algorithms. Ph.D. thesis, Computer Science and Electronic Engineering, University of Essex (2007)
3. Moraglio, A., Togelius, J.: Geometric differential evolution. In: Proceedings of GECCO-2009, pp. 1705–1712. ACM Press (2009)

4. Moraglio, A., Silva, S.: Geometric differential evolution on the space of genetic programs. Genet. Programming **6021**, 171–183 (2010)
5. Moraglio, A., Chio, C.D., Poli, R.: Geometric particle swarm optimization. In: M. Ebner et al. (eds.) Proceedings of the European conference on genetic programming (EuroGP). Lecture notes in computer science, vol. 4445 (Springer, Berlin, 2007), pp. 125–136
6. Togelius, J., Nardi, R.D., Moraglio, A.: Geometric pso + gp = particle swarm programming. In: 2008 IEEE Congress on Evolutionary computation (CEC 2008). (2008)
7. Poli, R., Vanneschi, L., Langdon, W.B., McPhee, N.F.: Theoretical results in genetic programming: the next ten years? Genet. Program Evolvable Mach. **11**, 285–320 (2010)
8. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor (1975)
9. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Stat. Comput. **4**(2), 87–112 (June 1994)
10. Augusto, D., Barbosa, H.J.C.: Symbolic regression via genetic programming. In: Proceedings of the Sixth Brazilian symposium on neural networks 2000, vol. 1, pp. 173–178 (2000)
11. Husselmann, A.V., Hawick, K.A.: Geometric optimisation using karva for graphical processing units. Technical Report CSTN-191, Computer Science, Massey University, Auckland (February 2013)
12. Augusto, D.A., Barbosa, H.J.C.: Accelerated parallel genetic programming tree evaluation with opencl. J. Parallel Distrib. Comput. **73**, 86–100 (2013)
13. Brameier, M.: On linear genetic programming. Ph.D. thesis, University of Dortmund (2004)
14. Ferreira, C.: Gene expression programming: a new adaptive algorithm for solving problems. Complex Syst. **13**(2), 87–129 (2001)
15. Ferreira, C.: Gene expression programming, vol. 21, 2nd edn. Studies in computational intelligence, (Springer, Berlin, 2006), ISBN 3-540-32796-7
16. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. Genet. Program Evolvable Mach. **11**, 339–363 (2010)
17. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical report, caltech concurrent computation program (1989)
18. Moscato, P., Cotta, C., Mendes, A.: Memetic algorithms. In: New optimization techniques in engineering, (Springer, 2004), pp. 53–85
19. Leist, A., Playne, D.P., Hawick, K.A.: Exploiting graphical processing units for data-parallel scientific applications. Concurrency Comput. Pract. Experience **21**(18), 2400–2437 CSTN-065 (2009)
20. Yu, Q., Chen, C., Pan, Z.: Parallel genetic algorithms on programmable graphics hardware. In: advances in natural computation, (Springer 2005), pp. 1051–1059
21. Basu, B., Mahanti, G.K.: Firefly and artificial bees colony algorithm for synthesis of scanned and broadside linear array antenna. Prog. Electromag. Res. **32**, 169–190 (2011)
22. Yang, X.S.: Nature-inspired metaheuristic algorithms. Luniver Press, Frome (2008)
23. Mussi, L., Daoilo, F., Cagoni, S.: Evaluation of parallel particle swarm optimization algorithms within the cuda architecture. Inf. Sci. **181**, 4642–4657 (2011)
24. Chitty, D.M.: Fast parallel genetic programming: multi-core cpu versus many-core gpu. Soft. Comput. **16**, 1795–1814 (2012)
25. Langdon, W.B.: A many-threaded cuda interpreter for genetic programming. In: Esparcia-Alcazar, A.I., Ekart, A., Silva, S., Dignum, S., Uyar, A.S. (eds.) Proceedings of the 13th European conference on genetic programming, (Springer, 2010), pp. 146–158
26. Cano, A., Olmo, J.L., Ventura, S.: Parallel multi-objective ant programming for classification using gpus. J. Parallel Distrib. Comput. **73**, 713–728 (2013)
27. Durkota, K.: Implementation of a discrete firefly algorithm for the qap problem within the seage framework. Technical report. Czech Technical University (2011)
28. Husselmann, A.V., Hawick, K.A.: Parallel parametric optimisation with firefly algorithms on graphical processing units. In: Proceedings of international conference on genetic and evolutionary methods (GEM'12). pp. 77–83 Number 141 in CSTN, CSREA, Las Vegas, 16–19 July 2012

29. Langdon, W.B.: Graphics processing units and genetic programming: an overview. Soft. Comput. **15**, 1657–1669 (March 2011)
30. Schulz, C., Hasle, G., Brodtkorb, A.R., Hagen, T.R.: Gpu computing in discrete optimization. part II: Survey focused on routing problems. Euro J. Transp. Logist. Online. pp 1–26 (2013)
31. NVIDIA: CUDA C Programming Guide, 5th edn. http://docs.nvidia.com/cuda/pdf/cuda_c_programming-Guide.pdf (2012)
32. Zhang, L., Zhao, Y., Hou, K.: The research of levenberg-marquardt algorithm in curve fittings on multiple gpus. In: Proceedings 2011 international joint conference IEEE trustCom-11, pp. 1355–1360 (2011)
33. Zhou, T.: Gpu-based parallel particle swarm optimization. Evol. Comput. (2009)
34. Cupertino, L., Silva, C., Dias, D., Pacheco, M.A., Bentes, C.: Evolving cuda ptx programs by quantum inspired linear genetic programming. In: Proceedings of GECCO'11 (2011)
35. Harding, S.L., Banzhaf, W.: Distributed genetic programming on GPUs using CUDA. Workshop on parallel Architecture and Bioinspired Algorithms, Raleigh, USA (2009)
36. Cavuoti, S., Garofalo, M., Brescia, M., Pescape, A., Longo, G., Ventre, G.: Genetic algorithm modeling with gpu parallel computing technology. In: Neural nets and surroundings, vietri sul mare, salerno, Italy, Springer, pp. 29–39 22nd Italian workshop on neural nets, WIRN 2012. 17–19 May 2013
37. Hoberok, B.: Thrust: a parallel template library. http://www.meganewtons.com/ (2011)
38. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of genetic algorithms. Morgan Kaufmann, San Mateo (1991), pp. 69–93
39. Weise, T.: Global optimization algorithms-theory and application. Self-Published, (2009)
40. Eiben, A., Raué, P.E., Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In: Proceedings of the 3rd conference on parallel problem solving from nature (1994)
41. Eiben, A.E.: Multi-parent recombination. Evol. comput. **1**, 289–307 (1997)
42. Husselmann, A.V., Hawick, K.A.: Visualisation of combinatorial program space and related metrics. Technical Report CSTN-190, computer science, Massey University, Auckland, 2013
43. Hawick, K.A., Playne, D.P.: Parallel algorithms for hybrid multi-core cpu-gpu implementations of component labelling in critical phase models. Technical Report CSTN-177, computer science, Massey University, Auckland, 2013
44. van Berkel, S.: Automatic discovery of distributed algorithms for large-scale systems. Master's thesis. Delft University of Technology (2012)
45. van Berkel, S., Turi, D., Pruteanu, A., Dulman, S.: Automatic discovery of algorithms for multi-agent systems. In: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion, pp. 337–334 (2012)

A Discrete Firefly Algorithm for Scheduling Jobs on Computational Grid

Adil Yousif, Sulaiman Mohd Nor, Abdul Hanan Abdullah and Mohammed Bakri Bashir

Abstract Computational grid emerged as a large scale distributed system to offer dynamic coordinated resources sharing and high performance computing. Due to the heterogeneity of grid resources scheduling jobs on computational grids is identified as NP-hard problem. This chapter introduces a job scheduling mechanism based on Discrete Firefly Algorithm (DFA) to map the grid jobs to available resources in order to finish the submitted jobs within a minimum makespan time. The proposed scheduling mechanism uses population based candidate solutions rather than single path solution as in traditional scheduling mechanism such as tabu search and hill climbing, which help avoids trapping in local optimum. We used simulation and real workload traces to evaluate the proposed scheduling mechanism. The simulation results of the proposed DFA scheduling mechanism are compared with Genetic Algorithm and Tabu Search scheduling mechanisms. The obtained results demonstrated that, the proposed DFA can avoid trapping in local optimal solutions and it could be efficiently utilized for scheduling jobs on computational grids. Furthermore, the results have shown that DFA outperforms the other scheduling mechanisms in the case of typical and heavy loads.

Keywords Computational grid · Discrete optimization · Firefly algorithm · Job scheduling

A. Yousif (✉) · A. H. Abdullah · M. B. Bashir
Faculty of Computing, Universiti Teknologi Malaysia, Skudai, Malaysia
e-mail: adiluofk@gmail.com

A. H. Abdullah
e-mail: hanan@utm.my

S. M. Nor
Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Skudai, Malaysia

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_13,

1 Introduction

Computational grid is a large scale distributed system consisting of a huge number of heterogeneous resources that belong to different virtual organizations. Scheduling jobs on such environments represents a great challenge and can be identified as NP-hard problem [1, 2]. Therefore, heuristics and metaheuristics mechanisms have been applied to handle the job scheduling problem on computational grid. However, metaheuristics are commonly able to find good but not necessarily efficient and optimal solutions for the job scheduling problem. Nonetheless, nature-inspired metaheuristics has demonstrated an excellent degree of effectiveness and efficiency for handling combinatorial optimization problems [3]. The remarkable rise in the size of the solution search space motivated researchers to employ nature-inspired metaheuristics mechanisms to solve computational grid scheduling problems.

Firefly Algorithm (FA) is a nature inspired metaheuristic algorithm, inspired by the flashing behavior of fireflies to search for food and to connect to each other [4]. The FA is a population-based technique with efficient and effective global search for combinatorial problems [5, 6]. FA is simple, distributed and does not have central control or data source, which enables the system to become more effective and scalable. Many researchers use FA to solve NP-hard problems such as clustering problem [6], flow shop scheduling problem [7] and traveling salesman problem [8], and so on.

This chapter applies a discrete version of FA algorithm to job schedule problems on computational grid. The proposed mechanism aims to create an optimal schedule that is able to finish the submitted jobs within a minimum makespan time. Each firefly is assumed to act as a scheduling candidate solution and it interacts with others fireflies in order to explore the search space for optimal solution. We compared the proposed discrete firefly scheduling with other metaheuristics scheduling mechanism such as tabu search and genetic algorithm. According to the simulation results, we can report that the proposed DFA outperforms the other job scheduling mechanisms in most cases.

The rest of the chapter is organized as follows. Section 2 reviews the related works on the literature; this is followed, in Sect. 3, by the formulation of the grid scheduling problem. Section 4 demonstrates the grid broker and management architecture. Section 5 describes the proposed DFA. Section 6 shows the implementation and experimental results. And finally, the conclusion and future works are presented in Sect. 7.

2 Related Work

This section reviews a number of job schedulers that use different metaheuristics techniques to map the clients' jobs to the available grid resources such as Hill Climbing, Tabu Search and Genetic Algorithm.

A metaheuristics is a set of algorithmic notions that can be employed to describe heuristics mechanisms appropriate to a broad set of different problems. Metaheuristics can be defined as a general-purpose heuristic mechanism intended to direct an underlying problem-specific heuristic toward promising area of good solutions in the solution search space. In other words metaheuristics can be defined as a heuristic for the heuristics [9]. The main challenge for optimization mechanisms is to increase the possibility of finding global optimal solutions. Greedy optimization mechanism such as Hill Climbing (HC) and Tabu Search (TS) strive to improve each single step. Greedy methods can find the solution quickly. However, greedy optimization mechanisms are often trapped in local optimal solutions [10]. HC is a local search optimization mechanism. It is an iterative technique that begins with a random solution in the search space, and then tries to discover optimized solutions by continuously modifying a single element of the current solution. If the modification generates a better candidate solution, the modification is considered, otherwise the modification is discarded [11]. HC is a local search mechanism which is suitable for finding local optimal solutions and so it is not appropriate in searching for global optimization. Besides, HC optimization mechanism suffers from the plateau problem when the solution search space is flat; in that situation, HC is not capable of finding out which way it should go, or it may choose directions that never lead to the optimal solution. Similar to the HC mechanism is TS which is a metaheuristic local search mechanism that can be used for handling optimization problems [12]. TS has been applied to handle job scheduling on computational grid [10]. TS has superiority over HC as it has a memory. This memory helps in keeping on with the exploration even if the improving movement is absent. Moreover, the memory prevents the TS scheduling mechanism from getting trapped in a local optimum that has been visited previously. However, TS uses a single search path of solutions and not population search or tree search. In the single search path technique, a set of moves throughout the solution search space are assessed to choose the best candidate solution.

Evolutionary Algorithm (EA) mechanisms, such as Differential Evolution (DE) generate a random initial population of feasible candidate solutions, that is a set of integer random numbers, and each solution represents a chromosome. Each chromosome is a vector indexed with a number from 1 to NP, where NP is the population size. After the initial population is generated, the population chromosomes are refined using crossover and mutation operations.

EAs have a limited range of movements, which reduces the likelihood of trapping in local or sub optimal solutions. However, they are slower in finding optimal solutions as a result of the complexity in managing the population movements [13]. GA and DE are used to schedule the jobs on the computational grid [14–16]. Evolutionary metaheuristics scheduling mechanisms outperform the grid basic scheduling mechanisms in most cases [17, 18]. Evolutionary algorithms such GA and DE in some cases trap in local optimal and cannot evolve any more. This is because the population diversity is becoming low after some number of iterations so diversity of individuals is lost and the crossover and mutation operations are no longer generates a better individual. To tackle this problem, GA applies a limited range of movements, which decreases the possibility of trapping in sub optimal. However, this makes GA

to be slower in finding optimal solutions. Furthermore, evolutionary algorithms may have a memory to store previous status. This memory may help in minimizing the number of individuals close to positions in candidate solutions that have been visited before. However, this may also decreases the search space converges since successive generations may die out.

GA and DE areused to schedule jobs on the computational grid [14–16]. Evolutionary metaheuristics scheduling mechanisms outperform the grid basic scheduling mechanisms in most cases in Ref. [17, 18]. De Falco et al. in Ref. [15] developed a multi objective Differential Evolution mechanism to schedule jobs on computational grid. The objective of the scheduling mechanism is to minimize the resources usage as well as to maximize the grid quality of service requirements. Another multi-objective differential evolution (MODE) for scheduling jobs on computational grid was introduced in the work by Talukder et al. in Ref. [19]. The aim of MODE is to minimize the job completion time and minimize the cost for executing the jobs. Selvi et al. introduced a new job scheduling mechanism based on DE to minimize the job completion time. The representation of the scheduling algorithm is based on array representation as they represent each valid solution as an array with a length equal to number of submitted jobs. Swarm Intelligence (SI) is a new class of nature-inspired metaheuristics based on population optimizations. The population elements are particles that aim to find the global optimal candidate solution by communicating with other particles and with the environment. In SI such as PSO, ACO and FA particles do not die; rather, they move throughout the search space themselves. PSO and ACO have been used as scheduling mechanisms to map the jobs to resources on computational grid in several research [1, 9, 20].

ACO has been applied as an optimization mechanism for scheduling jobs on computational grid [21–23]. The ACO is an optimization method inspired by the real ants in discovering the shortest path from source to destination. Real ants move randomly searching for food and go back to the nest while dropping pheromone on the path to identify their chosen path to encourage other ants to use [3]. If other ants use the same path, they will deposit more pheromone and if the path is no longer used, the pheromone will start to evaporate. The ants always choose the path that has higher pheromone concentration, and then they give feedback by depositing their own pheromone to make other ants use the path.

Particle Swarm Optimization (PSO) is one of the SI optimization methods, inspired by social behavior of swarms such as bird flocking or fish schooling [24]. Several works have been done to optimize job scheduling on computational grid using PSO [1, 2, 25, 26]. In PSO, particles never die. Particles are considered as simple agents that move and interact throughout the search space and record the best solution that they have visited. PSO technique is an adaptive optimization method [1, 24]. In particle swarm optimization, each particle represents a feasible solution in the search space which is a valid schedule of the client's submitted jobs to the available resources. Each particle in PSO has a position vector and velocity. After comparing the fitness of each schedule, the particles move based on their local knowledge which is represented by the particle knowledge and global knowledge which is the knowledge that the particles gain from the swarm.

A representation of grid job scheduling problem as a task resource assigning graph (T-RAG) is presented by Chen et al. in Ref. [25]. This representation handles the job scheduling problem as graph optimization problem. In the work presented in Ref. [25] PSO was employed as an optimization mechanism to find the optimal schedule for the job scheduling problem. The proposed mechanism represented each particle as an integer vector with values between 1 and m where m is the number of resources. The algorithm converts each real value to an integer value by upwarding the decimals places. Zhang et al. proposed an optimized job scheduling mechanism in Ref. [24, 27]. In their method, Zhang et al. used smallest position value technique to adapt the continuous PSO mechanism to be used for discrete permutation problems such as the process of scheduling jobs on computational grid. Their results showed that the discrete PSO based on the smallest position value outperformed genetic algorithm for scheduling jobs in term of average execution time. The PSO algorithm described previously are single objective optimization. In Ref. [28] a parallelized multi-objective particle swarms was introduced. This optimization method divided the swarm population into smaller sub swarm populations.

A discrete particle swarm optimization method (DPSO) mechanism for job scheduling on computational grid was introduced in Ref. [26]. In DPSO, particles are represented as a vector of natural numbers. The experimental evaluation for DPSO demonstrated that DPSO has a better performance than genetic algorithm for the job scheduling problem. A matrix representation called the position matrix of the job scheduling problem on computational grid was presented in Ref. [29] and enhanced in Ref. [2] by using different method for velocity updating. A fuzzy PSO method for scheduling jobs on computational grid was introduced in Ref. [1, 20]. The aim of the proposed fuzzy PSO is to minimize the scheduling time and to utilize the grid resources effectively. The empirical results demonstrated that the fuzzy PSO has performed better than genetic algorithm and tabu search optimization methods. Meihong et al. in Ref. [30] introduced a new PSO without velocity mechanism for grid job scheduling problem. In their mechanism they used sub swarms to update the particles positions. in Ref. [17] a PSO with two representations was introduced. In the first representation which called direct representation the particles are encoded as vectors of size $1 \times n$ where n is the number of jobs submitted to the broker. In the indirect representation the swarm is encoded as a matrix of size $m \times n$ where m is the number of resources and n is the number of jobs. The matrix is represented as binary numbers to show to which resource the job is allocated.

PSO has a number of disadvantages, for example, PSO slow its convergence speed when it is near the optimal solution. This is because PSO applies the linearly decreasing of inertia weights. Applying the linearly decreasing inertia weights affects the search capabilities at the end of run even if the global search capacity is needed to escape from local optimum in some cases [31]. Furthermore, PSO suffer suffers from the partial optimism. This problem affects the PSO speeds and directions [32].

3 Scheduling Problem Formulation

Let $R = \{r_1, r_2 \ldots r_m\}$ be m grid resources and $J = \{j_1, j_2 \ldots j_n\}$ be n independent client jobs. Job scheduling problem on computational grid can be defined as the process of mapping each client job to one or more time intervals on one or more grid resources.

Each job j_i consists of a set $n_i$of tasks $t_{i1}, t_{i2}, \ldots, t_{ini}$. The t_{ij} task has a number of processing requirements p_{ij}. In the exceptional case when the job j_i consists of only one task we recognize withand the processing requirements with. Each task t_{ij} is associated with a number of resources $r_{ij} \subseteq \{r_1, r_2, \ldots, r_m\}$ and t_{ij} possibly be processed on any of the resources of r_{ij}. To generalize the scheduling problem this chapter assumes that the task can be scheduled on any resource able to process it, which according to Brucker [11] is defined as multi-purpose resource. In this model all values of (p_i, p_{ij}, r_i) are assumed to be integer values.

Define C_{ij} as the time that resource r_i needs to finish job j_i; ΣC_i is the total time that resource r_i completes all the jobs submitted to it. The function described in Eq. (1) is makespan time.

$$f_{max}(c) := \max\{\Sigma C_i\} \tag{1}$$

The makespan time is the maximum completion time or the time when the grid system completes the latest job. The objective of the proposed DFA is to minimize the makespan time.

4 The Grid Brokering and Management Architecture

The architecture of the proposed scheduling mechanism is described in Fig. 1, which illustrates the interaction between grid clients, broker components, grid information service and grid resources. In this architecture, all resources joining the grid register their information in the Grid Information Service (GIS), an entity that provides grid resource registration, indexing and discovery services. This information includes the architecture of the resources, the operating system used, the number of processing elements the allocation policy, the speed of the resource and other information related to the resource characteristics. The grid resource broker is responsible for receiving jobs from grid clients, discovering the available resources, managing scheduling mechanisms and controlling the physical allocation of resources to the grid client's jobs.

The process of resource scheduling and allocation starts when grid clients submit the jobs to the broker through a portal. The broker starts the process of discovering the available resources by communicating with the GIS. After discovering the available resources that fulfill the lowest permitted level of the requirements, the broker starts the scheduling process using scheduling mechanisms to map the submitted jobs to available resources.

The scheduling mechanism consists of three main components i.e. the scheduling policies, the fitness functions and the scheduling algorithm. Generally, the scheduling policy is a set of rules that manage the process of resource allocation for the received

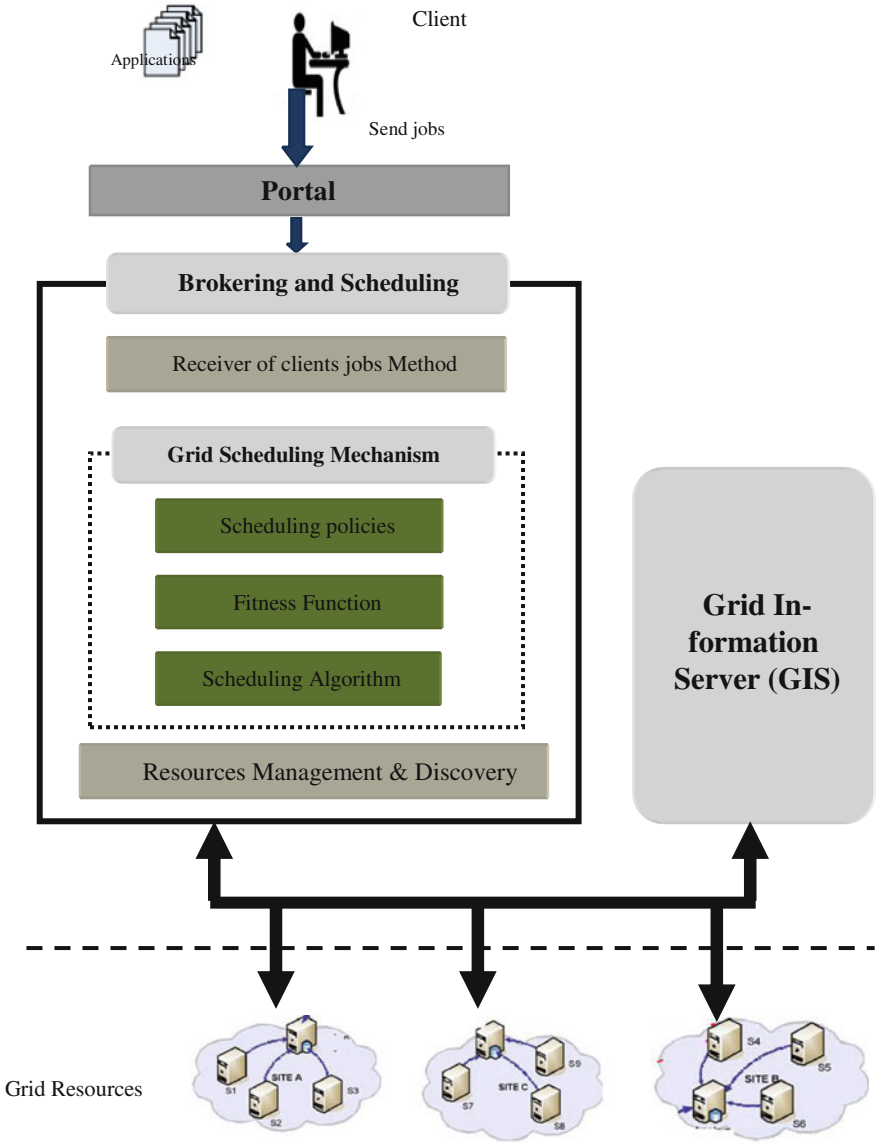

Fig. 1 DFA scheduling architecture

jobs. The fitness function is used to evaluate, rank and find out the quality of schedule. In the proposed scheduling mechanism, the attractiveness of the schedule is used to determine the quality of a given candidate solution within the population.

The goal of the job scheduling process is to allocate the submitted jobs to the available resources in order to complete the grid jobs within a minimum makespan. Thus, the attractiveness and fitness of the firefly correspond to the makespan func-

tion. The scheduling algorithm is responsible for generating valid schedules for the submitted jobs on the available resources. The scheduling algorithm is considered a good algorithm if it generates good or optimal schedule regarding the fitness function along with consuming few resources and spending not too much time to produce the schedule. The broker finally maintains the physical allocation of the jobs and manages the available resources constantly.

5 Job Scheduling Based on DFA

Combinatorial optimization problems are essential in various research fields, and recently the success of nature-inspired metaheuristics motivated researchers to apply those metaheuristics for solving combinatorial problems. For job scheduling on computational grid problems, the remarkable rise in the size of the solution search space motivated the researchers to use nature-inspired metaheuristics to handle the job scheduling problems. In this section, we introduce a discrete scheme based on firefly algorithm to optimize the process of scheduling jobs on computational grid.

5.1 Firefly Algorithm

Firefly algorithm (FA) is a nature inspired metaheuristic algorithm, inspired by the flashing behavior of fireflies to search for food and to connect to each other [4]. The FA is a population-based technique with efficient and effective global search for combinatorial problems [5, 6]. FA is simple, distributed and does not have central control or data source, which allows the system to become more effective and scalable.

Firefly optimization mechanism as described by [5, 6] can be summarized on the following steps:

- All fireflies are unisex, where every firefly can be attracted to every other firefly.
- The attractiveness of a firefly is relative to its light intensity.
- The attractiveness of a firefly is determined by its position within the search space.
- The less attractive fireflies are attracted by the brighter fireflies.
- The better value of the fitness function at certain position produces more attractive firefly.
- The brightest firefly moves randomly.

The initial population of fireflies is generated randomly and by using the fitness function of the optimization problem under study, the attractiveness β_0 is determined for each firefly in the population. All the fireflies move throughout the solution search space for a certain number of iterations. For every iteration, the attractiveness of each two fireflies f_i and f_j is compared; if firefly f_i is more attractive than firefly f_j then firefly f_j will move toward firefly f_i.

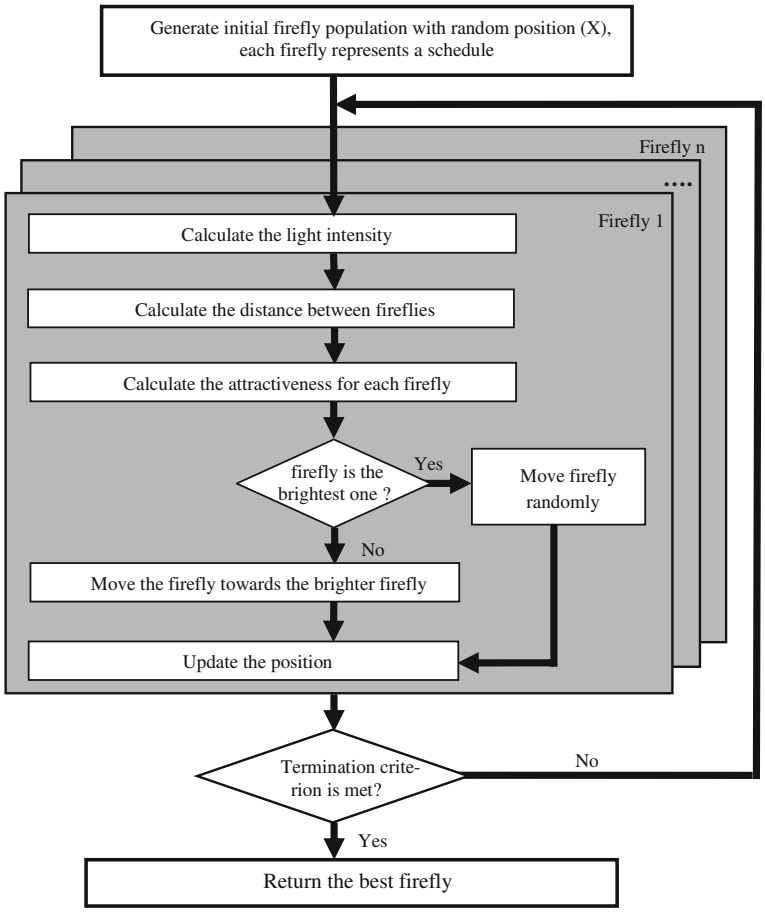

Fig. 2 The flowchart of proposed DFA scheduling mechanism

However, the light intensity or attractiveness value β depends on the distance r between the fireflies and the media light absorption coefficient γ. The attractiveness of each firefly is determined using the equation:

$$\beta\,(\mathrm{r}) = \beta_0 \mathrm{e}^{-\gamma r^2} \tag{2}$$

Where β_0 represents the attractiveness of the firefly at $\mathrm{r} = 0$.

The movement of a firefly $\mathrm{f_j}$ at position $\mathrm{x_j}$, attracted to a brighter firefly $\mathrm{f_i}$ at position is established by the equation:

$$x_j = x_j + \beta_0 e^{-\gamma r^2 ij}(x_j - x_i) + \alpha\epsilon_i \tag{3}$$

The main steps of the firefly algorithm are illustrated in Fig. 2 in the form of a flowchart.

5.2 Discrete Firefly Algorithm

The firefly algorithm has proven to be a good metaheuristic search mechanism on continuous optimization problems [33, 34]. Clearly, the standard firefly algorithm cannot be applied directly to tackle discrete problems as its positions are real numbers. Many researchers solved discrete optimization problems by applying adapted nature inspired metaheuristics optimization methods [35]. In our proposed scheduling mechanism, we used the Smallest Position Value (SPV) technique for firefly algorithm discretization [36].

SPV introduced by Tasgetiren, M.F., et al. [36] enables the continuous optimization mechanisms to be applied for all types of combinatorial optimization problems, which are NP-hard. SPV techniques find the permutation of the firefly through the firefly position values. The pseudo code for the SPV technique is illustrated in algorithm 1.

Algorithm 1 Smallest Position Value (SPV) Technique

(1) N = populationsize;

(2) K = number of iteration;

(3) X_i^k Represents the firefly i in the iteration number k ;

(4) $X_i^k = (X_{i,1}^k, X_{i,2}^k, \ldots, X_{i,n}^k)$

(5) Firefly population $X^k = (X_1^k, X_2^k, \ldots, X_N^k)$;

(6) Define S_i^k as the sequence of jobs implied by the firefly X_i^k

$S_i^k = (S_{i,1}^k, S_{i,2}^k, \ldots, S_{i,n}^k)$;

(7) Define the operation vector $R_i^k = (R_{i,1}^k, R_{i,2}^k, \ldots, R_{i,n}^k)$ as

$R_i^k = (S_i^k \bmod m) + 1$; //represents the permutation

5.3 Scheduling Jobs using DFA

In FA, fireflies are considered as simple agents that move and interact throughout the search space and record the best solution that they have visited. Hence, FA can be used to produce schedule candidate solutions in order to efficiently support the scheduling process on computational grid to map the received jobs to the available resources in order to finish job's execution within a minimum makespan time. The main steps of the DFA are finding the fireflies positions using the firefly algorithm movements and then using the smallest position value to find the permutation from the position values. The final step is calculating the fitness function using the permutation generated by SPV and determining the new movements. In DFA, we used the makespan time, which is the time when the grid finishes the latest job, as the attractiveness of the fireflies. The representation of the optimization methods is a crucial issue in designing

a successful solution. The proposed DFA considers each firefly as a vector of natural numbers. The element number i in the firefly vector represents the resource ID to which job number i is mapped.

Suppose $J = \{j_1, j_2, \ldots, j_n\}$ represents the jobs submitted to the grid broker and $R = \{r_1, r_2, \ldots, r_m\}$ are available resources. Let T be the execution time matrix as:

$$T = \begin{bmatrix} t_{11} & t_{12} & \ldots & t_{1n} \\ t_{21} & t_{22} & \ldots & t_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ t_{m1} & t_{m2} & \ldots & t_{mn} \end{bmatrix}. \tag{4}$$

Here t_{ij} represents the execution time of job number i on the resource j.

Suppose $X^k = (X_1^k, X_2^k, \ldots, X_N^k)$ represents the firefly population where $X_i{}^k$ indicates the i-th firefly in the k-th iteration. If the search space is n-dimensional space, then the i-th firefly can be defined as $X_i^k = (X_{i,1}^k, X_{i,2}^k, \ldots, X_{i,n}^k)$ where represents the dimension number j of the i-th firefly on the iteration number k. Each firefly position is a feasible schedule for the underlying job scheduling problem. Fireflies position are continuous values, to convert these values into discrete permutations we use the SPV. Consider $S_k^i = (s_{i,1}^k, s_{i,2}^k, \ldots, s_{i,n}^k)$ as the sequence of jobs implied by the firefly. Define the permutation vector $R_k^i = (R_{i,1}^k, R_{i,2}^k, \ldots, R_{i,n}^k)$ according to Eq. (5):

$$R_i^k = \left(S_i^k mod\, m\right) + 1 \tag{5}$$

The pseudo-code of the DFA scheduling mechanism is illustrated in algorithm 2.

Algorithm 2 Scheduling Jobs using DFA

```
Get the list of jobs to be scheduled;
Get the list of available resources from GIS;
Arrange the jobs and resources in an ascending order based on job Lengths and
resource processing speeds (this is because SPV works with sequences);
Initialize algorithm parameters;
Generate random initial population in which each firefly represents candidate
scheduling solution;
Convert the continuous values to discrete values using SPV;
Calculate the fitness values (makespan time) for the initial population;
While (Iteration< cycles) do
  For each firefly f_i do
    Calculate the fitness of other fireflies using Eq(1)
    Compare firefly f_i fitness with all other fireflies
    If firefly f_i is the brightest firefly
      Move f firefly f_i randomly
    Else
      Move firefly f_i towards the brighter firefly using Eq(2)
    end if
    Convert the continuous values to discrete values using SPV;
  end for
  Evaluate the new makespan times;
end while
Find the smallest makespan time
Choose the schedule that produce this makespan time as the optimal schedule
```

6 Implementation and Experimental Results

6.1 Simulation Framework

For DFA evaluation simulation methods have been employed. We used GridSim, which is a discrete-event grid simulation tool based on java [37]. GridSim is standard grid and distributed systems simulator that model heterogeneous resources and supports the allocation of resources in any time zone, and it allows advance reservation. Furthermore, the number of jobs that can be submitted to a resource is not limited and can occur simultaneously. Moreover, GridSim offers capabilities to specify the network speeds between grid resources and describe the resource characteristics [37].

The simulation entities are modeled and simulated based on the real DAS-2 system in terms of the number of resources, architecture, operating system and other resource characteristics. The main characteristics of the simulation model are as follows:

- Number of sites within the system: 5

- Number of CPUs in the trace: 400
- Number of users in the trace: 333
- Number of groups in the trace: 12
- Speed of CPUs ranges uniformly between 20 and 200 Million Instructions Per Second (MIPS).
- Resource operating systems: redhat, ubuntu, Centos, Solaris.
- Scheduling policy: Space shared.

To ensure that the simulation agrees with its requirements specifications, the correctness of the simulation has been verified using several verification techniques such as clean code functional program verification, checking of intermediate simulation outputs and comparing the final simulation outputs with analytical results.

6.2 The Workloads

In our simulation, we have used traces of the DAS-2 system from the Grid Workloads Archive (GWA) [38], which is a grid benchmarked workloads made available to help the grid researchers. The GWA trace contains real time workload of a huge number of resources obtained from various grid projects such as NorduGrid, which consists of more than 75 different clusters and Grid' 5000, an experimental grid environment contains 9 sites physically distributed in France; each site consists of a number of clusters.

In our experiments, we considered only a fraction of jobs submitted to the DAS-2 system compared to the total number of jobs submitted to the system which is 1124772 jobs submitted over a period of 2 years. Moreover, in our experiments we have not captured the warm-up period of DAS-2 workload trace, to avoid the unrepresentative trace fragments.

6.3 Parameter Selection

A suitable setting for the DFA parameters values are needed for the proposed mechanism in order to work effectively. Before comparing the proposed mechanism with other scheduling mechanisms, we have conducted experiments to find out the most suitable parameter values for the proposed mechanism. In the following subsections, we discuss some observations we have collected while setting the parameter values for the proposed DFA. To summarize the observations of the parameter settings, we will use the final selected parameter values as default values and change one value each time; the final parameter values are described in Table 3.

Table 1 Makespan time for different population sizes

2	4	6	8	10	12	16	20
735900	669000	669000	594480	432000	545222	543444	432000

6.4 The Population Size

To analyze the effect of the population size on the proposed mechanism, we conducted several experiments with different population sizes, and we calculated the makespan for each experiment. The results from those experiments are reported in Fig. 3 and Table 1. It can be seen that the population of size 20 has a less makespan time; indicating that the population size affects the searching quality of the proposed firefly scheduling mechanism. Generally, increasing the population size or number of fireflies enhance the search converge, because more fireflies are distributed throughout the solution search space. However, increasing the number of fireflies will result in longer scheduling time and hence affect the effectiveness of the scheduling algorithm. So from the observations on experiments done we noted that the population of size 10 is more suitable than other sizes of the population in which we get an optimal schedule in acceptable scheduling time.

6.4.1 Gamma Values:

The media light absorption coefficient γ measures the attractiveness absorption of a certain transmission media. As shown in Fig. 4, the makespan time increase when we increase the γ value more than 0.4 or decrease it less than 0.00006 and the suitable γ value is 0.02 (Table 2).

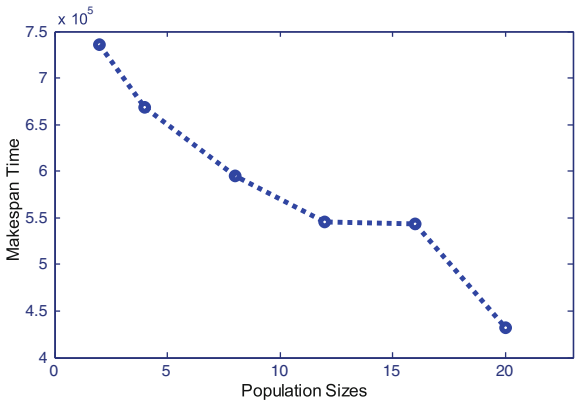

Fig. 3 Makespan time for different population sizes

6.4.2 Experimental Results

In our experiments, Genetic Algorithm (GA), and Tabu Search (TS) were used to compare the performance of the proposed DFA. The parameter values for GA, TS and DFA are described in Table 3

In our simulations, we have considered different sizes of workload trace ranging from a lightweight load contains only 500 jobs to heavy load contains 10,000 jobs. Each experiment was repeated several times with different random seeds, and the averages makespan times were calculated until the results are saturated.

Table 4 describes the results for makespan time for considered scheduling algorithms regarding different workload sizes. Furthermore, the results are visualized in Fig. 5.

It is observed from Table 4 and Fig. 5 that, the proposed DFA mechanism achieves the best results in terms of makespan time among all scheduling instances mostly. TS has longer makespan time compared with other scheduling mechanisms.

More specifically, in typical workload trace with 5000 jobs the makespan times as shown in Table 4 and Fig. 6 were {291878, 255182, 195350} for TS, GA and DFA, respectively. These results demonstrate that DFA requires significantly less makespan time than other scheduling mechanisms. Figure 7 shows the makespan time of the proposed mechanism and the other scheduling mechanisms when the heavy workload is considered, in which 10,000 jobs are submitted to the broker by the grid clients. The results obtained from this experiment mentioned the DFA has shown lower makespan time than the other scheduling mechanisms. TS scheduling mechanism obtained the worst makespan time.

Computational results for the makespan times for lightweight workload are shown in Table 4 and visualized in Fig. 8. The makespan time of GA was 2800 and for DFA was 1737 while the makespan time for TS was 1223 which is the minimum makespan time. This indicates TS performs better than DFA in case of lightweight workloads.

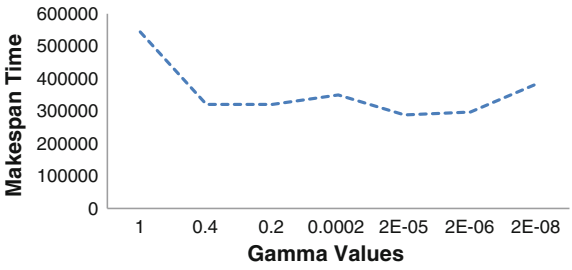

Fig. 4 Makespan time for different gamma values

Table 2 Makespan time for different gamma values

Gamma	1	0.4	0.2	0.0002	0.00002	0.000002	0.00000002
Makespan	543444	320580	320580	349623	288524	297393	383532

Table 3 Parameter values of DFA, GA, TS and HC

Algorithm	Parameter name	Parameter value
DFA	Size of the Population	10
	α	0.9
	γ	0.02
	β_0	1.0
GA	Size of the Population	30
	Crossover rate	0.8
	Mutation Rate	0.02
TS	Tabu list size	10
	Number of Iterations	300

Table 4 Makespan time of the scheduling process for different workload sizes

No. of Jobs	500	1000	1500	2000	3000	4000	5000	6000	7000	8000	9000	10000
TS	1500	1223	1800	164206	164605	262071	291878	379800	526560	601320	607080	731040
GA	2800	2800	4227	8303	164359	251084	255182	343129	387120	389298	390701	576000
DFA	1103	1737	2753	7762	195040	197064	195350	246090	251750	258417	257582	284519

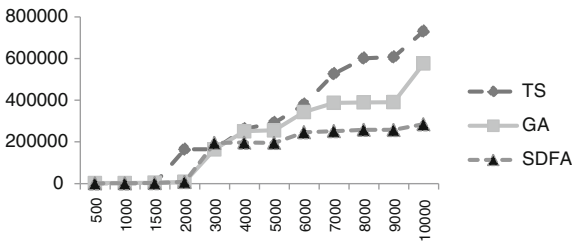

Fig. 5 The scheduling algorithms makespan time for different workload size

From the results we can note that, TS is suitable for scheduling problem with a lightweight workload trace in which the search space is not so huge and TS may find the optimal solution easily. However, in heavy workload, it is difficult for the single path mechanism such as TS to find the optimal solution. Moreover, TS suffers in handling the solution search space diversity as some neighborhood candidate solution is not necessarily to be generated. To tackle this problem, TS needs to be extended with other heuristics technique that can help avoid the unnecessarily neighborhood. However, the main drawback of this TS extended technique is the extra computational cost generated by the employed heuristic.

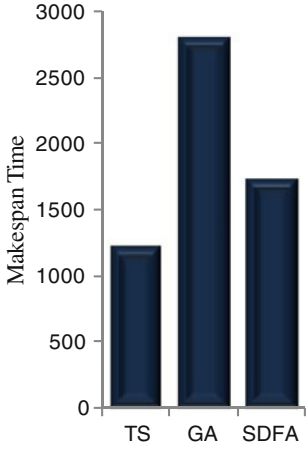

Fig. 6 Makespan time of the considered scheduling mechanism (Typical Load)

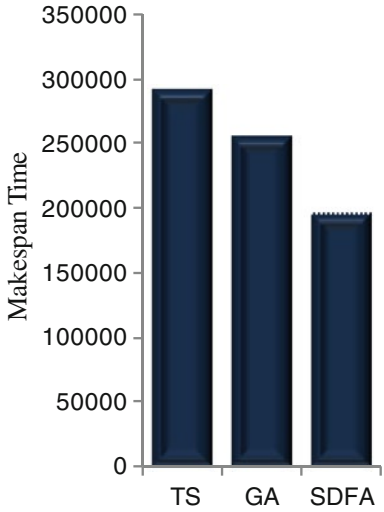

Fig. 7 Makespan time of the considered scheduling mechanism (Heavy Load)

7 Conclusion and Future Work

This chapter presented a new job scheduling mechanism for computational grid based on a discrete firefly algorithm. The objective of the proposed job scheduling mechanism is to allocate the grid jobs to the available resources in order to complete the jobs within a minimum makespan time. In the proposed DFA, we applied SPV to convert the continuous values of FA to discrete values. The proposed mechanism is evaluated based on simulation model using GridSim simulator and real workload data obtained from GWA. Empirical results revealed that the proposed scheduling mechanism outperforms other scheduling mechanisms in case of typical and heavy

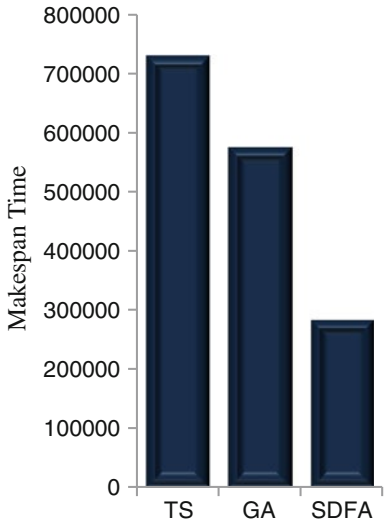

Fig. 8 Makespan time of the considered scheduling mechanism (Lightweight Load)

workloads. However, in the case of lightweight workload trace TS achieves less makespan time than DFA. The future works is directed towards handling distribution of data on the grid and the dependencies among grid resources.

Acknowledgments This research is supported by the Ministry of Higher Education Malaysia (MOHE) and collaboration with Research Management Center (RMC) Universiti Teknologi Malaysia. This paper is financially supported by GUP GRANT (No. Vot: Q.J130000.7128.00H55).

References

1. Liu, H., Abraham, A., Hassanien, A.E.: Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Gener. Comput. Syst. **26**(8), 1336–1343 (2010)
2. Izakian, H., et al.: A novel particle swarm optimization approach for grid job scheduling. Inf. Syst. Technol. Manag. **31**, pp. 100–109 (2009)
3. Zang, H., Zhang, S., Hapeshi, K.: A review of nature-inspired algorithms. J. Bionic Eng. **7**, S232–S237 (2010)
4. Yang, X.S.: Firefly algorithms for multimodal optimization. Stochastic Algorithms: Found. Appl. **5792**, 169–178 (2009)
5. Yang, X.S.: Nature-inspired metaheuristic algorithms: 1st Edn. Luniver Press, UK (2008)
6. Senthilnath, J., Omkar, S., Mani, V.: Clustering using firefly algorithm. Performance study. Swarm, Evol. Comput. **1**(3), pp. 164–171 (2011)
7. Sayadi, M.K., Ramezanian, R., Ghaffari-Nasab, N.: A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. Int. J. Ind. Eng. Comput. **1**, 1–10 (2010)
8. Jati, G., Suyanto, S.: Evolutionary discrete firefly algorithm for travelling salesman problem. Adapt. Intell. Syst. 393–403 (2011)
9. Dorigo, M., Stützle, T.: Ant colony optimization: the MIT Press Cambridge, Cambridge (2004)

10. Nayak, S.K., Padhy, S.K., Panigrahi, S.P.: A novel algorithm for dynamic task scheduling. Future Gener. Comput. Syst. **28**5, p. 709 (2012)
11. Yousif, A., et al.: Intelligent Task Scheduling for Computational Grid, In: 1st Taibah University International Conference on Computing and Information Technology pp. 670–674 (2012)
12. Brucker, P.: Sched. algorithms: Springer, Verlag (2007)
13. Li, S., et al.: A GA-based NN approach for makespan estimation. Appl. Math. Comput. **185**(2), 1003–1014 (2007)
14. Di Martino, V., Mililotti., .M :Scheduling in a grid computing environment using genetic algorithms **30**5-6, pp. 553-565 (2002)
15. De Falco, I., et al.: A distributed differential evolution approach for mapping in a grid environment. In: Parallel, Distributed and Network-Based Processing, 2007. PDP'07. 15th EUROMICRO International Conference on IEEE, Weimar, Germany (2007)
16. Selvi, S., Manimegalai, D., Suruliandi, A.: Efficient job scheduling on computational grid with differential evolution algorithm. Int. J. Comput. Theory Eng. **3**, 277–281 (2011)
17. Izakian, H., et al.: A discrete particle swarm optimization approach for grid job scheduling. Int. J. Innovative Comput Information Control **6**(9), 4219–4233 (2010)
18. Entezari M.R., Movaghar, A.: A genetic algorithm to increase the throughput of the computational grids. Int. J. Grid Distrib. Comput. **4**(2), (2011)
19. Talukder, A., Kirley, M., Buyya, R.: Multiobjective differential evolution for scheduling workflow applications on global Grids. Concurrency Comput. Pract. Experience **21**(13), 1742–1756 (2009)
20. Abraham, A., et al.: Scheduling jobs on computational grids using fuzzy particle swarm algorithm. Springer, Berlin Heidelberg (2006)
21. Xu, Z., X. Hou, and J. Sun.: Ant algorithm-based task scheduling in grid computing.In: Proceedings of the Canadian Conference on Electrical and Computer IEEE. (2003)
22. Yan, H., et al.: An improved ant algorithm for job scheduling in grid computing. In: Machine Learning and Cybernetics, 2005. In:Proceedings of 2005 International Conference on IEEE. (2005)
23. Basu, B., Mahanti, G.K.: Fire Fly and Artificial Bees Colony Algorithm for Synthesis of Scanned and Broadside Linear Array Antenna. Prog. Electromagnet. Res. **32**, 169–190 (2011)
24. Zhang, L., et al.: A task scheduling algorithm based on pso for grid computing. Int. J. Comput. Intell. Res. **4**(1), 37–43 (2008)
25. Chen, T., et al.: Task scheduling in grid based on particle swarm optimization. In: Parallel Distributed Computing, 2006. ISPDC'06. The Fifth International Symposium IEEE (2006)
26. Kang, Q., et al.: A novel discrete particle swarm optimization algorithm for job scheduling in grids. Nature of Computing ICNC'08. In: Fourth International Conference on IEEE (2008)
27. Zhang, L., Chen, Y., B. Yang.: Task scheduling based on PSO algorithm in computational grid. In: Intelligent Systems Design and Applications, 2006. ISDA'06. Sixth International Conference on IEEE, 696–704 (2006)
28. Mostaghim, S., Branke, J., Schmeck. H.: Multi-objective particle swarm optimization on computer grids. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation ACM (2007)
29. Yan-Ping, B.,Wei, Z., Jin S.: An improved PSO algorithm and its application to grid scheduling problem. In: Computer Science and Computational Technology, ISCSCT'08. International Symposium on IEEE (2008)
30. Meihong, W., Wenhua, Z., Keqing. W.: Grid Task Scheduling Based on Advanced No Velocity PSO. In: Internet Technology and Applications, 2010 International Conference on IEEE (2010)
31. Shi, Y., Eberhart, R.C.: Empirical study of particle swarm optimization. In: Evolutionary Computation, CEC 99. Proceedings of the Congress on IEEE (1999)
32. Dian, P.R., Siti, M.S., Siti, S.Y.: Particle Swarm Optimization: Technique, System and Challenges. Int. J. Comput. Appl. **14**(1), 19–27 (2011)
33. Yang, X.S., Firefly algorithm, Levy flights and global optimization. In: Bramer, M., Ellis, R., Petridis M. (Eds.) Research and Development in Intelligent Systems vol. XXVI, pp. 209–218. Springer, London (2010)

34. Jeklene, O.K.K.L.: OPTIMIZATION OF THE QUALITY OF CONTINUOUSLY CAST STEEL SLABS USING THE FIREFLY ALGORITHM. Materiali in tehnologije **45**(4), 347–350 (2011)
35. Onwubolu, G., Davendra, D.: Differential Evolution for Permutation-Based Combinatorial Problems. Differential Evolution: A Handbook for Global Permutation-Based Combinatorial, Optimization, pp. 13–34. (2009)
36. Tasgetiren, M.F., et al.: Particle swarm optimization algorithm for single machine total weighted tardiness problem. IEEE. (2004)
37. Buyya, R., Murshed, M.: Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency Comput.: Pract. Experience, **14**(13,15) 1175–1220 (2002)
38. Iosup, A., et al.: The grid workloads archive. Future Gener. Comput. Syst. **24**(7), 672–686 (2008)

A Parallelised Firefly Algorithm for Structural Size and Shape Optimisation with Multimodal Constraints

Herbert Martins Gomes and Adelano Esposito

Abstract In structural mechanics, mass reduction conflicts with frequency constraints when they are lower bounded since vibration mode shapes may easily switch due to shape modifications. This may impose severe restrictions to gradient-based optimisation methods. Here, in this chapter, it is investigated the use of the Firefly Algorithm (FA) as an optimization engine of such problems. It is suggested some new implementations in the basic algorithm, such as the parallelisation of the code, based on literature reports in order to improve its performance. It is presented several optimization examples of simple and complex trusses that are widely reported in the literature as benchmark examples solved with several non-heuristic and heuristic algorithms. The results show that the algorithm outperforms the deterministic algorithms in accuracy, particularly using the Parallel Synchronous FA version.

Keywords Firefly algorithm · Size and shape optimisation · Algorithm parallelisation

1 Introduction

During the last decades, the development of theories that aims at optimizing problems was based on mathematical programming. Methods like gradient descent, SIMPLEX, BFGS, Linear and Quadratic Sequential Programming are broadly used to solve a variety of engineering problems. The basic idea shared by these methods is that the gradient of the function to be optimized has important information to quickly find an optimum solution for a specific problem. However, when dealing with highly nonlinear, non-convex, non-differentiable, non-smooth and implicit problems (that are the opposite of the necessary conditions to the use of gradient based methods), these

H. M. Gomes (✉) · A. Esposito
Federal University of Rio Grande do Sul, Porto Alegre, RS, Brasil
e-mail: herbert@mecanica.ufrgs.br

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_14,

methods may present some convergence difficulties. Some of the today's engineering complex problems are challenging and may present such behaviour. Often, these methods get stuck in local optima.

Truss optimization is a parametric mechanical optimization problem. This type of mechanical structure is desirable when covering large areas with a minimum number of columns. This is a recurrent problem found in the literature since there are no theorems that define the theoretical optimum solution under stress/displacements and especially natural frequency constraints. This problem was unsuccessfully addressed by deterministic/gradient-based methods which presented convergence problems dealing with repeated natural frequencies and quick changes in mode shape in symmetrical structures when performing a parametric optimization. It is a very important real world problem since they can reduce drastically the amount of resources to build such structures. In this type of problem, mass reduction conflicts with frequency constraints when they are lower bounded since vibration shape modes may easily switch due to shape modifications. In addition, symmetrical structures present some repeated eigenvalues, which imposes extra difficulties to gradient methods. Thus, it is investigated the use of a Firefly Algorithm (FA) as an optimization engine. One important feature of the algorithm is based on the fact that it is non-gradient based, but based on single objective function evaluations. This is of major importance when dealing with non-linear optimization problems with several constraints, avoiding bad numerical behaviour due to gradient evaluations. The algorithm is revised, highlighting its most important features.

One of the reported disadvantages for the basic FA is its moderate performance related to processing time, which is more critical when exposed to complex optimisation problems where each function evaluation spent some considerable time. So, any improvement in the algorithm performance is welcome.

1.1 Literature Review

The use of Fireflies as an optimization tool was proposed by Yang [1] in 2008, when he conceived the algorithm. New researchers have been used the basic algorithm. However, some improvements in the basic algorithm have been proposed recently in order to enhance the method and compare with other metaheuristic algorithms. Few books related to this theme are available, such as Yang [1] and Yang [2].

The papers by Yang [3] in 2009 and [4] in 2010 conclude that the FA could carry out a nonlinear design optimization using stochastic test functions with singularities and this is potentially more powerful than other existing algorithms such as PSO. He warns that the convergence analysis still requires a theoretical framework.

Sayadia et al. [5] in 2010 successfully applied the FA as a minimisation engine in permutation flow shop scheduling problems (NP-Hard problem). Therefore, a direct solution is not available and Metaheuristic approaches need to be used to find the near-optimal solutions. The results of implementation of the proposed method were compared with other existing ant colony optimization technique. The results

indicated that the new proposed method performed better than the ant colony for some well-known benchmark problems.

More recently, Chai-ead et al. [6] in 2011, applied Bee Colony algorithm and Firefly algorithm to noisy non-linear optimization problems. They conducted numerical experimental tests that were analysed in terms of the best solution found so far, mean and standard deviation on both actual optimum and convergence times to the optimum. The Firefly algorithm seems to perform better when the noise levels are higher. The Bees algorithm provided the better levels of computation time and the speed of convergence. They concluded that the Firefly algorithm was more suitable to exploit a search space by improving the individuals' experience and simultaneously obtaining a population of local optimal solutions.

Finally, an enhanced variant of the algorithm was proposed by Yang [7], called Lévy-flight Firefly Algorithm. In the paper, it is claimed that the Lévy-flight firefly algorithm converges more quickly and deals with global optimization more naturally.

According to Yang [2], there is no universal method that provides a guaranteed optimum solution for any optimisation problem. In average, all the algorithms will present a similar performance over a broader number of problems.

In this way, in order to improve the performance of heuristic algorithms, several researchers have been successfully developed parallel versions for the basic serial versions. Basically, according to Subotic et al. [8, 9] there are two main group methods that define the goals of the parallelisation of bio-inspired parallelisation: (a) methods that aims at reducing running time and (b) methods that aims improving accuracy. Subotic et al. [8] presents one of the earlier parallelised versions of a FA which were applied to unconstrained continuum optimisation problems. Two approaches are implemented aiming the two previous goals. The idea of using more than one swarm in the same search space may give flexibility to the search and avoid getting stuck in local minima. They report that better results are obtained when the main firefly swarm is divided into two sub-swarms. New swarms are created by interchanging half of the best fireflies from each swarm. This allows sharing important information between swarms. It is reported that the algorithm improves accuracy and performance of the basic FA serial version.

Husselmann and Hawick [10, 11] in 2012 used a parallelised FA grid-boxing method in the CUDA (Compute Unified Device Architecture, a parallel computer platform created by NVIDIA for use of several Graphics Processing Units). The developed method could be used to find interactions between fireflies and improve performance by using a grid instead of just one list of "animats". The implementation of such algorithm allowed the parallel search of swarms keeping their interactions. Multimodal optimisation tests show good improvements.

In addition to what was mentioned, there are several other heuristic algorithms that are frequently explored and applied in parallel computation such as Parallel PSO (Belal and El-Ghazawi [12], Venter [13], Koh et al. [14], Mostaghim et al. [15], Esposito et al. [16], Parallel ABC (Narasimhan [17]), Parallel Ant Colony (Pedemonte et al. [18]), Parallel Cuckoo Search (Subotic et al. [9]), just to name a few.

In this chapter it is developed, described and compared four versions of a parallelised FA with serial algorithms applied to the structural size and shape optimisation with multimodal constraints.

2 The Firefly Algorithm

FA tries to mimic the flashing behaviour of swarms of fireflies. The bioluminescence serve as element of courtship rituals, methods of prey attraction, social orientation or as a warning signal to predators The rhythmic flash, the rate of flashing and the amount of time it remains on, form part of the system that brings both sexes together. Females respond to male's unique pattern of flashing in the same species.

Since the light intensity decays with the square of the distance, the fireflies have limited visibility to other fireflies. This plays an important role in the communication of the fireflies and the attractiveness, which may be impaired by the distance. The flashing light can be formulated in such a way that it is associated with the cost function to be optimized. In a maximization problem, the brightness is proportional to the objective function. In the development of the algorithm, some simplifications are assumed such as (Yang [1]): (a) it is assumed that all fireflies are unisex so they will be attracted to each other regardless of their sex; (b) the attractiveness is proportional do their brightness and they both decrease as the distance increases and (c) in the case of no existence of no brighter firefly on then, the fireflies will move randomly and (d) the brightness of a firefly is affected by its fitness (landscape of the objective function).

According to Yang [2], in the firefly algorithm, there are two important issues: (a) the variation of light intensity and (b) formulation of the attractiveness. For simplicity, one can always assume that the attractiveness of a firefly is determined by its brightness, which in turn is associated with the encoded objective function $f({}^{i}\mathbf{x})$. In the simplest case of maximization problems, the brightness I of a firefly i at a particular position ${}^{i}\mathbf{x} = (x_1, x_{2,\ldots,}x_{\mathrm{d}})^T$ (vector of design variables) can be chosen as $I({}^{i}\mathbf{x}) \propto f({}^{i}\mathbf{x})$. However, the attractiveness β is relative. Thus, it will vary with the distance r_{ij} between firefly i and firefly j. In addition, light intensity decreases with the distance from its source, and light is also absorbed by the media (air), so we should allow the attractiveness to vary with the degree of absorption (γ). In the simplest form, the light intensity $I(r_{ij})$ varies according to the inverse square law:

$$I(r_{ij}) = I_s/r_{ij}^2 \tag{1}$$

where I_s is the intensity at the source. For a given medium with a fixed light absorption coefficient γ, the light intensity I vary with the distance r_{ij} in the following form:

$$I(r_{ij}) = I_0 \exp(-\gamma r_{ij}^2) \tag{2}$$

where I_0 is the original light intensity.

As the firefly's attractiveness is proportional to the light intensity seen by adjacent fireflies, one can define the attractiveness β of a firefly by:

$$\beta(r_{ij}) = \beta_0 \exp(-\gamma r_{ij}^2) \tag{3}$$

where β_0 is the attractiveness at $r_{ij} = 0$. Equation (3) defines a characteristic distance $\Gamma = \gamma^{-1/2}$ over which the attractiveness changes significantly from it β_0 to $\beta_0 e^{-1}$. In fact, according to Yang [2] the attractiveness function $\beta(r_{ij})$ can be any monotonically decreasing functions such as $\beta(r_{ij}) = \beta_0 \exp(-\gamma r_{ij}^m)$. For a fixed γ, the characteristic length becomes $\Gamma = \gamma^{-1} \to 1$ when $m \to \infty$. Conversely, for a given characteristic length scale Γ in an optimization problem, the parameter γ is used as a typical initial value $\gamma = 1/\Gamma^m$. This was the way the heuristic parameters are tuned for the numerical examples.

The movement of a firefly i attracted to a brighter one j is simply determined by:

$$^i\mathbf{x} = {}^i\mathbf{x} + \beta_0 \exp(-\gamma r_{ij}^2)({}^i\mathbf{x} - {}^j\mathbf{x}) + \alpha {}^i\varepsilon \tag{4}$$

where the second term is due to the attraction and the third term $\alpha {}^i\boldsymbol{\varepsilon}$ is a randomization term composed by a vector of random numbers ($^i\boldsymbol{\varepsilon}$) drawn from a uniform distribution $[-1, 1]$ or a Gaussian distribution and a randomisation parameter α that can decrease along iterations in order to decrease the perturbation effect in the following firefly moves.

For most implementations, one can take $\beta_0 = 1$ and $\boldsymbol{\varepsilon} = \text{rand}[-1, 1]$ and $\alpha = 1$. At this point, it is worth noting that Eq. (4) is a random walk biased towards brighter fireflies. If $\beta_0 = 0$, it becomes a simple random walk. The parameter γ characterizes the variation of the attractiveness, and its value is crucially important in determining the speed of the convergence and how the FA algorithm behaves. As indicated by Yang [1], $\gamma \in [0, \infty)$ but for most applications, due to Γ values of the system to be optimized, it typically varies from 0.1 to 10. Figure 1 shows the pseudo-code of the firefly algorithm adapted to minimize functions.

Another important implementation that can improve the basic FA is the use of a decreasing alpha value along iterations. This implies random walks that are smaller along the iteration. This technique is borrowed from the Simulated Annealing Algorithm and improves the convergence behaviour. The alpha value is exponentially updated along iterations i (for all examples) as $^i\alpha = {}^{i-1}\alpha\,(0.99)$.

In order to compare (when possible) algorithm's speed and efficiency, the same set of random numbers was used by fixing the seed of the random number generator. For statistical purposes, when comparing accuracy, the random number generator was set free for random independent runs.

2.1 The Proposed Parallelised Versions of FA

In the last decade there were great advances in computer cluster systems composed by several computer unities each with multiple cores. This made affordable the use of parallel solvers for large scale problems. This favoured parallelised algorithms, particularly those bio-inspired. In this context, the implementation of algorithms that benefits the parallel way of work could make feasible the use of such algorithms in complex and time consuming problems.

The previously introduced FA is now presented using multiple cores by an implementation in the Matlab® environment assuming minimisation of cost function, which is the case of structural mass minimisation that will be treated in this paper. As indicated by the pseudo code of Fig. 1, the basic FA algorithm calls the objective function for each firefly at the beginning of the process. This obviously is an independent task that can be performed completely in parallel. In the other hand, once started the main loop, when comparing brightness of fireflies, each time a firefly moves towards a brighter one, the objective function should be evaluated again, since the movement means that the design variables had been changed as the fitness functions too. In this case, each firefly is evaluated once in a sequential (serial) way being not possible a parallel evaluation.

This can be reduced if the fitness function evaluation is delayed and take place outside the "*i*" main loop. One can say that this means performing attractiveness

Objective Function $f(\mathbf{x})$ with $\mathbf{x} = (x_1, x_1, ..., x_d)^T$, *d*= no. of design variables
Generate an initial population of fireflies randomly ${}^i\mathbf{x}$, $i = 1, 2, ..., n$, *n*=no. of fireflies, ranked in ascending order.
Light intensity *I* at ${}^i\mathbf{x}$ is determined by $f({}^i\mathbf{x})$ for all fireflies.
Define the light absorption coefficient γ based on the Γ value of the initial population
While *t* < maximum number of generations or convergence criteria are met
  For *i*=1 to *n*
    For *j*=1 to n
      If (${}^iI > {}^jI$), move *i* towards *j*
        Calculate the distance $r_{ij} = \| {}^i\mathbf{x} - {}^j\mathbf{x} \|$
        Calculate $\beta = \beta_0 \exp(-\gamma r_{ij}^2)$
        Update design variables ${}^i\mathbf{x} = {}^i\mathbf{x} + \beta({}^i\mathbf{x} - {}^j\mathbf{x}) + \alpha\, {}^i\boldsymbol{\varepsilon}$
        Update ${}^iI = f({}^i\mathbf{x})$
      End if
    End For *j*
  End For *i*
  Rank fireflies in ascending order and find the best firefly so far (the first).
End While
Post process results

Fig. 1 Pseudocode for sequential asynchronous firefly algorithm (SAFA) to minimise functions. Adapted from Yang [2]

comparisons that are not precise, since there were movements of the fireflies during the comparisons. In fact, since the swarm of fireflies are ranked in ascending order, for a given firefly i from the main outer loop, this firefly i will move (will be updated) towards any other firefly j ($j < i$) provided it (j) is more attractive. So, updating the attractiveness at the end of "j" inner loop means assuming a movement of firefly i that is weighted by all the other more attractive fireflies. In other words, the movement of the firefly i is a weighted sum (in a vector sense of Eq. (4)) towards just the other more attractive fireflies.

When the evaluation of the attractiveness of the fireflies are performed at the end of the comparisons, it is said that this is a Synchronous algorithm, since all evaluations are performed without exchange of information between fireflies (no matter whether evaluated sequentially one by one, or in parallel, all at the same time).

If during comparisons of attractiveness, re-assed values of attractiveness for those fireflies that had moved are made available for the remaining comparisons, this is said to be an Asynchronous versions of the algorithm. With this in mind, there are four possible versions for the algorithm: (a) a Sequential Synchronous (SSFA), (b) a Sequential Asynchronous (SAFA), (c) a Parallel Synchronous (PSFA) and (d) a Parallel Asynchronous (PAFA). In the author's point of view, a Parallel Asynchronous (PAFA) version of the algorithm is partially possible. In other words, this can be achieved by performing the evaluation of the updated position of the fireflies that had been moved at the end of the "j" loop and not at the end of "i" loop. Moreover, the comparisons should be initialized from the most attractive firefly to the less ones (loop $i = 1$ to n) and then compared to the rest of the fireflies ($j = i + 1$ to n). So, for those fireflies in the "j" loop that had been compared to firefly "i" and moved, its attractiveness can be updated at the end of the "j" loop in order to be available to the other fireflies in the next step of the "i" loop. This makes sense only if the fireflies are ranked in ascending order in the minimisation problem.

These assumptions allow the algorithm to be stated as indicated by the pseudocodes in Figs. 2 and 3 and can be readily parallelised. The pseudocode for the version SSFA is the same for the PSFA, except for the final evaluation that is performed sequentially.

3 Performance Metrics

It is important to define metrics in order to compare the proposed versions of algorithms. In a broad sense, metrics related to accuracy and performance can be defined in many ways. The "execution time", i.e., the time from the start of the problem till the end of the problem is a very popular one (wall-clock time). Although there is a dependency of this metric on the machine used, it can be useful when working with the same machine and running several versions of a code. One way to avoid this problem is to assume the number of objective function calls as a measure of execution time or speed that is independent of machine characteristics.

Another metric is the "speed-up". The speed-up refers to how much a parallel algorithm is faster than a corresponding sequential algorithm. The robustness of an

Objective Function $f(\mathbf{x})$, with $\mathbf{x} = (x_1, x_1, ..., x_d)^T$, d= no. of design variables

Generate an initial population of fireflies randomly ${}^i\mathbf{x}$, i=1,2,…,n where n=no. of fireflies, ranked in ascending order.

Light intensity ${}^iI$ at ${}^i\mathbf{x}$ is determined by ${}^iI = f({}^i\mathbf{x})$ in parallel

Define the light absorption coefficient γ based on the Γ value of the initial population

While t < maximum number of generations or convergence criteria are met

  For i=1 to n

    For j=1 to n

      If (${}^iI > {}^jI$), move i towards j

        Calculate the distance $r_{ij} = \| {}^i\mathbf{x} - {}^j\mathbf{x} \|$

        Calculate $\beta = \beta_0 \exp(-\gamma r_{ij}^2)$

        Update design variables ${}^i\mathbf{x} = {}^i\mathbf{x} + \beta({}^i\mathbf{x} - {}^j\mathbf{x}) + \alpha\, {}^i\boldsymbol{\varepsilon}$

      End if

    End For j

  End For i

  Update ${}^kI = f({}^k\mathbf{x})$ for k=1 to no. of fireflies that moved (in parallel)

  Rank fireflies in ascending order and find the best firefly so far (the first).

End While

Post process results

Fig. 2 Pseudocode for PSFA to minimise functions

Objective Function $f(\mathbf{x})$, with $\mathbf{x} = (x_1, x_1, ..., x_d)^T$, d= no. of design variables

Generate an initial population of fireflies randomly ${}^i\mathbf{x}$, i=1,2,…,n where n=no. of fireflies, ranked in ascending order.

Light intensity ${}^iI$ at ${}^i\mathbf{x}$ is determined by ${}^iI = f({}^i\mathbf{x})$ in parallel

Define the light absorption coefficient γ based on the Γ value of the initial population

While t < maximum number of generations or convergence criteria are met

  For i=1 to n

    For j=1 to n

      If (${}^iI > {}^jI$), move i towards j

        Calculate the distance $r_{ij} = \| {}^i\mathbf{x} - {}^j\mathbf{x} \|$

        Calculate $\beta = \beta_0 \exp(-\gamma r_{ij}^2)$

        Update design variables ${}^i\mathbf{x} = {}^i\mathbf{x} + \beta({}^i\mathbf{x} - {}^j\mathbf{x}) + \alpha\, {}^i\boldsymbol{\varepsilon}$

      End if

    End For j

  End For i

  Update ${}^kI = f({}^k\mathbf{x})$ for k=1 to no. of fireflies that moved (in parallel)

  Rank fireflies in ascending order and find the best firefly so far (the first).

End While

Post process results

Fig. 3 Pseudocode for PAFA to minimise functions

optimisation algorithm is related to the scatter of obtained results (for instance, measuring the standard deviation or the coefficient of variation) for several independent runs of the algorithm.

The algorithm's "accuracy" is related to how accurate is the mean value of several independent runs when comparing to a benchmark value (mathematically obtained for closed-form solutions, or the best result obtained by some author so far) And finally, the algorithm's "efficiency", that is the ratio of speed-up and the number of processors used to solve a given problem, which measures how efficiently the algorithm uses the parallel resources. These simple metrics will be used in this chapter to evaluate the performance of the proposed algorithms.

4 Structural Size and Shape Optimization with Multimodal Constraints

The problem of truss structural size and shape optimization with multimodal constraints can be mathematically stated as:

$$\begin{array}{ll} \text{Minimise Mass} = \sum_{i=1}^{n} L_i \rho_i A_i \; i = 1, \ldots, n \text{ for all bars} & \\ \text{Subjected to } \omega_j \geq \omega_j^* & \text{for some Circular frequencies } j \\ \omega_k \leq \omega_k^* & \text{for some Circular frequencies } k \\ \text{and} & \\ A_{l\,\min} \leq A_l & \text{for some bar cross sectional areas } l \\ \mathbf{x}_{q\,\min} \leq \mathbf{x}_q \leq \mathbf{x}_{q\,\max} & \text{for some nodal coordinates } q \end{array} \tag{5}$$

where L is the bar length, ρ is the material density, A is the bar cross-sectional area, ω is the circular frequency, $\mathbf{x}$ is the coordinate vector of truss nodes and PF is the Penalization Factor defined as the sum of all active constraint violations. The constraint violations are treated with the penalty function technique so the objective function to be minimised is modified to:

$$\text{Mass} = \left(\sum_{i=1}^{n} L_i \rho_i A_i \right) (1 + PF) \text{ for all bars} \tag{6}$$

This penalization is applied to the frequency constraints, as indicated by Eq. (7), but is also valid to stress and displacement constraints when necessary.

$$PF = \sum_{i=1}^{nc} \left| \frac{\omega_i}{\omega_i^*} \pm 1 \right| \text{ for all active constraints} \tag{7}$$

where nc is the number of equality or inequality constraints.

This formulation allows, for solutions with active constraints, objective function values that are always greater than the non-violated ones.

5 Numerical Examples

The synchronous and asynchronous implementations as well as their respective parallel extensions were used to solve some of the several structural problems often explored in literature. The objective of the present study basically relies on: to measure the quality of the proposed implementations in terms of the accuracy and robustness. For the values of the accuracy and robustness, it was executed ten independent realisation for each implementation, whereas for the accuracy it was considered as reference value, the small mass found amongst all four implemented algorithms.

The mean number of function calls ($\text{Call}_{\text{mean}}$) was considered as convergence criteria for stopping iterations. This mean value considers the number of iterations to achieve a tolerance value of 10^{-3} for the best objective function value between iterations. The parallel approaches were implemented in an 8-core processor, AMD FX-8110, 2.81 GHz, 8.0 GB of RAM, in a Windows 7 64 bits platform, using the software Matlab®, which allows performing the computations in a parallel form using the resources available in the Parallel Computing Toolbox.

5.1 Ten Bar Truss Problem

The optimization of a 10-bar truss problem using the previously described FA algorithms is addressed here. It has fixed shape and variable continuous cross sectional area. At each free node it is attached a non-structural mass of 454.0 kg as depicted in Fig. 4. The material properties and design variable ranges are listed in Table 1.

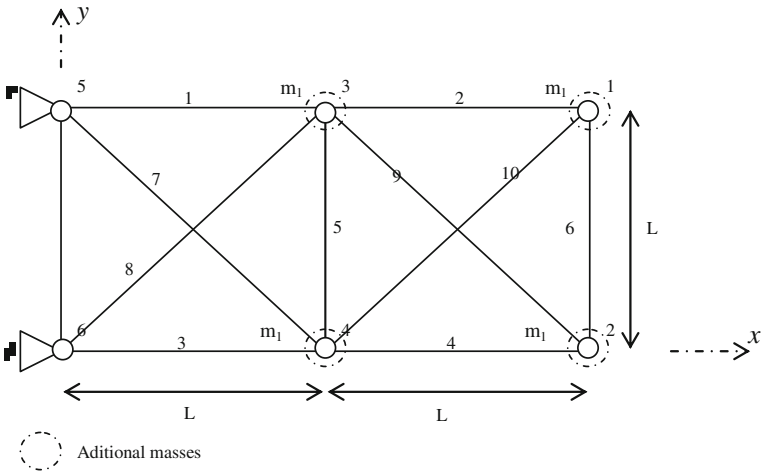

Fig. 4 10-bar truss structure with added masses

Table 1 Material properties and frequency constraints for the 10-bar truss structure

Property	Value	Unit
E (Young modulus)	6.89×10^{10}	N/m^2
ρ (Material density)	2770.0	kg/m^3
Added mass	454.0	kg
Design's variable lower bound	0.645×10^{-4}	m^2
Main bar's dimension	9.144	m
Constraints	$f_1 = 7, f_2 \geq 15, f_3 \geq 20$	Hz

This example was first solved by Grandhi and Venkayya [19] using the Optimality Criterion Algorithm (OCA). Sedeghati et al. [20] used a Sequential Quadratic Programming (SQP) with the finite element force method alongside to solve the problem. Wang et al. [21] used an Evolutionary Node Shift Method (ENSM) and Lingyum et al. [22] used Niche Hybrid Genetic Algorithm (NH-GA). Gomes [23, 24] used a PSO algorithm to solve the same problem. Table 2 shows the design variable results and the final mass for the optimized truss. It should be highlighted the good results obtained with the FA algorithm. In general, the FA algorithm implemented in this work reached the lowest mass among those found in the literature, as can be seen in Table 2. Particularly, it was obtained the lowest mass using serial asynchronous version of firefly (ASFA), i.e., 524.511 kg, 11.7 % lower than the other presented results (Grandhi [25]).

Table 5 shows a comparison between proposed algorithms by the optimum value of mass, accuracy, robustness and number of objective function calls. The comparison is performed in terms of mean values. For the accuracy, it is assumed with best mass value found by the SAFA, which is the lowest mass found between the algorithms so far, as can be seen in Table 2.

A common aspect for the four implementations consists in the quality of the solution of algorithm. It can be noticed in Table 5 good values for accuracy (low values for the difference between for mean mass for independent runs and the optimum value) and robustness (low standard deviation) indicating repeatability of the method in finding optimum solution reinforcing thus the reliability in the implementations. This feature is more expressive in the asynchronous version, presenting lower values for mean mass than synchronous versions, as less number of objective function calls than synchronous version. Table 4 shows the used heuristic parameters in the simulations with the four versions of the firefly algorithm.

Table 3 shows the first seven natural frequencies constraints obtained by each of the proposed methods showing that none of them is violated.

Figures 5 and 6 shows the performance of the algorithms using the following metrics: execution time, speed-up and efficiency. Observing Fig. 5 it is possible to say that the parallel extensions needed longer execution times. This computation cost becomes noticeable in the parallel asynchronous version of the FA. This fact is confirmed in terms of speedup, which has registered values lower than one, making

Table 2 Optimal design cross sections (cm^2) for several methods (total mass does not consider added masses, 10-bar truss problem)

Elem. no.	Wang [21]	Grandhi [26]	Sedaghati [20]	Lingyum [22]	Gomes [25]	Present work			
	ENSM	OCA	SQP	NH-GA	FA	SSFA	SAFA	PSFA	PAFA
1	32.456	36.584	38.245	42.234	31.198	35.582	35.070	34.996	35.514
2	16.577	24.658	09.916	18.555	14.030	14.818	14.622	14.104	14.679
3	32.456	36.584	38.619	38.851	34.754	34.667	35.076	35.437	35.115
4	16.577	24.658	18.232	11.222	14.900	14.902	14.606	14.845	14.825
5	02.115	04.167	04.419	04.783	00.645	00.645	00.645	00.645	00.645
6	04.467	02.070	04.419	04.451	04.672	04.548	04.583	04.588	04.565
7	22.810	27.032	20.097	21.049	23.467	23.506	23.819	23.133	23.591
8	22.810	27.032	24.097	20.949	25.508	23.630	23.759	24.756	23.811
9	17.490	10.346	13.890	10.257	12.707	12.482	12.453	12.655	12.325
10	17.490	10.346	11.452	14.342	12.351	12.422	12.418	11.940	12.216
Mass (kg)	553.80	594.00	537.01	542.75	531.28	524.534	524.511	524.680	524.562

Table 3 Optimized frequencies (Hz) with several methods for the 10-bar truss structure

f (Hz)	Wang [21]	Grandhi [26]	Sedaghati [20]	Lingyum [22]	Gomes [25]	Present work			
	ENSM	OCA	SQP	NH-GA	FA	SSFA	SAFA	PSFA	PAFA
1	7.011	7.059	6.992	7.008	7.000	7.000	7.000	7.000	7.000
2	17.302	15.895	17.599	18.148	16.136	16.209	16.185	16.187	16.215
3	20.001	20.425	19.973	20.000	20.000	20.003	20.000	19.999	19.999
4	20.100	21.528	19.977	20.508	20.152	20.005	20.016	20.010	20.003
5	30.869	28.978	28. 173	27.797	28.645	28.492	28.588	28.371	28.448
6	32.666	30.189	31.029	31.281	29.443	29.004	28.957	29.003	28.880
7	48.282	54.286	47.628	48.304	48.779	48.712	48.516	48.492	48.623

Table 4 Heuristic parameters used in the simulations (10-bar truss problem)

No. of fireflies n	Alpha coefficient α	Absorption coefficient γ	Minimum attractiveness β_0
30	0.6	50	0.4

Table 5 Statistical results for the 10-bar truss structure problem (10 independent runs)

Method	$\text{Mass}_{\text{mean}}$ (kg)	Accuracy (kg)	Robustness (kg)	$\text{Call}_{\text{mean}}$
SSFA	526.072	1.560	2.604	17181
SAFA	524.579	0.067	0.052	15000
PSFA	526.107	1.595	2.514	17181
PAFA	524.604	0.092	0.032	14400

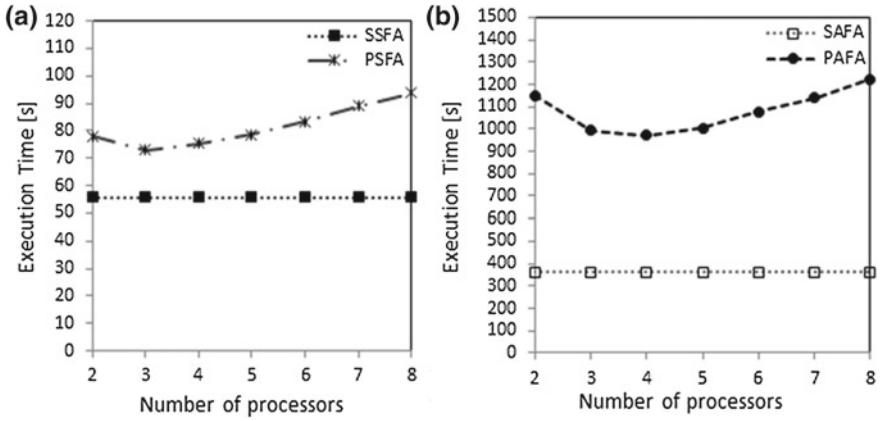

Fig. 5 Execution time of the (**a**) synchronous and (**b**) asynchronous algorithms (10-bar truss problem)

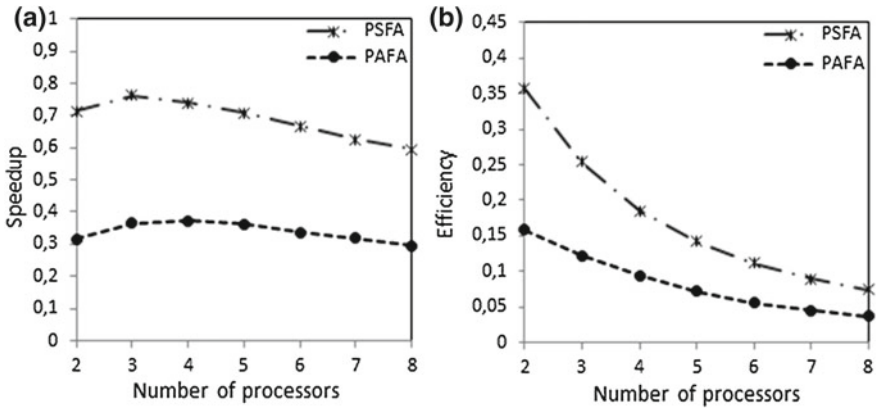

Fig. 6 (**a**) Speed-up and (**b**) Efficiency of the parallel algorithms (10-bar truss problem)

the application of both parallel extensions not so worth in the case of the simple 10-bar truss problem (the size of the problem is not worth for parallelisation).

5.2 Shape and Size Optimisation of 37-bar Truss Problem

It is a simple supported Pratt Type 37-bar truss as indicated in Fig. 7. There are non-structural masses of $m = 10\,\text{kg}$ attached to each of the bottom nodes of the lower chord, which are modelled as bar elements with rectangular cross sectional areas of $4 \times 10^{-3}\text{m}^2$. The other bars are modelled as simple bar elements with initial sectional areas of $1 \times 10^{-4}\text{m}^2$. The material property for the bar elements are set as $\text{E} = 2.1 \times 10^{11}\text{N/m}^2$ and $\rho = 7800\,\text{kg/m}^3$. This example has been investigated by Wang et al. [21] using the Evolutionary Node Shift Method (ENSM), by Lingyum et al. [22] using the NH-GA algorithm and by Gomes [26] using a simple FA algorithm.

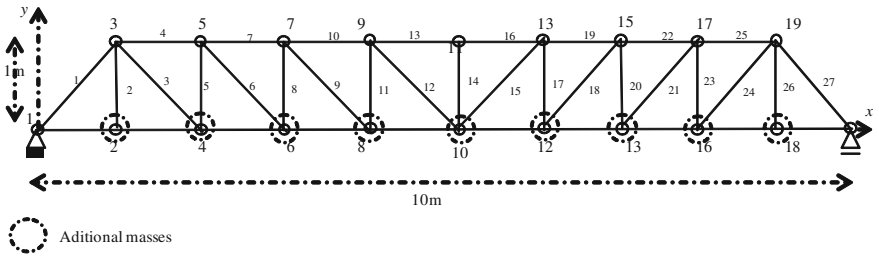

Fig. 7 37-bar truss structure with added masses

Table 6 Heuristic parameters used in the simulations of the algorithms (37-bar truss problem)

No. of fireflies n	Alpha coefficient α	Absorption coefficient γ	Minimum attractiveness β_0
40	0.6	50	0.4

This is considered a truss optimization on size and shape since all nodes of the upper chord are allowed to vary in the y-axis in a symmetrical way and all the diagonal and upper chord bars are allowed to vary its cross sectional areas starting from $A = 1 \times 10^{-4}\ m^2$. There are three constraints in the first three natural frequencies so that $f_1 \geq 20$ Hz, $f_2 \geq 40$ Hz, $f_3 \geq 60$ Hz. Therefore, it is considered a truss optimization problem with three frequency constraints and 19 design variables (five shape variables plus 14 sizing variables). Figure 7 shows a sketch of the 37-bar truss initial shape design.

Table 6 shows the used heuristic parameters for 10 independent runs of FA in the 37-bar truss problem. Table 7 shows the optimal design cross sections for several

Table 7 Optimal cross sections for several methods in the 37-bar truss problem (total mass does not consider added masses)

Variable no.	Initial design (Const. violated)	Wang [21] ENSM	Lingyum [22] NH-GA	Gomes [25] FA	Present work			
					SSFA	SAFA	PSFA	PAFA
Y_3, Y_{19}(m)	01.0	01.2086	01.1998	00.9311	00.9363	00.9563	00.9493	00.9491
Y_5, Y_{17}(m)	01.0	01.5788	01.6553	01.2978	01.3405	01.3725	01.3579	01.3618
Y_7, Y_{15}(m)	01.0	01.6719	01.9652	01.4694	01.5138	01.5507	01.5356	01.5382
Y_9, Y_{13}(m)	01.0	01.7703	02.0737	01.5948	01.6509	01.6844	01.6709	01.6741
Y_{11}(m)	01.0	01.8502	02.3050	01.7069	01.7015	01.7330	01.7244	01.7214
$A_1, A_{27}(cm^2)$	01.0	03.2508	02.8932	02.5706	02.9790	02.9304	02.6890	02.9981
$A_2, A_{26}(cm^2)$	01.0	01.2364	01.1201	00.8136	01.0788	00.9969	00.9698	00.9679
$A_3, A_{24}(cm^2)$	01.0	01.0000	01.0000	00.9543	00.7208	00.7377	00.7352	00.7090
$A_4, A_{25}(cm^2)$	01.0	02.5386	01.8655	02.3675	02.5407	02.5752	02.7749	02.5372
$A_5, A_{23}(cm^2)$	01.0	01.3714	01.5962	01.4265	01.2676	01.2482	01.4303	01.2188
$A_6, A_{21}(cm^2)$	01.0	01.3681	01.2642	01.4254	01.2121	01.2570	01.2192	01.2605
$A_7, A_{22}(cm^2)$	01.0	02.4290	01.8254	02.3638	02.6366	02.4628	02.4073	02.5734
$A_8, A_{20}(cm^2)$	01.0	01.6522	02.0009	04.3521	01.3225	01.3786	01.3722	01.4084
$A_9, A_{18}(cm^2)$	01.0	01.8257	01.9526	01.4801	01.4906	01.5377	01.4872	01.5193
$A_{10}, A_{19}(cm^2)$	01.0	02.3022	01.9705	03.4074	02.9578	02.4524	02.8217	02.5358
$A_{11}, A_{17}(cm^2)$	01.0	01.3103	01.8294	01.7063	01.3072	01.2233	01.2477	01.2174
$A_{12}, A_{15}(cm^2)$	01.0	01.4067	01.2358	01.2123	01.2572	01.3177	01.3355	01.3115
$A_{13}, A_{16}(cm^2)$	01.0	02.1896	01.4049	02.4078	02.3027	02.4001	02.3755	02.3752
$A_{14}(cm^2)$	01.0	01.0000	01.0000	01.0199	00.5005	00.5000	00.5000	00.5000
Total mass(kg)	336.3	366.50	368.84	361.75	359.16	358.96	359.20	358.95

Table 8 Optimised frequencies (Hz) with several methods for the 37-bar truss problem

f (Hz)	Initial design	Wang [21] ENSM	Lingyum [22] NH-GA	Gomes [25] FA	Present work SSFA	SAFA	PSFA	PAFA
1	8.89	20.0850	8.89	20.0005	20.0009	20.0002	20.0018	20.0001
2	28.82	42.0743	28.82	40.0168	39.9994	40.0006	40.0024	40.0029
3	46.92	62.9383	46.92	60.0561	60.0011	60.0044	59.9981	60.0058
4	63.62	74.4539	63.62	76.6520	75.6486	75.2654	75.5710	74.8859
5	76.87	90.0576	76.87	92.4318	93.6842	93.2407	92.9383	92.5635

Table 9 Statistical results for the 37-bar truss problem (10 independent runs)

Method	$\text{Mass}_{\text{mean}}$ (kg)	Accuracy (kg)	Robustness (kg)	$\text{Call}_{\text{mean}}$
SSFA	360.188	1.242	0.962	18432
SAFA	359.350	0.403	0.401	15004
PSFA	360.150	1.204	1.182	19576
PAFA	359.117	0.171	0.232	18411

methods. In this case, again the implementations of the FA algorithm designed in the present work found the best solution amongst all found in literature. Particularly, for this problem, the asynchronous parallel version outperformed in accuracy the remaining versions, finding a mass of 358.95 kg.

Table 8 shows the optimized frequencies for several methods and those obtained by the present work. It can be noticed that none of the frequency constraints is violated. Table 9 shows the statistical results for 10 independent runs for each FA approach in the 37-bar truss problem. Again, it can be seen the good accuracy of the implemented versions by the low values for accuracy and robustness. It is worth mentioning the good performance of the asynchronous versions that showed the best accuracy and robustness amongst algorithms, and the versions that used the lowest number of cost function calls.

It is possible to evaluate the performance of the proposed algorithms in the solution of the 37-bar truss problem. Regarding the execution time (see Fig. 8), it was noticed a different behaviour from the previous example (10-bar truss). It was verified that the parallel version of the synchronous algorithm achieved better processing speeds when compared with its serial version. This speed gain is measured in terms of speed-up and efficiency as indicated by Fig. 9, which shown the maximum speed-up of 2.72 obtained by the PSFA using four parallel processors. The PAFA continued to present the worst performance when compared to its serial version counterpart. Related to PAFA efficiency, despite being low, it shows a less accentuated trend than that presented by the PSFA.

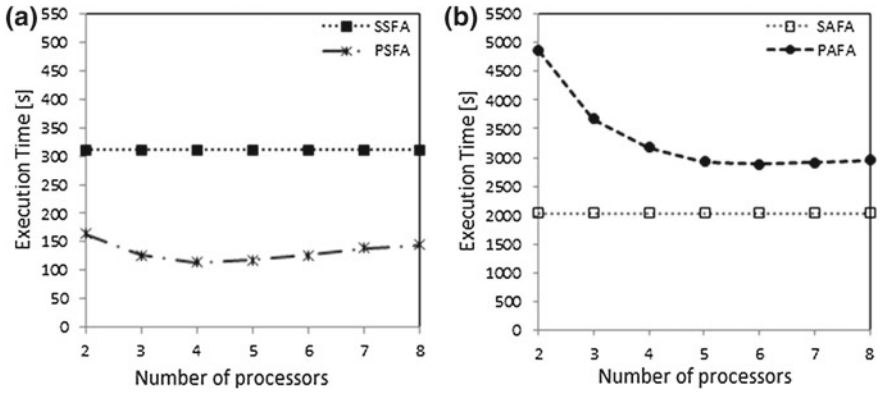

Fig. 8 Execution time of the (**a**) synchronous algorithms and (**b**) asynchronous algorithms

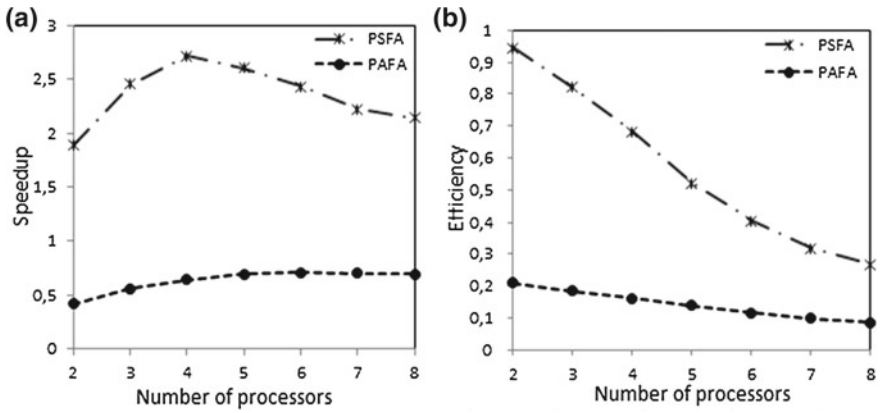

Fig. 9 (**a**) Speed-up and (**b**) efficiency of the parallel algorithms (37-bar truss problem)

5.3 Shape and Size Optimisation of a 52-Bar Dome Problem

In this example, a hemispherical space truss (like a dome) is optimized in shape and size with constraints in the first two natural frequencies. The space truss has 52 bars and non-structural masses of $m = 50\,\text{kg}$ are added to the free nodes. The cross-sectional areas are permitted to vary between 0.0001 and 0.001 m^2. The shape optimization is performed taking into account that the symmetry should be kept in the design process. Each movable node is allowed to vary $\pm 2\,\text{m}$. For the frequency constraint it is set that $f_1 \leq 15.916\,\text{Hz}$ and $f_2 \geq 28.649\,\text{Hz}$. A sketch of the initial design is depicted in Figs. 10 and 11. This example is considered to be a truss optimization problem with two frequency constraints and 13 design variables (five shape variables plus eight sizing variables). Table 10 summarizes the heuristic parameters used in the simulations.

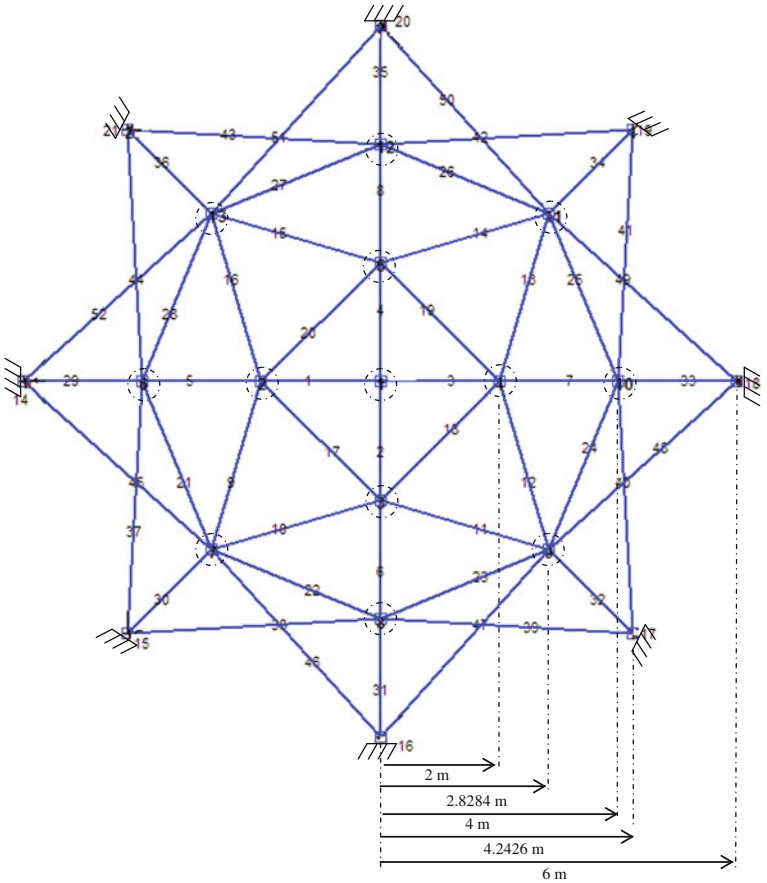

Fig. 10 Initial design of the 52-bar dome truss structure

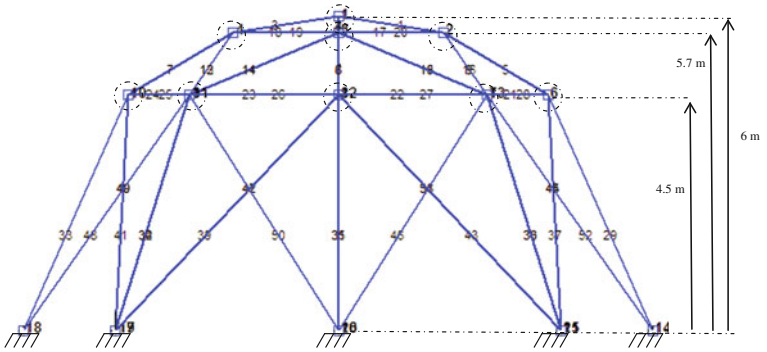

Fig. 11 Initial design of the 52-bar dome truss structure (*lateral view*)

Table 10 Heuristic parameters used in the simulations (52-bar dome problem)

No. of fireflies n	Alpha coefficient α	Absorption coefficient γ	Minimum attractiveness β_0
40	0.3	0.1	0.5

Table 11 Optimal cross section designs for several methods in the 52-bar truss problem (total mass does not consider added masses)

Variable no.	Initial design	Lin [27]	Lingyum [22]	Gomes [25]	Present work			
		OCA	NH-GA	FA	SSFA	SAFA	PSFA	PAFA
Z_A (m)	06.000	04.3201	05.8851	04.1700	05.9423	05.9385	06.0510	05.8213
X_B(m)	02.000	01.3153	01.7623	02.7575	02.6275	02.6169	02.6019	02.6025
Z_B (m)	05.700	04.1740	04.4091	03.8028	03.7000	03.7000	03.7000	03.7000
X_F (m)	02.828	02.9169	03.4406	04.2988	04.1967	04.1906	04.1829	04.1835
Z_F(m)	04.500	03.2676	03.1874	02.6011	02.5000	02.5000	02.5000	02.5000
A_1(cm^2)	02.000	01.00	01.0000	01.0000	01.0000	01.0000	01.0000	01.0000
A_2(cm^2)	02.000	01.33	02.1417	01.6905	01.5573	01.5311	01.5152	01.4780
A_3(cm^2)	02.000	01.58	01.4858	01.4776	01.0539	01.0573	01.0486	01.0727
A_4(cm^2)	02.000	01.00	01.4018	01.2130	01.2492	01.2595	01.2581	01.2945
A_5(cm^2)	02.000	01.71	01.9110	01.3697	01.2609	01.2709	01.2885	01.2782
A_6(cm^2)	02.000	01.54	01.0109	01.0070	01.0000	01.0000	01.0000	01.0000
A_7(cm^2)	02.000	02.65	01.4693	01.3383	01.2445	01.2472	01.2633	01.2455
A_8(cm^2)	02.000	02.87	2.1411	01.6682	01.6648	01.6570	01.6344	01.6558
Total mass (kg)	338.69	298.0	236.046	202.842	189.617	190.108	188.933	201.290

Table 11 shows the initial and final optimised coordinates, areas and mass. FA optimum mass is about 12 kg heavier than Gomes [24] optimum using PSO. It can be noticed that the PSO performed better than the other methods. FA performed better than Lin [27] that used Optimality Criterion Algorithm (OCA) and Lingyum [22] which used the NH-GA.

Table 11 shows the optimal design cross sections for several methods. Again in this case, the current implementation of the FA algorithm found the best solutions amongst all the other methods reported in the literature. For the current problem, the synchronous algorithm outperformed the three other versions, finding a minimum mass of 189.617 kg, 36.4 % lower than the worst of the reported results from the literature (Lin et al. [27]).

Table 12 shows the final optimized frequencies (Hz) for the methods. It can be noticed that none of constraints is violated.

Table 13 shows the statistical results for 10 independent runs for each version of FA in the 52-bar truss problem. It is also shown in the same Table 13 the respective values for accuracy, robustness and mean number of objective function calls. In this

Table 12 Optimized frequencies (Hz) with several methods for the 52-bar truss problem

f (Hz)	Initial design	Lin [27]	Lingyum [22]	Gomes [25]	Present work			
		OCA	NH-GA	FA	SSFA	SAFA	PSFA	PAFA
1	8.89	15.22	12.81	13.242	14.603	14.333	15.441	15.497
2	28.82	29.28	28.65	28.671	28.649	28.649	28.649	28.649
3	46.92	29.28	28.65	28.671	28.649	28.649	28.650	28.657
4	63.62	31.68	29.54	29.245	28.653	28.650	28.650	28.657
5	76.87	33.15	30.24	29.342	28.653	28.812	28.984	28.889

Table 13 Statistical results for the 52-bar truss problem (10 independent runs)

Method	$\text{Mass}_{\text{mean}}$ (kg)	Accuracy (kg)	Robustness (kg)	$\text{Call}_{\text{mean}}$
SSFA	189.617	0.739	1.025	26660
SAFA	190.108	1.231	1.132	23408
PSFA	188.933	0.056	0.047	27400
PAFA	201.290	12.413	14.402	21143

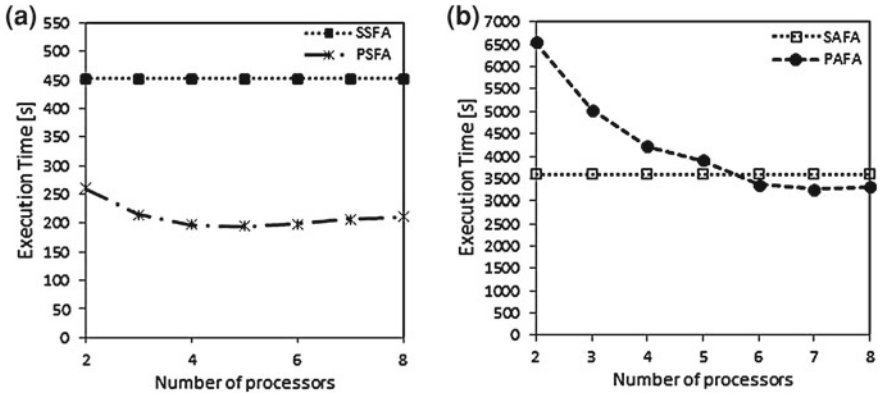

Fig. 12 Execution time for the (**a**) synchronous algorithms and (**b**) asynchronous algorithms

example, it was noticed that the PAFA algorithm did not present good result when compared to its serial counterpart (SSFA) as well as others versions (SAFA and PAFA). The other versions showed again a good behaviour with good accuracy and robustness. It can be noticed that the asynchronous versions were able to find good solutions within a small number of objective function calls when compared to their synchronous counterparts, despite the slightly higher mass value.

Figures 12 and 13 show the behaviour of the algorithms. As in the previous problem, the parallel extension of synchronous algorithm presented lower processing times.

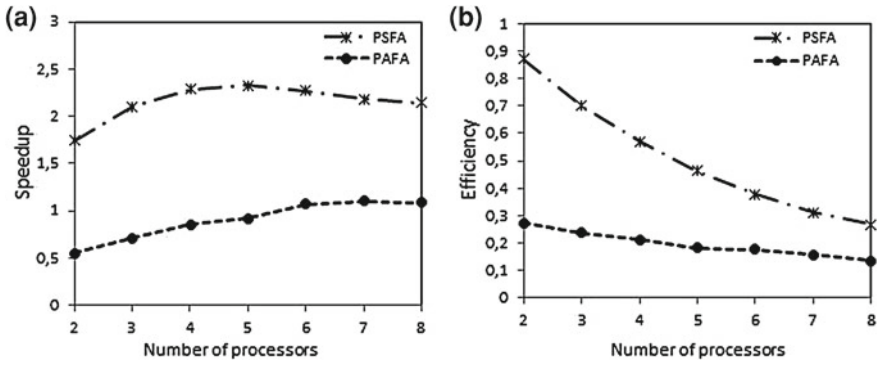

Fig. 13 (**a**) Speed-up and (**b**) efficiency of the parallel algorithms (52-bar truss problem)

Considering the asynchronous version, its parallel counterpart (PAFA) showed an improved behaviour so that with six processors, the processing times of the PAFA become less that its serial extension (SAFA) resulting in a small gain of the speed-up. This behaviour was not verified in the previous examples. Related to the efficiency in parallel processing, the proposed algorithms keep the previous behaviour shown in previous problems, exception for the efficiency of the PAFA, which showed a slight increase, approaching the PSFA efficiency.

6 Final Remarks

In this chapter, the problem of truss design optimization in shape and size with multimodal constraints was addressed. The constraints were treated as usual with the penalisation method. It is well known that this kind of optimization problem has high-nonlinear behaviour related to the frequency constraints especially for shape optimization, since eigenvalues are very sensitive to shape modifications. In the literature it is reported several methods that address this problem. In all the investigated problems one has to deal with eigenvalue-eigenvector symmetrical problem, a condition that may happen during the shape optimisation which represents a severe difficulty using gradient based optimisation algorithms.

An optimisation methodology was proposed based on the Firefly Algorithm (FA). The fact that the algorithm works with a population of probable solutions and heuristic parameters allows exploration/exploitation capabilities and this makes it likely to escape from local minima in the search process. It is well known that FA requires just a few heuristic parameters to be tuned and those are generally easy to link with the cost function magnitude of the analysed problem.

Four modified versions of the FA algorithm were proposed. Two of them take into account the possible parallelisation of the code and the other two take into account the possible synchronization of the cost function evaluation, namely: (a)

SSFA (Sequential Synchronous FA), (b) SAFA (Sequential Asynchronous FA), (c) PSFA (Parallel Synchronous FA) and (d) PAFA (Parallel Asynchronous FA). The first version (a) is the archetypal basic FA algorithm.

It was presented three examples of increasing difficulty that were compared with results in the literature. The well-known execution time(wall-clock time), speed-up and efficiency metrics were used in order to trace comparisons allowing the enhancements and issues related to the quality of solutions obtained with each one of the proposed versions.

Regarding the quality of the results, it can be noticed that in the three presented examples the Synchronous and Asynchronous versions presented the best optimum results amongst all other reported values in the literature. It is worth mentioning that the Asynchronous version presented in most of the cases (except for the third example) the best mass solution, accuracy and robustness.

The numerical results show that the Asynchronous version, particularly the Parallelised version, can found more easily found a better solution. In the other hand the Synchronous version presented a computational cost significantly higher than the Asynchronous version in all the examples. This behaviour may be easily explained by the fact that allowing the attractiveness to be updated along comparisons between fireflies and making the new values available for the following comparisons; this shortens the path toward the optimum, reducing the function calls. In fact, looking for a nature counterpart, the comparisons may occur in a random way and the attractiveness is updated instantaneously. This will be subject for future research.

What can be said is that the parallel extension of the FA allowed enhancing performance as the computation time of the structural analysis increased. This can be seen by the efficiency obtained with the PAFA in the optimization of the 52-bar dome example with an increasing speed-up for an increasing number of processors. Such feature can be attributed to the behaviour presented by the PSFA, which showed superior performance gains against PAFA, even for the simple 37-bar example, which presents the smallest computational time.

In fact, the addressed problems showed the overall behaviour of the proposed versions of the FA. Large scale problems are being envisioned in order to highlight some of the presented trends.

Acknowledgments The authors acknowledge the CNPq and CAPES support for this work.

References

1. Yang, X.-S.: Nature-Inspired Metaheuristic Algorithms, p. 115, 1st edn. Luniver Press, United Kingdom (2008)
2. Yang, X.-S.: Engineering Optimization: An Introduction with Metaheuristic Applications, p. 343. Wiley, USA (2010)
3. Yang, X.-S.: Firefly algorithms for multimodal optimization. In: Stochastic Algorithms: Foundations and Applications, SAGA 2009, Lecture Notes in Computer Science, No. 5792, pp. 169–178 (2009)

4. Yang, X.-S.: Firefly algorithm, stochastic test functions and design optimization. Int. J. Bio-Inspired Comput. **2**(2), 78–84 (2010)
5. Sayadia, M.K., Ramezaniana, R., Ghaffari-Nasaba, N.: A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. Int. J. Ind. Eng. Comput. **1**, 1–10 (2010)
6. Chai-ead, N., Aungkulanon, P., Luangpaiboon, P.: Bees and firefly algorithms for noisy non-linear optimization problems. In: Proceedings of the International Multi Conference of Engineering and Computer Scientists, IMECS 2011, vol. II, Hong-Kong (2011)
7. Yang, X.-S.: Firefly algorithm, lévy flights and global optimization. In: Bramer, M., Ellis, R., Petridis, M. (eds.) Research and Development in Intelligent Systems, vol. XXVI, pp. 209–218. Springer, London (2010)
8. Subtonic, M., Tuba, M., Stanarevic, N.: Parallelization of the firefly algorithm for unconstrained optimization problems, In: Astorakis, N., Mladenov, V., Bojkovic, Z. (eds.) Latest Advances in Information Science and Application, 11–13 May, pp. 264–269, WSEAS Press, Singapore. ISSN:1790-5109, ISBN: 978-1-61804-092-3 (2012)
9. Subtonic, M., Tuba, M., Stanarevic, N., Bacanin, N., Simian, D.: Parallelization cuckoo search algorithm for unconstrained optimization. In: Astorakis, N., Mladenov, V., Bojkovic Z. (ed.) Recent Researches in Applied Information Science, pp. 151–156, WSEAS Press, Faro. ISSN:1790-5109, ISBN: 978-1-61804-092-3 (2012)
10. Husselmann, A.V., Hawick, K.A.: Parrallel parametric optimisation with firefly algorithms on graphical processing units. In: Proceedings of the 2012 World Congress in Computer Science, Computer Engineering, and Applied Computing (2012)
11. Hawick, K.A., James, H.A., Scogings, C.J.: Grid-boxing for spatial simulation performance optimisation, In: Proceedings of the 39th Annual Simulation, Symposium (ANSS'06) (2012)
12. Belal, M., El-Ghazawi, T.: Parallel models for particle swarm optimizers: a powerful and promising approach. Stigmergic Optim. Stud. Comput. Intell. **31**, 239–259 (2006)
13. Venter, G., Sobieski, J.S.: A parallel PSO algorithm accelerated by asynchronous evaluations. In: Proceedings of the 6th World Congresses of Structural and Multidisciplinary Optimization, Rio de Janeiro (2005)
14. Koh, B., George, A.D., Haftka, R.T., Fregly, B.J.: Parallel asynchronous particles swarm optimization. Int. J. Numer. Meth. Eng. **67**, 578–595 (2006)
15. Mostaghim, S., Branke, J., Schmeck, H.: Multi-objective particle swarm optimization on computer grids. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO 2007, pp. 869–875 (2007)
16. Esposito, A., Gomes, H.M., Miguel, L.F.F.: Parallel computation applied to optimization of engineering problems by particle swarms (in Portuguese). In: Cardona, A., Kohan, P.H., Quinteros, R.D., Storti M.A. (eds.) Proceedings of the Mecánica Computacional, vol. XXXI, pp. 925–943. Salta (2012)
17. Narisimhan, H.: Parallel artificial bee colony (PABC) algorithm. In: Proceedings of 2009 World Congress on Nature and Biologically Inspired Computing, NaBIC 2009, pp. 306–311 (2009)
18. Pedemonte, M., Nesmachnow, S., Cancela, H.: A survey on parallel ant colony optimization. Appl. Soft Comput. **11**, 5181–5197 (2011)
19. Grandhi, R.V., Venkayya, V.B.: Structural optimization with frequency constraints. AIAA J. **26**(7), 858–866 (1988)
20. Sedaghati, R., Suleman, A., Tabarrok, B.: Structural optimization with frequency constraints using finite element force method. AIAA J. **40**(2), 382–388 (2002)
21. Wang, D., Zhang, W.H., Jiang, J.S.: Truss optimization on shape and sizing with frequency constraints. AIAA J. **42**(3), 1452–1456 (2004)
22. Lingyun, W., Mei, Z., Guangming, W., Guang, M.: Truss optimization on shape and sizing with frequency constraints based on genetic algorithm. J. Comput. Mech. **25**, 361–368 (2005)
23. Gomes, H.M.: A particle swarm optimization algorithm for truss optimization on shape and size with dynamic constraints. In: Proceedings of the 20th International Congress of Mechanical Engineering COBEM 2009, November, pp. 15–20. Brazil (2009)

24. Gomes, H.M.: Truss optimization with dynamic constraints using particle swarm algorithm. J. Expert Syst. Appl. **38**, 957–968 (2011)
25. Grandhi, R.V.: Structural optimization with frequency constraints—a review. AIAA J. **31**(12), 2296–2303 (1993)
26. Gomes, H.M.: A firefly metaheuristic structural size and shape optimisation with natural frequency constraints. Int. J. Metaheuristics **2**, 38–55 (2012)
27. Lin, J.H., Chen, W.Y., Yu, Y.S.: Structural optimization on geometrical configuration and element sizing with static and dynamic constraints. Comput. Struct. **15**(5), 507–515 (1982)

Intelligent Firefly Algorithm for Global Optimization

Seif-Eddeen K. Fateen and Adrián Bonilla-Petriciolet

Abstract Intelligent firefly algorithm (IFA) is a novel global optimization algorithm that aims to improve the performance of the firefly algorithm (FA), which was inspired by the flashing communication signals among firefly swarms. This chapter introduces the IFA modification and evaluates its performance in comparison with the original algorithm in twenty multi-dimensional benchmark problems. The results of those numerical experiments show that IFA outperformed FA in terms of reliability and effectiveness in all tested benchmark problems. In some cases, the global minimum could not have been successfully identified via the firefly algorithm, except with the proposed modification for FA.

Keywords Global optimization · Nature-inspired methods · Intelligent firefly algorithm.

1 Introduction

To date, a significant work has performed on global optimization and there have been significant algorithmic and computational developments including their applications to a wide variety of real-life and engineering problems. For illustration, recent advances in global optimization have been discussed by Floudas and Gounaris [1]. In particular, global optimization has and continues to play a major role in a number of engineering and science applications. As expected, finding the global optimum

S.-E. K. Fateen
Department of Chemical Engineering, Cairo University, Giza, Egypt
e-mail: sfateen@alum.mit.edu

A. Bonilla-Petriciolet (✉)
Department of Chemical Engineering, Instituto Tecnológico de Aguascalientes, Aguascalientes, México
e-mail: petriciolet@hotmail.com

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_15,

is more challenging than finding a local optimum and, in some real-life applications, the location of this global optimum is crucial because it corresponds to the correct and desirable solution. Basically, global optimization methods can be classified into two broad categories: deterministic and stochastic methods. The former methods can provide a guaranteed global optimum but they require certain properties of objective function and constraints such as continuity and convexity. On the other hand, stochastic global optimization methods aim at finding the global minimum of a given function reliably and effectively. These methods are attractive for solving challenging global optimization problems arising from real-life applications because they are applicable to any problem without the assumptions of continuity and differentiability of the objective function. Until now, several stochastic methods have been proposed and tested in challenging optimization problems using continuous variables and they include simulated annealing, genetic algorithms, differential evolution, particle swarm optimization, harmony search, and ant colony optimization. Recent nature-inspired optimization algorithms include cuckoo optimization [2], artificial bee colony optimization [3–5], honey bee mating optimization [6], and multi-colony bacteria foraging optimization [7]. In general, these methods may show different numerical performances and, consequently, the search for more effective and reliable stochastic global optimization methods has been an active area of research.

In particular, the Firefly algorithm (FA) [8] is a novel nature-inspired stochastic optimization method. It was inspired by the flashing behavior of fireflies. A few variants of this algorithm were recently developed, e.g.: discrete FA [9], multiobjective FA [10], Lagranian FA [11], Chaotic FA [12] and a hybrid between FA and ant colony optimization [13]. This relatively new method has gained popularity in finding the global minimum of diverse science and engineering application problems. For example, it was rigorously evaluated by Gandomi et al. [14] for solving global optimization problems related to structural optimization problems, and has been recently used to solve the flow shop scheduling problem [9], financial portfolio optimization [13], economic dispatch problems with valve loading effects [15] and phase and chemical equilibrium problems. Even FA is usually robust for continuous global optimization; the performance of available FA algorithms may fail to solve challenging application problems. Therefore, it is convenient to study new algorithm modifications and improvements to enhance its reliability and efficiency especially for large-scale problems. The aim of this chapter is to present a modification to the existing FA algorithm and to evaluate its performance in comparison with the original algorithm. The remainder of this chapter is divided as follows: Sect. 2 introduces the Firefly algorithm. Section 3 introduces the proposed modification and our new FA algorithm, namely the Intelligent Firefly Algorithm. The numerical experiments performed to evaluate the modification are presented in Sect. 4. The results of the numerical experiments are presented and discussed in Sect. 5. Section 6 summarizes the conclusions of this chapter.

2 Firefly Algorithm (FA)

As stated in this book, FA is a nature-inspired stochastic global optimization method that was developed by Yang [8]. The FA algorithm imitates the mechanism of firefly communications via luminescent flashes. In the FA algorithm, the two important issues are the variation of light intensity and the formulation of attractiveness. The brightness of a firefly is determined by the landscape of the objective function. Attractiveness is proportional to brightness and, thus, for any two flashing fireflies, the less bright one moves towards the brighter one.

In FA, the attractiveness of a firefly is determined by its brightness, which is equal to the objective function. The brightness of a firefly at a particular location x was chosen as $I(x) = f(x)$. The attractiveness is judged by the other fireflies. Thus, it was made to vary with the distance between firefly i and firefly j. The attractiveness was made to vary with the degree of absorption of light in the medium between the two fireflies. Thus, the attractiveness is given by

$$\beta = \beta_{min} + (\beta_o - \beta_{min})\, e^{-\gamma r^2} \tag{1}$$

The distance between any two fireflies i and j at x_i and x_j is the Cartesian distance:

$$r_{ij} = \boldsymbol{x}_i - \boldsymbol{x}_j = \sqrt{\sum_{k=1}^{d} \left(x_{i,k} - x_{j,k}\right)^2} \tag{2}$$

The movement of a firefly attracted to another more attractive (brighter) firefly j is determined by

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \beta \left(\boldsymbol{x}_j - \boldsymbol{x}_i\right) + \alpha \epsilon_i \tag{3}$$

The second term is due to the attraction, while ϵ_i in the third term is a vector of random numbers usually drawn from a uniform distribution in the range $[-0.5, 0.5]$. α is a parameter that controls the step. It can also vary with time, gradually reducing to zero, as used in many studies [8, 14, 16] which can help to improve the convergence rate of the optimization method.

3 Intelligent Firefly Algorithm (IFA)

In the original FA, the move, i.e., Eq. (3), is determined mainly by the attractiveness of the other fireflies; the attractiveness is a strong function of the inter distance between the fireflies. Thus, a firefly can be attracted to another firefly merely because it is close, which may take it away from the global minimum. The fireflies are ranked according to their brightness, i.e. according to the values of the objective function at their respective locations. However, this ranking, which is a valuable piece of

information *per se*, is not utilized in the move equation. A firefly is pulled towards each other firefly as each of them contributes to the move by its attractiveness. This behavior may lead to a delay in the collective move towards the global minimum. The idea behind of our Intelligent Firefly Algorithm (IFA) is to make use of the ranking information such that every firefly is moved by the attractiveness of a fraction of fireflies only and not by all of them. This fraction represents a top portion of the fireflies based on their rank. Thus, a firefly is acting intelligently by basing its move on the top ranking fireflies only and not merely on attractiveness.

A simplified algorithm for the IFA technique is presented in Fig. 1. The new parameter ϕ is the fraction of the fireflies utilized in the determination of the move. The original firefly algorithm is retained by setting ϕ to 1. This parameter is used as the upper limit for the index j in the inner loop. Thus, each firefly is moved by the top ϕ fraction of the fireflies only.

The strength of FA is that the location of the best firefly does not influence the direction of the search. Thus, the fireflies are not trapped in a local minimum. However, the search for the global minimum requires additional computational effort as many fireflies wander around uninteresting areas. With the intelligent firefly modifications, the right value of the parameter ϕ can maintain the advantage of not being trapped in a local minimum while speeding up the search for the global minimum. The right value of ϕ gives a balance between the ability of the algorithm not to be trapped in a local minimum and its ability to exploit the best solutions found. An iterative procedure can be used to reach a good value of ϕ that is suitable for

Step 1: Set parameters and Initialize

Generate initial population of n fireflies x_i
Define light absorption coefficient γ

Step 2: Main evolution loop

```
for k = 1 : MaxGeneration
    Evaluate new solutions and update light intensity determined by f_obj(x_i)
    Rank the fireflies and find the current best solution.
    Determine the number of top performers n_top = n * ϕ
    for i = 1 : n
        for j = 1 : n_top
            if (I_i > I_j)
                Calculate distance r between two butterflies
                Calculate attractiveness via exp[-γr^2]
                Move firefly i towards j via Eq. 3
            end if
        end for j
    end for i
end for k
```

Fig. 1 Simplified algorithm of intelligent firefly algorithm (IFA)

the problem to be solved. This iterative procedure is demonstrated in the numerical experiments below. We will show that this straightforward modification can improve significantly the performance of FA for continuous global optimization.

4 Numerical Experiments

Twenty classical benchmark functions were used to evaluate the performance of IFA as compared to the original FA. Tables 1 and 2 show the benchmark functions used along with their names, number of variables, variable limits and the value of the global minimum. The value of the new FA parameter ϕ was made to vary at four different values: 0.05, 0.1, 0.25 and 0.5. The effect of this parameter was studied for the twenty benchmark problems. The parameters for the original FA were kept constant in all experiments, i.e.: we used the value of 1 for β_o, 0.2 for β_{min}, 1 for γ, and α was made to decrease with the increase in the iteration number, k, in order to reduce the randomness according to the following formula:

$$\alpha_k = \left(1.11 \times 10^{-4}\right)^{b/iter_{max}} \alpha_{k-1} \tag{4}$$

Thus, the randomness is decreased gradually as the optima are approached. This formula was adapted from Yang [16]. The value of the parameter b was taken equal to 5 except for the Himmelblau, Powell, Wood, Rastrigin, Rosenbrock, Sine envelope sine wave, and Zacharov functions. This change in value was necessary to obtain solutions closer to the global optimum for those functions.

The 20 problems constitute a comprehensive testing for the reliability and effectiveness of the suggestion modification to the original FA. Eight functions have two variables only, yet some of them are very difficult to optimize. Surface plots of these functions are shown in Fig. 2.

To complete the evaluation of the IFE in comparison with the original FA algorithm, we have employed performance profile (PP) reported by Dolan and Moré [17], who introduced PP as a tool for evaluating and comparing the performance of optimization software. In particular, PP has been proposed to represent compactly and comprehensively the data collected from a set of solvers for a specified performance metric. For instance, number of function evaluations or computing time can be considered performance metrics for solver comparison. The PP plot allows visualization of the expected performance differences among several solvers and to compare the quality of their solutions by eliminating the bias of failures obtained in a small number of problems.

To introduce PP, consider n_s solvers (i.e. optimization methods) to be tested over a set of n_p problems. For each problem p and solver s, the performance metric t_{ps} must be defined. In our study, reliability of the stochastic method in accurately finding the global minimum of the objective function is considered as the principal goal, and hence the performance metric is defined as

Table 1 Benchmark functions used for testing the performance of FA and IFA

Name	Objective function
Ackley	$f_1 = 20\left(1 - e^{-0.2\sqrt{0.5\left(x_1^2+x_2^2\right)}}\right) - e^{0.5(\cos 2\pi x_1 + \cos 2\pi x_2)} + e^1$
Beale	$f_2 = (1.5 - x_1 + x_1x_2)^2 + \left(2.25 - x_1 + x_1x_2^2\right)^2 + \left(2.625 - x_1 + x_1x_2^3\right)^2$
Booth	$f_3 = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$
Carrom table	$f_4 = -\left[\cos x_1 \cos x_2 e^{\left\lvert 1-\sqrt{x_1^2+x_2^2}/\pi \right\rvert}\right]^2 /30$
Cross-leg table	$f_5 = -\left[\left\lvert \sin(x_1)\sin(x_2)\, e^{\left\lvert 100-\sqrt{x_1^2+x_2^2}/\pi \right\rvert}\right\rvert + 1\right]^{-0.1}$
Himmelblau	$f_6 = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_2^2 + x_1 - 7\right)^2$
Levy 13	$f_7 = \sin^2(3\pi x_1) + (x_1 - 1)^2\left[1 + \sin^2(3\pi x_2)\right] + (x_2 - 1)^2\left[1 + \sin^2(2\pi x_2)\right]$
Schaffer	$f_8 = 0.5 + \frac{sin^2\left[\sqrt{x_1^2+x_2^2}\right]-0.5}{\left[0.001\left(x_1^2+x_2^2\right)+1\right]^2}$
Helical valley	$f_9 = 100\left[(x_3 - 10\theta)^2 + \left(\sqrt{x_1^2 + x_2^2} - 1\right)^2\right] + x_3^2 \; where\; 2\pi\theta = \tan^{-1}\left(\frac{x_1}{x_2}\right)$
Powell	$f_{10} = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$
Wood	$f_{11} = 100\left(x_1^2 - x_2\right)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90\left(x_3^2 - x_4\right)^2 + 10.1\left[(x_2 - 1)^2 + (x_4 - 1)^2\right] + 19.8(x_2 - 1)(x_4 - 1)$
Cube	$f_{12} = \sum_{i=1}^{m-1} 100\left(x_{i+1} - x_i^3\right)^2 + (1 - x_i)^2$
Sphere	$f_{13} = \sum_{i=1}^{m} x_i^2$
Egg holder	$f_{14} = \sum_{i=1}^{m-1}\left\{-(x_{i+1} + 47)\sin\left(\sqrt{\lvert x_{i+1} + x_i/2 + 47\rvert}\right) + \sin\left[\sqrt{\lvert x_i - (x_{i+1} + 47)\rvert}\right](-x_i)\right\}$
Griewank	$f_{15} = \frac{1}{4000}\left[\sum_{i=1}^{m}(x_i - 100)^2\right] - \left[\prod_{i=1}^{m}\cos\left(\frac{x_i-100}{\sqrt{i}}\right)\right] + 1$
Rastrigin	$f_{16} = \sum_{i=1}^{m}\left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$
Rosenbrock	$f_{17} = \sum_{i=1}^{m-1} 100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2$
SES wave	$f_{18} = \sum_{i=1}^{m-1}\left\{0.5 + \frac{sin^2\left[\sqrt{x_{i+1}^2+x_i^2}\right]-0.5}{\left[0.001\left(x_{i+1}^2+x_i^2\right)+1\right]^2}\right\}$
Trigonometric	$f_{19} = \sum_{i=1}^{m}\left[m + i(1 - \cos x_i) - \sin x_i - \sum_{j=1}^{m}\cos x_j\right]^2$
Zacharov	$f_{20} = \sum_{i=1}^{m} x_i^2 + \left(\sum_{i=1}^{m} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{m} 0.5 i x_i\right)^4$

Table 2 Decision variables and global optimum of benchmark functions used for testing the performance of FA and IFA

Objective function	n_{var}	Search domain	Global minimum
Ackley	2	[−35, 35]	0
Beale	2	[−4.5, 4.5]	0
Booth	2	[−10, 10]	0
Carrom table	2	[−10, 10]	−24.157
Cross-leg table	2	[−10, 10]	−1
Himmelblau	2	[−5, 5]	0
Levy 13	2	[−10, 10]	0
Schaffer	2	[−100, 100]	0
Helical valley	3	[−1000,1000]	0
Powell	4	[−1000, 1000]	0
Wood	4	[−1000, 1000]	0
Cube	5	[−100, 100]	0
Sphere	5	[−100, 100]	0
Egg holder	50	[−512, 512]	959.64
Griewank	50	[−600, 600]	0
Rastrigin	50	[−5.12, 5.12]	0
Rosenbrock	50	[−50, 50]	0
Sine envelope sine wave	50	[−100, 100]	0
Trigonometric	50	[−1000, 1000]	0
Zacharov	50	[−5, 10]	0

$$t_{ps} = f_{calc} - f^* \tag{5}$$

where f^* is the known global optimum of the objective function and f_{calc} is the mean value of that objective function calculated by the stochastic method over several runs. In our study, f_{calc} is calculated from 30 runs to solve each test problem by each solver; note that each run is different because of random number seed used and the stochastic nature of the method. So, the focus is on the average performance of stochastic methods, which is desirable [18].

For the performance metric of interest, the performance ratio r_{ps} is used to compare the performance on problem p by solver s with the best performance by any solver on this problem. This performance ratio is given by

$$r_{ps} = \frac{t_{ps}}{\min\{t_{ps} : 1 \leq s \leq n_s\}}. \tag{6}$$

The value of r_{ps} is 1 for the solver that performs the best on a specific problem p. To obtain an overall assessment of the performance of solver$s$ on n_p problems, the following cumulative function for r_{ps} is used:

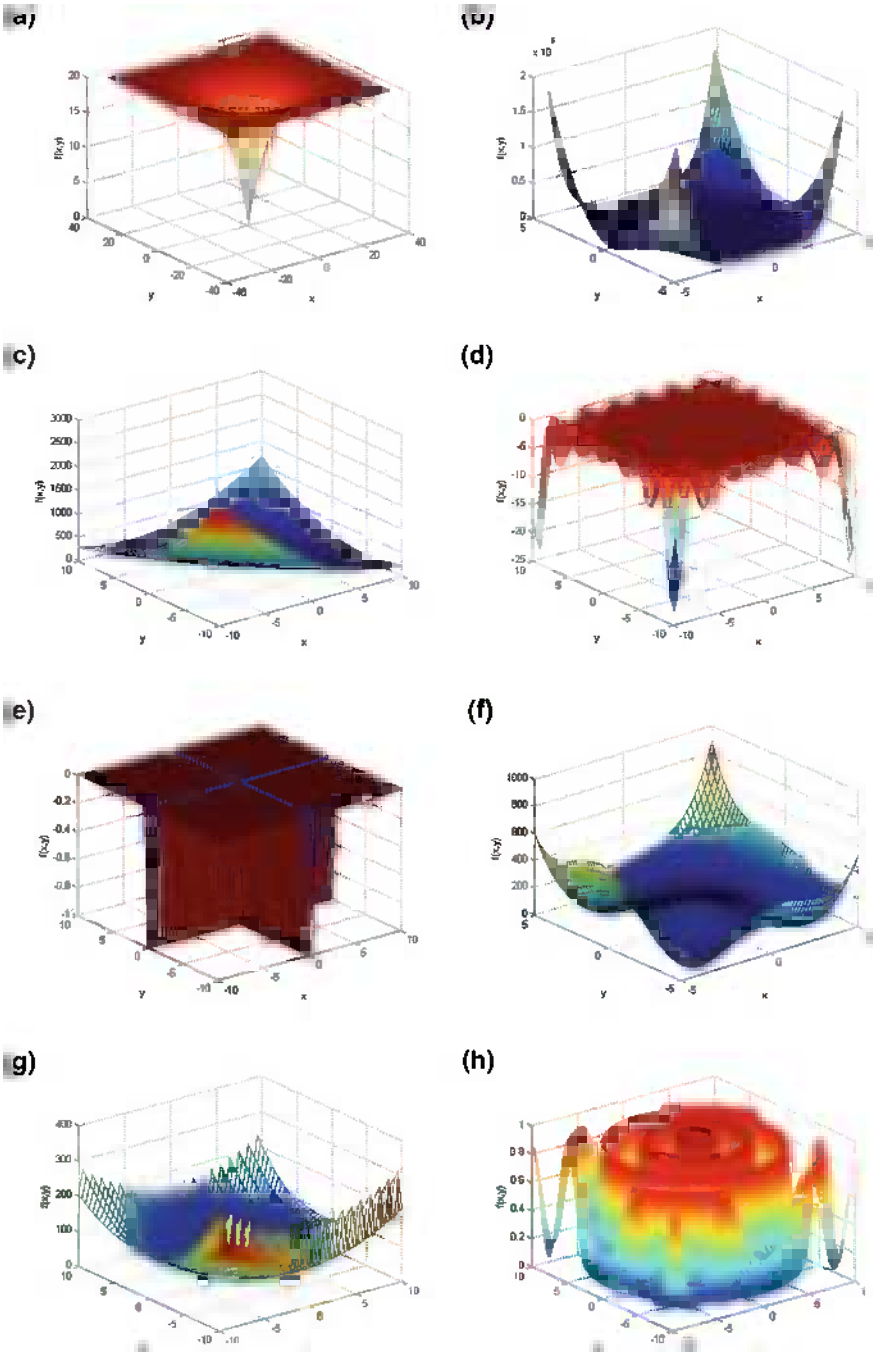

Fig. 2 Surface plots of the two-variable benchmark functions used in this study: **a** Ackley, **b** Beale, **c** Booth, **d** Carrom table, **e** Cross-leg table, **f** Himmelblau, **g** Levy 13, and **h** Schaffer

$$\rho_s(\varsigma) = \frac{1}{n_p} size\{p : r_{ps} \leq \zeta\} \tag{7}$$

where $\rho(\varsigma)$ is the fraction of the total number of problems, for which solver s has a performance ratio r_{ps} within a factor of ς of the best possible ratio. The PP of a solver is a plot of $\rho_s(\varsigma)$ versus ς; it is a non-decreasing, piece-wise constant function, continuous from the right at each of the breakpoints [17]. To identify the best solver, it is only necessary to compare the values of $\rho_s(\varsigma)$ for all solvers and to select the highest one, which is the probability that a specific solver will "win" over the rest of solvers used. In our case, the PP plot compares how accurately the stochastic methods can find the global optimum value relative to one another, and so the term "win" refers to the stochastic method that provides the most accurate value of the global minimum in the benchmark problems used.

5 Results and Discussion

As stated, each of the numerical experiments was repeated 30 times with different random seeds for IFA with four different-parameter values ($\phi = 0.05, 0.1, 0.25, 0.5$) and for the original FA algorithm ($\phi = 1$). The objective function value at each iteration for each trial was recorded. The mean and the standard deviation of the function values were calculated at each iteration. The global optimum was considered to be obtained by the method if it finds a solution within a tolerance value of 10^{-10}. The progress of the mean values is presented in Figs. 3, 4 and 5 for each benchmark function and a brief discussion of those results follows.

The Ackley function has one minimum only. The global optimum was obtained using all methods, as shown in Fig. 3a. However, the effectiveness of IFA is better than FA for all parameter values. No difference in the impact of the parameter value was observed. The improvement in performance was also clear with the Beale function Fig. 3b for ϕ values of 0.05 and 0.1. Beale has one minimum only, which was obtained satisfactory by all methods. Further iterations will improve their accuracies. The most effective method was the IFA with $\phi = 0.05$. The pattern of behavior for the five methods was also observed for the Booth function; IFA was significantly more effective than the FA as shown in Fig. 3c. The best performance was with the parameter value of 0.05, 0.1 and 0.25. The first three functions are relatively easy to optimize; the global optima were easily obtained.

For the Carrom table function, FA was unable to obtain the global minimum within the used tolerance, as shown in Fig. 3d. The global minimum was obtained by IFA with parameter value of 0.05, 0.1 and 0.25. On the other hand, the cross-leg table function is a difficult one to minimize. Its value at the global minimum is -1. IFA performed significantly better than FA with no apparent differences in performance between the four values of ϕ, as shown in Fig. 3e.

FA and IFA were both able to identify the global minimum of the Himmelblau function. Figure 3f shows the evolution of the mean best values. IFA performed more

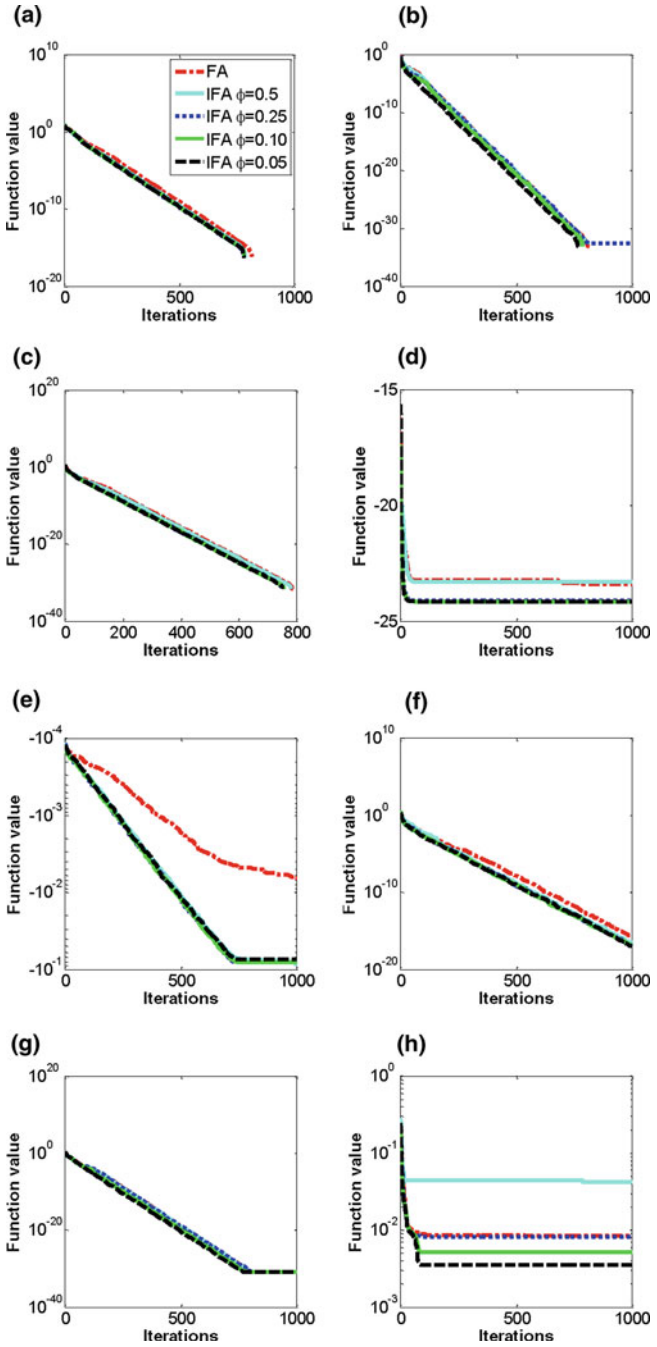

Fig. 3 Evolution of mean best values for IFA (with $\phi = 0.05, 0.1, 0.25, 0.5$) and the original FA algorithm ($\phi = 1$) for: **a** Ackley, **b** Beale, **c** Booth, **d** Carrom table, **e** Cross-leg table, **f** Himmelblau, **g** Levy 13 and **h** Schaffer functions

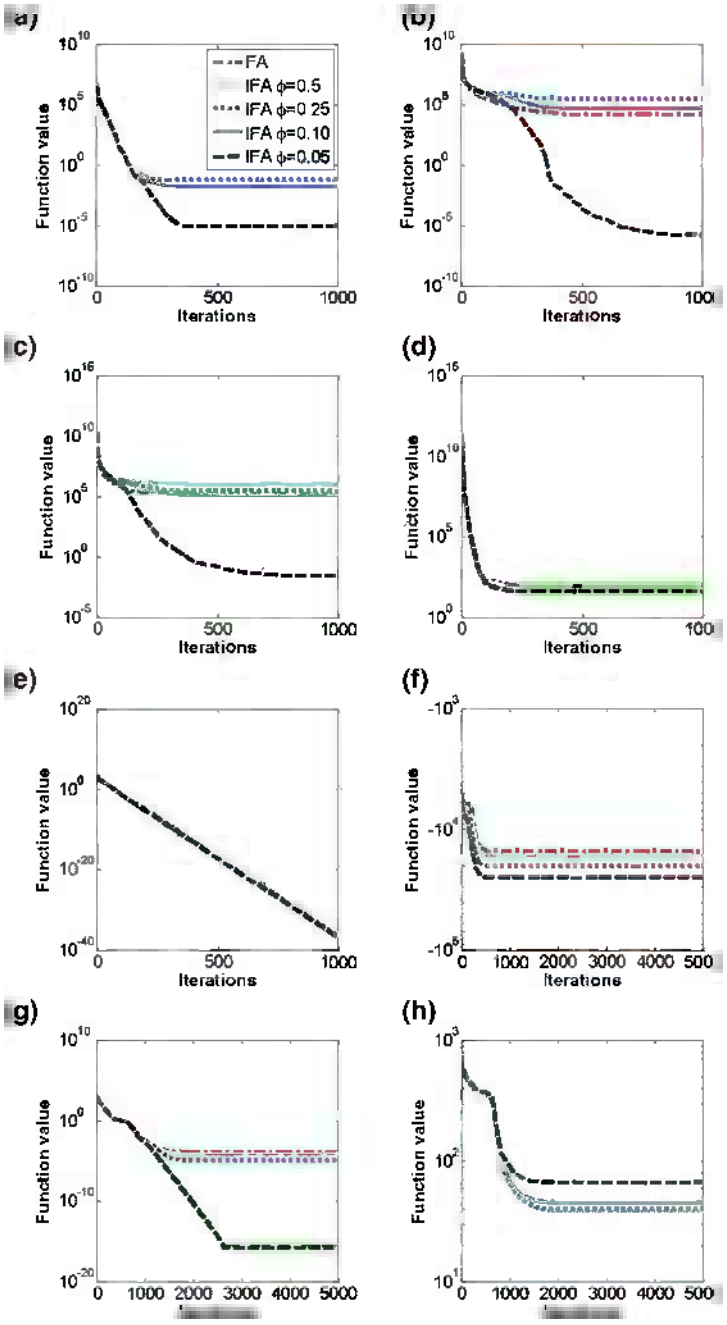

Fig. 4 Evolution of mean best values for IFA (with $\phi = 0.05, 0.1, 0.25, 0.5$) and the original FA algorithm ($\phi = 1$) for: **a** Helical valley, **b** Powell, **c** Wood, **d** Cube, **e** Sphere, **f** Egg holder, **g** Griewank and **h** Rastrigin functions

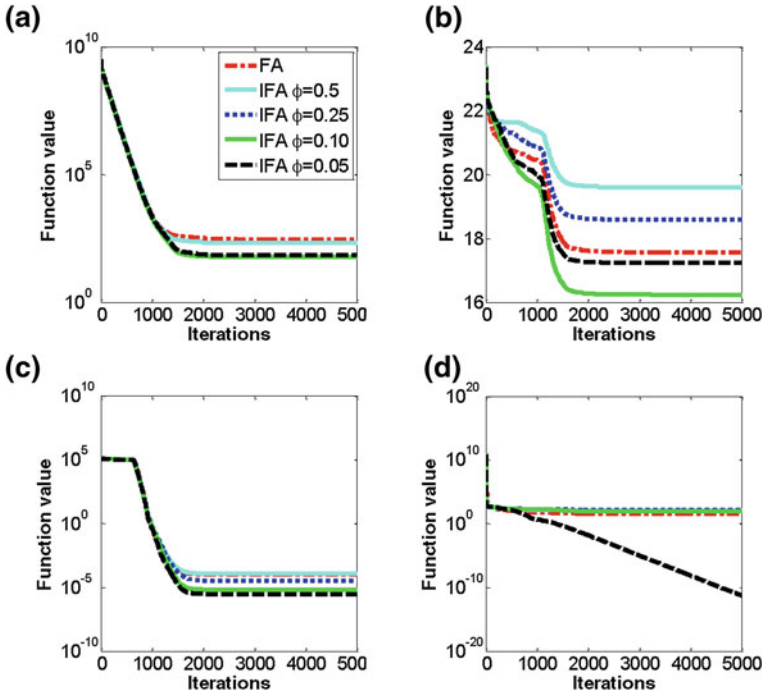

Fig. 5 Evolution of mean best values for IFA (with $\phi = 0.05, 0.1, 0.25, 0.5$) and the original FA algorithm ($\phi = 1$) for: **a** Rosenbrock, **b** Sine envelope sine wave, **c** Trigonometric and **d** Zacharov functions

effectively than FA. The effect of the value of ϕ was minor. Both algorithms were also able to identify the minimum of the Levy 13 function Fig. 3g. However, IFA was significantly more effective when the ϕ parameter values of 0.1 and 0.05 were used.

The Schaffer function is multimodal. Both FA and IFA failed to converge to the global minimum within the used tolerance. However, IFA was able to arrive at a better solution when the ϕ parameter values of 0.1 and 0.05 were used, as shown in Fig. 3h. IFA with $\phi = 2.5$ and 0.5 failed to improve the solution in comparison with that obtained by FA. Schaffer function concludes the 2-variable functions. IFA, with $\phi = 0.05$ and 0.1, performed better than FA in all of them.

The helical valley function has three variables. The evolution of the mean best values of FA, shown in Fig. 4a, showed that performance of IFA with $\phi = 0.05$ was considerably better than the performance of FA. No improvement were observed, when $\phi = 0.5$, 0.25 or 0.1. In addition, all methods failed to obtain the global minimum for the used tolerance. For the Powell function, which has four variables, IFA with $\phi = 0.05$ obtained ten orders of magnitude improvement in the solution, as shown in Fig. 4b. Although the solution was not obtained within the acceptable tolerance but it was very close. IFA with high values of the ϕ parameter were also

unable to identify the global minimum and converged to a higher solution. Those three algorithms seem to have been trapped in a local minimum.

A similar pattern was observed with the Wood function, which has four variables as well. The global minimum was not obtained within the acceptable tolerance used in this study. However, IFA with $\phi = 0.05$ obtained a six orders of magnitude improvement in the solution, as shown in Fig. 4c. Higher values of the ϕ-parameter failed to improve the performance of the FA algorithm. In fact, IFA with $\phi = 0.5$ and 0.25 obtained worse results. For the cube function, which has five variables, Fig. 4d shows that IFA with $\phi = 0.05$ improved the solution slightly. However, none of the methods was able to identify successfully the global minimum for this function. The sphere function also has five variables but it is easier to solve for its global optimum than the cube function. All methods successfully identified the global optimum with slight improvement in performance when IFA was used as shown in Fig. 4e.

The egg holder function was used with 50 variables and the stochastic methods were run up to 5,000 iterations. IFA obtained solutions lower than that of FA, regardless of the value of the parameter ϕ, as shown in Fig. 4f. IFA with $\phi = 0.1$ and 0.05 obtained the best result. Griewank function has 50 variables also. IFA with

Table 3 Values of the mean minima (f_{calc}) and standard deviations (σ) obtained by the FA and IFA algorithms with different values of the ϕ—parameter for the benchmark problems used in this study

Objective function	Numerical performance of					
	FA		IFA ($\phi = 0.5$)		IFA ($\phi = 0.05$)	
	f_{calc}	σ	f_{calc}	σ	f_{calc}	σ
Ackley	0	0	0	0	0	0
Beale	0	0	0	0	0	0
Booth	0	0	0	0	0	0
Carrom table	−23.43	1.75	−23.31	2.52	**−24.16**	**0**
Cross-leg table	−6.6E−3	1.0E−2	**−0.0857**	**8.0E − 2**	−7.46E−2	2.6E−2
Himmelblau	0	0	0	0	0	0
Levy 13	0	0	0	0	0	0
Schaffer	8.5E−3	3.4E−3	0.0427	2.6E−2	**0.0035**	**3.4E − 3**
Helical valley	2.6E−2	5.4E−2	0.0389	8.5E−2	**9.8E − 6**	**2.5E − 5**
Powell	1.84E4	2.7E4	1.54E5	2.5E5	**1.7E − 6**	**2.9E − 6**
Wood	1.78E5	2.9E5	1.06E6	1.9E6	**2.9E − 2**	**0.13**
Cube	88.69	77.05	90.03	92.97	**41.39**	**65.09**
Sphere	0	0	0	0	0	0
Egg holder	−1.52E4	1.6E3	−1.64E4	1.2E3	**−2.5E4**	**2.1E3**
Griewank	1.62E−4	6.1E−5	8.54E−5	5.1E−5	**0**	**0**
Rastrigin	45.47	12.7	**38.14**	**7.92**	66.63	16.44
Rosenbrock	296.8	652.3	213.1	572.9	**70.57**	**109.6**
Sine envelope sine wave	17.56	1.13	19.61	71.85	**17.25**	**1.21**
Trigonometric	1.05E−4	6.1E−5	1.19E−4	9.1E−5	**2.9E − 6**	**2.9E − 6**
Zacharov	36.68	13.2	106.9	30.42	**0**	**0**

$\phi = 0.1$ and 0.05 successfully identified the global minimum, whereas the other methods failed, as shown in Fig. 4g. Improvement of performance with the IFA algorithm is shown clearly with this benchmark problem.

The results of the optimization of Rastrigin function were peculiar. The global optimum was not obtained by FA or IFA and the performance was improved with the use of high values of ϕ (0.5 and 0.25) as shown in Fig. 4h. This is the only function for which the use of IFA with $\phi = 0.05$ did not improve the result.

The global minimum of the Rosenbrock function was also not successfully identified by all methods, as shown in Fig. 5a. IFA, in general, gave better results than FA. The global minimum of the sine-envelope-sine function was also not successfully identified by all methods. This is a multimodal function and it is easy for any optimization method to be trapped in one of the local minima. IFA with $\phi = 0.1$ gave the best solution followed by IFA with $\phi = 0.05$. The results of IFA with higher values of ϕ were worse than the FA result, as shown in Fig. 5b.

The familiar improvement pattern with the use of IFA was obtained with the trigonometric function, as shown in Fig. 5c. The lowest value of ϕ gave the best solution but failed to find the global minimum within the acceptable tolerance used in this study. This was not the case with the Zacharov function (Fig. 5d) since IFA with $\phi = 0.05$ successfully identified the global minimum within the required tolerance with twelve orders of magnitude improvement in the solution. All other methods were trapped in some local minima and no improvement in performance was obtained with IFA with other values of the parameter ϕ.

Table 3 shows a summary of the evaluation results for the twenty benchmark problems. IFA was able to provide better solutions to all challenging problems.

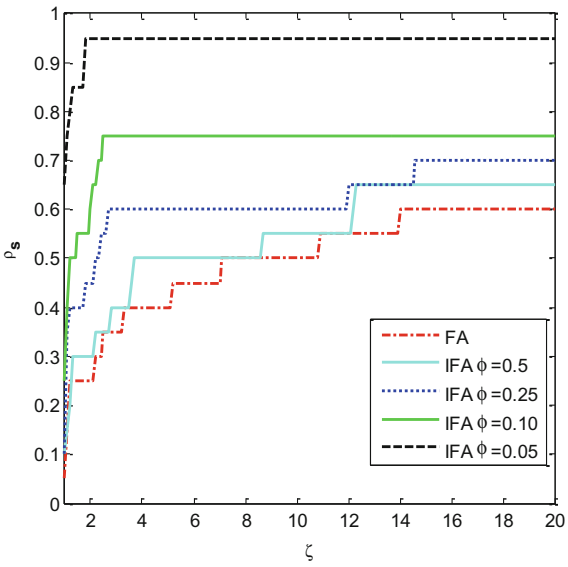

Fig. 6 Performance profiles of the FA and IFA methods for the global optimization of the benchmark problems used in this study

For example, IFA with $\phi = 0.05$, found the global optimum of the helical valley function within 10^{-4} tolerance, Powell and Trigonometric functions within 10^{-5}, and Zacharov function, while the FA obtained a solution several orders of magnitude higher as shown in Table 3. The performance profiles, shown in Fig. 6, summarize the results of the IFA evaluation with the four different values of the ϕ-parameter in comparison with FA. Figure 6 clearly shows IFA with $\phi = 0.05$ was the best performing algorithm in 18 out of the 20 cases considered. FA has not outperformed IFA in any of the benchmark problems tested. Further, PP indicates that ϕ is an important parameter to improve the performance of IFA. The value of $\phi = 0.05$ appears to be the best value among those tested for reliability as indicated by the cumulative highest performance ratio in Fig. 6.

6 Conclusions

In this chapter, we propose a modification to FA through limiting the number of fireflies affecting the moves to a fraction of top performing fireflies. This modification was evaluated by attempting to find the global optimum of twenty benchmark functions. The newly developed IFA led to improved reliability and effectiveness of the algorithm in all tested benchmark problems. In some cases, the global minimum could not have been obtained via the firefly algorithm, except with this modification. IFA was found to perform best, when the new parameter ϕ was set to 0.05, which was the lowest value evaluated.

References

1. Floudas, C.A., Gounaris, C.E.: A review of recent advances in global optimization. J. Glob. Optim. **45**, 3–38 (2008)
2. Rajabioun, R.: Cuckoo optimization algorithm. Appl. Soft Comput. **11**, 5508–5518 (2011)
3. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Appl. Soft Comp. **8**, 687–697 (2008)
4. Li, G., Niu, P., Xiao, X.: Development and investigation of efficient artificial bee colony algorithm for numerical function optimization. Appl. Soft Comp. **12**, 320–332 (2012)
5. Omkar, S.N., Senthilnath, J., Khandelwal, R., Narayana Naik, G., Gopalakrishnan, S.: Artificial bee colony (ABC) for multi-objective design optimization of composite structures. Appl. Soft Comp. **11**, 489–499 (2011)
6. Marinaki, M., Marinakis, Y., Zopounidis, C.: Honey bees mating optimization algorithm for financial classification problems. Appl. Soft Comp. **10**, 806–812 (2010)
7. Chen, H., Zhu, Y., Hu, K.: Multi-colony bacteria foraging optimization with cell-to-cell communication for RFID network planning. Appl. Soft Comp. **10**, 539–547 (2010)
8. Yang, X.S.: Firefly algorithm. Nature-Inspired Metaheuristic AlgorithmsLuniver Press, UK (2008)
9. Sayadi, M.K., Ramezanian, R., Ghaffari-Nasab, N.: A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. Int. J. Ind. Eng. Comput. **1**, 1–10 (2010)

10. Apostolopoulos, T., Vlachos, A.: Application of the firefly algorithm for solving the economic emissions load dispatch problem. Int. J. Comb. **ID 523806**, 1–23 (2011)
11. B. Rampriya, K. Mahadevan, and S. Kannan, Unit commitment in deregulated power system using Lagrangian firefly algorithm, In: International Conference on Communication Control and Computing Technologies, pp. 389–393, 2010.
12. dos Santos Coelho, L., de Andrade Bernert, D.L., Mariani, V.C.: A chaotic firefly algorithm applied to reliability-redundancy optimization. In: IEEE Congr. Evol. Comput. **2011**, 517–521 (2011)
13. Giannakouris, G., Vassiliadis, V., and Dounias, G. Experimental Study on a Hybrid Nature-Inspired Algorithm for Financial Portfolio Optimizatio. In: Konstantopoulos, S., Perantonis, S., Karkaletsis, V., Spyropoulos, C. and Vouros, G. (eds.) Artificial Intelligence: Theories, Models and Applications, vol. 6040, pp. 101–111. Springer, Berlin / Heidelberg, (2010)
14. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Mixed variable structural optimization using firefly algorithm. Comp. Struct. **89**, 2325–2336 (2011)
15. Yang, X.S., Sadat Hosseini, S.S., Gandomi, A.H.: Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect. Appl. Soft Comp. **12**, 1180–1186 (2012)
16. Yang, X.S.: Review of meta-heuristics and generalised evolutionary walk algorithm. Int. J. Bio-Insp. Comput. **3**, 77–84 (2011)
17. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
18. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. J. Glob. Optim. **31**, 635–672 (2005)

Optimization of Queueing Structures by Firefly Algorithm

Joanna Kwiecień and Bogusław Filipowicz

Abstract In the chapter we describe the application of firefly algorithm in discrete optimization of simple queueing structures such as queueing systems. The optimization of these systems is complicated and there is not any universal method to solve such problem. We briefly cover basic queueing systems. Hence, Markovian systems with exponential service times and a Poisson arrival process with losses, with finite capacity and impatient customers and closed queueing system with finite number of jobs are presented. We consider structural optimization, for example maximization of overall profits and minimizing costs controlled by the number of servers. We show the results of performed experiments.

Keywords Queueing systems · Queueing structures · Structural optimization · Cost optimization · Optimization algorithm · Firefly algorithm

1 Introduction

Many practical problems are so complex therefore to study them it is necessary to use models. Conducting experiments on the model gives us many benefits, as:

- simplified reality, models are easier and less complex,
- less risk—with the help of the model, we can compare different variants of solutions and we can introduce the best solution in practice,
- developed mathematical models.

Queueing theory is one of the branches of operational research, which enables the modeling of many real situations. Originally it was closely associated with the technique, but due to its efficiency and flexibility, it has been used in modeling and

J. Kwiecień (✉) · B. Filipowicz
AGH University of Science and Technology, 30 Mickiewicza Ave., 30-059 Krakow, Poland
e-mail: kwiecien@agh.edu.pl

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_16,

performance analysis of computer science, telecommunication systems, industry, call centers or service systems. Currently, queueing theory which includes the queueing systems and networks arouses interest among mathematicians, engineers and economists. To develop such methods, which allow for complete characterization of the service process, the estimation of the quality of systems' work and allow to optimize the structure, and thereby allow for the efficient organization of service is the main task of queueing theory [2–4]. Many standard books describe the basic queueing model and a variety of applications [1–3, 5, 6]. Kleinrock in [7, 8] gives the key foundations of queueing theory. Gautam in [5] shows a broad range of applications including e.g. healthare systems, information systems, transportation or restaurants.

When we consider the optimization problem of queuing structures, we usually mean the selection of appropriate values in the service and selection of the appropriate stream of arriving requests so that the total cost of the system and the waiting time of requests in the queue were minimal. So-called congestion level is often defined as the average waiting time in the queue or the average number of requests in the queue. There are several ways to reduce the "congestion". We can improve system performance by increasing the number of service channels or by increasing the speed of service. Sometimes, the procedure of increasing the number of service channels is hard to perform and requires major investment.

Many optimization problems of queueing theory require efficient algorithms, which can easily be adapted to existing restrictions. Researchers are searching for techniques which give a solution within a reasonable time. One group consists of metaheuristics inspired by nature, which are increasingly used to solve many optimization problems. Firefly Algorithm (FA) belongs to this type of methods. Recently, in [9] we presented the application of firefly algorithm in optimization problem of two queueing systems.

Cost analysis is an important issue in queueing system. Mishra and Yadaw in [12] discussed profit analysis of a loss queueing system and its optimization problem with respect to service rate. We should mention that the geometric programming plays an important role in optimization of several queuing systems [14]. Lin and Ke in [10] describe the use of the genetic algorithms to optimization problem of sample queuing system. Stidham in [13] presents how to set selected parameters of queueing systems and networks, and how control flow of arriving entities to achieve the required purpose.

To present the review of the most popular queueing systems and their optimization problems solved by firefly algorithm is the main aim of the chapter. The chapter is organized as follows. In Sects. 2 and 3 of this chapter, discussions on selected queueing systems (system with losses, system with finite capacity and impatient customers, closed queueing system with finite number of jobs) and related optimization issues of these models are presented. Section 4 describes how firefly algorithm can be applied to optimization problems. Finally, Sect. 5 is devoted to present the selected results for different types of queueing systems.

2 Queueing Systems

A queuing system consists of input, queue and service centres what often means the output of entities. The service centres consist of one or more channels (servers) for serving entities arriving in a certain manner and having some service requirements. If a service channel is available, new entity will seize it for some period. If a service channel is busy, the arriving entity takes specific action: waiting or leaving. The entities represent something that requires maintenance, for example users, customers, jobs or transactions. They can wait for service if there is a waiting room and leave the system after being served. In many cases entities are lost. The performance characteristics of queueing systems include the mean number of entities in a system/a queue, their mean waiting time or the mean response time [2–4, 6].

To describe queueing systems, we use Kendall's notation. Thus, the service system is described by shorthand notation *X/Y/m/D/L*. Systems are described by distribution of inter-arrival times (X), distribution of service times (Y), the number of servers (m), the service discipline (D) and the maximum total numbers of entities called capacity (L). Here, we consider only Markovian systems with exponential service times, in which the arrival process is a Poisson. Commonly used symbol for the first two positions (for X and Y) in the notation is M that denotes exponential distribution of the service time required by each entity and Poisson arrival process. The fourth position in this notation is used for the order in which the entities are served in the queue. Here, we analyze the models with m parallel identical servers and FIFO queueing discipline (first in-first out). In all equations traffic intensity ρ is the ratio of arrival λ to service rate μ. If the stationary probabilities are known, all performance characteristics can be derived, for example the mean number of jobs in a system and in a queue. All probabilities can be derived according to the Chapman-Kolmogorov equations that are very important in the queueing theory. On example of M/M/m/−/m system we present the rule of obtaining these equations and the stationary probabilities. The method of parameters obtaining is precisely described in [3].

2.1 M/M/m/−/m System with Losses

In the system we consider that arriving customer is served if at least one service channel is available. If all servers are busy, the newly arriving customer departs the queueing systems without service and it is lost [2–4, 9]. Let p_i means the probability of the number of entities in the system and it denotes that the system is in state E_i. Note that the system can be in one of the following characteristc states:

- E_0—all service channels are available,
- E_i—i channels are occupied,
- E_m—m channels are busy, the next entity is not accepted—the locking system.

The set of equations obtained from the so-called mnemonic rule represents the dynamics of the system. Using the rule, we can write the differential equation for

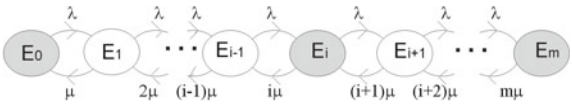

Fig. 1 The state-transition-rate diagram of M/M/m/−/m queueing system

the state probabilities based on state-transition-rate diagram. In such diagram, each state is represented by an oval and each nonzero density rate is represented by a directed branch pointing from one state to another. The branches represent the permitted transitions. We assume that the system begins with zero entities at time 0. The state-transition-rate diagram is shown in Fig. 1 [3, 7]. Characteristic states are marked on the graph as shaded ovals.

On the left side of each equation, we write the derivative of the state probability. On the right side there are so many factors as the graph edges are associated with a particular state. If the edge comes out the state, its corresponding factor has a minus sign, otherwise—a plus sign. Each factor is equal to the probability density function, which identifies a particular edge, multiplied by the probability of the state from which the edge comes out [3].

Consequently, we have the resulting set of equations:

$$\begin{aligned} &p_0'(t) = -\lambda p_0(t) + \mu p_1(t) \\ &\vdots \\ &p_i'(t) = \lambda p_{i-1}(t) - (\lambda + i\mu) p_i(t) + (i+1)\mu p_{i+1}(t) \\ &\qquad\qquad\qquad\qquad\qquad\qquad for\ 1 \le i \le m-1 \\ &\vdots \\ &p_m'(t) = \lambda p_{m-1}(t) - m\mu p_m(t) \end{aligned} \tag{1}$$

In order to solve the set of equations for the time-dependent behavior $p_k(t)$, we require initial conditions. In most applications, knowledge of the service characteristics in the steady state is sufficient. Here, we assume that all probabilities tend to constant value and their derivatives tend to zero. Carrying out this assumption our equations convert to [3, 7]:

$$\begin{aligned} &0 = -\lambda p_0 + \mu p_1 \\ &\vdots \\ &0 = \lambda p_{i-1} - (\lambda + i\mu) p_i + (i+1)\mu p_{i+1} \\ &\qquad\qquad\qquad\qquad for\ 1 \le i \le m-1 \\ &\vdots \\ &0 = \lambda p_{m-1} - m\mu p_m \end{aligned} \tag{2}$$

The sum of the steady-state probabilities must be equal to one, so these probabilities also satisfy so-called normalization equation:

$$\sum_{i=0}^{m} p_i = 1. \tag{3}$$

Note that from Eqs. (2) and (3), we obtain the stationary probabilities of the system. The steady-state probability of no jobs in the system is given by:

$$p_0 = \frac{1}{\sum_{k=0}^{m} \frac{\rho^k}{k!}}, \quad \rho = \frac{\lambda}{\mu}. \tag{4}$$

The stationary probability of k jobs in the system can be expressed by the following equation:

$$p_k = \frac{\rho^k}{k!} \cdot p_0. \tag{5}$$

The most important quantity out of the values obtained by Eqs. (2) and (3) is the probability that the newly arriving customers are lost, because all servers are busy:

$$p_{lost} = \frac{\rho^m}{m!} \cdot p_0 = p_m. \tag{6}$$

The average number of jobs in the system is one of the parameters characterizing the queuing system, which is defined as:

$$\overline{K} = \sum_{k=0}^{m} k p_k = \rho(1 - p_{lost}), \tag{7}$$

with p_{lost} from Eq. (6).

2.2 *M/M/m/FIFO/m+N System with Finite Capacity and Impatient Customers*

In this system we have situations that the customers should be served before their respective deadlines. We assume the maximum number of customers is $m+N$ and waiting room is limited to N. Each customer arriving to the system has a maximum waiting time in queue, which is distributed exponentially with parameter δ. If the waiting time for access to the server exceeds this maximum waiting time, the customer leaves the system without receiving service. An arriving entity is forced to join the queue if there are less than $m+N$ customers in the system. Otherwise, it is lost [9]. The state-transition-rate diagram is shown in Fig. 2.

In the stationary case, we obtain all interested parameters from the following set of equations:

Fig. 2 The state-transition-rate diagram of M/M/m/FIFO/m+N system with impatient jobs

$$
\begin{aligned}
&0 = -\lambda p_0 + \mu p_1 \\
&\vdots \\
&0 = \lambda p_{i-1} - (\lambda + i\mu) p_i + (i+1)\mu p_{i+1}, \quad for\ 1 \le i \le m-1 \\
&\vdots \\
&0 = \lambda p_{m-1} - (\lambda + m\mu) p_m + (m\mu + \delta) p_{m+1} \\
&\vdots \\
&0 = \lambda p_{m+k-1} - (\lambda + m\mu + k\delta) p_{m+k} + [m\mu + (k+1)\delta] p_{m+k+1}, \\
&\qquad\qquad for\ 1 \le k \le N-1 \\
&\vdots \\
&0 = \lambda p_{m+N-1} - (m\mu + N\delta) p_{m+N} \\
&\textstyle\sum_{j=0}^{m+N} p_j = 1
\end{aligned}
\tag{8}
$$

The stationary probability in which there are no entries can be derived from:

$$
p_0 = \left[\sum_{k=0}^{m} \frac{\rho^k}{k!} + \frac{\rho^m}{m!} \sum_{r=1}^{m+N} \frac{\rho^r}{\prod_{n=1}^{r} \left(m + n\frac{\delta}{\mu} \right)} \right]^{-1}. \tag{9}
$$

Using Eq. (9), we solve the steady-state probability that jobs are lost because of exceeding the time limit as follows:

$$
p_{\mathrm{w}} = \left[\frac{\delta}{\lambda} \cdot \frac{\rho^m}{m!} \sum_{r=1}^{m+N} \frac{r\rho^r}{\prod_{n=1}^{r} \left(m + n\frac{\delta}{\mu} \right)} \right] \cdot p_0. \tag{10}
$$

The stationary probability that the arriving jobs are lost due to occupy all service channels and places in queue is given by:

$$
p_{m+N} = \frac{\rho^{m+N}}{m! \prod_{n=1}^{N} \left(m + n\frac{\delta}{\mu} \right)} \cdot p_0. \tag{11}
$$

We may easily calculate the probability that jobs will be lost in this queueing system from:

$$
p_{lost} = p_{\mathrm{w}} \cdot p_{m+N}. \tag{12}
$$

In practice we find these systems, where the entities are waiting in a queue for certain time and the capacity of a system is limited. For example such systems can be used to model the existing structures in the health service when the patient waiting time for a visit is limited. In the case of call centers, customer hangs up when he has to wait too long because an operator is busy.

2.3 The M/M/m/FIFO/N/F Closed Queueing System with Finite Population of N Jobs

In the system we assume a finite number of identical machines that generate population of N customers. Therefore, the total number of jobs in the whole system is no more than N. The system has m identical service channels. If the total number of customers does not exceed the number of servers, all jobs are served immediately, without waiting in a queue [3, 4, 7]. Figure 3 shows its state-transition-rate diagram.

The steady-state probabilities we can calculate from the following set of equations:

$$\begin{aligned}
&0 = -N\lambda p_0 + \mu p_1 \\
&\vdots \\
&0 = (N-i+1)\lambda p_{i-1} - [(N-i)\lambda + i\mu]p_i + (i+1)\mu p_{i+1} \\
&\qquad\qquad for\ 1 \le i \le m-1 \\
&\vdots \\
&0 = (N-m+1)\lambda p_{m-1} - [(N-m)\lambda + m\mu]p_m + m\mu p_{m+1} \\
&\vdots \\
&0 = (N-j+1)\lambda p_{j-1} - [(N-j)\lambda + m\mu]p_j + m\mu p_{j+1} \\
&\qquad\qquad for\ m \le j \le N-1 \\
&\vdots \\
&0 = \lambda p_{N-1} - m\mu p_N \\
&\textstyle\sum_{k=0}^{N} p_k = 1
\end{aligned} \tag{13}$$

The steady-state probability in which there are no jobs in system is an important quantity. It follows from Eq. (13), yielding:

$$p_0 = \left[\sum_{i=0}^{m} \frac{N!}{i!(N-i)!} \rho^i + \sum_{j=m+1}^{N} \frac{N!}{m!(N-j)!m^{j-m}} \rho^j \right]^{-1}. \tag{14}$$

Fig. 3 Diagram of transition for closed queueing system

The steady-state probability of k jobs in the system is given by:

$$\begin{aligned} p_k &= \frac{N!}{k!(N-k)!}\rho^k \cdot p_0, \quad 1 \le k \le m \\ p_k &= \frac{N!}{m!(N-k)!m^{k-m}}\rho^k \cdot p_0, \quad m < k \le N \end{aligned} . \tag{15}$$

Using Eq. (14), we calculate the mean number of jobs in the whole system as follows:

$$\overline{K} = p_0 N! \left[\sum_{i=0}^{m} \frac{i}{i!(N-i)!}\rho^i + \sum_{j=m+1}^{N} \frac{j}{m!m^{j-m}(N-j)!}\rho^j \right]. \tag{16}$$

Obviously for any real case, depending on the modeled situation, various machines, operators can be modeled as a source that requests a service (e.g. a fault). The rapid development of technology and the automation of many processes caused the machine/computer is increasingly carrying out the works that were reserved for a man not so long ago. Thus, the number of different devices that require maintenance or repair increases. The main task of the people who are responsible for technical service among others is to optimize the life of equipment, and in case of failures its rapid identification and removal. To choose an appropriate service technicians able to ensure the reliability and quality of the equipment is one of the important problems of managers.

3 Optimization Problems

Queueing models provide a powerful tool, which allows us to make the optimal decisions about the structure of the system, in order to improve the organization of the service. Funding for the maintenance of an adequate number of employees is an important component of the costs incurred by the company. Therefore, they have to be optimized, which will ensure the proper handling of equipment operated.

Optimization of queuing systems described above is a difficult problem. There are no universal methods of solving it. In order to formulate the optimization problem we need to specify the parameters that will be the decision variables. There are three basic types of optimization [2]:

- minimization of costs for a given stream of incoming requests by changing the parameters of the service time,
- maximization of stream of incoming requests with limited cost by seeking service ratio,
- finding the optimal topology (we are looking for the number of service channels, for which labor costs are lowest).

Here, we consider the structural problem of optimization, which is to find the correct number of servers, what optimizes costs [3, 9].

In the case of M/M/m/−/m queueing system with losses we wish to maximize the overall profits. The number of channels that maximizes the objective function is to be calculated. Let us assume that r denotes the cost associated with the entity in the system and c denotes the cost of channel depreciation. Therefore, the objective function has the following form:

$$f(m) = r\rho\left(1 - \frac{\rho^m}{m!\sum_{k=0}^{m}\frac{\rho^k}{k!}}\right) - cm, \quad \rho = \frac{\lambda}{\mu}. \tag{17}$$

For considered M/M/m/FIFO/m+N system with impatient customers we have to maximize all profits subject to the number of channels m and the number of waiting places N. Let us denote the profit on customer service by r_1 and the cost of server depreciation by r_2 . Thus, the objective function is calculated with the help of Eq. (12) from the formula:

$$f(m, N) = r_1\lambda(1 - p_{lost}) - r_2(m + N). \tag{18}$$

In the closed queueing systems M/M/m/FIFO/N/F we seek the number of servers that minimize the overall costs of its operation. Taking into account the cost of server maintenance (c_1) and the cost of existing jobs in the systems (c_2), the objective function can be determined with the help of Eq. (16):

$$f(m) = c_1 m + c_2\overline{K}. \tag{19}$$

It is clear that the optimization of these objective functions is complicated. Therefore, we have to use one of the optimization algorithms.

4 Firefly Algorithm

The firefly algorithm is a metaheuristic algorithm developed by Yang [15, 16]. Its idea is based on the behavior of fireflies. In the algorithm we use the difference in light intensity, that is proportional to the value of the objective function. Each individual has a certain attractiveness, which determines the direction of movement. All fireflies are characterized by light intensity associated with the objective function. Yang in [15] describes the idea and three rules of firefly algorithm. One rule is that a firefly follows the brighter one. If there is no brighter firefly, firefly will move randomly. In problems described in this chapter, the firefly algorithm is executed as follows [11, 15, 16]:

- Step 1: Initialize algorithm's parameters and generate population of fireflies: n—number of fireflies, the maximum attractiveness β_0, the light absorption coefficient

γ, randomization parameter α, maximum number of generations *MaxGen*; the objective function $f(x)$ is given by one of Eqs. (17)–(19). Assuming the defined maximum and minimum value of x_i, the initial populations of individuals are initialized randomly over the entire search space as follows:

$$x_i = x_i^{\min} + rand \cdot (x_i^{\max} - x_i^{\min}). \tag{20}$$

- Step 2: Compute light intensity for each firefly, which is determined by the value of objective function $f(x_i)$. For a maximization problem, the brightness is proportional to the value of the objective function.
- Step 3: For each generation (1,..., *MaxGen*):
  – Find attractive individual. If it exists then move firefly i towards a brighter firefly j according to:

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha \cdot (rand - 0.5), \tag{21}$$

  where distance r_{ij} between two fireflies is calculated using the Euclidian distance. Otherwise, randomly movement.
  – Obtain attractiveness, which varies with distance r according to:

$$\beta(r) = \beta_0 e^{-\gamma r^2}. \tag{22}$$

  – Select the brightest firefly as the potential optimum.
- Step 4: Terminate if a predefined stopping criterion is met, otherwise go back to Step 3.

Detailed descriptions of the firefly algorithm can be found in [11, 15, 16]. The optimization problems presented above are discrete. Therefore, in all experiments, the positions of fireflies have been rounded to the nearest integer value using $round(x_i)$. The best firefly gives the value of decision variable that is optimal in considered case. We have implemented all calculations in Matlab, with help of files presented in [17].

5 Results of Experiments

We conducted a number of computational experiments on prepared test instances. For each queueing system presented in this chapter, we performed several experiments, depending on the size of fireflies population and number of iterations. In most problems, the population of 20 individuals was sufficient, except for the queueing system with impatient customers. Firefly algorithm has found the optimal value of the objective function within 20 iterations. For each case, we have performed 10 calculations. In all presented examples the values of algorithm parameters are: $\alpha = 0, 1$, $\beta_0 = 0, 1$, $\gamma = 1$. We have tried to check the influence of these parameters on solution. For example, reducing α to 0,01 gives worse results. For increased ab-

Table 1 The results of FA optimization for $m \in [1, 50]$

M/M/m/—/m, $r = 4$, $c = 1$, decision variable: m				
Case	λ	μ	m =?	value of $f(m)$
A	1	1	2	1.20
B	10	5	3	3.316
C	10	10	2	1.20
D	10	15	1	0.60
E	25	5	8	10.599
F	50	5	14	23.727

sorption coefficient ($\gamma = 10$) we have the solution a few iterations later. For higher value of attractiveness ($\beta_0 = 0, 5$) the results are comparable.

Table 1 shows the best results for M/M/m/−/m system in six cases of several settings of system parameters and values of the objective function obtained by the the firefly algorithm. For example, in first case (A) the optimum value of the objective function given by Eq. (17) is 1,20 and the number of service channels is equal to 2. Figures 4, 5 and 6 present the change of the objective function during optimization process.

For the optimization of queueing system with impatient customers, in which the decision variables are the number of servers (m) and the number of waiting places (N), we use 50 fireflies. Using 20 fireflies was not sufficient to obtain optimal value of sought-after parameters. Table 2 shows the best results for four cases of different settings of the service rate μ and the ratio of arrival λ. Last column presents the maximal value of the objective function obtained by FA. Figures 7 and 8 show the relationship of the objective function with iterations in considered cases.

For closed queueing system we use 20 fireflies. In Table 3 and Figs. 9, 10, 11 and 12 we present the best results. Note that for this system we minimize the objective function.

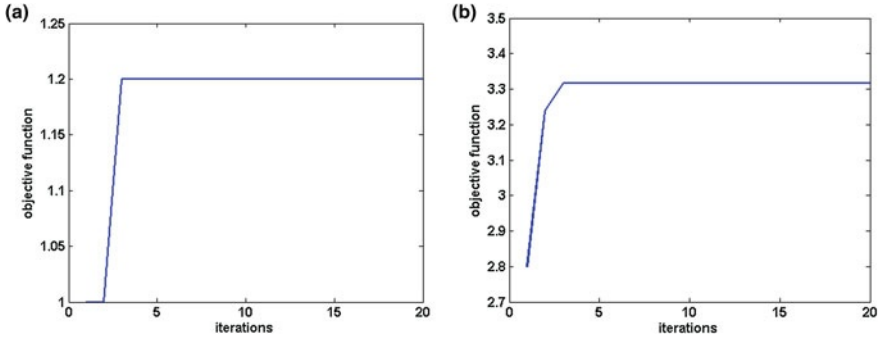

Fig. 4 The change of the objective function during optimization process; **a** case A, **b** case B

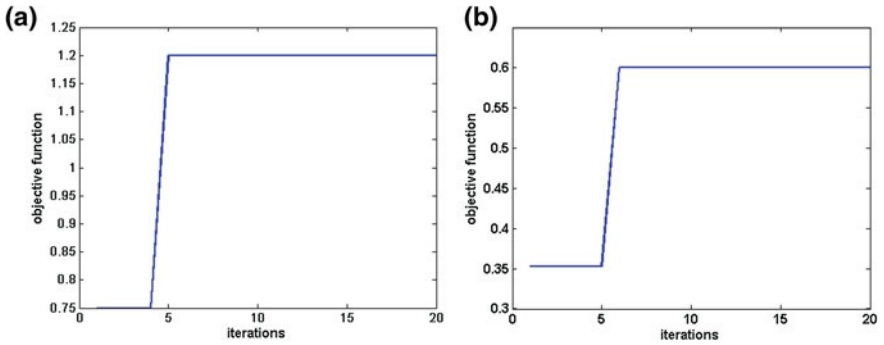

Fig. 5 The change of the objective function during optimization process; **a** case C, **b** case D

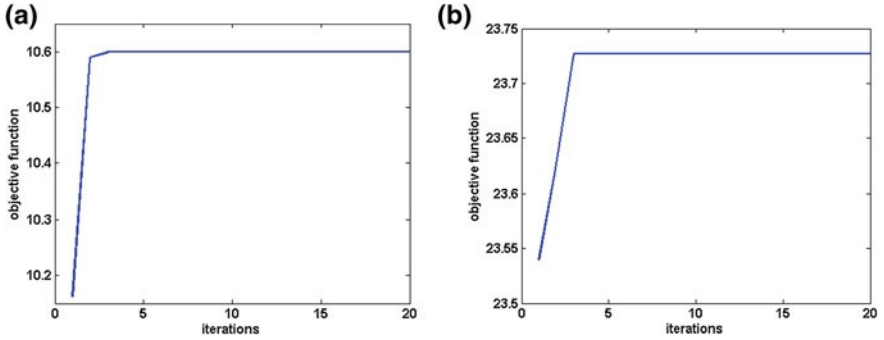

Fig. 6 The change of the objective function during optimization process; **a** case E, **b** case F

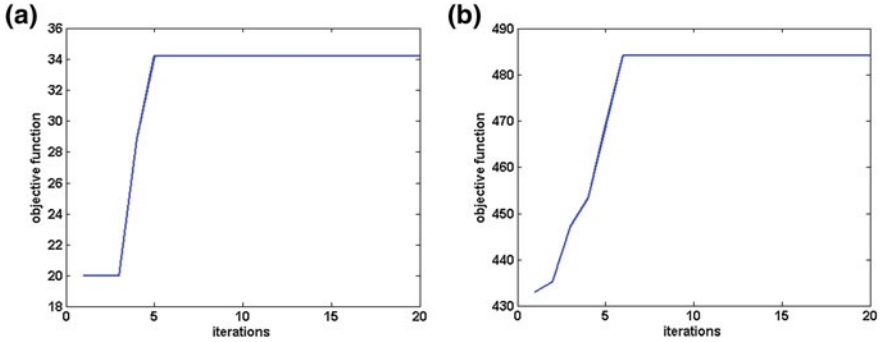

Fig. 7 The change of the objective function during optimization process; **a** case A, **b** case B

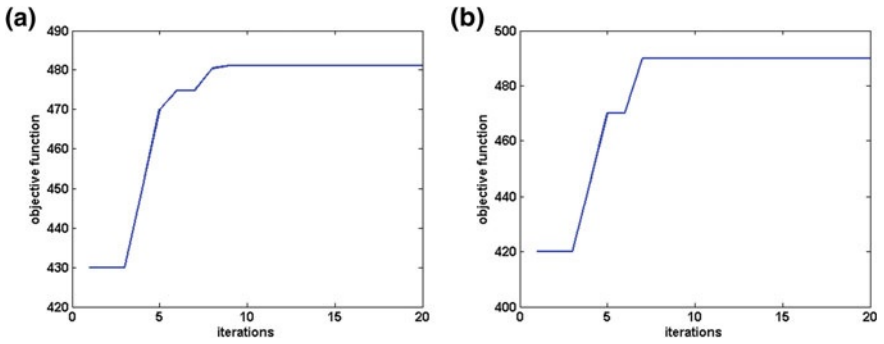

Fig. 8 The change of the objective function during optimization process; **a** case C, **b** case D

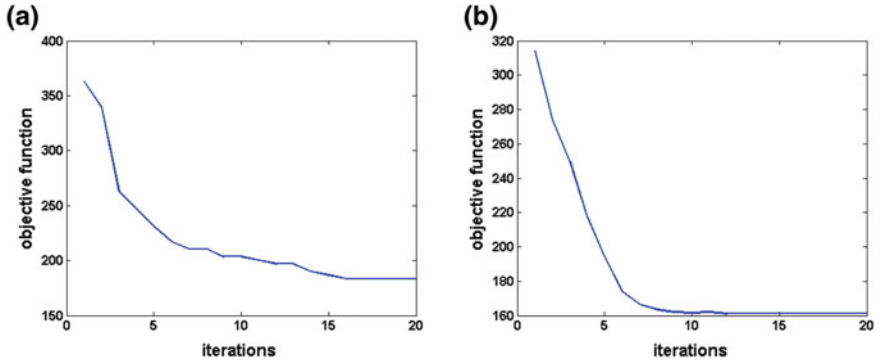

Fig. 9 The change of the objective function during optimization process; **a** case A, **b** case B

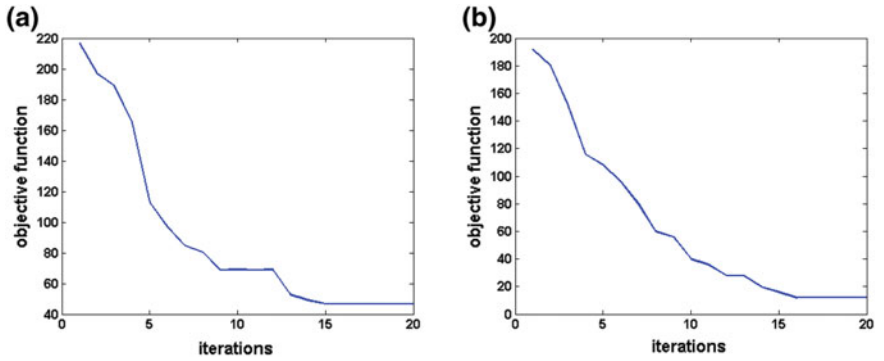

Fig. 10 The change of the objective function during optimization process; **a** case C, **b** case D

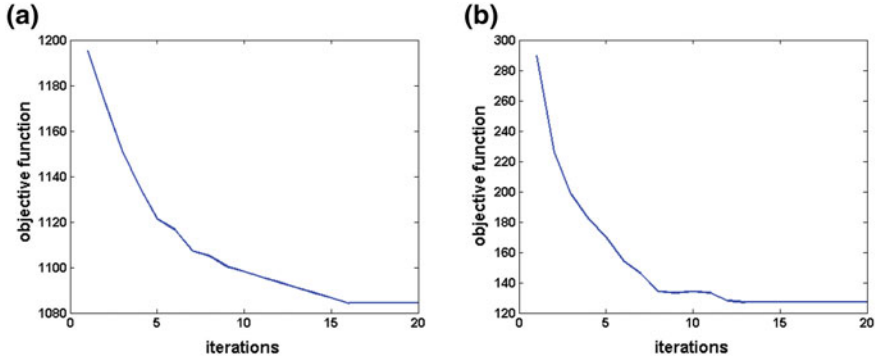

Fig. 11 The change of the objective function during optimization process; **a** case E, **b** case F

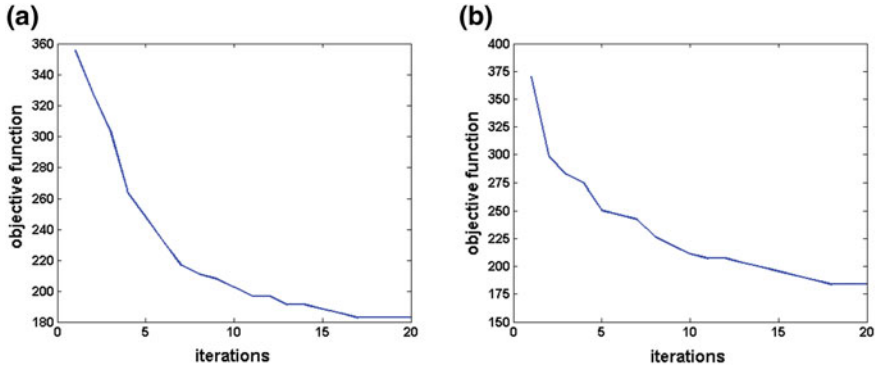

Fig. 12 The change of the objective function during optimization process; **a** case G, **b** case H

Table 2 The results of FA optimization, for $m \in [1, 50]$ and $N \in [1, 50]$

M/M/m/FIFO/m+N, $r_1 = r_2 = \delta = 5$, decision variable: m, N					
Case	λ	μ	$m = ?$	$N = ?$	value of $f(m, N)$
A	10	5	1	1	34.18
B	100	5	1	1	484.07
C	100	50	1	1	481.11
D	100	500	1	1	489.96

Table 3 The results of FA optimization for $m \in [1, 50]$

M/M/m/FIFO/N/F , $c_1 = 4$, $c_2 = 12$, decision variable: m					
Case	λ	μ	N	$m =?$	value of $f(m)$
A	19	1	15	1	183.37
B	19	10	15	9	161
C	19	100	15	4	46.76
D	19	10	1	1	11.86
E	19	10	100	50	1084.21
F	1	1	15	8	126.93
G	10	1	15	1	182.80
H	100	1	15	1	183.88

6 Conclusion

The computational results show that the firefly algorithm is a very powerful tool, allowing to solve the optimization of queueing systems. The algorithm is simple to implement, but its parameters may depend on the type and size of the optimized problems. In our experiments the decision variable was the number of servers or the number of servers and the queue capacity. The other values were assumed to be fixed parameters. It should be noted that this algorithm can be used to determine other characteristics, including the determination of the optimal policy of queue liquidation. In some real situations there may be more than one objective function, especially in the use of more complex structures such as tandem model or queueing networks. The analysis and application of queueing systems, tandem models and queueuing networks with losses is the topic of many research in computer science, traffic or industrial engineering.

References

1. Bhat, U. N.: An Introduction to Queueing Theory. In: Modeling and Analysis in Applications. Birkhauser, Verlag, Boston (2008)
2. Bolch, G. et al.: Queueing networks and markov chains. In: Modeling and Performance Evaluation with Computer Science Applications. Wiley, New York (1998)
3. Filipowicz, B.: Stochastic Models in Operation Research: Analysis and Synthesis of Queueing Systems and Networks. Warsaw, WNT (1996). (in Polish)
4. Filipowicz, B., Kwiecień, J.: Queueing systems and networks: models and applications. Bulletin of the Polish Academy of Sciences. Tech. Sci. **56**, 379–390 (2008)
5. Gautam, N.: Analysis of Queues: Methods and Applications. CRC Press, Taylor and Francis Group, Boca Raton (2012)
6. Gross, D. et al.: Fundamentals of Queueing Theory, 4th edn. Wiley, New York (2008)
7. Kleinrock, L.: Queueing Systems, vol. I: Theory. Wiley, New York (1975)
8. Kleinrock, L.: Queueing Systems, vol. II: Computer Applications. Wiley, New York (1976)
9. Kwiecień, J., Filipowicz, B.: Firefly algorithm in optimization of queueing systems. Bulletin of the Polish Academy of Sciences. Tech. Scie. **60**, 363–368 (2012)

10. Lin, ChH, Ke, J.Ch.: Optimization analysis for an infinite capacity queueing system with multiple queue-dependent servers: genetic algorithms. Int. J. Comp. Math. **88**(7), 1430–1442 (2011)
11. Łukasik, S., Żak, S.: Firefly algorithm for continuous constrained optimization task, computational collective intelligence. Semantic web, social networks and multiagent systems. LNCS **5796**, 97–106 (2009)
12. Mishra, S.S., Yadav, D.K.: Computational approach to profit optimization of a loss-queueing system. J. Appl. Comput. Sci. Math. **9**, 78–82 (2010)
13. Stidham, S. Jr.: Optimal Design of Queueing Systems. CRC Press, Taylor and Francis Group, Boca Raton (2009)
14. Verma, R.K.: Multiobjective optimization of a queueing system. J. Math. Program. Oper. Res. **17**, 103–111 (1986)
15. Yang, X.S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press, Frome (2008)
16. Yang, X.S.: Firefly algorithms for multimodal optimization. Stochastic algorithms: foundations and applications. SAGA, LNCS **5792**, 169–178 (2009)
17. Yang, X.S.: Firefly Algorithm—Matlab files, available: http://www.mathworks.com/matlabcentral/fileexchange/29693-firefly-algorithm. Accessed 25 Feb 2012

Firefly Algorithm: A Brief Review of the Expanding Literature

Iztok Fister, Xin-She Yang, Dušan Fister and Iztok Fister Jr.

Abstract Firefly algorithm (FA) was developed by Xin-She Yang in 2008 and it has become an important tool for solving the hardest optimization problems in almost all areas of optimization as well as engineering practice. The literature has expanded significantly in the last few years. Various FA variants have been developed to suit different applications. This chapter provides a brief review of this expanding and state-of-the-art literature on this dynamic and rapidly evolving domain of swarm intelligence.

Keywords Firefly algorithm · Discrete firefly algorithm · Nature-inspired algorithm · Scheduling · Combinatorial optimization · Engineering optimization

1 Introduction

Among swarm-intelligence-based algorithms, firefly algorithm (FA) is now one of the most widely used. Firefly algorithm was developed by Xin-She Yang in 2008 [1], based on inspiration from the natural behavior of tropical fireflies. FA tries to mimic the flashing pattern and attraction behaviour of fireflies. The purpose of these flash-

I. Fister (✉) · D. Fister · I. Fister Jr.
Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia
e-mail: iztok.fister@uni-mb.si

D. Fister
e-mail: dusan.fister@uni-mb.si

I. Fister Jr.
e-mail: iztok.fister2@uni-mb.si

X.-S. Yang
School of Science and Technology, Middlesex University, North London, UK
e-mail: x.yang@mdx.ac.uk

X.-S. Yang (ed.), *Cuckoo Search and Firefly Algorithm*,
Studies in Computational Intelligence 516, DOI: 10.1007/978-3-319-02141-6_17,

ing lights are twofold: to attract mating partners and to warn potential predators. Obviously, these flashing light and its intensity can obey some rules, including physical laws. In essence, FA uses the following three idealized rules [1]:

- Fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex.
- The attractiveness is proportional to the brightness and they both decrease as their distance increases. Thus for any two flashing fireflies, the less brighter one will move towards the brighter one. If there is no brighter one than a particular firefly, it will move randomly.
- The brightness of a firefly is determined by the landscape of the objective function.

As a firefly's attractiveness is proportional to the light intensity seen by adjacent fireflies, we can now define the variation of attractiveness β with the distance r by

$$\beta = \beta_0 e^{-\gamma r^2}, \tag{1}$$

where β_0 is the attractiveness at $r = 0$. The movement of a firefly i is attracted to another more attractive (brighter) firefly j is determined by

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2}(x_j^t - x_i^t) + \alpha\, \varepsilon_i^t, \tag{2}$$

where the second term is due to the attraction. The third term is randomization with α being the randomization parameter, and ε_i^t is a vector of random numbers drawn from a Gaussian distribution at time t. Other studies also use the randomization ε_i^t can easily be extended to other distributions such as Lévy flights. It is worth pointing out that γ controls the scaling, while α controls the randomness. For the algorithm to convergence properly, randomness should be gradually reduced, and one way to achieve this is to use

$$\alpha = \alpha_0 \theta^t, \quad \theta \in (0, 1), \tag{3}$$

where t is the index of iterations/generations. Here α_0 is the initial randomness factor, and we can set $\alpha_0 = O(1)$ without losing generality.

Studies have shown that FA is very efficient [2–5]. Fister et al. provided a comprehensive review of the current literature of the firefly algorithm and its variants [6]. Since then, about 30 more journal papers published in the last a few months alone. In fact, a quick Google scholar search using firefly algorithm as the keyword returned 625 hits at the time of writing this chapter in July 2013. A similar search using Scirus gave 658 hits with 158 peer-reviewed journal papers. Therefore, it seems impossible to review every single piece of research work concerning firefly algorithms, however, it would be useful to summarize the key works/papers that we can get hold of and highlight the main and representative results.

Therefore, the main aim of this chapter is to briefly introduce the readers the state-of-the-art developments so as to provide classifications of variants, research works, and provide a good snapshot of the current literature. The rest of chapter

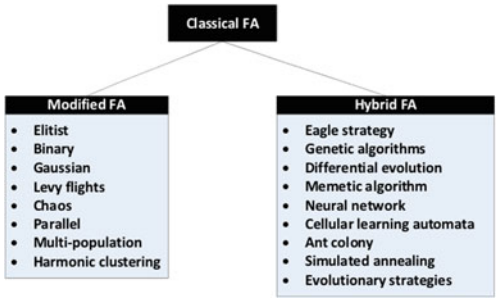

Fig. 1 Variants of the firefly algorithm

is organized as follows. In Sect. 2, a brief review of the modified and hybridized firefly algorithms is presented. Section 3 deals with the application domains where the firefly algorithms were successfully used, while Sect. 4 focuses on the application of the firefly algorithm in engineering optimization. Finally, conclusions are drawn briefly and the directions for future work are discussed in Sect. 5.

2 Classifications of Firefly Algorithms

The standard firefly algorithm has been proved very efficient and it has three key advantages

- Automatic subdivision of the whole population into subgroups so that each subgroup can swarm around a local mode. Among all the local modes, there exists the global optimality. Therefore, FA can deal with multimodal optimization naturally.
- FA has the novel attraction mechanism among its multiple agents, and this attraction can speed up the convergence. The attractiveness term is nonlinear, and thus may be richer in terms of dynamical characteristics.
- FA can include PSO, DE and SA as its special cases as shown in Chap. 1. Therefore, it is no surprise that FA can efficiently deal with a diverse range of optimization problems.

Many researchers use FA to solve a diverse range of problems, and they have also tried to develop various variants to suit for specific types of applications with improved efficiency. Using similar classification as proposed in [6], the variants of the firefly algorithm can be divided into modified and hybridized algorithms (Fig. 1). In total, there are more than 20 different FA variants.

The short review of research papers concerning the classical firefly algorithms can be summarized in Table 1.

Table 1 Classification of the firefly algorithms

Topic	References
The original firefly algorithm	[1]
Multi-modal test functions	[3]
Continuous and combinatorial optimization	[7]
Review of nature-inspired meta-heuristics	[8–10]

2.1 Modified FA

The modified firefly algorithms can be analyzed according to the setting of their algorithm-dependent parameters. In line with this, the firefly algorithms are classified according to the following criteria:

- representation of fireflies (binary, real-valued);
- population scheme (swarm, multi-swarm);
- evaluation of fitness function;
- determination of the best solution (non-elitism, elitism);
- randomization of moving the fireflies (uniform, Gaussian, Lévy flights, chaos distributions).

As a results, the existing studies concerning the modified algorithms can be presented in Table 2.

2.2 Hybrid Firefly Algorithms

The firefly algorithm has been designed as a global problem solver that should obtain the good results on the all classes of optimization problems. However, the No-Free-Launch theorem usually poses some limitations [42]. In order to overcome the limitations imposed by this theorem, hybrid methods tend to be used to develop new variants of nature-inspired algorithms including the variants of firefly algorithms. Various hybridizations have been applied on the firefly algorithm to seek improvements. Hybridization incorporates some problem-specific knowledge to the firefly algorithm. Normally, it was hybridized with other optimization algorithms, machine learning techniques, heuristics, etc. The short review of the major hybrid firefly algorithms is illustrated in Table 3.

3 Applications

Since its first appearance in 2008, in the last few years, the firefly algorithm has been used in almost every area of sciences and engineering, including optimization, classifications, travelling salesman problem, scheduling, image processing, and

Table 2 Modified firefly algorithms

Topic	References
Elitist firefly algorithm	[11]
Binary represented firefly algorithm	[12–16]
Gaussian randomized firefly algorithm	[17, 18]
Lévy flights randomized firefly algorithm	[4, 18, 19]
Chaos randomized firefly algorithm	[20–22]
Parallel firefly algorithm	[23, 24]
Multi-population	[25]
Harmonic clustering	[26, 27]
Quaternion firefly algorithm	[28]

Table 3 Hybrid firefly algorithms

Topic (with which the firefly algorithm hybridizes)	References
Eagle strategy using Lévy walk	[29]
Genetic algorithms	[15, 30]
Differential evolution	[31, 32]
Memetic algorithm	[33, 34]
Neural network	[35–37]
Cellular learning automata	[15, 38]
Ant colony	[39]
Simulated annealing	[40]
Evolutionary strategies	[41]

engineering designs. Some application domains are more theoretical, whilst others have focused on solving the real-world problems. The taxonomy of firefly algorithm applications can be seen in Fig. 2 where we have focused on three major areas of applications: optimization, classification and engineering designs.

3.1 Optimization

The firefly algorithm has been applied into the following classes of problems:

- continuous,
- combinatorial,
- constraint,
- multi-objective,
- multi-modal,
- dynamic and noisy.

Continuous optimization problems often concern a set of real numbers or functions, whilst in the combinatorial optimization problems, solutions are sought from a

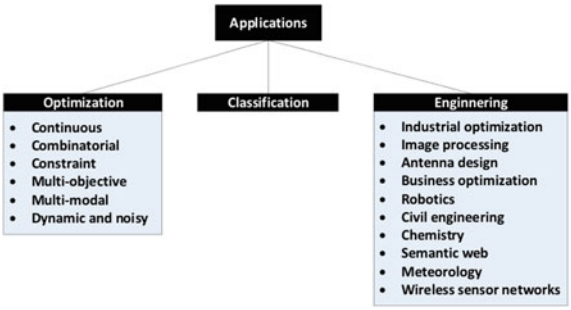

Fig. 2 Taxonomy of firefly algorithm applications

Table 4 Optimization applications

Topic	References
Continuous optimization	[2, 4, 7, 9, 18, 19, 46]
Combinatorial optimization	[47–55]
Constrained optimization	[56, 57]
Multi-objective optimization	[5, 58–63]
Multi-modal optimization	[64]
Dynamic and noisy environment	[65–69]

finite or infinite set, typically, of integers, sets, permutations, or graphs [43]. The latter class of optimization problems can also be called discrete optimization. Solutions of constrained problems must obey some limitations (also known as constraints). In the multi-objective problems, the quality of a solution is defined by its performance in relation to several, possibly conflicting, objectives. On the other hand, for multi-modal problems, there are a (large) number of local modes that are better than all their neighboring solutions, but do not have as good a fitness as the globally optimal solution [44]. The dynamic and noisy problems are non-stationary. That is, they change over time [45].

Various studies of the firefly algorithm in this application domain can be summarized in Table 4.

3.2 Classifications

Classification algorithms are procedures for selecting a hypothesis from a set of alternatives that best fits a set of observations or data. Usually, these algorithms are more relevant in machine learning, data mining, and neural networks. A review of existing studies from this area can be summarized as follows:

- The firefly algorithm was hybridized with the Rough Set Theory (RST) for finding a subset of features [70].
- The firefly algorithm was used for training the radial basis function (RBF) network [71].

Table 5 Engineering applications

Engineering area	References	Total
Industrial optimization	[73–94]	22
Image processing	[95–103]	9
Antenna design	[104–108]	5
Business optimization	[109–112]	4
Robotics	[113–115]	3
Civil engineering	[116, 117]	2
Chemistry	[118, 119]	2
Semantic web	[120]	1
Meteorology	[121]	1
Wireless sensor networks	[122]	1

- The firefly algorithm was used for clustering data objects into groups according to the values of their attributes [72].

4 Engineering Optimization

The firefly algorithm has become one of the most important tools for solving the design optimization problems in routine engineering practice. As can be seen from Table 5, almost every engineering domain has been covered by the applications of this algorithm. The majority of studies come from engineering and industries.

In summary, the rapid expansion of the research literature on the firefly algorithms in engineering optimization proves that the firefly algorithms enter in its matured phase. That is, after initial theoretical studies, more and more applications from real-world case studies have been emerged, which means that this algorithm has become a serious tool for solving various challenging real-world problems.

5 Conclusion

Though with a relative short history up to now, the firefly algorithm has become a matured optimization tool for solving a diverse of range of optimization problems such as engineering designs, scheduling, feature selection, travelling salesman problems, image processing, classifications and industrial applications. Over 20 new FA variants have been developed and new applications and studies are emerging almost daily.

The popularity of the firefly algorithm and its variants may be due to their simplicity, flexibility, versatility and superior efficiency. It is no surprise that FA has been used in almost every area of sciences, engineering and industry.

However, there is still room for improvements. Firstly, theoretical analysis is still very limited, and this is also true for many other nature-inspired algorithms. Mathematical analysis is challenging, but it is possible to use theories such as dynamical systems, Markov chains and statistics to gain insights into the working mechanisms of various variants. Secondly, more applications should focus on large-scale real-world applications. Thirdly, parameter tuning and parameter control can be a very useful area for further research. Finally, the current research communities strive to find better and smarter algorithms. It can be expected that the firefly algorithm and its variants may be extended and further developed into some sort of self-evolving and truly intelligent algorithms.

References

1. Yang, X. S.: Firefly algorithm (chapter 8). Nature-Inspired Metaheuristic Algorithms, pp. 79–90, Luniver Press, Cambridge (2008)
2. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Mixed variable structural optimization using firefly algorithm. Comput. Struct. **89**(23–24), 2325–2336 (2011)
3. Yang, X. S.: Firefly algorithms for multimodal optimization. In: Proceeding of the Conference on Stochastic Algorithms: Foundations and Applications, pp. 169–178. Springer (2009)
4. Yang, X. S.: Firefly algorithm, levy flights and global optimization. In: Watanabe, O., Zeugmann, T. (eds.) Research and Development in Intelligent Systems XXVI, pp. 209–218. Springer, Berlin (2010)
5. Yang, X.S.: Multiobjective firefly algorithm for continuous optimization. Eng. Computers **29**, 175–184 (2013)
6. Fister, I, Fister, I.Jr., Yang, X.-S., Bret, J.: A comprehensive review of firefly algorithms. Swarm and Evolutionary Computation, http://dx.doi.org/10.1016/j.swevo.2013.06.001, (2013 In press)
7. Yang, X.S.: Firefly algorithm, stochastic test functions and design optimisation. Int. J. Bio-Inspired Comput. **2**(2), 78–84 (2010)
8. Parpinelli, R.S., Lopes, H.S.: New inspirations in swarm intelligence: a survey. Int. J. Bio-Inspired Comput. **3**(1), 1–16 (2011)
9. Yang, X.S.: Review of meta-heuristics and generalised evolutionary walk algorithm. Int. J. Bio-Inspired Comput. **3**(2), 77–84 (2011)
10. Zang, H., Zhang, S., Hapeshi, K.: A review of nature-inspired algorithms. J. Bionic Eng. **7**, 232–237 (2010)
11. Ong, H.C., Luleseged Tilahun, S.: Modified firefly algorithm. J. Appl. Math. **2012**, 12 (2012)
12. Chandrasekaran, K., Simon, S.P., Padhy, N.P.: Binary real coded firefly algorithm for solving unit commitment problem. Inf. Sci. (2013) http://dx.doi.org/10.1016/j.ins.2013.06.022
13. Chandrasekaran, K., Simon, S.P.: Network and reliability constrained unit commitment problem using binary real coded firefly algorithm. Int. J. Electr. Power Energy Syst. **43**(1), 921–932 (2012)
14. Falcon, R., Almeida, M., Nayak, A.: Fault identification with binary adaptive fireflies in parallel and distributed systems. In: Evolutionary Computation (CEC), 2011 IEEE Congress on, pp. 1359–1366. IEEE (2011)
15. Farahani, S.M., Abshouri, A.A., Nasiri, B., Meybodi, M.R.: Some hybrid models to improve firefly algorithm performance. Int. J. Artif. Intel. **8**(12), 97–117 (2012)
16. Palit, S., Sinha, S.N., Molla, M.A., Khanra, A., Kule, M.: A cryptanalytic attack on the knapsack cryptosystem using binary firefly algorithm. In: Computer and Communication Technology (ICCCT), 2011 2nd International Conference on, pp. 428–432. IEEE (2011)

17. Farahani, S.M., Abshouri, A.A., Nasiri, B., Meybodi, M.R.: A gaussian firefly algorithm. Int. J. Machine Learn. Comput. **1**(5), 448–454 (2011)
18. Yang, X.S.: Metaheuristic optimization: algorithm analysis and open problems. In: Pardalos, P.M., Rebennack, S. (eds.) Experimental Algorithms, pp. 21–32. Lecture notes in computer science, volume 6630Springer Verlag, Berlin (2011)
19. Yang, X.S.: Efficiency analysis of swarm intelligence and randomization techniques. J. Comput. Theor. Nanosci. **9**(2), 189–198 (2012)
20. dos Santos Coelho, L., de Andrade Bernert, D. L., Mariani, V. C.: A chaotic firefly algorithm applied to reliability-redundancy optimization. In: Evolutionary Computation (CEC), 2011 IEEE Congress on, vol. 18, pp. 89–98, IEEE (2013)
21. Gandomi, A.H., Yang, X.-S., Talatahari, S., Alavi, A.H.: Firefly algorithm with chaos. Commun. Nonlinear Sci. Numer. Simul. **18**(1), 89–98 (2013)
22. Yang, X.-S.: Chaos-enhanced firefly algorithm with automatic parameter tuning. Int. J. Swarm Intell. Res. **2**(4), 1–11 (2011)
23. Husselmann, A.V., Hawick, K.A.: Parallel parametric optimisation with firefly algorithms on graphical processing units. Technical, Report CSTN-141 (2012)
24. Subutic, M., Tuba, M., Stanarevic, N.: Parallelization of the firefly algorithm for unconstrained optimization problems. In: Latest Advances in Information Science and Applications, pp. 264–269 (2012)
25. Liu, G.: A multipopulation firefly algorithm for correlated data routing in underwater wireless sensor networks. Int. J. Distrib. Sens. Netw. (2013)
26. Adaniya, M.H.A.C., et al.: Anomaly detection using metaheuristic firefly harmonic clustering. J. Netw. **8**(1), 82–91 (2013)
27. Adaniya, M.H.A.C, Lima, F.M., Rodrigues, J.J.P.C., Abrao, T., Proenca, M.L.: Anomaly detection using dsns and firefly harmonic clustering algorithm. In: Communications (ICC), 2012 IEEE International Conference on, pp. 1183–1187. IEEE (2012)
28. Fister, I., Yang, X.-S., Brest, J., Fister, I.Jr.: Modified firefly algoirthm using quaternion representation. Expert Systems with Applications, http://dx.doi.org/10.1016/j.eswa.2013.06.070, (2013)
29. Yang, X. S., Deb, S.: Eagle strategy using levy walk and firefly algorithms for stochastic optimization. In: Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), pp. 101–111 (2010)
30. Luthra, J., Pal, S.K.: A hybrid firefly algorithm using genetic operators for the cryptanalysis of a monoalphabetic substitution cipher. In: Information and Communication Technologies (WICT), 2011 World Congress on, pp. 202–206. IEEE (2011)
31. Abdullah, A., Deris, S., Mohamad, M., Hashim, S.: A new hybrid firefly algorithm for complex and nonlinear problem. In: Omatu, S., et al. (eds.) Distributed Computing and, Artificial Intelligence, vol. 151, pp. 673–680. Springer, Berlin (2012)
32. Abdullah, A., Deris, S., Anwar, S., Arjunan, S.N.V.: An evolutionary firefly algorithm for the estimation of nonlinear biological model parameters. PloS one. **8**(3), e56310 (2013)
33. Fister, I.Jr., Yang, X.-S., Fister, I., Brest, J.: Memetic firefly algorithm for combinatorial optimization. pp. 75–86. Jožef Stefan Institute (2012)
34. Srivastava, A., Chakrabarti, S., Das, S., Ghosh, S., Jayaraman, V.K.: Hybrid firefly based simultaneous gene selection and cancer classification using support vector machines and random forests. In; Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012), pp. 485–494. Springer (2013)
35. Hassanzadeh, T., Faez, K., Seyfi, G.: A speech recognition system based on structure equivalent fuzzy neural network trained by firefly algorithm. In: Biomedical Engineering (ICoBE), 2012 International Conference on, pp. 63–67. IEEE (2012)
36. Nandy, S., Sarkar, P.P., Das, A.: Analysis of a nature inspired firefly algorithm based back-propagation neural network training. arXiv, preprint arXiv:1206.5360 (2012)
37. Ranjan Senapati, M., Dash, P.K.: Local linear wavelet neural network based breast tumor classification using firefly algorithm. Neural Comput. Appl. **30**, pp. 1–8 (2013)

38. Hassanzadeh, T., Meybodi, M.R.: A new hybrid algorithm based on firefly algorithm and cellular learning automata. In: 20th Iranian Conference on Electrical Engineering, pp. 628–633. IEEE (2012)
39. Aruchamy, R., Vasantha, K.D.D.: A comparative performance study on hybrid swarm model for micro array data. Int. J. Comput. Appl. **30**(6), 10–14 (2011)
40. Vahedi Nouri, B., Fattahi, P., Ramezanian, R.: Hybrid firefly-simulated annealing algorithm for the flow shop problem with learning effects and flexible maintenance activities. Int. J. Prod. Res. (ahead-of-print), 1–15 (2013)
41. Luleseged Tilahun, S., Ong, H.C.: Vector optimisation using fuzzy preference in evolutionary strategy based firefly algorithm. Int. J. Oper. Res. **16**(1), 81–95 (2013)
42. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)
43. Papadimitriou, H., Steglitz, I.: Copmbinatorial Optimization: Algorithms and Complexity. Dover Publications, Inc., Mineola, NY (1998)
44. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer-Verlag, Berlin (2003)
45. Morrison, R.W.: Designing Evolutionary Algorithms for Dynamic Environments. Springer Verlag, Berlin (2004)
46. Poursalehi, N., Zolfaghari, A., Minuchehr, A., Moghaddam, H.K.: Continuous firefly algorithm applied to pwr core pattern enhancement. Nucl. Eng. Des. **258**, 107–115 (2013)
47. Durkota, K.: Implementation of a discrete firefly algorithm for the qap problem within the sage framework. Czech Technical University, Prague, Master's thesis (2009)
48. Hönig, U.: A firefly algorithm-based approach for scheduling task graphs in homogeneous systems. In: Informatics, pp. 24–33. ACTA Press (2010)
49. G. Jati. Evolutionary discrete firefly algorithm for travelling salesman problem. In: Adaptive and Intelligent Systems, pp. 393–403 (2011)
50. Khadwilard, A., Chansombat, S., Thepphakorn, T., Thapatsuwan, P., Chainate, W., Pongcharoen, P.: Application of firefly algorithm and its parameter setting for job shop scheduling. In: 1st Symposium on Hands-On Research and, Development, pp. 1–10 (2011)
51. Kwiecień, J., Filipowicz, B.: Firefly algorithm in optimization of queueing systems. Tech. Sci. **60**(2), 363–368 (2012)
52. Liu, C., Gao, Z., Zhao, W.: A new path planning method based on firefly algorithm. In: Computational Sciences and Optimization (CSO), 2012 Fifth International Joint Conference on, pp. 775–778. IEEE (2012)
53. Marichelvam, M.K., Prabaharan, T., Yang, X.-S.: A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. IEEE Trans. Evol. Comput. TEVC-00124-2012 (2012)
54. Sayadi, M.K., Ramezanian, R., Ghaffari-Nasab, N.: A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. Int. J. Industrial Eng. Comput. **1**(1), 1–10 (2010)
55. Wang, G., Guo, L., Duan, H., Liu, L., Wang, H.: A modified firefly algorithm for ucav path planning. Int. J. Hybrid Inf. Technol. **5**(3), 123–144 (2012)
56. Gomes, H.M.: A firefly metaheuristic structural size and shape optimisation with natural frequency constraints. Int. J. Metaheuristics **2**(1), 38–55 (2012)
57. Łukasik, S., Żak, S.: Firefly algorithm for continuous constrained optimization tasks. In: Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems, pp. 97–106. Springer, 2009.
58. Abedinia, O., Amjady, N., Naderi, M.S.: Multi-objective environmental/economic dispatch using firefly technique. In: Environment and Electrical Engineering (EEEIC), 2012 11th International Conference on, pp. 461–466. IEEE (2012)
59. Amiri, B.k, Hossain, L., Crawford, J.W., Wigand, R.T.: Community detection in complex networks: Multi-objective enhanced firefly algorithm. Knowl.-Based Syst. **46**, 1–11 (2013)
60. dos Santos Coelho, L., Bora, L.C.: Felipe Schauenburg, and Piergiorgio Alotto. A multiobjective firefly approach using beta probability distribution for electromagnetic optimization problems. IEEE Trans. Magn. **49**(5), 2085 (2013)

61. Poursalehi, N., Zolfaghari, A., Minuchehr, A.: Multi-objective loading pattern enhancement of pwr based on the discrete firefly algorithm. Ann. Nucl. Energy **57**, 151–163 (2013)
62. Niknam, T., Azizipanah-Abarghooee, R., Roosta, A., Amiri, B.: A new multi-objective reserve constrained combined heat and power dynamic economic emission dispatch. Energy **42**(1), 530–545. Elsevier (2012)
63. Santander-Jiménez, S., Vega-Rodríguez, M.A.: A multiobjective proposal based on the firefly algorithm for inferring phylogenies. In: Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, pp. 141–152. Springer (2013)
64. Miguel, L.F.F.: Rafael Holdorf Lopez, and Letícia Fleck Fadel Miguel. Multimodal size, shape, and topology optimisation of truss structures using the firefly algorithm. Adv. Eng. Softw. **56**, 23–37 (2013)
65. Abshouri, A.A., Meybodi, M.R., Bakhtiary, A.: New firefly algorithm based on multi swarm & learning automata in dynamic environments. In: IEEE proceedings, pp. 73–77 (2011)
66. Chai-Ead, N., Aungkulanon, P., Luangpaiboon, P.: Bees and firefly algorithms for noisy non-linear optimization problems. In: Proceedings of the International Multi Conference of Engineering and Computer Scientists **2**, 1–6 (2011)
67. Farahani, S.M., Nasiri, B., Meybodi. M.R.: A multiswarm based firefly algorithm in dynamic environments. In: Third International Conference on Signal Processing Systems (ICSPS2011), vol. 3, pp. 68–72 (2011)
68. Nasiri, B., Meybodi, M.R.: Speciation based firefly algorithm for optimization in dynamic environments. Int. J. Artif. Intell. **8**(12), 118–132 (2012)
69. Mustafa, M.W., Azmi, A., Aliman, O., Abdul Rahim, S.R.: Optimal allocation and sizing of distributed generation in distribution system via firefly algorithm. In: Power Engineering and Optimization Conference (PEDCO) Melaka, Malaysia, 2012 IEEE International, pp. 84–89. IEEE (2012)
70. Banati, H., Bajaj, M.: Firefly based feature selection approach. IJCSI Int. J. Comput. Sci. Issues **8**(4), 473–480 (2011)
71. Horng, M.H., Lee, Y.X., Lee, M.C., Liou, R.J.: Firefly meta-heuristic algorithm for training the radial basis function network for data classification and disease diagnosis. In: Parpinelli, R., Lopes, H.S. (eds.) Theory and New Applications of Swarm Intelligence, pp. 1–19. InTech, Rijeka (2012)
72. Senthilnath, J.: SN Omkar, and V. Mani. Clustering using firefly algorithm: Performance study. Swarm Evol. Comput. **1**(3), 164–171 (2011)
73. Abedinia, O., Amjady, N., Kiani, K., Shayanfar, H.A.: Fuzzy pid based on firefly algorithm: Load frequency control in deregulated environment. In: The 2012 International Conference on Bioinformatics and Computational Biology, pp. 1–7 (2012)
74. Apostolopoulos, T., Vlachos, A.: Application of the firefly algorithm for solving the economic emissions load dispatch problem. In: International Journal of Combinatorics, 2011, 23 p., (2011)
75. Aungkulanon, P., Chai-Ead, N., Luangpaiboon, P.: Simulated manufacturing process improvement via particle swarm optimisation and firefly algorithms. In Proceedings of the International MultiConference of Engineers and Computer Scientists **2**, 1–6 (2011)
76. Chandrasekaran, K., Simon, S.P.: Optimal deviation based firefly algorithm tuned fuzzy design for multi-objective ucp. IEEE Trans. Power Syst. **28**(1), 460–471 (2013)
77. handrasekaran, K., Simon, S.P.: Demand response scheduling in scuc problem for solar integrated thermal system using firefly algorithm. In: Renewable Power Generation (RPG 2011), IET Conference on, pp. 1–8. IET (2011)
78. Chatterjee, A., Mahanti, G.K., Chatterjee, A.: Design of a fully digital controlled reconfigurable switched beam concentric ring array antenna using firefly and particle swarm optimization algorithm. Prog. Electromagnet Res. B **36**, 113–131. EMW Publishing (2012)
79. dos Santos Coelho, L., Mariani, V.C.: Improved firefly algorithm approach for optimal chiller loading for energy conservation. Energy Buildings **59**, 1–320 (2012)
80. Dekhici, L., Borne, P., Khaled, B., et al.: Firefly algorithm for economic power dispatching with pollutants emission. Informatica Economică **16**(2), 45–57 (2012)

81. Dutta, R., Ganguli, R., Mani, V.: Exploring isospectral spring-mass systems with firefly algorithm. In: Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science, vol. 467, pp. 3222–3240. The Royal Society (2011)
82. Hu, H.: Fa-based optimal strategy of trains energy saving with energy materials. Adv. Mater. Res. **485**, 93–96 (2012)
83. Kazemzadeh, A.S.: Optimum design of structures using an improved firefly algorithm. Int. J. Optim. Civ. Eng. **2**(1), 327–340 (2011)
84. Mauder, T., Sandera, C., Stetina, J., Seda, M.: Optimization of the quality of continuously cast steel slabs using the firefly algorithm. Materiali in tehnologije **45**(4), 347–350 (2011)
85. Mohammadi, s., Mozafari, B., Solimani, S., Niknam, T.: An adaptive modified firefly optimisation algorithm based on hong's point estimate method to optimal operation management in a microgrid with consideration of uncertainties. Energy (2013)
86. Bharathi Raja, S., Srinivas Pramod, C.V., Vamshee Krishna, K., Ragunathan, A., Vinesh, S., Optimization of electrical discharge machining parameters on hardened die steel using firefly algorithm. Engineering with Computers **36**, 1–9 (2013)
87. Rampriya, B., Mahadevan, K., Kannan, S.: Unit commitment in deregulated power system using Lagrangian firefly algorithm. In: Communication Control and Computing Technologies (ICCCCT), 2010 IEEE International Conference on, pp. 389–393. IEEE (2010)
88. Roeva, O.: Optimization of e. coli cultivation model parameters using firefly algorithm. Int. J. Bioautomation **16**, 23–32 (2012)
89. Roeva, O., Slavov, T.: Firefly algorithm tuning of pid controller for glucose concentration control during e. coli fed-batch cultivation process. In: Proceedings of the Federated Conference on Computer Science and Information Systems, pp. 455–462. IEEE (2012)
90. Rubio-Largo, Á., Vega-Rodríguez, M. A.: Routing low-speed traffic requests onto high-speed lightpaths by using a multiobjective firefly algorithm. In Applications of Evolutionary Computation, p. 12–21. Springer (2013)
91. Chandra Saikia, L., Kant Sahu, S.: Automatic generation control of a combined cycle gas turbine plant with classical controllers using firefly algorithm. Int. J. Electr. Power Energy Syst, **53**, 27–33 (2013)
92. Sanaei, P., Akbari, R., Zeighami, V., Shams, S.: Using firefly algorithm to solve resource constrained project scheduling problem. In: Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012), pp. 417–428. Springer (2013)
93. Yang, X.S., Hosseini, S.S.S., Gandomi, A.H.: Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect. Appl. Soft Comput. **12**(3), 1180–1186 (2011)
94. Yazdani, A., Jayabarathi, T., Ramesh, V., Raghunathan, T.: Combined heat and power economic dispatch problem using firefly algorithm. Front. Energy **7**, 1–7 (2013)
95. Hassanzadeh, T., Vojodi, H., Mahmoudi, F.: Non-linear grayscale image enhancement based on firefly algorithm. In: Swarm, Evolutionary, and Memetic Computing, pp. 174–181. Springer (2011)
96. Hassanzadeh, T., Vojodi, H., Moghadam, A.M.E.: An image segmentation approach based on maximum variance intra-cluster method and firefly algorithm. In: Natural Computation (ICNC), 2011 Seventh International Conference on, vol. 3, pp. 1817–1821. IEEE (2011)
97. Horng, M.H.: Vector quantization using the firefly algorithm for image compression. Expert Syst. Appl. **39**(1), 1078–1091 (2012)
98. Horng, M.H., Jiang, T.W.: The codebook design of image vector quantization based on the firefly algorithm. In: Computational Collective Intelligence. Technologies and Applications, pp. 438–447 (2010)
99. Horng, M.H., Jiang, T.W.: Multilevel image thresholding selection based on the firefly algorithm. In: Ubiquitous Intelligence and Computing and 7th International Conference on Autonomic and Trusted Computing (UIC/ATC), 2010 7th International Conference on, pp. 58–63. IEEE (2010)

100. Horng, M.H., Liou, R.J.: Multilevel minimum cross entropy threshold selection based on the firefly algorithm. Expert Syst. Appl. **38**(12), 14805–14811. Elsevier (2011)
101. Mohd Noor, M.H., Ahmad, A.R., Hussain, Z., Ahmad, K.A., Ainihayati, A.R.: Multilevel thresholding of gel electrophoresis images using firefly algorithm. In: Control System, Computing and Engineering (ICCSCE), 2011 IEEE International Conference on, pp. 18–21. IEEE (2011)
102. Xiaogang, D., Jianwu, D., Yangping, W., Xinguo, L., Sha, L.: An algorithm multi-resolution medical image registration based on firefly algorithm and powell. In: Intelligent System Design and Engineering Applications (ISDEA), 2013 Third International Conference, pp. 274–277. IEEE (2013)
103. Zhang, Y., Wu, L.: A novel method for rigid image registration based on firefly algorithm. Int. J. Res. Rev. Soft Intell. Comput. (IJRRSIC) **2**(2), 141–146 (2012)
104. Basu, B., Mahanti, G.K.: Firefly and artificial bees colony algorithm for synthesis of scanned and broadside linear array antenna. Prog. Electromagnet Res. B **32**, 169–190 (2011)
105. Basu, B., Mahanti, G.K.: Thinning of concentric two-ring circular array antenna using fire fly algorithm. Scientia Iranica, 19(6), 1802–1809 (2012)
106. Chatterjee, A., Mahanti, G.K.: Minimization in variations of different parameters in different φ planes of a small-size concentric ring array antenna using firefly algorithm. Ann. Telecommun. **68**, 1–8 (2012)
107. Sharaqa, A., Dib, N.: Circular antenna array synthesis using firefly algorithm. Int. J. RF Microwave Comput. Aided Eng. Article in press, Wiley Online Library (2013)
108. Zaman, M.A., Matin, A., et al.: Nonuniformly spaced linear antenna array design using firefly algorithm. Int. J. Microwave Sci. Technol. 2012, 8 (2012)
109. Banati, H., Bajaj, M.: Promoting products online using firefly algorithm. In: Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on, pp. 580–585, IEEE (2012)
110. Giannakouris, G., Vassiliadis, V., Dounias, G.: Experimental study on a hybrid nature-inspired algorithm for financial portfolio optimization. In: Artificial Intelligence: Theories, Models and Applications, pp. 101–111 (2010)
111. Salomie, I., Chifu, V.R., Pop, C.B., Suciu, R.: Firefly-based business process optimization. pp. 49–56 (2012), cited By (since 1996)
112. Yang, X. S., Deb, S., Fong, S.: Accelerated particle swarm optimization and support vector machine for business optimization and applications. In: Networked Digital Technologies, pp. 53–66 (2011)
113. Jakimovski, B., Meyer, B., Maehle, E.: Firefly flashing synchronization as inspiration for self-synchronization of walking robot gait patterns using a decentralized robot control architecture. Archit. Comput. Sys. ARCS 2010, 61–72 (2010)
114. Mardlijah, A.J., Widodo, B., Santoso, A.: A new combination method of firefly algorithm and t2fsmc for mobile inverted pendulum robot. J. Theor. Appl. Inf. Technol. **47**(2):824–831 (2013)
115. Severin, S., Rossmann, J.: A comparison of different metaheuristic algorithms for optimizing blended ptp movements for industrial robots. In: Intelligent Robotics and Applications, pp. 321–330 (2012)
116. Gholizadeh, S., Barati, H.: A comparative study of three metaheuristics for optimum design of trusses. Int. J. Optim. Civ. Eng. **3**, 423–441 (2012)
117. Talatahari, S., Gandomi, A.H., Yun, G.J.: Optimum design of tower structures using firefly algorithm. The Structural Design of Tall and Special Buildings (2012)
118. Fateen, S.E., Bonilla-Petriciolet, A., Rangaiah, G.P.: Evaluation of covariance matrix adaptation evolution strategy, shuffled complex evolution and firefly algorithms for phase stability, phase equilibrium and chemical equilibrium problems. Chem. Eng. Res. Des. **90**(12), 2051–2071 (2012)
119. Zhang, Y., Wang, S.: Solving two-dimensional hp model by firefly algorithm and simplified energy function. Mathematical Problems in Engineering. vol. 2013, 398141, 9 p (2013). doi:10.1155/2013/398141

120. Pop, C.B., Chifu, V.R., Salomie, I., Baico, R.B., Dinsoreanu, M., Copil, G.: A hybrid firefly-inspired approach for optimal semantic web service composition. Scal. Comput. Pract. Exp. vol. 12(3), pp. 363–369 (2011)
121. dos Santos, A.F., de Campos Velho, H.F., Luz, E.F.P., Freitas, S.R., Grell, G., Gan, M.A.: A Firefly optimization to determine the precipitation field on South, America. Inverse Prob. Sci. Eng. **21**, 417–428 (2013)
122. Breza, M., McCann, J.A.: Lessons in implementing bio-inspired algorithms on wireless sensor networks. In Adaptive Hardware and Systems, 2008. AHS'08. NASA/ESA Conference on, pp. 271–276. IEEE (2008)