Institute for Mathematics and its Applications IMA

The **Institute for Mathematics and its Applications** was established by a grant from the National Science Foundation to the University of Minnesota in 1982. The IMA seeks to encourage the development and study of fresh mathematical concepts and questions of concern to the other sciences by bringing together mathematicians and scientists from diverse fields in an atmosphere that will stimulate discussion and collaboration.

The IMA Volumes are intended to involve the broader scientific community in this process.

Willard Miller, Jr., Professor and Director

* * * * * * * * * *

IMA ANNUAL PROGRAMS

1982–1983	Statistical and Continuum Approaches to Phase Transition
1983–1984	Mathematical Models for the Economics of Decentralized Resource Allocation
1984–1985	Continuum Physics and Partial Differential Equations
1985–1986	Stochastic Differential Equations and Their Applications
1986–1987	Scientific Computation
1987–1988	Applied Combinatorics
1988–1989	Nonlinear Waves
1989–1990	Dynamical Systems and Their Applications
1990–1991	Phase Transitions and Free Boundaries
1991–1992	Applied Linear Algebra
1992–1993	Control Theory and its Applications
1993–1994	Emerging Applications of Probability
1994–1995	Waves and Scattering
1995–1996	Mathematical Methods in Material Science
1996–1997	Mathematics of High Performance Computing
1997–1998	Emerging Applications of Dynamical Systems
1998–1999	Mathematics in Biology
1999–2000	Reactive Flows and Transport Phenomena
2000–2001	Mathematics in Multi-Media

Continued at the back

Lawrence David Davis Kenneth De Jong
Michael D. Vose L. Darrell Whitley
Editors

Evolutionary Algorithms

With 81 Illustrations

Springer

Lawrence David Davis
Tica Associates
Newbury, MA 019151, USA

Kenneth De Jong
Computer Science Department
George Mason University
Fairfax, VA 22030-4444, USA

Michael D. Vose
Computer Science Department
The University of Tennessee
Knoxville, TN 37996-1301, USA

L. Darrell Whitley
Computer Science Department
Colorado State University
Fort Collins, CO 80523, USA

Series Editor:
Willard Miller, Jr.
Institute for Mathematics and its Applications
University of Minnesota
Minneapolis, MN 55455, USA

Mathematics Subject Classifications (1991): 68T05, 90B40

Library of Congress Cataloging-in-Publication Data
Evolutionary algorithms / Lawrence David Davis . . . [et al.].
p. cm. — (The IMA volumes in mathematics and its applications ; v. 111)
Includes bibliographical references and index.
ISBN 0-387-98826-2 (alk. paper)
1. Genetic algorithms—Congresses. I. Davis, Lawrence David.
II. Series.
QA402.5.E95 1999
519.3—dc21 99-18387

Printed on acid-free paper.

Production managed by Allan Abrams; manufacturing supervised by Nancy Wu.
Camera-ready copy prepared by the IMA.
Printed and bound by Braun-Brumfield, Inc., Ann Arbor, MI.
Printed in the United States of America.

9 8 7 6 5 4 3 2 1

ISBN 0-387-98826-2 Springer-Verlag New York Berlin Heidelberg SPIN 10717722

FOREWORD

This IMA Volume in Mathematics and its Applications

EVOLUTIONARY ALGORITHMS

is based on the proceedings of a workshop that was an integral part of the 1996–97 IMA program on "MATHEMATICS IN HIGH-PERFORMANCE COMPUTING."

I thank Lawrence David Davis (Tica Associates), Kenneth De Jong (Computer Science, George Mason University), Michael D. Vose (Computer Science, The University of Tennessee), and L. Darrell Whitley (Computer Science, Colorado State University) for their excellent work in organizing the workshop and for editing the proceedings. Further appreciation is extended to Donald G. Truhlar (Chemistry and Supercomputing Institute, University of Minnesota) who was also one of the workshop organizers.

In addition, I also take this opportunity to thank the National Science Foundation (NSF), Minnesota Supercomputing Institute (MSI), and the Army Research Office (ARO), whose financial support made the workshop possible.

Willard Miller, Jr., Professor and Director

PREFACE

The IMA Workshop on Evolutionary Algorithms brought together many of the top researchers working in the area of Evolutionary Computation for a week of intensive interaction. The field of Evolutionary Computation has developed significantly over the past 30 years and today consists a variety of subfields such as genetic algorithms, evolution strategies, evolutionary programming, and genetic programming, each with their own algorithmic perspectives and goals.

By bringing together key individuals from the various subfields in one place in a relative small forum of fifty individuals, commonalities and differences in these approaches were made much clearer. The field of Evolutionary Computation is not mature, and is still enjoying youthful exuberance. The workshop facilitated communication among many of the various camps in the evolutionary computation field, encouraging mutual cooperation and challenging members of each field to translate and integrate within their own perspectives the principles and approaches of the others. The workshop also provided opportunities for people who belong to the same subdiscipline but who infrequently have the opportunity to meet with each other to share and discuss ideas.

The workshop did a great deal to clarify the current state of the theory in Evolutionary Algorithms. The existing theory might be characterized as deriving from two principal approaches. There is a high level macro-theory that looks at the processing of "building blocks" and "schemata" that are shared by many good solutions when searching a problem space. There is also a low level micro-theory that builds exact Markov models of the search process. The macro-level theory ignores certain detail; the micro-level theory involves working with detailed models that unfortunately grow exponentially as a function of problem size, since the sizes of the models are larger than the search space itself. It is sometimes hard for researchers working at such different levels of abstraction to interact. The IMA workshop allowed researchers working at these different levels to present their points of view and to move toward common ground.

Another area in which there was real progress was in communication between theorist and practitioners in the evolutionary computation field. Speakers presented applications across a wide range of problem areas. In some of those cases, theoretically motivated methods work quite well. In other cases, practitioners used domain-based methods to obtain better performance than could be achieved by using a "pure" evolutionary algorithm. Individuals on both sides went away with a better appreciation of the successes and failures of current theory. The workshop should help to change what practitioners say about the current state of theory in the field.

Some very surprising and impressive applications were presented at

the workshop, including data decomposition, deriving control parameters for modeling systems using partial differential equations, and even new best known solutions to bin packing problems.

Several of the parallel computing speakers discussed techniques for parallelizing algorithms at a meta-level. This approach is not one typically followed in the evolutionary algorithm field, where hybridization of algorithms tends to occur in the operator set. The result of this is likely to be renewed experimentation with hybridization techniques in the evolutionary algorithm field.

Overall, the workshop was not only useful for bring people together and strengthening the evolutionary computation community; it was also a valuable learning experience. The IMA did a wonderful job in creating the environment for the workshop, in promoting it, and in structuring it so that there was extensive time for offline interaction.

Lawrence David Davis (Tica Associates)

Kenneth De Jong (George Mason University)

Michael D. Vose (The University of Tennessee)

L. Darrell Whitley (Colorado State University)

CONTENTS

GENETIC ALGORITHMS AS MULTI-COORDINATORS IN LARGE-SCALE OPTIMIZATION*

IOANNIS T. CHRISTOU†, WAYNE MARTIN†, AND ROBERT R. MEYER†

Abstract. We present high-level, decomposition-based algorithms for large-scale block-angular optimization problems containing integer variables, and demonstrate their effectiveness in the solution of large-scale graph partitioning problems. These algorithms combine the subproblem-coordination paradigm (and lower bounds) of price-directive decomposition methods with knapsack and genetic approaches to the utilization of "building blocks" of partial solutions. Even for graph partitioning problems requiring billions of variables in a standard 0-1 formulation, this approach produces high-quality solutions (as measured by deviations from an easily computed lower bound), and substantially outperforms widely-used graph partitioning techniques based on heuristics and spectral methods.

1. Introduction. Most large-scale mathematical programming problems are constructed by concatenating and linking together blocks of data corresponding to major components of the problem. In the case of network optimization, for example, these components may correspond to different commodities or goods with different priorities or origins. There may also be multiple time periods or scenarios, and even combinations of all of the preceding factors. In many important cases it is thus possible to formulate such problems as block-angular problems of the following form:

$$
\begin{array}{llllllllll}
\min\limits_{x} & f_1(x_1) & + & f_2(x_2) & + & \ldots & + & f_P(x_P) & & \\
s.t. & A_1 x_1 & & & & & & & = & b_1 \\
& & & A_2 x_2 & & & & & = & b_2 \\
& & & & & \ddots & & & & \\
& & & & & & & A_P x_P & = & b_P \\
& D_1 x_1 & + & D_2 x_2 & + & \ldots & + & D_P x_P & \leq & d \\
& & & 0 \leq x_p \leq u_p, & & p = 1, \ldots, P & & & &
\end{array}
\tag{1}
$$

In addition, we assume that some or all of the variables are constrained to be integer. Note that x_p represents a block of variables and the matrices A_p and D_p, $p = 1, \ldots, P$ correspond to blocks of constraints. For large-scale problems, the number of variables in each block may be in the thousands and there may be thousands of constraints. (In the largest problem that we have solved by our approach, $P = 100,000$ and each x_p is a block of $100,000,000$ variables.)

*This research was partially supported by the Air Force Office of Scientific Research under grant F49620-94-1-0036, and by the NSF under grants CDA-9024618 and CCR-9306807.

†Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706.

The constraints $\sum D_p x_p \leq d$ are called *coupling constraints* since the deletion of these constraints removes the coupling between the blocks of variables x_p. Thus, the resulting "relaxed" problem may be solved by decomposing it into P smaller optimization problems (called subproblems) of the form:

$$\begin{aligned} \min_x \quad & f_p(x_p) \\ s.t. \quad & A_p x_p = b_p \\ & 0 \leq x_p \leq u_p \end{aligned} \tag{2}$$

Because of the relationship between their feasible sets, the optimal value (which is the sum of the optimal values of the subproblems) of the full relaxed problem provides a *lower bound* for the objective of the original problem, but an optimal solution of the relaxed problem (the concatenation of the subproblem solutions) will generally violate the coupling constraints. In order to satisfy these constraints, a major iteration of a traditional price-directive decomposition approach for convex optimization utilizes a subproblem phase in which the problems (2) are modified by altering their objective functions (usually by adding infeasibility penalties via Lagrangian-related terms) to produce subproblems of the form

$$\begin{aligned} \min_x \quad & f_p^*(\lambda_p, x_p) \\ s.t. \quad & A_p x_p = b_p \\ & 0 \leq x_p \leq u_p \end{aligned} \tag{3}$$

where λ_p represents the penalty multipliers (on the coupling constraints) used in defining the subproblem for the block of variables x_p in this iteration. This phase is followed by a coordination phase in which the solutions of these modified subproblems are combined (usually via linear weights) subject to the coupling constraints to obtain feasible solutions of the original problem. For continuous optimization problems, these weights are continuous variables and the coordination problem is generally a tractable continuous optimization problem. However, for discrete optimization problems, continuous weights would generally destroy the integrality of the subproblem solutions, so the coordination problem in this case is a more difficult discrete optimization problem. The approach that we consider here can also be viewed within this subproblem-coordination framework, but the techniques used for both subproblem generation and for coordination differ significantly from traditional decomposition methods. One major difference in our approach is that we employ the techniques of genetic algorithms (GA's) in the coordination phase. The concept of combining parts of solutions to obtain better solutions is central to GA's and hence

meshes well with the coordination paradigm. However, while traditional GA's focus on the original problem variables and encode solutions as binary strings, the "genes" in our high-level GA correspond to encodings of subproblem solutions or collections of subproblem solutions. In particular, we approximate the original problem by replacing the original variables by a vector variable w of length $P' \leq P$, in which w_i designates the index of a subproblem solution (or collection of subproblem solutions) chosen from a library of such objects. (We consider below how a vector w may be mapped into a feasible solution of the original problem.) With this encoding, the general approach is as follows:

0) (Initialization) Initialize the library of subproblem solutions and select an initial set of values of w as the initial GA population.

1) (Multi-coordination) Run a GA for one generation to produce a new population.

2) (Subproblem phase) Using the results of the multi-coordination step, adjust penalty weights to obtain a new family of subproblems, generate the corresponding subproblem solutions, augment the subproblem solution library, and then repeat the multi-coordination step.

As we will see in the application below, there need not be a strict separation between steps 1) and 2) of this procedure, since the adjustment of penalty weights may take place as the w vectors are expanded to feasible solutions.

2. Minimum-perimeter equi-partition. We will first outline our approach in the case of a particular class of graph partitioning problems, and then consider generalizations to other block-angular problems. In order to establish a simple geometric viewpoint, consider a rectangular grid $\mathcal{G}$ of unit-area cells and define the *minimum perimeter problem* MP($\mathcal{G}$, P) to be the equi-partition of the grid into P components (whose areas differ by at most one unit) of minimum total perimeter. Graph partitioning of large 5-point uniform grids arises in the context of the parallel solution of PDEs using finite difference schemes [Str89] in a parallel computing environment, in computer vision [Sch89] and in the simulation of molecular behavior in chemical engineering. It also is needed in the parallel solution of large-scale maximum-flow problems arising from the study of magnetic flux lines in super-conductors. The size of the graph required for the study of the asymptotic behavior of these flux lines renders large versions of this problem impossible to solve in a single workstation mainly because of memory requirements (up to 127MB of memory is required even for relatively small problems of this category). Therefore, a decomposition of the graph into smaller pieces that fit in the memory of a single node of a Network of Workstations or a CM-5 or an SP2 is required. Since preflow push maintains a list of active nodes and performs "flow push" on the immediate neighbors of the selected node, it performs local computations on a graph that is logically a 5-point rectangular grid. To minimize interprocessor com-

munication we need to minimize the borders of the region each processor occupies. Under the assumption that all processors are equally powerful, load balancing becomes equivalent to equi-partitioning, i.e., the area of the domain assigned to each processor differs by no more than one from the area of the domain assigned to any other processor.

Letting the binary variable x_i^p correspond to the possible assignment of processor p to cell i, $\mathcal{I}$ denote the set of pairs of adjacent cells, and A_p the area (i.e., total number of cells to be assigned) for processor p, the simplest algebraic expression of the minimum perimeter problem is as a Quadratic Assignment Problem (QAP) [PRW93]:

$$\min \sum_{i,j \in \mathcal{G}} \sum_{\substack{p,p'=1 \\ p \neq p'}}^{P} c_{ij} x_i^p x_j^{p'} \tag{4}$$

$$s.t. \begin{cases} \sum_{i \in \mathcal{G}} x_i^p = A_p & p = 1 \ldots P \\ \sum_{p=1}^{P} x_i^p = 1 & i \in \mathcal{G} \\ x_i^p \in \mathbf{B} = \{0,1\} \end{cases}$$

$$\text{where } c_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{I} \\ 0 & \text{else} \end{cases}$$

and the partition sizes A_p are chosen such that their cardinalities differ by at most 1. Note that this objective function does not count the edges of the boundary of the original domain $\mathcal{G}$, which are part of the total perimeter of any partition, but since this contribution is simply a fixed constant, it may be ignored in setting up an optimization problem, and then added to the optimal value of that problem to obtain the value of the total perimeter of the solution. While this algebraic formulation of the objective function is compact and is useful for comparisons with methods for the QAP, it does not match the block separable format of the block angular problem (1) that we wish to use for algorithmic purposes below, so we observe that the objective can also expressed as $\sum_{p=1}^{P} \mathcal{P}(x_p)$, where $\mathcal{P}(x_p)$ is the perimeter of component p and x_p is the block of variables x_i^p where p is fixed and i ranges over $\mathcal{G}$.

Deletion of the coupling constraints $\sum_{p=1}^{P} x_i^p = 1$, $i \in \mathcal{G}$ yields a relaxed problem that decomposes into P independent subproblems of the form:

$$\min \mathcal{P}(x_p) \tag{5}$$

$$s.t. \begin{cases} \sum_{i \in \mathcal{G}} x_i^p = A_p \\ x_i^p \in \mathbf{B} = \{0,1\}. \end{cases}$$

Each such subproblem requires the determination of the minimum perimeter of a collection of A_p cells in the domain. In most cases of interest, the domain $\mathcal{G}$ will contain at least one embedded square of area at least A_p, in which case it can be shown that the optimal value of the subproblem is precisely $2\left\lceil 2\,A_p^{1/2}\right\rceil$. (It is possible to establish optimal subproblem values [CM96a] even if $\mathcal{G}$ is so elongated that it does not contain even one such square, but we will not consider such unusual cases here.) In addition, as shown in [YM92] it is possible to construct all possible optimal solutions to the subproblems, essentially by determining those rectangular "frames" with perimeter is $2\left\lceil 2\,A_p^{1/2}\right\rceil$ and area at least A_p (there are $O(A_p^{1/4})$ such frames). If h is the height of such an optimal frame, we will refer to h as an optimal height in the discussion below. As we will see, the subproblem solutions, which we will refer to as "optimal shapes" and the corresponding optimal heights are related to the building blocks of the GA's that we develop below.

If one can tile together P such optimal shapes to completely cover the grid, it follows that one has an optimal solution to the minimum perimeter problem, since the value of the solution will match the lower bound (obtained by summing the optimal values of the subproblems):

$$LB = \sum_p 2\left\lceil 2\,A_p^{1/2}\right\rceil. \tag{6}$$

As we will see below, this lower bound tends to be quite good in cases in which the original domain is rectangular and P is relatively large. On the other hand, it is easy to construct examples in which a tiling that uses only optimal shapes does not exist and hence this lower bound cannot be attained (consider, for example, the minimum perimeter equi-partition of a 3×3 grid into two components of areas 4 and 5). Hence, the general approach that we will consider involves the generation of optimal solutions to the subproblems above (or variants of these subproblems with suitably modified objective functions), accompanied by coordination procedures that combine (and sometimes modify) subproblem solutions to produce good feasible solutions to the original problem. We first consider rectangular domains, and then show how a GA for that problem class may generalized and applied to non-rectangular domains.

3. Stripe decomposition for rectangular domains. In this section we consider the case in which the domain $\mathcal{G}$ is an $M \times N$ rectangular grid and we will initially assume that P divides MN (so that each component has equal area $A = MN/P$). For this class of problems, we will first describe an algorithm based on using a knapsack algorithm for coordination, and then consider a GA that represents an alternative coordination approach and applies to more general problem classes.

Under these assumptions, the rectangular domain may be partitioned into a collection of horizontal "stripes" with heights $h_i \leq A$ with the property that the total area of each stripe is a multiple of A and that the sum of the stripe heights is M [Mar96]. Heights $h \leq M$ with the property that hN (the area of the corresponding stripe) is a multiple of A will be termed "knapsack-valid"; observe that $min\{A, M\}$ is always a knapsack-valid height. Both the knapsack and GA approaches that we now discuss utilize stripe heights as their primary variables, w_i. That is, w_1 corresponds to the height of the first stripe, etc. In this case, each w_i corresponds to a collection of subproblem solutions that fill a stripe of height given by the value of w_i. These are generated by considering penalty weights λ constructed as follows: cells not in the stripe are assigned high penalty weights, and cells within the stripe are initially assigned weights that force assignment of the leftmost cells to the initial block of variables associated with the stripe, and then for the second block of variables the weights are set to force assignments to adjacent unassigned cells remaining in the stripe, etc.

Before detailing the knapsack approach, we observe that even the single parameter family corresponding to the case of $M = N = P$ yields graph partition problems that are extremely difficult for commonly used techniques. To provide some insight into why this is true, note that the symmetry of the problem implies that in any optimal solution each component may be assigned an arbitrary distinct index p. This implies that there are at least $N!$ algebraically distinct optimal solutions. While this property is advantageous for GA's and other local search procedures, in branch-and-bound procedures it requires the consideration of enormous numbers of intermediate nodes in the tree in order to prove optimality. An additional disadvantage in the case of branch-and-bound applied to a standard equivalent linear integer programming formulation of (4) is that an optimal solution of the initial linear programming relaxation is obtained by setting all $x_i^p = 1/N$. To see this, note that for $N \geq 2$, the objective value of this feasible solution will match the linear program's lower bound of 0 (this is true for any term-wise equivalent linear formulation of the QAP, such as that obtained by replacing each quadratic objective term that has a non-zero coefficient by a non-negative linear variable $y(i, p, j, p')$ subject to the added constraint $y(i, p, j, p') \geq x_i^p + x_j^{p'} - 1$). Branch-and-bound can be expected to behave poorly when there is a large gap between the optimal value of the initial linear programming relaxation and the true optimal value (which in this case is $O(N^{3/2})$). Test runs with a variety of strategies using the well-known CPLEX commercial mixed-integer-programming branch-and-bound code verified that even with a good branching strategy on the very small problem corresponding to N=5, the branch-and-bound method was unable to prove optimality after running for 6000 seconds and generating more than 45,000 nodes in the tree. (In contrast, the knapsack approach solved to optimality problems with N as large as 1000 in less than 1 second.)

The knapsack approach was motivated by error bound results [CM96b] for the special case $P = M \geq N$. These error bound results demonstrated the asymptotic optimality of solutions corresponding to a particular stripe decomposition. Specifically, it was shown that stripes of optimal or near-optimal height could be employed in a partition of the rows of the domain, and that these stripes could be filled with optimal or near-optimal shapes. The result of coordinating subproblem solutions in this way was to produce a feasible solution with objective value v. Letting δ denote the relative distance of this solution from the lower bound, i.e., $\delta = (v - LB)/LB$, where LB is the lower bound, we showed

$$\delta < 1\left\lceil 2\sqrt{N} \right\rceil. \tag{7}$$

Thus, as $N \to \infty$, this relative gap tends to 0, and we say that this family of solutions is asymptotically optimal. (See [CM96a] for analogous results for more general domains.) While this result establishes the quality of a particular stripe decomposition under certain assumptions on the parameters of the problem, the knapsack approach assumes only that P divides MN and determines a stripe decomposition that is optimal with respect to a particular technique for assigning cells within stripes. This is done by computing for each knapsack-valid stripe height the total perimeter of the shapes obtained by filling the stripe in a column-wise fashion from left to right. The corresponding shapes can be considered as optimal solutions of modified subproblems in which Lagrangian penalty terms are added to the objective to force assignments to the leftmost unassigned cells within the stripe. The corresponding total perimeter for the filled stripe is used as the objective coefficient associated with this stripe height, and coordination of stripe heights for this class of problems may be accomplished via a knapsack problem that minimizes the corresponding objective function subject to the single constraint that the the stripe heights add to M.

Algebraically, assuming n valid heights and letting h_i denote a valid stripe height and c_i its associated perimeter contribution, this knapsack problem is:

$$\min \sum_{i=1}^{n} c_i y_i$$
$$s.t. \begin{cases} \sum_{i=1}^{n} h_i y_i = M \\ y_i \in \mathbb{N} \qquad\qquad i = 1 \ldots n \end{cases} \tag{8}$$

Figure 1 shows the relative distance from the lower bound of the best stripe form partition computed by the knapsack method (denoted as MSP for minimum stripe partition) for a set of problems of the form MP($10N \times 10N$, N); here the area assigned to each processor is $100N$ and since the

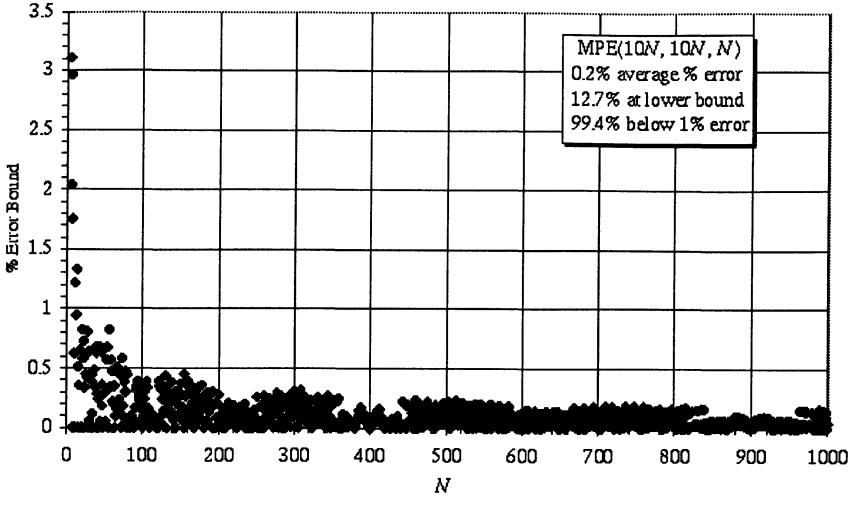


FIG. 1. *Distance from Lower Bound of Knapsack Solutions.*

number of processors is 10 times smaller than each dimension of the grid, the error bound (7) does not apply. However, one can clearly see that the resulting partitions are of excellent quality. For example, the solution obtained for the $10,000 \times 10,000$ grid partitioned among 1000 processors was within 0.042% of the lower bound.

The MSP algorithm is very fast; this is due to the fact that stripe filling is linear in the size of the grid, and that the knapsack problem is of relatively small dimension. Table 1 gives the results of comparisons between our approach and two other popular (and well established) graph partitioning methods: recursive spectral bisection (RSB) and the Geometric Mesh Partitioner (GEOMETRIC). For RSB, we used the Chaco package [HL95] that was provided to us by B. Hendrickson and R. Leland. Chaco is written entirely in ANSI C. We obtained a version of the Geometric Mesh Partitioner from the ftp site indicated in [GMT95]. It is written in MATLAB which helps explain the long response times for some of the larger problems in our test-suite. MSP is written in FORTRAN 77. All programs run on a Sun SPARC-Station 20 workstation. The column labeled Gap gives the relative distance (in percentage terms) from lower bound. (Note that since the lower bound is not necessarily the optimal value, the distance from optimality may be much less.)

Note that neither RSB nor GEOMETRIC could solve the first problem because these recursive methods require the number of partitions to be a power of two. For the last problem, RSB and GEOMETRIC ran out of memory while trying to construct the adjacency matrix of the graph. In those cases in which all of the algorithms produced solutions, the solution

TABLE 1
Comparison of Spectral Bisection, Geometric and MSP.

PROBLEM		RSB		GEOMETRIC		MSP	
$M \times N$	P	Time	Gap%	Time	Gap%	Time	Gap%
7 x 7	7	-	-	-	-	0.01	0.00
32 x 31	8	1.8	6.52	43.6	5.43	0.01	1.09
32 x 30	64	3.0	6.25	90.4	6.25	0.01	0.00
100 x 100	8	9.0	9.33	111.0	7.39	0.04	5.63
128 x 128	128	85.5	14.13	539.9	7.13	0.04	1.63
256 x 256	256	227.8	13.25	3304.2	4.15	0.09	0.00
512 x 512	512	-	-	-	-	0.25	0.14

quality (as measured by the relative gap) of MSP was markedly superior. (We observe here that these test problems were also run on the widely-used METIS graph partitioner, which employs multi-level heuristics, and the GA to be described below. The quality of the METIS solutions was comparable to that of RSB, while the GA produced solutions nearly identical to those of MSP, but required run times comparable to RSB.) These results show that by using well-chosen building blocks corresponding to groups of subproblem solutions and by appropriately coordinating these groups either by the knapsack or GA method, it is possible to substantially outperform state-of-the-art graph partitioning codes, which do not take advantage of problem structure. (Details of additional computational tests and comparisons are given in [Mar96].)

However, it is difficult to generalize this knapsack coordination approach to handle cases in which P does not divide MN or cases of non-rectangular domains. In the next section, we consider a more general approach based on utilizing GA's for the coordination of stripe heights. This GA approach produces essentially the same solutions as MSP for the class of rectangular domains to which MSP applies, but in addition has produced excellent results over a broad range of large-scale problems.

4. Snake decomposition for general domains.

4.1. Snake decomposition . We now describe how stripe decomposition can be modified in order to handle general domains. Figure 2 shows the solution produced by applying a GA based on this generalized approach to an discretized ellipsoidal domain. We need not assume that P divides the area of the grid and we will relax the requirement of dealing with "knapsack-valid" stripe heights. Observe that a general 5-point grid domain can be enclosed within its "rectangular hull", i.e., the smallest rectangle enclosing all of the grid. *Snake decomposition* extends stripe decomposition by accepting as input a partition of the rows of the rectangular hull of the grid, and then "snaking" through the domain, filling the columns of these stripes with processor indices top to bottom, column after column (within

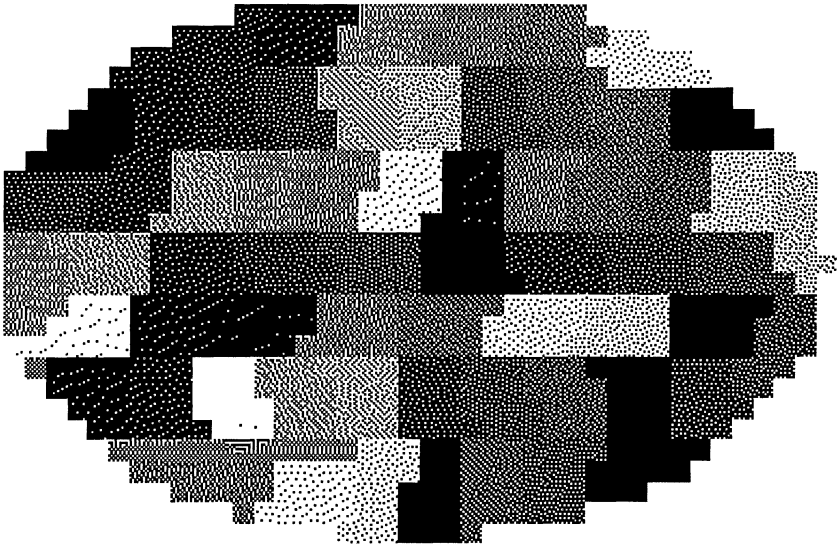


FIG. 2. *Partition of an ellipsoidal domain among 64 procs.*

each stripe) first going left to right, then right to left for the next stripe. (As we will discuss below, this is essentially a procedure for transforming a genotype (comprised of an array of stripe heights) into a phenotype (a set of assignments of cells to processors). Of course, alternative such transformation procedures are possible. For example, the assignments could be done row-wise (with bounds on row lengths related to optimal widths), allowing each component to "spill" into the next stripe.) Snake decomposition differs from stripe decomposition in the sense that stripes that cannot be filled with an integer number of components are permitted and are handled by allowing the fill procedure to "overflow" to the next stripe so that stripes are interdependent. A component corresponding to such an overflow can be thought of as an optimal solution to a subproblem in which there is a hierarchy of penalties added to the original objective: very large penalties for cells already assigned or not in the region spanned by the two stripes under consideration and smaller penalties for assignments to appropriate cells of the second stripe. Once a partition of the rows of the rectangular hull of the grid is given, the `snake` process assigns the cells of the grid to the available processors in linear time ($O(|\mathcal{G}|)$). However, because the overflow process blurs the boundaries between stripes, the determination of good partitions becomes more difficult because it is no longer possible to associate a unique perimeter with a stripe height as was the case with the knapsack approach. Since an objective function analogous to the one in the knapsack problem that provides total perimeter as a function of stripe heights cannot be constructed for these more general cases, we consider

more general mechanisms for coordinating stripes. In the next section, we describe multi-coordination techniques, i.e., procedures in which multiple coordination steps are performed in parallel, based on the use of genetic algorithms applied to high-level encodings.

4.2. Multi-coordination via a genetic algorithm. The initial GA that we considered for this problem class used an encoding in which each individual was represented by a vector of optimal shape indices [YM92]. That is, an allele corresponded to a particular optimal shape chosen from a library of optimal shapes, and the procedure for cell assignment was designed to construct an assignment similar to the one corresponding to that optimal shape. This approach was similar to the Grouping GA approach of [Fal94]. However, we have now developed a higher-level GA in which the alleles correspond to stripe heights, which can be regarded as collections of subproblem solutions. This higher-level approach is faster and has provided better results for our collection of test problems.

To evaluate the "fitness" of an individual produced by crossover and mutation we first "repair" the height values so that they sum to the height of the rectangular hull (at present this is done by changing each allele as little as possible, but other alternatives, such as directing changes toward the set of optimal heights, are also under consideration). We then employ a snake process to generate a feasible solution (optionally followed by a local search starting at that feasible solution). We use a distributed and asynchronous GA, termed DGA, that employs the *island model* of computations [MSB91, D. LEVINE95] where the algorithm spawns a number of processes, each of which is a GA running independently and asynchronously of the others

Each island GA process is equipped with a communication mechanism that is essentially nothing more than simple send and receive calls and a broadcast feature available in most message passing systems that allows it to send or receive incoming individuals according to certain migration criteria that are based on island load (population density of each island). The population in each island can vary (as the DNA length of each individual can, since an individual is an array of integers adding up to the number of rows in the rectangular hull of the grid) but cannot exceed a population limit. Depopulated islands attract well-fit immigrants from other islands.

To help prevent premature convergence phenomena due to the appearance of "super-individuals" early on in the evolutionary process, "age" is utilized to eliminate from the population any individual that has reached its life-limit (which is a random variable following the normal distribution with a mean that is directly proportional to its fitness value).

Each island process uses 1-point crossover with a crossover rate of 0.8%, it has a mutation rate of 0.1% and a maximum population of 16 individuals. A Roullete-wheel strategy was used for selecting the mating individuals. Since DGA follows the *steady-state* approach, approximately

70% of the population is considered for mating at each iteration. Finally, when an individual is born, if there is an empty slot in the island, it is inserted there, else it replaces the worst individual in the population. Migration occurs when an island's population is less than half the population of another island, and the populated island's best individuals have stayed in their home island for at least 3 years (iterations). DGA was allowed to run for 20 generations. It was implemented in C and uses the PVM 3.3.10 message passing interface for all communication required [GBD+94]. DGA runs on a Cluster of Workstations (COW) available at the Computer Sciences Dept. at the University of Wisconsin - Madison. Each workstation is a Sun SPARC-Server 20 running at 66MHz, and comes equipped with 64MB of RAM. We used an 8 node partition of the cluster for our experiments. We tested DGA against RSB and RSQ (Recursive Spectral Quadrisection) as implemented in the Chaco package from Sandia National Laboratories, as described earlier. The results for some non-rectangular domains (see table 2, in which SIZE denotes the number of cells of the domain) are shown in table 3 and Gap denotes the relative distance from lower bound. (It should be noted that the quality of the lower bound from the theory of optimal shapes is not as good for non-rectangular domains as for rectangular grids, since the theory does not take into account that shapes containing portions of the irregular boundary will generally have non-minimal perimeter. Hence, the gaps for DGA do not necessarily mean that its solutions are far from optimal.)

TABLE 2
Sizes of Non-rectangular Test Problems.

PROBLEM	SIZE
circle	7800
torus	7696
diamond	4019
ellipse	823

TABLE 3
Spectral Methods vs. DGA on non-rectangular grids.

PROBLEM		RSB		RSQ		DGA (8 procs)	
Shape	P	Time	Gap%	Time	Gap%	Time	Gap%
circle	16	23.3	24.44	9.1	21.80	20.29	8.47
circle	64	34.7	16.87	14.5	28.34	17.5	5.87
ellipse	16	2.3	10.83	1.4	13.33	6.4	8.33
ellipse	64	3.5	5.16	2.2	15.10	7.5	5.56
torus	16	27.3	28.97	12.5	32.67	16.7	11.50
torus	64	36.5	22.86	18.5	34.3	13.8	11.00
diamond	16	14.0	38.67	6.5	35.74	10.4	16.40
diamond	64	18.7	29.78	9.0	28.80	14.7	13.37

It is clear that DGA can find significantly better partitions than RSB or RSQ for uniform 5-point (regular or irregular boundary) grids, and that it compares well with them in response time as well. DGA was also compared against the widely-used METIS mesh partitioner, which uses a variety of multilevel heuristics. The quality of solutions produced by METIS was comparable to that RSB and RSQ and therefore had an optimality gap typically 2 to 3 times poorer than that of DGA on the non-rectangular grids. (For rectangular grids, the difference in performance was much more dramatic, as may be seen from Table 1, noting that the quality of the DGA solutions was essentially the same as that of MSB.)

5. Directions for further research. We plan to investigate the augmentation of the **snake** procedure for fitness evaluation by a post-processing local search mechanism. Promising candidates include variants of Kenighan-Lin [KL70] tailored to focus on boundary cells [KK95], tabu search, and simulated annealing. In addition, we will consider methods involving more general swap cycles (as opposed to pairwise swaps) based on linear network approximations to the QAP and focused on boundary cells of components with poor perimeter and neighboring components. Finally, the **snake** process will be generalized to allow the user to specify a range of acceptable processor areas as opposed to strict balancing constraints. The user will thus be allowed to assess the corresponding tradeoff between balance and communications cost in the original application.

An interesting aspect of the encoding in terms of stripe heights is that the fitness computation requires the application of **snake** (optionally followed by a local search), which can be time-consuming for large problems. Under these circumstances the investigation of GA variants by means of virtual GA's [Gre95] (in which fitness evaluations are replaced by statistical models of fitness) offers the possibility of streamlining the GA construction process. Our statistical models could extend the approach of [Gre95] by taking advantage of data on quality indices for slices or slice "overflow" at breakpoints.

The approach described above for two-dimensional graph partitioning can be readily extended to three dimensions. The slices in that case are determined by planes orthogonal to one of the axes. In fact, the approach of fitting together, via multi-coordination techniques such as GA's, good or optimal solutions to subproblems, applies to a broad range of areas. In [YM92] we reported the successful application of a similar decomposition approach to data partitioning problems arising from parallel database design. This problem class has the same constraints as (4) but an objective function that represents the minimization of communication with a centralized query server as opposed to the decentralized communication represented by the minimum perimeter problem. Note that while obtaining true optimal solutions to subproblems such as minimal perimeter and surface area problems is of theoretic interest, computationally it is suffi-

cient to simply generate good subproblem solutions, since these objects only play the role of starting points in a multi-stage coordination process that includes the capability of repairing or modifying subproblem solutions. Thus, heuristics such as GA's may be utilized for the subproblems (see [PJ94]) as well as for the coordination step. Extensions of the results to non-uniform grids, triangulations, other data partitioning problems arising from parallel database design, and other types of fixed-charge networks provide promising areas for further research for this approach. In multi-commodity fixed-charge networks, for example, the subproblems would be single-commodity subproblems corresponding to various approximations of the the fixed-charges. Recent results in telecommunications network design [Dav97] and pharmaceutical design [Har94] demonstrate that this paradigm of utilizing subproblem solutions as building blocks within the context of genetic algorithms is very effective in those problem domains as well. By focusing on the "island" GA model with appropriately designed high-level problem representations to reduce communication costs and well-designed procedures for subproblem multi-coordination, we anticipate that the evolutionary paradigm of genetic algorithms will prove to be a very effective technique for solving structured large-scale combinatorial optimization problems in distributed computing environments.

REFERENCES

[CM96a] I.T. Christou and R.R. Meyer. Optimal and asymptotically optimal equi-partition of rectangular domains via stripe decomposition. In H. Fischer, B. Riedmuller, and S. Schaffler, editors, *Applied Mathematics and Parallel Computing - Festschrift for Klaus Ritter*, pages 77–96. Physica-Verlag, 1996.

[CM96b] I.T. Christou and R.R. Meyer. Optimal equi-partition of rectangular domains for parallel computation. *Journal of Global Optimization*, 8:15–34, January 1996.

[Dav97] L.D. Davis. Telecommunication network optimization with genetic algorithms: A decade of progress. In *Evolutionary Algorithms*. Springer-Verlag, 1997. To appear.

[D. LEVINE95] D. LEVINE. *User's Guide to the PGAPack Parallel Genetic Algorithm Library Version 0.2.* Argonne National Laboratory, June 1995.

[Fal94] Emanuel Falkenauer. A new representation and operations for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Laboratory, 1994.

[GMT95] J.R. Gilbert, G.L. Miller, and S.H. Teng. Geometric mesh partitioning: Implementation and experiments. In *Proceedings of the 9th International Symposium on Parallel Processing*, pages 418–427, 1995.

[Gre95] J.J. Grefenstette. Virtual genetic algorithms: First results. Technical Report AIC-95-013, Navy Center for Applied Research in AI, 1995.

[Har94] W.E. Hart. *Adaptive global optimization with local search.* PhD thesis, University of California, San Diego, 1994.

[HL95] B. Hendrickson and R. Leland. *The Chaco User's Guide Version 2.0.* Sandia National Laboratories, July 1995.

[KK95] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report 95-064, Department of Computer Science, University of Minnesota, 1995.

[KL70] B.W. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *Bell Systems Tech. Journal*, pages 291–308, February 1970.

[Mar96] W. Martin. Fast equi-partitioning of rectangular domains using stripe decomposition. Discrete Applied Mathematics, 82 (1998) 193-207.

[MSB91] H. Muhlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In R. Belew and L. Booker, editors, *Proceedings of the Fourth Intl. Conference on Genetic Algorithms*, pages 45–52. Morgan Kaufmann Publishers, Los Altos, CA, 1991.

[PJ94] Mitchell A. Potter and Kenneth De Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings of the Third International Conference on Parallel Problem Solving from Nature.* Springer-Verlag, 1994.

[PRW93] P.M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems.* American Mathematical Society, 1993.

[Sch89] R.J. Schalkoff. *Digital Image Processing and Computer Vision.* John Wiley & Sons, 1989.

[Str89] J. Strikwerda. *Finite Difference Schemes and Partial Differential Equations.* Wadsworth & Brooks, 1989.

[YM92] J. Yackel and R.R. Meyer. Optimal tilings for parallel database design. In P.M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 293–309. North - Holland, 1992.

TELECOMMUNICATION NETWORK OPTIMIZATION WITH GENETIC ALGORITHMS: A DECADE OF PRACTICE

LAWRENCE DAVIS*

I. Introduction. This paper describes five applications of genetic algorithm technology to problems in telecommunication network design, configuration, and message routing. The applications were carried out over the past ten years. (It seems quite appropriate somehow that the date of the conference that this volume chronicles is almost exactly ten years after the date on which I began work on my first telecommunication network problem.)

This paper focuses on the evolution of our approaches to these problems rather than the specifics of the problems themselves. We will concentrate on three aspects of the solutions: the *representation* used, the nature and workings of the *evaluation function*, and the inclusion of *domain knowledge* in the genetic algorithms.

II. The three themes. This paper traces the development of approaches to telecommunication network optimization problems by describing five applications from a series I have worked on in collaboration with a number of talented and ingenious genetic algorithm and telecommunication specialists. Before describing these five projects, I would like to note several of the higher–level lessons that have become apparent in carrying out the work that this paper describes.

The *representation* used in a genetic algorithm is critical to the speed of the algorithm and the quality of its solutions. It is probably best to adopt a representation that is like those used by human experts in the domain. A good representation facilitates the use of operators, allows good evaluation functions to be created, and responds well to the knowledge–based and syntactic operators that will be used in optimization. A bad representation responds badly to the operators in the operator set, interacts badly with the evaluation function, and hinders the use of domain–based operators.

The *evaluation function* provides a measure of chromosome fitness to the genetic algorithm, but it can be made to do much more. It should be constructed to assist the genetic algorithm in moving toward good solutions, even at the beginning of the search process when solution quality is bad. It should make the information it has gleaned available where possible. In my view, a good evaluation function will incorporate whatever domain–based knowledge is available and rapidly computable in constructing a solution from the chromosomal encoding. Often, a good evaluation function can be used to cache information gained during the evaluation of a

*Tica Associates, 36 Low Street, Newbury, MA 01951. E–mail: davis@tica.com.

chromosome, for use in deciding what operators to apply to children of that chromosome. Finally, where possible, an evaluation function that rapidly repairs violations of constraints rather than penalizing for such violations produces much faster optimization.

For a real–world application, *domain knowledge* ought to be incorporated into the genetic algorithm whenever feasible. (The result almost always improves the genetic algorithm.) The local heuristics that are typically applied to optimization problems of the sort considered in this paper tend to exploit features of the topology of the solution space. Domain–based heuristics form the basis of good operators, construction principles, and repair heuristics. The global search strategy of evolutionary algorithms allows the algorithm to consider candidates for such heuristics over wide areas of the search space

III. Problem 1: Assigning bandwidths to a network of fixed topology. In 1986, as an employee of Bolt Beranek and Newman Inc, I became involved in solving a problem frequently encountered by network analysts in BBN Communications' Network Analysis group. The problem was one of assigning bandwidths to a network whose topology had already been determined. Bandwidth assignment took place as the final stage in a network design process that used queuing models of routing and network behavior in order to assess a design's satisfaction of hard and soft performance constraints.

The design process began with the homing of terminals into clusters and clusters into nodes, continued with the determination of a "backbone" network topology consisting of links with high bandwidth that were certain to be capable of carrying all the anticipated network traffic, and ended with the reduction in bandwidth of the backbone links in order to save link costs, while continuing to satisfy the network constraints.

The most interesting part of this process from the network designers' point of view was the determination of the network topology. Most of the designers were mathematicians, and enjoyed working with the graph–theoretic aspects of network design. The least interesting part was the final bandwidth allocation part, for several reasons. One was that the response time of the computers (Lisp Machines) that we used in 1986 was fairly slow; the queuing models took several minutes to run in the hardware environment of that time. Further, the models were stochastic; a designer would modify the network bandwidths, wait for several minutes for the simulation to complete, and attempt to analyze the results, knowing that the results might possibly be quite different if the simulation process were repeated. Finally, the problem was less interesting from a theoretical point of view than that of designing the network topology. For these reasons, the human designers thought that the bandwidth assignment problem would

be the best place to apply genetic algorithms in order to improve the results of the design process.

Susan Coombs of the Network Analysis group and I attacked the problem of assigning bandwidths by using a genetic algorithm. We have reported the results of our research in Coombs and Davis 1987 and Davis and Coombs 1987. The reader is referred to these papers for a more detailed exposition of the problem and our approach to it.

Neither we nor the other network analysts were optimistic about our prospects for success when we began, for two reasons. The first was that there was no agreed–on evaluation function for comparing two designs with different bandwidth assignments. The second was that there could be no fixed evaluation function, as evidenced by the fact that none of the four network design problems solved by the design group during our three–month attack on the problem had identical requirements, and the critical requirements varied from design to design. If there could be no evaluation function, how could a genetic algorithm be made to work?

We rapidly learned that the range of requirements across the set of design problems being solved at the time we were working on the problem was great. Every client wished to save money, but this was the only requirement in common. Some wished the maximum utilization of various links in the network to be below specified thresholds, in order to allow for future growth without network modification. One, a dialup service, required that no message in the network take longer than a certain amount of time to be delivered so that no customer would be unduly charged for connect time. One required that the corporate headquarters be able to deliver all its traffic after the failure of any link in the network. One required that a certain percentage of the traffic in the network be deliverable after the failure of any link in the network. And so on. No set of requirements encountered during the three–month period was a subset of or identical to any other set. Given this diversity in evaluation and feasibility criteria, it seemed unlikely that a genetic algorithm evaluation function could be crafted that would work for all network design problems.

Most of the effort that we expended in working on the bandwidth allocation problem was directed to the issue of design requirements. We spend a great deal of time in determining the extent and nature of the set of requirements that might be possibly be applied to network designs, and in devising genetic algorithm operators that might help to meet those requirements. The result was a twofold approach to the construction of our evaluation function. First, we created a library of network design requirements. Each file in this library contained a method that could be used to compute the degree of violation of the requirement. Each file also contained one or more genetic algorithm operators that could be used to promote satisfaction of the requirement. Second, to optimize the allocation of bandwidths to a specific network design, we selected the set of files that were related to the requirements of that network, added the operators in

those files to the genetic algorithm operator set, and weighted the various requirements as we believed the network designer would weight them in an evaluation function. A good deal of knowledge acquisition, testing, and revision was required to accomplish the task of setting appropriate weights on the various design requirements.

The result of this knowledge acquisition effort was a library of requirements and knowledge–based operators that incorporated much of what we had learned from the analysts in the Network Analysis group. After a good deal of trial and error, we also developed sets of weights on requirements such that our evaluation function (a weighted sum of requirements violations, which was to be minimized) matched the preferences of the network analysts in nearly every case (the cases where preferences did not match tended to be ones in which many requirements were violated, and so it was not as important to be precise about the degree of badness of a solution).

We applied genetic algorithms with evaluation functions and operator sets built from this library to four real network design problems. These were the four problems that were currently being solved by network analysts during the three–month period in which we carried out this research. We used twelve–hour computer runs (approximately the same amount of time that the designers spent in carrying out this task). In each case, the network designer judged the design produced by the genetic algorithm to be superior to that produced by the network designer. In terms of the weighted evaluation function developed as described above, the average improvement over these four design problems was 4%.

The *representation* we used was a list of bandwidths, with one list member for each link in the network. The members of this list were taken from a list of allowable bandwidths. Thus, the representation we used reflected the fact that the problem had highly–constrained solution components. The operators were not allowed to produce values that were not on the list of allowable bandwidths.

The *evaluation function* was created as described above. To evaluate a chromosome, each bandwidth was assigned to its corresponding link and the queuing model was run. Each constraint appropriate to the network being designed was applied, and the constraint penalties were weighted and summed. Some information derived during the run of the model was cached, as noted below, and occasionally a chromosome was "repaired" and evaluated again, as noted below.

A good deal of *domain knowledge* was built into the genetic algorithm. The constraint library had associated operators designed to minimize penalties for that constraint. There were "Lamarckian" features of the evaluation process that noted defects in chromosomes that were only detectable after the chromosome's performance had been simulated, repaired the defects, and re–evaluated the chromosomes. And there were knowledge–based mutation operators.

Two knowledge–based mutation operators used in our system are prob-

ably worthy of note, because they were based on heuristics used extensively by the human designers and because they exploited information derived during a parent's evaluation process. They were *creep–up* and *creep–down.*

When human experts use the heuristics on which these operators were based, their reasoning might be as follows: "I am attempting to satisfy constraints and minimize cost. If I have a link that was under-utilized during simulation, I might be able to save money by decreasing its bandwidth. If I have a link that was over–utilized during simulation, I might be able to minimize constraint violation if I increase its bandwidth. So I'll try decreasing the bandwidth of some underutilized links and increasing the bandwidth of some overutilized links."

There are three interesting things about these heuristics. One is that they often produce the desired effect. As rules of thumb, they have good performance. The second is that they sometimes fail to produce the desired effect. A link that is underutilized might be causing the network routers to behave in a beneficial way. Decreasing the size of the link might cause the routers to route in highly unsatisfactory ways. (And of course the stochastic nature of the simulation process means that a link might appear underutilized while further simulation would show that, on average, it would not be.) The third fact about these heuristics is that if they are the only heuristics that are used, and they are applied to only one link at a time, then they can rapidly lead the designer into a local optimum which cannot be escaped by a single application of either heuristic.

These three facts suggested to us that a genetic algorithm–one that moved link bandwidths up or down based on link utilization, and that had the capability of modifying several bandwidths in a single pass–might be better than a more directed procedure, such as the one human designers used. (The procedure typically followed by the human designers was somewhat akin to opportunistic hill–climbing.)

Our hypothesis turned out to be correct. Our *creep–up* operator increased the bandwidth of links with a probability influenced by the amount of underutilization of that link when the parent was evaluated. Our *creep–down* operator decreased the bandwidth of links with a probability influenced by the amount of overutilization of that link when the parent was evaluated. Our generational genetic algorithm using these operators was capable of modifying the bandwidth of several links at a time, and was able to escape local optima and handle noisy feedback.

The inclusion of these and other heuristics developed by domain experts allowed our genetic algorithm to proceed much faster than a genetic algorithm without them, and to produce better results.

IV. Routing messages in a telecommunications network. In 1988, while still an employee of Bolt Beranek and Newman, Inc., I carried out a series of studies as a consultant to and collaborator with a group of operations research scientists at U S West. This group was headed

by Tony Cox, and later was a primary factor in U S West's co–winning the prestigious 1994 ORSA Prize, awarded by the Operations Research Society of America (now INFORMS) to the company judged best in the world in applying operations research in ways that have profound business impact. One of the first problems we tackled together in a long and highly satisfying collaboration that has continued to the present time was that of routing a group of messages from their origin to destination nodes in a telecommunications network of fixed topology and bandwidth, in order to minimize the penalty from unplaced messages, and in order to make it easy to accommodate future messages. This problem, described in Cox, Davis, and Qiu 1991, was ancillary to the problem of deciding which requests for dedicated bandwidth to accept. The reader is referred to the paper for a fuller description of the problem.

This problem seemed similar to a problem I had worked on in the early 1980s at Texas Instruments. The Texas Instruments problem was to route connections between devices in a semiconductor so as to minimize the surface area used after all routing was completed. The U S West problem involved routing messages through a network. Although many particulars of these two problems were different, they had two features in common. First, when solving these problems, the human experts tended to take the things to be routed (connections or messages), rank them in some order, and then process them in order using local, greedy heuristics. Second, the problems were very difficult, and the solutions obtained were highly dependent on the order in which elements were considered. Placing a single route early in the solution development might block the best placement of many other routes in semiconductor design. Placing a single message early on its favored route might block a number of other messages.

In my experience, these two characteristics are diagnostic of a problem in which an order–based chromosomal *representation* is applicable, and in both cases this is the approach that was (successfully) used in the genetic algorithm. For the message routing case described here, the chromosome representation was a permutation of the list of messages to be routed. All operators were applied to such permutations.

Before beginning a run of the genetic algorithm, the shortest n routes for each chromosome were computed using Dijkstra's algorithm and cached (see Taha 1975 or a graph theory textbook for a description of Dijkstra's algorithm). To evaluate any chromosome, the *evaluation function* set the network to its initial state and then, for each message in the order given on the chromosome, the evaluation function assigned that message its shortest feasible path. After assigning a message, the available bandwidth on each link in the path was appropriately reduced, and the algorithm was applied to the next message on the chromosome. This greedy evaluation function was quite similar to the process a human expert would carry out in attempting to route the set of messages, except that the expert would experiment with only a few, heuristically–derived orderings of the mes-

sages. For example, the messages might be sorted by a human expert in decreasing order of expected profit, or according to some other criterion. The genetic algorithm, in experimenting widely over the space of message orderings, always improved significantly on the performance of such faster heuristics.

The operators used in this genetic algorithm were primarily syntactic. When mutating, they carried out order–based crossover of two parents and scrambled sublists of a single. Thus, in this case there was little *domain knowledge* built into the genetic algorithm. Crossover was a simple, order–based crossover.

The performance of this algorithm was quite good, and was at its comparative best as the size of the networks under consideration increased. The algorithm was compared with LINDO, a commercial optimization package for solving small–to–medium–size problems. The genetic algorithm found solutions in minutes that LINDO required two weeks to find. The principal benefit from this approach, it seemed, was that one could quite rapidly generate feasible solutions that were slower to produce but much better than faster, domain–based heuristics. These solutions were also much faster to produce but only marginally inferior to the optimal solutions generated by LINDO. Thus, the genetic algorithm occupied a useful niche in the space of optimizers between the one–pass optimizers and the global optimizers, and many real–world problems fell into this niche.

V. The single–link problem. In an attempt to understand further the nature of the message routing problem, Tony Cox and his group spent some time considering a problem involving a single link with fixed bandwidth. Messages arrive, randomly generated from a known joint frequency distribution. One can accept or reject each message. If one accepts a message, one must have sufficient bandwidth available to handle it, and one gains a reward equal to a fixed amount plus a constant times the message's length. An accepted message consumes a specified amount of bandwidth in the link from the present time to the message's end. If one rejects a message, one gains no reward, but one will keep the link open in hopes of receiving more profitable messages.

In designing an optimization strategy for this problem, one quickly notes two things: the shorter a message, the more profitable it is per unit of message time to accept it; and it is advantageous to accept more messages if the link is at low levels of utilization. But these observations do not determine a solution to the problem, which is to find a decision criterion that maximizes profit by accepting or rejecting messages based on the current state of the link.

A stochastic dynamic programming solution to this problem had already been implemented at U S West (for a description of this algorithm, see Chiu, Lu, and Cox 1995). On receiving a message, the dynamic programming algorithm broke down the possible future events in the queue into

small detail and, based on the characteristics of the message–generating probability distribution and the number of messages currently in the link, decided whether to accept or reject the new message, summing the expected utilities of the various outcomes. These utilities were computed using the characteristics of the probability distribution. This approach provided a useful, approximate solution to the problem, but the approach was CPU–intensive to run. The group wondered whether a genetic algorithm might produce approximate solutions with much less computation time, and whether the genetic algorithm might be applicable to problems without analytic characterizations.

At that time, in 1991, I had just founded Tica Associates, a consulting practice specializing in optimization, often with the use of genetic algorithms. This was one of the first problems that was worked on at Tica Associates. It seemed that the message acceptance decision could be formulated in terms of deciding what it cost to accept a new message. If a message's expected cost was greater than or equal to its profit, then the message should be rejected. If the message's expected cost was less than its profit, then the message should be accepted. But how would one assess the cost incurred by accepting a message? It was clear that the cost incurred by a message's acceptance depended on how many messages were in the link at the time that the new message arrived, and what the state of the link would be if the message were accepted. It was not clear how to compute this cost precisely, but it seemed that it was not necessary to know how to do this, if the genetic algorithm could be made to evolve an expression of the cost. This is exactly what it was made to do.

The *representation* used for this problem was a chromosome with n fields, where n is the bandwidth of the link. Each chromosome field was a number between zero and a heuristically–derived upper limit. The interpretation of the chromosome was as follows. The number in the first field of the chromosome was the cost incurred by a message per unit of time when the link was empty. The number in the second field was the cost incurred by a message per unit of time when the link already had one message in it. And so on.

The genetic algorithm was generational and elitist. *The evaluation function* was stochastic. In each generation, the evaluation function randomly generated a number of messages from the probability distribution. These messages were stored in order of increasing arrival time. To evaluate a chromosome, the evaluation function took each cached message in order and determined the cost of accepting that message, using the chromosome fields to compute the message's cost. For example, if the message were of length three, and the link currently contained one message ending in one time step, then the cost of the new message was the cost of occupying one unit of the second level of bandwidth, and two units of the first level of bandwidth. These costs were read from the chromosome and summed. If the profit to be obtained from the message were greater than this amount,

the message was accepted. If not, it was rejected.

The effect of this evaluation function was to select for chromosomes encoding cost functions that led to profit. In each generation, all chromosomes were given the same set of messages to process, and their rankings of the population members tended to be correlated with how well those members would handle larger sets of messages generated from the same probability distribution. The size of the message set for each generation was interpolated from a small number at the beginning of the run to a large number at the end. The result returned from a run of the genetic algorithm was the average of fields in the top five chromosomes in the final generation.

Domain knowledge was included in this algorithm in the following way. The operators used were syntactic in nature: arithmetic crossovers, uniform crossovers, creeps, and mutations. After applying operators, however, the chromosome was processed so that it was monotonically–increasing. This procedure was applied based on the observation that the cost of accepting a message when the link is empty must be less than the cost of accepting the same message when the link is full, and so it is plausible to assume that the weights on an optimal chromosome will be monotonically increasing.

The genetic algorithm and the evaluation function were created in a day or so. The results of the early genetic algorithm runs surprised us. They generated solutions that were slightly better than the ones generated by the dynamic programming approach–solutions that we had believed to be nearly optimal. The reason was that the genetic algorithm took as much account of the cost of accepting a message at the end of the message's duration as it did at the beginning of the message's duration, while the dynamic programming approach, for reasons of efficiency, sampled the possibilities at the end of the message's duration in much cruder detail. If it had not done this, the CPU requirements for running the dynamic programming approach over a million messages or so would have been on the order of days rather than hours. (The genetic algorithm required twenty minutes or so to run).

We noted two interesting features of the genetic algorithm after this experiment was completed. One was that the genetic algorithm used no knowledge about the probability distribution that generated the messages, while the dynamic programming approach relied on this knowledge in order to construct its estimate of message utility. Thus, the genetic algorithm could be applied without modification to problems in which the probability distribution was unknown or could not be analytically expressed. Another was that the genetic algorithm could with minor modification be applied to messages arriving at the present time but starting in the future. This change in the problem would have greatly added to the time required to run the dynamic programming algorithm. These facts, together with the much faster run time, suggested that this genetic algorithm technique for evolving a policy that accepted or rejected messages based on evolved measures

of utility might be superior to others we had been using for the case of full networks, but we never carried out the research required to determine whether this was the case.

VI. Survivable network design. In 1992 Tony Cox's group at U S West asked David Orvosh and me at Tica Associates to consider a problem for which exact methods could not be applied in any reasonable amount of time. The problem was to specify the design of a telecommunication network such that all messages in a traffic table were routable, and such that after the failure of any single link in the network, at least a specified proportion of the traffic could be routed.

This problem combined elements of the network design and routing problems discussed earlier in this paper. To demonstrate that a solution is acceptable, one must show that all the traffic can be routed. Thus, solving the problem involves solving a message routing problem. Further, the problem includes the bandwidth assignment problem considered by Susan Coombs and me as described above. Finally, the solution itself must include the specification of a network topology. Thus, this problem included the part of the network design problem solved by human designers before Susan Coombs and I worked on the bandwidth assignment aspect of network design.

The problem is made most difficult by the survivability constraint–the requirement that a proportion of the traffic be routable after the failure of any link in the network. Recall that assessment of network performance in the bandwidth allocation problem could take several minutes. In evaluating a solution to the survivable network problem, one must assess network performance once to determine that the traffic can be routed, and then once for each link in the topology, to determine whether, when that link is eliminated from the network, the required proportion of each entry in the traffic table can be routed. Thus, the number of tests of network performance that might be required to evaluate a chromosome is $n + 1$, where n is the number of links in the topology specified by the chromosome.

One of the network design problems that Susan Coombs and I solved in 1986 involved just this survivability constraint. We had realized then that we would be unable to solve the problem in the twelve hours of CPU time allotted us if each chromosome were evaluated fully, and so we experimented with what we called an "ice age" evaluation procedure. We eliminated consideration of the survivability constraint in general, but every m generations (where m was a parameter of our genetic algorithm) we applied the constraint to every chromosome in the population. This procedure worked, but just barely. The populations had the quite understandable tendency to reduce bandwidth and evolve toward less expensive (and less acceptable) solutions as soon as an ice age generation was over. In some ice age generations, no solution satisfied the survivability constraint, and postprocessing was required to repair the final solution.

For this reason, David Orvosh and I decided in 1992 that we would include the survivability test in each chromosome evaluation. However, we would not carry out an extensive version of the test. Instead, we would run a fast, greedy algorithm that would give an estimate of how much traffic could be routed, and use that as a surrogate for a more nearly optimal analysis of the routability of the messages given a network design. We would, however, allow the genetic algorithm to influence the fast, greedy routing by evolving the order in which the messages were to be routed in the initial topology, and in the various topologies produced by eliminating a single link from the network.

It was clear to us that evaluation of literally–encoded network topologies was likely to waste a good deal of time unless we included a repair mechanism in our evaluation function. For example, a mutation operator might delete a link and disconnect all traffic in a topology that is otherwise quite good. It might be quite expensive to determine that this is the case, but once determined, it is not difficult to carry out a local, greedy repair procedure that produces a legal design. Accordingly, we based our evaluation procedure on the idea that we would take whatever topology and bandwidth assignments the chromosome encoded and do whatever it took in the process of evaluation to guarantee that the result satisfied all the problem constraints. The result was a type of genetic algorithm we have come to call a "Seed, Repair, Replace Genetic Algorithm," or SRR–GA. (We have since applied SRR–GAs to a variety of problems, often to good effect.)

For the survivable network design problem, our chromosome *representation* was tripartite. We relied on a preprocessing of the problem in which the set of all possible links was reduced to a small subset s that was considered to be the set of links to be considered in a solution. This preprocessing step, carried out at U S West, reduced the problem from one of finding a network topology using a subset of all possible links in the network, to one of finding a network topology using the links in s. Given this preprocessing step, part 1 of a chromosome was a list of numbers, one for each member of s. The interpretation of part 1 is that each item in the list is the bandwidth of the corresponding member of s. A zero bandwidth means that the corresponding link does not exist. Any other entry means that the link does exist, and has bandwidth equal to the corresponding part 1 field. Part 2 of a chromosome is a permutation of the list of entries in the traffic table. This permutation is used during chromosome evaluation to guide the routing of traffic over the full network topology. Part 3 of a chromosome is another permutation of the list of entries in the traffic table. This permutation is used during chromosome evaluation to guide the routing of traffic when the survivability test is applied.

The *evaluation function* applied to a three–part chromosome of this type was rather complex. It tested the chromosome to verify that all traffic could be routed, and then it deleted each link in the chromosome in turn to

verify that the required proportion of traffic could be carried by the network if that link failed. Whenever the evaluation function determined that traffic could not be routed, the evaluation function modified, or "repaired," the network to make routing possible. The repair technique was local and greedy. It is specified more fully in the 1993 paper. Here let me just give the flavor of it.

Suppose that we ask the evaluation function to evaluate a chromosome in which part 1 consists of all zeroes, and parts 2 and 3 consist of permutations of the list of traffic. The evaluation function would take the first member of part 2 and find the shortest path that member could take in the existing topology. There would be no such path, since at this time the network has no links. The evaluation function would then add the lowest–cost links to the network with sufficient bandwidth to route the message. It would carry out this procedure for each message in part 2 of the chromosome, adding links and/or bandwidth as required to ensure that the traffic is routable. (If part 1 had had positive entries, the bandwidth consumed by routed messages would be subtracted from the amount of bandwidth available for each link over which the traffic were routed.)

During the survivability test, a similar procedure is carried out for each link in the topology derived from carrying out the first test. First, the link is deleted from the network. Next the available bandwidths of the links are set to the bandwidths derived in carrying out prior tests. Finally, the required proportion of each entry in part 3 of the chromosome is routed, and bandwidth and/or links are added as required to ensure routability (the deleted link may not be added while it is being tested). Any links added in this part of the procedure are also tested for survivability.

This procedure produces a repaired correlate of the original chromosome that is guaranteed to satisfy the problem constraints. The network may well be suboptimal; the repair and testing procedures are fast, greedy heuristics rather than optimal ones. But the genetic algorithm evolves to provide network topologies that are well–suited to the greedy heuristics, and parts 2 and 3 of the chromosome evolve to make the heuristics more and more effective as the population converges.

The result of applying this algorithm is a network design that satisfies the constraints. At the conclusion of the run, the best design found can be optimized in various ways. Each link can be tested for possible bandwidth reduction or elimination, using more intensive algorithms than those used by the genetic algorithm. However, we found that the results produced relatively quickly by the genetic algorithm were fairly close in quality to those achieved by applying these optimization processes to them.

There is an important point to note concerning the representation used in an SRR–GA. The network encoding in part 1 of a chromosome at the start of the evaluation process may be quite different from that derived through the repair mechanism. It is an interesting question whether it is better to place the original chromosome in the population, or to place

the chromosome's repaired correlate in the population. This is an issue that David Orvosh and I have studied in some detail in Orvosh and Davis 1993. Our conclusions were that across a variety of problems in which the repair technique was used, the strategy of always replacing the original chromosome with its repaired correlate and the strategy of never replacing the original chromosome with its repaired correlate were both inferior to a mixed strategy in which the original part 1 was replaced with its repaired correlated with probability .05. The interested reader is referred to Orvosh and Davis 1993 for a fuller discussion of our hypotheses as to why this is the case. At any rate, this mixed approach was superior for the survivable network problem, and we used it here.

VII. Dynamical hierarchical packing. In the past year or so we have carried out two studies in collaboration with Tony Cox (currently with Cox Associates, of Denver, Colorado). These studies centered on the problem of deciding how to upgrade a configuration of switches and switch components in order to minimize costs and accommodate demands that are changing with time. Here we consider our second study, with results that are described more fully in Davis, Cox, Lu, Kuehner, and Orvosh 1997.

The problem that we solved concerned switch capacity planning for wireless telephony. The client needed to add to a configuration of switch modules and switch components in order to satisfy demand over a number of time periods. The problem was to determine how many of each of four types of switch components should be added to the configuration at each time step, so that all demands at that time step were satisfied, and so that the cost of the components and the switch modules required to contain them were minimized. Costs were discounted, so that expenses incurred farther in the future counted less than expenses incurred earlier. The most interesting feature of this problem was the requirement that a relatively expensive switch module must be purchased for every nine switch components. Thus, there was a hierarchical packing aspect to the problem, which was christened the "dynamic, hierarchical packing problem" by Tony Cox.

The problem could be formulated in a spreadsheet and solved with the spreadsheet solver (a subroutine that used a branch and bound heuristic), but response times were very slow. A realistic problem with ten time steps could require several hours, and the solution provided by the solver was generally suboptimal. A realistic problem with twenty time steps could require thirty or more hours, and the solution was generally suboptimal. In order for the client to consider alternate designs and alternate assumptions, it was important to have more rapid feedback, and so Tony Cox asked us at Tica Associates to apply a genetic algorithm to the problem. Although this problem is quite different from the survivable network problem just discussed, we were able to achieve good results using an SRR–GA.

The *representation* that we used for the chromosomes was a two–

dimensional matrix. There were four columns, one for each type of switch component. There were as many rows as there were time steps in the problem.

The *evaluation function* carried out extensive repairs when evaluating a chromosome. Before the run began, the genetic algorithm cached the most cost–effective switch module type for each type of demand. The evaluation function carried out three repair passes over the chromosome and evaluated the result.

On pass 1, the evaluation function noted any point at which a column entry was less than the entry on the preceding row, and increased the entry in order to enforce montonicity. (We enforced monotonicity because switch modules, once purchased, were assumed to be in place forever, and deleting them from the configuration would not be meaningful for this problem). On pass 1, the evaluation function also noted any demand type that was not satisfied by the configuration represented at a time step, and modified the configuration by adding units of the most cost–effective switch component to satisfy that demand. Pass 1 constituted a repairing of the chromosome; at the conclusion of pass 1, the chromosome represented a feasible solution to the problem.

Pass 2 carried out a trimming procedure. It began with the final time step and proceeded step by step back to the first time step. At each step, the evaluation function reduced the amount of switch components as long as such reductions continued to satisfy the demand for that time step. The evaluation function also enforced monotonicity by reducing any fields that were greater than the corresponding field in the next time step. At the conclusion of Pass 2, it was possible that the solution was infeasible.

Pass 3 was a repeat of the Pass 1 procedure. Pass 3 repaired any infeasibilities introduced by the Pass 2 trimming procedure. At the conclusion of Pass 3, the chromosome again represented a feasible solution to the problem.

The evaluation of a chromosome was the cost of the repaired correlate of the chromosome after this three–pass procedure had been applied. Thus, the cost was the sum, for each time step, of the discounted cost of new switch components, together with the discounted cost of any new switch modules required to house them.

A good deal of *domain knowledge* was included in the design of this SRR–GA. The repair procedure invoked during evaluation used quick, greedy procedures for satisfying unsatisfied demand. We enforced monotonicity because we had observed that for this particular type of hierarchical, dynamic packing problem monotonicity would be a property of an optimal solution. We trimmed the chromosome because we observed that this procedure would be very fast to execute, and it would speed up the genetic algorithm tremendously (rather than finding reductions in field values by creeping or mutating, the reductions were found quickly and greedily, and were achieved immediately).

In our paper we considered several variations on the problem, and proposed changes to the evaluation function and operator set that were based on our analysis of the problem modifications. The conclusion we reached in the paper was that an SRR–GA was a very effective solution technique for this type of problem (the SRR–GA found solutions in thirteen seconds that took hours for the solver to find, and solution quality was at least as good as that of the solver). We also concluded that the performance of the SRR–GA could be maintained when the problem was varied, if appropriate domain–based heuristics were added to the SRR–GA to track these changes.

VIII. Conclusions. This paper has described five selected applications of genetic algorithm technology to a series of problems in the design, configuration, and routing of telecommunication networks. The problems we tackled grew more difficult over the decade in which these applications were carried out, partly because the speed of the computers available to us increased by two orders of magnitude, and partly because we learned a great deal about applying genetic algorithms to combinatorial optimization problems, and so were able to tackle more difficult problems successfully. One goal here has been to acquaint the reader with the techniques used in solving these very important problems and, by describing our approaches in a longitudinal way, to stress three points: the *representation* used in solving a hard real–world problem can be critical to the success or failure of the algorithm; the *evaluation function* can often track information that will be valuable to subsequent operators or repair operations; when working with real problems, one maximizes one's chances of success when one incorporates as much existing *domain knowledge* into one's algorithm as is practicable.

IX. Acknowledgements. This paper has been greatly improved by the comments of Tony Cox and Susan Coombs. Steve Chiu and Leonard Lu provided helpful references. The projects described here were extensive collaborations between the authors of the multi–author papers cited below.

REFERENCES

[1] Chiu, S. Y., Lu, Leonard, and Cox, Tony (1995), *Optimal Access Control for Broadband Services: Stochastic Knapsack with Advanced Information*, European Journal of Operations Research **89, No. 1**, pp. 127–134.

[2] Coombs, Susan and Davis, Lawrence (1987), *Genetic Algorithms and Communication Link Speed Design: Constraints and Operators*, Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms, John J. Grefenstette (ed), Hillsdale: Lawrence Erlbaum Associates.

[3] Cox, Tony, Davis, Lawrence, and Qiu, Yuping (1991), *Dynamic Anticipatory Routing*, Handbook of Genetic Algorithms, Lawrence Davis (ed), New York: Van Nostrand Reinhold.

[4] DAVIS, LAWRENCE, AND COOMBS, SUSAN (1987), *Genetic Algorithms and Communication Link Speed Design: Theoretical Considerations*, Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms, John J. Grefenstette (ed), Hillsdale: Lawrence Erlbaum Associates.

[5] DAVIS, LAWRENCE, COX, TONY, KUEHNER, WARREN, LU, LEONARD, AND ORVOSH, DAVID (TO APPEAR), *Dynamic Hierarchical Packing of Wireless Switches Using a Seed, Repair, and Replace (SRR) Genetic Algorithm*, Heuristics, Kluwer Academic Publishers.

[6] DAVIS, LAWRENCE, ORVOSH, DAVID, COX, ANTHONY, AND QIU, YUPING (1993), *A Genetic Algorithm for Survivable Network Design*, Proceedings of the Fifth International Conference on Genetic Algorithms, Stephanie Forrest (ed), San Mateo: Morgan Kaufmann.

[7] DAVIS, LAWRENCE AND ORVOSH, DAVID (1993), *Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints*, Proceedings of the Fifth International Conference on Genetic Algorithms, Stephanie Forrest (ed), San Mateo: Morgan Kaufmann.

[8] TAHA, H. A. (1975), *Integer Programming, Theory, Applications, and Computation*, Academic Press.

USING EVOLUTIONARY ALGORITHMS TO SEARCH FOR CONTROL PARAMETERS IN A NONLINEAR PARTIAL DIFFERENTIAL EQUATION

ROGENE M. EICHLER WEST[†‡*], ERIK DE SCHUTTER[*], AND GEORGE L. WILCOX[‡]

0. Overview. Many physical systems of interest to scientists and engineers can be modeled using a partial differential equation extended along the dimensions of time and space. These equations are typically nonlinear with real-valued parameters that control the classes of behaviors that the model is able to produce. Unfortunately, these control parameters are often difficult to measure in the physical system. Consequently, the first task in developing a model is usually to search for appropriate parameter values. In a high dimensional system, this task potentially requires a prohibitive number of evaluations and it may be impossible or inappropriate to select a unique solution.

We have applied evolutionary algorithms (EAs) to the problem of parameter selection in models of biologically realistic neurons. Our objective was not to find the "best" solution, but rather we sought to produce the manifold of high fitness solutions that best accounts for biological variability. The search space was high dimensional (> 100) and each function evaluation required from one minute to several hours of CPU time on high performance computers. Using this model and our goals as an example, we will: 1) review the problem from the neuroscience perspective; 2) discuss high performance computing aspects of the problem; 3) examine the suitability of EAs for the efficient optimization of this class of problems; and 4) describe and justify the specific EA implementation used to solve this problem.

1. The problem in neuroscience. Neurons are the fundamental units responsible for the transmittal and transduction of information in the brain. Information is transmitted between neurons via diffusable chemicals. The chemical signals from many convergent presynaptic neurons are spatiotemporally integrated and transduced into an electrical response by the postsynaptic neuron (Figure 1a). If the electrical response is sufficiently large, then the postsynaptic neuron will propagate a momentary event and release its own chemical transmitters, carrying the signal to the next set of neurons in the network. These suprathreshold electrical events are referred to as action potentials, bursts, or spikes.

[*] Born-Bunge Foundation, University of Antwerp - UIA, B2610 Antwerp, Belgium.
[†] Division of Biology, 216-76, Caltech, Pasadena, CA, 91125 USA.
[‡] Minnesota Supercomputer Institute, University of Minnesota, MPLS, MN 55455, USA.
Corresponding author: rogene@bbf.uia.ac.be.

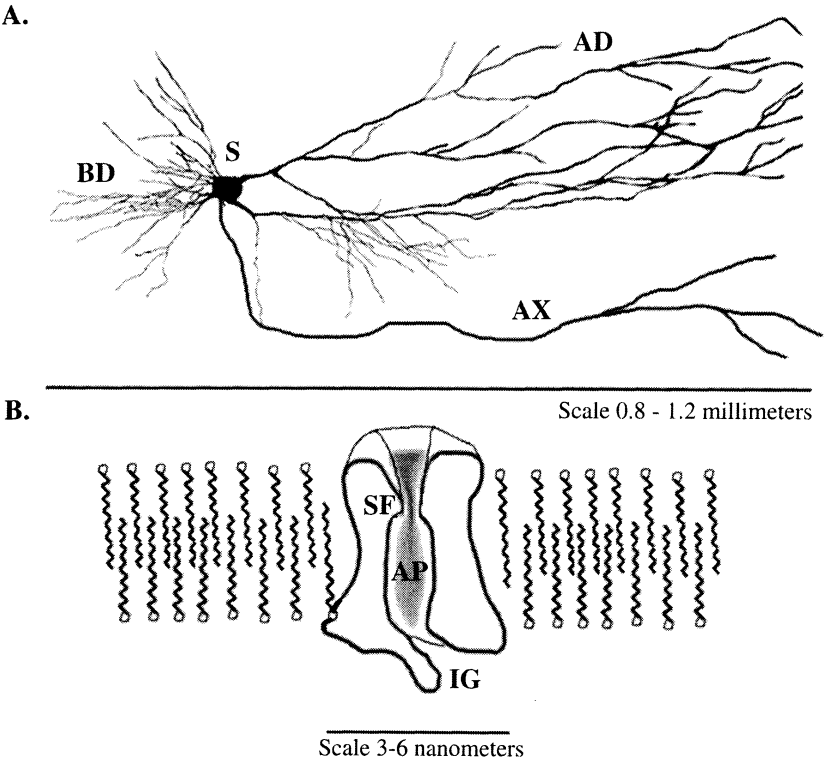


FIG. 1. *(A) CA3 hippocampal neuron morphology digitized from an experimental preparation. Presynaptic neurons transmit chemical signals onto receptor sites in the basal and apical dendrites. If the spatiotemporally propagated electrical signal becomes sufficiently depolarized in the region where the axon extends away from the soma (the axon hillock), action potentials are generated and transmitted via the axon to the next set of neurons in the network. S, soma; BD, basal dendrites; AD, apical dendrites; AX, axon. (B) VGCD channels are distributed throughout the lipid bilayer membrane of the neuron. Electrochemical conditions induce conformational changes in the three dimensional structure of the protein channel. Some conformations are more conductive to the flow of ions by, for example, releasing an inactivation gate blocking the aqueous pore. A region of charged animo acid residues at the mouth of the channel serves as a filter for ions of a particular charge or size. AP, aqueous pore; SF, selectivity filter; IG, inactivation gate.*

How information is encoded in the rate and pattern of electrical signaling events in single neurons and populations of neurons is currently the subject of hot debate among neuroscientists (Ferster and Spruston 1995). Interpretations of the *neural code* have been proposed for many systems, for example: the representation of faces in monkeys (Abbott et al 1996); the force exerted in a voluntary motor behavior (Monster and Chan 1977); sensory information in the cricket cercal system (Theunissen et al. 1996); the representation of odors (Laurent 1996); and cognitive encoding of di-

rection in the motor cortex (Georgopoulos et al. 1993). However, it is fair to say that at this time, *we don't really know how neurons encode information* and the answer to this question is at the heart of understanding brain function.

Spike production in neurons is governed by a class of membrane-spanning proteins known as voltage-gated and/or calcium-dependent (VGCD) channels (Figure 1b). These channels contribute in a nonlinear manner to the electrical response of the neuron by varying their ionic conductances in response to the voltage and/or calcium concentration. A large variety of VGCD channels exist, and they contribute substantially to the electrical properties of the neuron (Johnston et al. 1996, Spruston et al. 1994, Hille 1992). The contribution of VGCD channel subtypes to information encoding varies adaptively, however. Intracellular signaling compounds, whose concentrations are elevated in response to activity, functionally modulate the kinetics and maximum conductances of VGCD channels (Perezreyes and Schneider 1994, Kallen et al. 1993, Li et al. 1993, Toro and Stefani 1991, Numann et al. 1991, Chetkovich et al. 1991). A significant body of work in Aplysia suggests that some forms of animal learning can be attributed to the activity- dependent modulations of VGCD channels (Klein and Kandel 1978, Fitzgerald et al. 1990, Edmonds et al. 1990). Therefore, there are many combinations of VGCD channel dynamics that are congruent with the "normal" functioning of neurons. However, specific classes of combinations may be required for or achieved during information processing associated with certain cognitive or behavioral states.

To address the extent of VGCD channel influence on information processing, we need a substantial amount of information about the types and numbers of VGCD channels present in a given class of neurons. We need to know the kinetics of the channel conductances with respect to their voltage and calcium sensitivities. We need to know their distribution along the spatial extent of the neuron and how these distributions may change over time. Finally, we need tools to manipulate the channel distributions, conductances and/or kinetics so that we can observe how channel modulations affect the behavior of the entire neuron.

Substantial advances have been made over the past decade in the tools and techniques available for the measurement and manipulation of VGCD channels and currents. These methods include single channel recording (Sakmann and Neher 1983, Wonderlin et al. 1990), voltage-sensitive dyes (Ebner and Chen 1995), optical imaging of calcium ion concentrations (Denk et al. 1990), and localization of channels via immunohistochemistry or autoradiography (Rhodes et al. 1996, Maletic-Savatic et al. 1995). However, we still don't have the ability to experimentally identify and control VGCD channels in a precise manner. Spatial limitations prevent the measurement of individual channels on the smaller dendritic processes. Channel localization methods are limited by the specificity of the marker

compounds and the resolution of the optical signal (Kuhar and Unnerstall 1985). At this point in time, it is not possible to manipulate the spatial expression of channels nor is it possible to impose a specific localized change in the conductance of a single channel type.

As an alternative approach, biophysically realistic computational models of neurons (Segev et al. 1989, De Schutter and Bower 1994) allow observation of the state variables, such as voltage and calcium concentration, and control of the scaling parameters, such as the kinetics of channels and their spatial distributions, with a precision unattainable in experimental preparations. The past decade has seen the development of several public domain neural simulators (De Schutter 1992). The power of modeling and ease of model implementation are persuasive arguments for studying the influence that VGCD channels can have on the bioelectrical behaviors of neurons within a computational framework.

The outstanding problem in the development of these models has been to determine parameter sets that represent the spatial distributions of the VGCD channels. Models of neurons typically have from tens to thousands of unconstrained parameters. Because these models exist in high dimensional spaces, it is not possible to evaluate every combination of plausible parameters. For example, to enumeratively search through each combination of 100 binary parameters $\{0, 1\}$, a total of 2^{100}, or 10^{30}, simulations must be performed. If each simulation requires one second to be run and evaluated, $4 * 10^{20}$ centuries would be required to complete this evaluation[1].

We would like to know ALL the parameter sets that produce behaviors appropriate to a given electrical signaling behavior. Such an encompassing manifold, or *boundary enclosing a volume of solutions*, captures the reality of biology-like diversity. We would like to examine the relative ranges of each parameter manifold as a measure of sensitivity of the model along each parameter axis. Finally, we would like an efficient method of producing these solutions.

2. High performance computing aspects of the problem. Because this IMA volume and associated workshop were intended to focus on the high performance computing (HPC) aspects of EA implementations, it seems reasonable to briefly review the computational structure of the model. This will additionally help the reader to better understand the source and the significance of the parameters we would like to optimize. Our model is structurally similar to systems in other HPC fields such as fluid dynamics and geophysics. In many ways, the process and goals of model building are shared across HPC applications. We have a computational model of a physical process. We have many unconstrained parameters in our description. We want the behaviors, e.g. the time series, from our simulated model to correspond well with data collected from the

[1]Incidentally, this is older than the universe is estimated to have existed (Jantsch 1980).

physical system. We would like to interpret parameter sets that yield good correspondences to validate the appropriateness of the model or to make predictions about the physical system. Our approach should therefore be generalizable to a number of HPC endeavors.

The physics of electrical behaviors in biophysically realistic neurons is described by the cable equation (Rall 1989). The cable equation is a second order partial differential equation *(PDE)* of the parabolic type. It is analytically similar to the heat or diffusion equations. The equation describes voltage as a function of space and time.

$$\frac{\partial^2 V}{\partial X^2} - V - \frac{\partial V}{\partial T} = 0 \tag{1}$$

The voltage term, V, represents the transmembrane voltage difference as a deviation from the resting value. X and T represent the dimensions of distance and time over a specified domain. The Method of Lines replaces the *PDE* problem by a mathematically equivalent system of ordinary differential equations *(ODE)* by discretizing the spatial domain. Within each local domain, or spatial compartment, kinetic equations such as those describing VGCD conductances can be incorporated. This resulting ODE system can be subsequently solved using numerical integration. The derivation follows.

The following substitutions

$$X = \frac{x}{\lambda} \tag{2}$$

$$T = \frac{t}{\tau} \tag{3}$$

produce a form of the cable equation that incorporates biological parameters.

$$\lambda^2 \frac{\partial^2 V}{\partial x^2} - V - \tau \frac{\partial V}{\partial t} = 0 \tag{4}$$

x is the distance along the axis of a membrane cylinder (cm) and t is time (ms). λ (cm) and τ (ms) are scaling metrics for the electrical properties of the system. Known as the length constant and the time constant, respectively, they are metrics for the distance or time required for the voltage to attenuate to $1/e$ of the initial value when a deviation from the resting value is applied. They are defined in terms of the resistance and capacitance per unit length or area.

$$\tau = R_m C_m \tag{5}$$

$$\lambda = \sqrt{\frac{r R_m}{2 R_i}} \tag{6}$$

R_m is the specific membrane resistance (Ωcm^2), C_m is the specific membrane capacitance ($\mu F/cm^2$), R_i is the axial resistance (Ωcm), and r is the radius of the cylinder (cm).

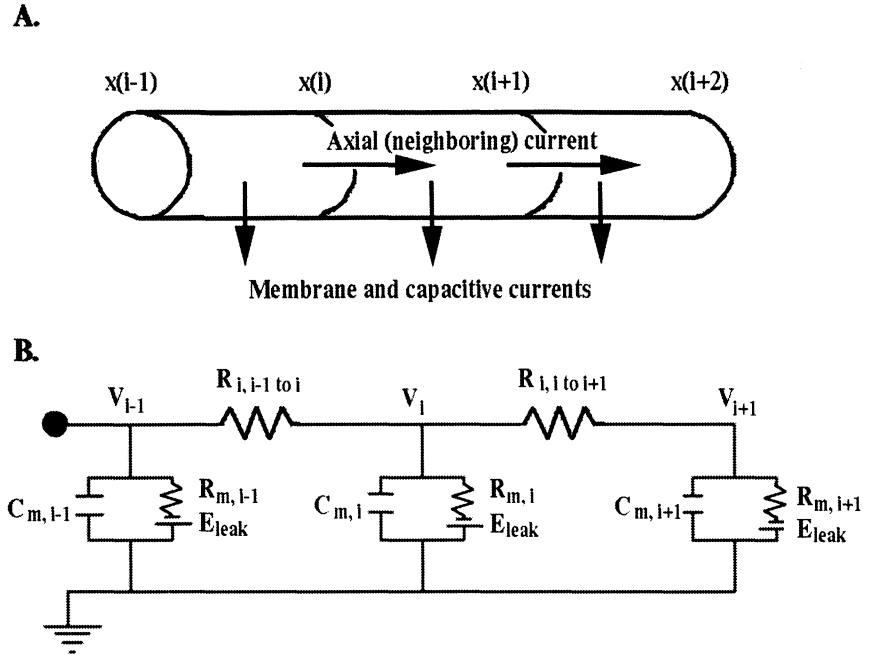


FIG. 2. *(A) A length of a neuronal process is discretized into three cables that are spatially adjacent to each other and coupled through a linear resistance term. (B) The electric circuit diagram for three electrical compartments has an axial resistance term coupling the compartments. Each region of membrane in this figure has two sources of local current: membrane (passive leak) and capacitive.*

The *PDE* is replaced by a vector equation of N *ODEs* through the approximation of the second order spatial derivative with a central differences expression. The number of compartments[2], N, depends on the number of discretized elements required for the system to numerically converge. The number of adjacent elements is determined by the specific morphological representation of a given compartment[3]. Equation (7) describes the Laplacian for an element with two neighboring compartments.

$$\frac{\partial^2 V_i}{\partial x^2} \cong \frac{\left(\frac{\partial V_{i+1/2}}{\partial x}\right) - \left(\frac{\partial V_{i-1/2}}{\partial x}\right)}{(\Delta x_{i+1/2,i-1/2})} \cong \frac{\frac{(V_{i+1}-V_i)}{\Delta x_{i+1,i}} - \frac{(V_i-V_{i-1})}{\Delta x_{i,i-1}}}{(\Delta x_{i+1/2,i-1/2})}; i, i+1, i-1 \in N \tag{7}$$

[2]The terms "element" and "compartment" are used interchangeably in this passage. "Element" is often used in fluid dynamics where PDEs are solved by Finite Element Methods. The term "compartment" is predominately used by computational neuroscientists.

[3]For example: terminal endings have one neighbor, cable continuation regions have two neighbors, branch points have three neighbors, and the soma can have an arbitrary number of neighbors.

To clarify the notation, consider the simplifying assumption that all compartments have uniform radius and length. Rearrangement and substitution yields:

$$C_m \frac{\partial V_i}{\partial t} = \frac{r_i}{2R_i}\left(\frac{(V_{i+1} + V_{i-1} - 2V_i)}{(\Delta x^2)}\right) - \frac{V_i}{R_m} \tag{8}$$

The axial (neighbor) and membrane currents are defined by application of Ohm's Law.

$$I_{\text{neigh}} = \frac{r_i}{2R_i}\left(\frac{V_{i+1} + V_{i-1} - 2V_i}{(\Delta x^2)}\right) \tag{9}$$

$$I_{\text{mem}} = -\frac{V_i}{R_m} = -(G_{\text{mem}} V_i) \tag{10}$$

Thus by substitution, the single compartment ODE can be described by three current sources; capacitive, neighboring (axial), and membrane.

$$\frac{\partial V_i}{\partial t} = -\left(\frac{I_{\text{neigh}} + I_{\text{mem}}}{C_m}\right) \tag{11}$$

It is generally assumed that C_m and R_i are constant for all compartments in the model, while R_m is often varied (Holmes and Rall 1992). The capacitance of the membrane is typically estimated to be a constant value of $1.0 \mu F/cm^2$ (Hodgkin and Rushton 1946).

The voltages representing spatially adjacent compartments in the I_{neigh} term *couple* the system of *N ODEs*. This co-dependency requires that the entire system be solved simultaneously as a vector-matrix equation. The matrix is structurally symmetric, although not necessarily numerically symmetric when compartments of nonuniform physical dimensions are allowed. The matrix is sparse and constant (with respect to the location of zeros) for a given discretized morphology (Hines 1984, Mascagni 1989, Eichler West and Wilcox 1996). Numerical methods optimized for sparse matrices, such as the minimal ordering method, eliminate unnecessary operations on zero-valued matrix elements (Press et al. 1992).

I_{mem} is composed of several current sources: injected, passive leak, and VGCD channel currents.

$$I_{\text{mem}} = I_{\text{stim}} + I_{\text{leak}} + I_{\text{VGCD}} \tag{12}$$

Any carrier, pore, or mechanism for transferring ions across the membrane is considered a subset of this current type. Injected current, I_{stim}, is a driving term typically applied to the soma compartment. This term models microelectrode current injections during biological experiments. The membrane leak current, I_{leak}, stabilizes the voltage near the resting potential.

$$I_{\text{leak}} = G_{\text{leak}}(V - E_{\text{leak}}) \tag{13}$$

G_{leak} is the conductance (μS) and E_{leak} is the reversal potential (mV) determined by the Nernst potential of the permeable ion species. G_{leak} and E_{leak} are typically constant-valued. I_{VGCD} is the sum of the currents produced by multipleVGCD channel types. Using the classic Hodgkin and Huxley model (1952) as an example, we would include currents representing the fast sodium (I_{Na}) and potassium delayed rectifier ($I_{\text{K(DR)}}$) channels in the I_{VGCD} expression.

$$I_{\text{VGCD}} = I_{Na} + I_{K(DR)} \tag{14}$$

The conductances of the VGCD channel subtypes are functions of voltage, calcium concentration, and/or time. Ion channels are assumed to exist in either an open (conductive) or closed state. Using the potassium delayed rectifier parameter as an example, n represents the normalized fraction of the population that is in an open state, α is the rate at which the channels open, and β is the rate at which the channels close. The rate constants α_n and β_n are functions of voltage and/or calcium concentration[4].

$$\text{open}(n) \underset{\beta_n}{\overset{\alpha_n}{\rightleftharpoons}} \text{closed}(1-n) \tag{15}$$

From this relationship, the first order kinetic equation for the open state is:

$$\frac{\partial n}{\partial t} = -\beta_n n + \alpha_n(1-n) = -(\alpha_n + \beta_n)n + \alpha_n \tag{16}$$

To produce the current term for a given channel subtype, n is multiplied by a conductance scaling factor (G_n in μS) and the driving potential for the ionic species permeable to that channel ($V - E_n$ in mV). The driving potential, determined in part by the concentration gradient of the major ion species, favors inward currents through sodium and calcium permeable channels and outward currents through the potassium permeable channels.

$$I_n = G_n n(V - E_n) \tag{17}$$

The conductance scaling factors, G_n, are typically the unknown parameters in compartmental models. As *described in the next sections, EA strategies can be used to determine appropriate values for these parameters.*

Many channel types have more complicated kinetics due to the existence of multiple subconductance states (m, eqn. 18; n, eqn. 19) and/or

[4]The voltage-dependent rate constants are nonlinear terms typically fit by experimental data to the form $\frac{A(V-B)}{\left(\left(\exp\frac{V-C}{D}\right)-E\right)}$, where A, B, C, D, and E are constants. Calcium-dependent rate constants are not typically as standard in form, but are also nonlinear and fit from experimental data.

inactivation gates (h, eqn. 19). For example, the Hodgkin and Huxley (1952) sodium and potassium delayed rectifier currents are defined as:

$$I_{Na} = G_{Na} m^3 h (V - E_{Na}) \tag{18}$$

$$I_{K(DR)} = G_{K(DR)} n^4 (V - E_K) \tag{19}$$

It should be noted that the classic biophysical kinetic description of Hodgkin-Huxley is only an approximation of the ensemble behavior of channels (Hille 1992, Keynes 1992). Further, this model may be inappropriate to describe the behavior of some channel types (Goldman 1943).

The state parameter ODEs (eqn. 16) must be solved concurrently with the voltage ODEs (eqn. 11); however, the state parameter equations depend only on the local values of voltage and/or calcium. Inclusion of these current sources potentially increases the problem size substantially. For the Hodgkin-Huxley (1952) model with N compartments, $4N$ equations (m, h, n, V) must be solved simultaneously. The Traub (1991) CA3 hippocampal neuron model, which we have used in the modeling work presented in this paper, contains 11N equations. This model uses a simplified 19 compartment representation of a neuron for a total of 209 ODEs and 114 unknown conductance scaling factors. The De Schutter-Bower (1994) cerebellar Purkinje cell model contains many more channel types and an experimentally-derived morphology for a total of 32,000 ODEs and 19,200 unknown conductance scaling factors. The trend in neuroscientific modeling is to build larger models that include more experimental detail. Thus, the need to determine appropriate areas of parameter space will only continue to grow.

The Traub (1991) model offered many advantages for a proof-of-concept EA experiment; each simulation required minimal computer resources and the model expressed a range of characteristic hippocampal neuron behaviors. A depolarizing afterpotential (DAP) was produced after a stimulus subthreshold for burst elicitation and suprathreshold bursts were followed by afterhyperpolarizations (AHPs). The model demonstrated a transition between single spiking to bursting regimes that corresponded well with experimentally-observed frequency-intensity responses. Finally, a network of these neurons was able to reproduce picrotoxin-induced synchronized multiple bursts and has been since refined as a model of gamma-frequency EEG activity (Traub et al. 1996).

Traub's model reduced the morphology of a CA3 hippocampal neuron to a 19 compartment equivalent cylinder (Rall 1962). The passive membrane properties, $Rm = 10,000\ \Omega cm^2$ and $Cm = 3.0\ \mu F/cm^2$, yield a time constant[5] of $30ms$ consistent with experimental observations (Turner

[5]The time constant, tau, is the product of the membrane capacitance and resistance in a passive membrane model. $\tau = R_m C_m$. However, Traub's capacitive value is considered unphysiologically high and may have the effect of excessively scaling the voltage derivative. Likewise, the membrane resistance is low according to recent experimental

and Schwartzkroin 1980, Turner 1984). The test stimulus was a 0.3 nA depolarizing current injection into the soma compartment because Traub described physiologically interesting behaviors produced by this stimulus. The model defined kinetic equations for six channel conductances within each compartment: a sodium channel, a hybrid high threshold calcium channel, a potassium delayed rectifier channel, a potassium A channel, a potassium AHP channel, and a calcium-dependent potassium channel.

The EA in this study searched the 114 dimensional space containing the six channel conductances in each of the 19 compartments. Traub selected his parameters by trial-and-error. One consequence of this approach was the predominant use of arbitrary zeros and the report of a single "best" solution of the search rather than a manifold. We wanted to demonstrate the ability of an EA to automatically select a volume of solutions that either included the parameter set of Traub or demonstrated an improved correspondence to experimental observations in comparison to Traub's parameter choice.

We defined two low dimensional (single value) metrics to characterize the parameter space for the purposes of post-analysis and initialization of the random seed parameters. The *total conductance* is the sum of all conductances over all compartments and the *excitatory/inhibitory conductance ratio* (EIR) is the sum of all excitatory (inward) conductances over all compartments divided by the sum of all inhibitory (outward) conductances over all compartments.

3. Why use evolutionary algorithms? Evolutionary Algorithms (EAs) represent an efficient and robust method for parameter fitting in high dimensional spaces (Holland 1975, Forrest 1993). This algorithm has been successful when applied to difficult optimization problems: for example, in job shop scheduling (Whitley et al. 1995), high-performance network design (Davis et al. 1993), DNA fragment assembly (Parsons et al. 1995), and models of the earth's mantle viscosity (Kido et al. 1996). A statistical proof called the Schema Theorem guarantees improvement of the search over time (Holland 1975).

EAs have not previously been applied to the field of compartmental neuronal modeling, although reduced dimensional parameter searches using systematic (Bhalla and Bower 1993) or stochastic (Foster et al. 1993) search methods have been reported. Would these or methods other than an EA perform better? Descriptions of many optimization methods are available both conceptually in textbooks and algorithmically in computer executable codes obtainable from public domain sites on the Internet. These methods include hill climbing, simulated annealing, neural networks, ge-

---

estimates (Spruston and Johnston 1992). $Rm = 30,000\ \Omega cm^2$ and $Cm = 1.0\mu F/cm^2$ are more reasonable values. We did not deviate from Traub's values in this study such that our final results could be more directly compared to his.

netic algorithms, local search, and heuristic search. With the large number of methods and their variations available, how does one choose the best for one's application?

Wolpert and Macready (1995) propose that there are "No Free Lunches" (NFL) for effective optimization. Using theoretical arguments, they claim that all search algorithms perform exactly the same when averaged over all cost functions. In other words, the set of functions for which algorithm A outperforms algorithm B is complementary to the set for which B outperforms A. Furthermore, observing how well an algorithm has done so far in a search will tell us little about how well it will continue to do. Consequently, it is a flawed strategy to choose a method by comparing initial algorithmic performance and then to search completely based on these preliminary results. Wolpert and Macready warn that if one does not incorporate any information about the function into the algorithm, then success must rely on a fortuitous matching between the features of function and the search path determined by the algorithm.

The NFL paper produced a great deal of discussion because the implication that choosing a successful method for a given application may be akin to the luck of a coin flip stands in contrast to empirical experience. Culberson (1996) extended the NFL theorem with a corollary proposing that *all algorithms perform the same as a random enumerative search when the method is blind.* These assertions suggest that we will not likely find a canned method to solve our optimization problem. A hand-tuned method is required, taking into account as much information about the function and solution space as we know.

The need to incorporate domain knowledge into a search is not newly revealed by the NFL-related reports. Based on empirical studies, Davis (1991) suggested that the difficulty of search is not a property of a given problem, but rather depends on the relationship between the problem and the algorithm. Using formal language theory, Hart and Belew (1991) also conclude that analyses of optimization methods (in particular, a GA), which do not specify the class of functions being analyzed, can make few claims regarding the efficiency for an arbitrary function. Specifically, they state:

> "These results suggest that future analyses of the Genetic Algorithm must pay close attention to the relationship between the algorithmic parameters of the GA and the function space from which the fitness function is selected. Since any fixed set of algorithmic parameters cannot enable the GA to efficiently optimize an arbitrary function in a broad class like the one we have considered, we must consider smaller classes or distributions of functions for which those parameters are most appropriate. Alternatively, if we have a function we wish to optimize, we must carefully select our algorithmic parameters if we wish to

optimize the function efficiently with the Genetic Algorithm."

This statement implies that there is a *specific* implementation of an algorithm that will be effective for each class[6] of functions. For an algorithm to be effective, the biases in how search space is explored must correlate well with salient features in the cost function. The problem is, of course, to know what is salient about the function when the whole point of the search is that little is known! (Otherwise, why would one be searching over it?)

To optimize the parameters for the compartmental neuronal model, one must first choose a method that sounds reasonable, gain some intuition into how the optimization algorithm works, and attempt to modify it with domain-based knowledge. Evolutionary Algorithms (EAs) are a popular and successful parameter optimization technique inspired by biological principles. Armed only with intuition and a predisposition towards this technique because of the familiarity of the biological metaphor, our initial optimization efforts focused on understanding the applicability of EAs for the compartmental neuron channel distribution problem.

4. Designing an EA strategy. The specific mechanisms of selection and recombination applied to a particular problem space determine the success of an EA. Selection identifies and retains favorable parameter combinations. Crossover enhances the search mechanism through the breakup and recombination of parameter combinations, maximizing a good balance between these two extremes. Thus, the representation of the problem along the string and the type of crossover function applied are important factors in the rate of success of a search because probability favors the breakup of longer parameter combinations. In the following sections, we will discuss and justify our implementation with respect to: 1) representation; 2) recombination; 3) selection; 4) initialization; and 5) termination criteria.

Representation

The representation of a problem is the mapping between problem space and the data space operated on by the algorithm. Three aspects of representation must be considered for an EA application: 1) embedding information onto string values; 2) ordering values onto the linear array of the string; and 3) choosing the cardinality of the alphabet (binary vs. real-valued). These aspects should be examined with respect to the choice of recombination operators one intends to apply.

We considered two embedding spaces for our problem: 1) a string of values representing each of the individual parameters; and 2) a string of scaled functions which could be applied over the normalized distance from the soma. We use the soma compartment as a spatial reference point in our system because most experimental recordings are made in the soma.

[6] Unfortunately, it remains unclear what distinguishes the differences between function classes.

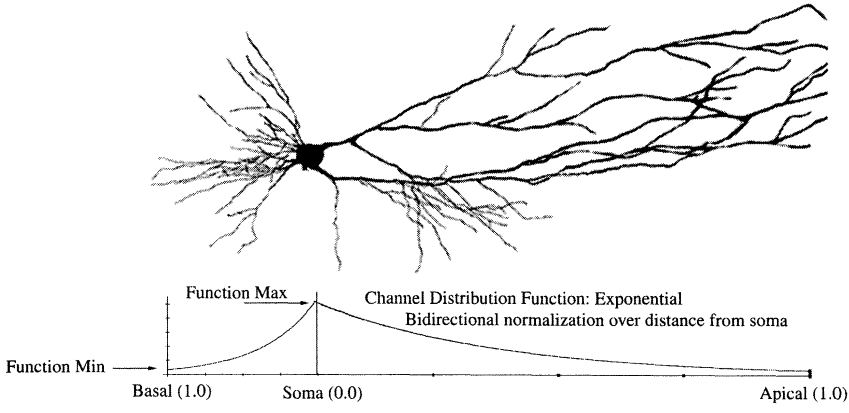


FIG. 3. *Only three parameters are needed to encode channel distributions as functions: function type (ex. linear, exponential, sinusoidal); a maximum function value; and a minimum function value. In this example, the function was applied over a bidirectional normalization of distance. However, this choice was arbitrary and other pertinent system metrics might be more appropriate.*

However, other PDE systems might consider the utility of applying scaled functions of the normalized length of the spatial domain along one or more dimensions.

For the string of individual values, two orderings appear to be natural for the problem: parameter ordering and compartment ordering. For parameter ordering, like- parameter types are ordered adjacently along the string. For example (using two channel conductance parameters):

$$\mathrm{Na}_1\mathrm{Na}_2 \ldots \mathrm{Na}_N \ldots \mathrm{K(DR)}_1\mathrm{K(DR)}_2 \ldots \mathrm{K(DR)}_N \tag{20}$$

where Na and K(DR) represent the sodium and potassium delayed rectifier conductance parameters, and the subscripts refer to compartment number (N total compartments). In compartment ordering, all parameters from the same compartment maintain adjacency along the string. For example:

$$\mathrm{Na}_1\mathrm{K(DR)}_1 \ldots \mathrm{Na}_2\mathrm{K(DR)}_2 \ldots \mathrm{Na}_N\mathrm{K(DR)}_N \tag{21}$$

Such an encoding would yield a string of length MN, where M is the number of parameter *types* and N is the number of compartments. Thus, this representation scales with the number of compartments. For the Traub model, we have a string length of 114. The De Schutter-Bower model would

be searched rather inefficiently with this representation, as the string would contain a total of 19,200 parameters.

A scaled function embedding requires assumptions about the parameter distributions that we expect to see in nature. It is reasonable to assume that the actual distributions, with respect to the linear distance from the soma, may be described as noisy instances of a constant, linear, exponential, or sinusoidal function (Stuart and Sakmann 1994, Masukawa et al. 1991). Additionally, structure-specific functions may be used to account for the observations that some channels may be present only at, for example, branch points (Catterall et al. 1991). While this approach has the advantage of incorporating domain-based observations, we are limiting the EA to exploring combinations of these plausible distributions. It would be somehow more satisfying to start from random conditions and have the EA converge on a set of parameters that could be fit by scaled functions.

In this embedding space (Figure 3), each parameter distribution is defined by three string values: a value encoding function type; a maximum function value; and a minimum function value. An additional term could be optionally added to include a degree of noise to the function distributions.

$$\mathrm{Na}_{\mathrm{Funct}}\ \mathrm{Na}_{\mathrm{Max}}\ \mathrm{Na}_{\mathrm{Min}}\ [\mathrm{Na}_{\mathrm{Noise}}] \ldots \mathrm{K(DR)}_{\mathrm{Funct}}\ \mathrm{K(DR)}_{\mathrm{Max}}\ \mathrm{K(DR)}_{\mathrm{Min}}\ [\mathrm{K(DR)}_{\mathrm{Noise}}] \tag{22}$$

Such an encoding would yield a string of length $4M$, where M is the number of parameter *types*. This representation is independent of any assumptions regarding the number of compartments appropriate for the model. The function would be scaled over the range determined by the maximum and minimum values. The parameter value for each compartment would be the evaluation of the function relative to the position of that compartment over the normalized length of the neuron.

The ordering of parameters along the string affects the defining length, e.g., the difference in the position along the string of those parameters that contribute most to achieving high fitness. Strings with longer defining lengths have a higher probability of being disrupted during crossover. Consequently, different orderings of the same problem will be differentially affected by the same crossover operator. On one hand, it is useful for crossover to be disruptive so that a thorough exploration of parameter combinations can take place. However, too much disruption may destroy any high order combinations of optimal parameters that the EA has discovered, rendering the overall search inefficient.

It is difficult to know *a priori* which of the three embedding descriptions would be most effective. If we chose a parameter ordering, the first parameter type and the last parameter type along the length of the string would be disrupted with a high probability (with respect to 1-pt crossover analysis). If we chose a compartment ordering, compartments numbered furthest apart would be disrupted most often. Which is more important to maintain - the longitudinal compartment distribution of parameters or

the inner- compartment ratio of parameters types to one another? Without any empirical or theoretical basis for the decision, one approach is to develop more than one crossover operator so that both orderings will be operated upon. Estimating the disruptive effect of crossover on the scaled function embedding is not quite as intuitive because each *parameter value for the model* is embedded in three (four) dimensions instead of one in the *string parameter* space. Previous work suggests that EAs are most effective when each string parameter can be optimized independently (Davis 1991). Consequently, a function-based search may be difficult for the EA to optimize. Determining the pros and cons of each ordering will require empirical exploration.

Genetic algorithms have traditionally used binary representations, primarily because of the relative simplicity in analysis offered by low cardinality alphabets. The parameters in neuron models are real-valued, however. Transformation of real values into a binary representation would have required a trade-off between the precision and efficiency of search. For example, if the resolution we desire for a given parameter is 0.1 arbitrary units, and if we assume a search range of $0.0 - 100.0$, then each binary representation of a parameter would require 10 bits. The 114 parameters of the Traub model would be encoded in a string length of 1140. This is a relatively long string over which we would expect recombination to be inefficient. Increasing the number of compartments would exacerbate this problem. The EA would spend too much time searching the least significant digits of the string early in the search. Once the significant digits had converged, they would no longer require searching. However, genetic drift would have caused a certain degree of convergence on the least significant digits such that there would no longer be a random distribution in the population to sufficiently explore that area of parameter space. Real encodings offer many advantages over binary encodings, including the convenience and psychological comfort that comes with the direct correspondence between the mapping spaces. Technical benefits of real codes include avoidance of Hamming Cliffs, elimination of crossover disruption *within* a parameter, and a reduced susceptibility to deceptive path traversal (Goldberg 1991).

For reasons of convenience, we decided to implement a real- valued string. To enable a more direct comparison to Traub's parameters, we chose to represent the space as a string of 114 values. Since we lacked sufficient information to choose between parameter ordering or compartment ordering, we decided to develop crossover operators that transposed and exploited both orderings.

Recombination Operators

Historically, crossover has been associated with genetic algorithms and genetic programming, whereas mutation had been predominantly studied in evolutionary programming and evolution strategies. Exclusive use of either operator is becoming less prevalent, especially in applications. The-

oretical and empirical studies have compared the conditions under which one operator is more successful than the other. The results do not provide us with much guidance, however. The main conclusion is that the success of a recombination strategy is specific to the problem being optimized and cannot be predicted *a priori*.

Commonly studied and applied forms of crossover include the 2-point and uniform operators. Early research suggested that the number of crossover points should be low to minimize disruption (Holland 1975, De Jong 1975). In contrast, many recent applications have demonstrated superior performance with more disruptive recombination operators (Syswerda 1989). It has been theoretically demonstrated that 2-pt crossover[7] is the least disruptive crossover operator. In contrast, uniform crossover[8] is the most disruptive but is without a positional bias (Spears and De Jong 1990). The question remains, which crossover operator should we apply to which ordering of our problem?

Mutation is both a mechanism for local search and a mechanism to prevent solution components lost during crossover from being eliminated for all successive generations. Mutation is applied in binary representations by flipping the value of the bit in the randomly selected position. In real-valued strings, more sophisticated scaling or replacement methods are required. Genetic algorithms typically apply mutation as a background operator at low rates, for example 0.1% (Goldberg 1989). In contrast, it has been shown that a high mutation rate can be beneficial in some applications (Bramlette 1991). In particular, a rate of 20% is recommended in some studies (Bagchi et al. 1991).

Because we were left with questions about which rate and kind of real-valued mutation would be successful for our problem, we decided to let the EA tell us which strategy worked best. This was implemented as a self-adaptive strategy for choosing the rate of application of *all* the operators of potential interest to us. This approach has been used to select mutation rates in evolution strategies (Bäck et al. 1991, Saravanan et al. 1995) and crossover rates in genetic algorithms (Spears 1995). Further, Parsons et al. (1995) report a synergistic effect gained by retaining many types of operator, suggesting additional benefits gained from this strategy.

---

[7]Two identical points are randomly selected along the length of two parent strings, dividing each into three segments. The first and third segments from the first parent and the second segment from the second parent are recombined to produce the offspring strings. 2-point crossover can be generalized to n-point crossover, where n is the number of crossover points.

[8]A probability P_o characterizes the degree of disruption. A random number is drawn for each parameter along the string. If the value of this random number is greater than P_o, then the value from parent 1 is used to produce the offspring string. If the random number is less than P_o, parent 2's value is used. The greatest disruption occurs when $P_o = 0.5$.

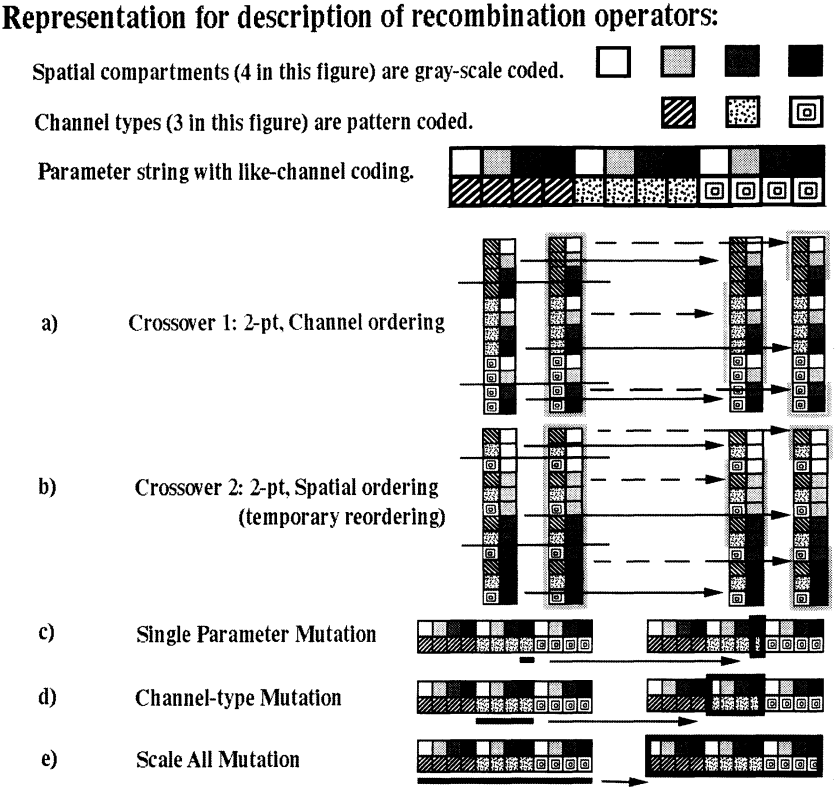


FIG. 4. *The crossover and mutation operators for the EA parameter search. Two parent parameter strings (represented without and with gray borders) were randomly selected from the mating pool for crossover operations. One parameter string was randomly selected from the mating pool for mutation operations. a, Crossover 1 (2-pt, compartment ordering). Two points along the string are randomly selected. The parent strings contribute alternate regions to the offspring string. b, Crossover 2 (2- pt, parameter ordering). The procedure is the same as in (a), except the parameters were first reordered such the same compartment parameters were adjacent. c, Single Parameter Mutate. Two random numbers were selected. The first number determined which of the parameters would be operated on. The second number was scaled to the range appropriate to that parameter and replaced the original value of the selected parameter. d, Parameter-type Scale Mutation. Two random numbers were selected. The first number determined which of the parameter types would be operated on. The second number was a scaling factor that ranged from* 0.0 *to* 2.0. *All conductances of the selected channel type were multiplied by the scaling factor. e, Scale All Mutation. Random number selection (a coin flip with probability 0.5) determined whether all values in a string were scaled by constant multiplicative factor* 0.9 *or* 1.1.

a highly fit individual to contribute to the mating pool for multiple generations if its fitness was sufficiently high, thus preventing the loss of good strings (Bäck and Hoffmeister 1991). Many EA implementations allow multiple copies of high fitness strings to contribute to the mating pool in

We have custom-designed recombination operators[9] to explore the real-valued parameter space potentially suitable for the compartmental neuron problem. The set of recombination operators includes: *crossover 1 (2-point, compartment ordering)*, *crossover 2 (2-point, parameter ordering)*, *single parameter mutate, parameter-type scale, and scale all* (Figure 4). Two versions of the 2-point crossover operator were applied to exploit both *compartment ordering* and *parameter ordering*. This allowed us to reduce positional bias while maintaining low disruption. The *single parameter mutation* explored solutions adjacent in a single dimension to current solutions. We assumed that the ratios of the conductances of each of the channel types were important determinants of high fitness solutions and subsequently developed scaling operators, *scale all* and *parameter-type scale*, to explore this assumption.

All operators had the same initial probability of being chosen. Subsequent probabilities were based on the success of a given operator relative to that of the others. The minimum probability of operator selection was fixed at 1%. A success was awarded when a parameter set produced by a given operator scored a fitness value above the average fitness of the population.

Selection Mechanisms

Selection identifies and retains high fitness strings. Selection mechanisms are characterized based on strength. Strong selection concentrates quickly on exploiting the best individuals, favoring fast convergence at the expense of minimally sampling the search space. Such a strategy could potentially lead to premature convergence on a less than optimal solution, but the answer is obtained quickly. Weak selection maintains a highly diverse population, but at the expense of increased search time. Very weak selection results in a random walk search. Since we want to evolve a *manifold* and not just a "best" single solution, we will avoid the stronger methods. This would potentially cost us an increased number of function evaluations, but we hoped that we would find better solutions.

We developed the following selection mechanism. An extinction threshold, a variable fitness criterion, was automatically adjusted to maintain a population size between 500 and 1000 population members. Population members from both current and previous generations were admitted to the mating pool if their individual fitness values exceeded the extinction threshold. This threshold-based "elite selection" mechanism allowed

[9]We experimented with other recombination operators, but they consistently yielded success rates less than 2.5%. These included: 1) inversion operators that reordered the 114 parameter *from* the left to right ordering $[1, 2, 3, ..., 112, 113, 114]$ *to* the right to left ordering $[114, 113, 112, ..., 3, 2, 1]$; and 2) alignment operators that performed a self-crossover by reordering adjacent sections of a string, for example *from* $[1, 2, 3, ..., 56, 57, 58, ..., 112, 113, 114]$ *to* $[56, 57, 58, ..., 112, 113, 114, ..., 1, 2, 3, ...]$. Two versions of each operator were developed to exploit both spatial ordering and channel ordering.

5. Designing a fitness measure. It was not clear how much information and what kind of information should be included in our fitness criteria. However, it would seem to be possible to manipulate the algorithm to produce whatever results we want. Therefore, any claims regarding the power of the EA parameter optimizing approach would be compromised if we *tweaked* the fitness measure until we obtained good performance. Therefore, we attempted to objectively design and adhere to a fitness measure philosophy before we began the EA search. While the specific quantitative aspects of our criteria were extracted from the biological literature, our approach to fitness measure design was intended to be a generally applicable to any time series.

We partitioned the information into three categories: 1) mathematical requirements; 2) etiological satisfaction; and 3) waveform decomposition. The decision to award 20% of the fitness points to the mathematical requirements and etiological satisfaction categories and 60% to the waveform decomposition was arbitrary. Given the variability in real physiological responses, we distributed the assignment of scores using a gradient rather than a step function. In other words, a *range* of values received the full score for satisfaction of each individual criterion and those values that were close to the optimal range received partial credit. By *coaxing* the evolution along through the assignment of partial credit, we provide the EA with more information to guide the search given a constant number of fitness criteria.

Mathematical requirements. Time series exhibit damped, periodic, chaotic, or random characteristics. The stimulus we applied to the model should generate *approximately* periodic bursting. Therefore, the first goal was to partition the space of all solutions to eliminate parameter spaces that yield damped or steady state time series[11]. Credit was assigned for fulfilling the following successive criteria. Does the solution demonstrate spiking at all? Does it spike beyond the initial transient? Does it spike during the last 20% of the time series?

Etiological criteria. Does the parameter set produce periodic solutions for the wrong reasons? The somatic time series should yield spiking in response to the depolarizing current injection. Dendritic compartments with sustained, elevated voltages could provide a source of depolarizing axial current that would inappropriately (in the absence of synaptic currents in this model) lead to somatic spiking. Thinking like an experimental biologist, we considered these solutions equivalent to neurons which had their dendritic membranes damaged during experimental preparation and rejected them.

[11]The data collected from neurons is far too noisy and neurons themselves are far too adaptive to become overly concerned with distinguishing between the other three classes of time series behavior at this point. However, a healthy neuron would respond to the current injection by producing spikes or bursts.

proportion to the absolute or scaled fitness value of that string relative to the other strings (Forrest 1993, Goldberg 1989). Our implementation limited membership in the mating pool to a single copy of each suprathreshold string. This strategy reduced the possibility of premature convergence caused by the overexpression of proportionately higher fitness strings.

The initial population size was 1000 parameter sets. Preliminary results suggested that this population size provided a sufficient compromise between diversity and efficiency[10]. The population size must be large enough to maintain a diverse gene pool. Otherwise, the EA must primarily rely on mutations to explore the space. In such a case, the rate of improvement is reduced to that of a random walk search. However, an extremely large population could increase the amount of computer time needed to solve the problem and might not improve the rate at which the problem is solved (Macready et al. 1996).

Initialization

The initial parameters were randomly selected and scaled as follows: 1) 114 real- valued random numbers were drawn between 0 and 1; 2) a random number was selected between 0.5 and 2.0 to scale the *excitatory-to-inhibitory conductance ratio*; and 3) a random number between 0 and $2,500 mS/cm^2$ was selected to scale the *total conductance*. Preliminary results suggested that parameter sets with values outside of these bounds did not yield fitnesses greater than 0.1. The recombination operators allowed subsequent generations to explore beyond these initial bounds.

Termination

Termination criteria in the EA literature are somewhat arbitrary because the statistical nature of the search prohibits knowing *when* the population will evolve to a given fitness value. Typically, EA searches are terminated after: 1) a predetermined number of generations or individual population members have been evaluated; 2) a goal fitness (such as 1.0) has been achieved; or, 3) the diversity of the string values in the population has been significantly reduced. Our arbitrary termination criterion was satisfied after 200 CPU hours had elapsed on each of eight dedicated Silicon Graphics Power Challenge R8000 90 MHz processors (approximately 125 Tera floating point operations total). At most HPC research centers, researchers are allocated a certain number of CPU cycles on various machines during six month to one year grant periods. We find that a CPU-based termination criteria gives us a much more informative measure of what we can potentially solve with the CPU time we have available.

[10]Essentially, we increased the population size until no further gains were made in the rate of "best population member" improvement over the evolution of twenty generations.

Waveform decomposition. The fitness filter is not a temporal point-by-point curve fitter. Instead, we decomposed the evaluation of each spatiotemporal dataset into a set of characteristics extended along multiple scales (Figure 5). The largest scale feature was the somatic burst rate. We then *zoomed* into the characteristics of the individual bursts to evaluate such features as the peak-to-burst ratio, burst width, and the least squares fit to an experimental CA3 burst waveform. At the finest scale, we evaluated the postburst afterhyperpotential amplitude as well as quantitative aspects of individual spikes such as the width and height. Lastly, spatial behaviors such as the relative calcium influx and peak height were examined and scored.

Finally, we specifically included observations noted by Traub into our evaluation. It does not appear that Traub proposed a list of criteria and then optimized the parameters to meet these criteria. Rather, it is simply reported that the model is able to replicate experimental observations such as the following: 1) afterhyperpotentials drop 5 to 10 mV below resting potential; 2) somatic bursts are produced at approximately 1 Hz in response to the application of 0.3 nA depolarizing current in the soma; 3) apical and basal dendrites yield spikes; 4) the maximum calcium influx is observed in the midapical dendrites; and 5) spike width at half amplitude in soma should be approximately 0.8 ms.

6. HPC and parallelism. Each simulation executed for five seconds of neuron-time so that sufficient data were available for evaluation by the fitness filter. Each simulation required 49 seconds on a Silicon Graphics R8000 90 MHz processor. For a model such as the Purkinje cell (De Schutter and Bower 1994), each simulation could require hours. Because each parameter set can be evaluated independently, EA applications have a natural parallel decomposition. Our EA searches have been implemented in parallel in the following four ways: 1) fork/exec functions; 2) rcp/rsh scripts; 3) Loadleveler/DQS application management; and 4) MPI/PVM libraries. We used combinations of these approaches to maximize our access to heterogeneous computing environments. This environment included a 9 processor Cray C916/10512, a 12 processor Silicon Graphics Power Challenge (R8000 75 MHz), a 8 processor Silicon Graphics Power Onyx (R8000 90 MHz), 8 IBM RS6000 Model 590 workstations, and a 14 processor IBM SP Supercomputer.

In the simplest case, parallelism can be implemented at the process level using standard ANSI C system calls to *fork/exec* functions. No special software or libraries are required. Load balancing is handled by the operating system. However, the implementation is limited to shared memory architectures such as machines offered by Cray and Silicon Graphics. Our EA program executed each simulation independently as a child process. Each simulation was assigned an identification number using the *putenv* function. The simulator used the identification number to open

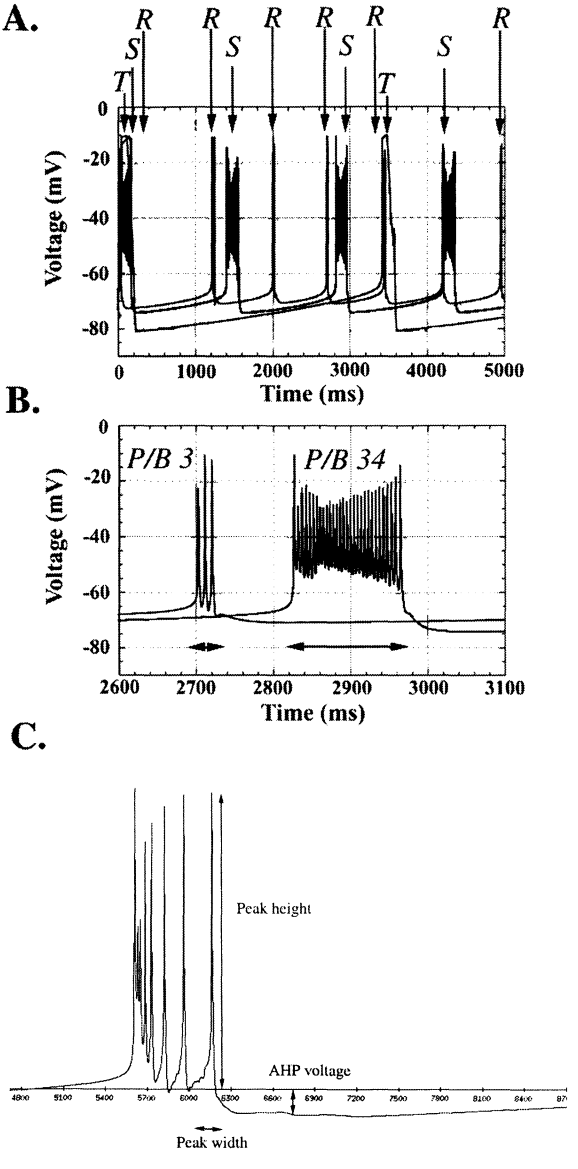


FIG. 5. *Waveform decomposition for the somatic time series. (A) Global behavior, such as the burst rate, was evaluated and scored. Time series R and S received full credit for satisfaction of this criteria because the burst rates, approximatley* 0.8 *and* 1.2 *Hz respectively, are within the experimentally observed range. (B) Qualities of the individual bursts, such as peak-to-burst ratio and burst width, were evaluated and scored. The left burst received maximal credit for a peak-to-burst ratio and burst width within the acceptable ranges. (C) The fine detail characteristics of the spikes within a burst, such as peak height and width and the afterhyperpotential (AHP) voltage, were the final behaviors to be evaluated.*

the correct parameter file and to name the output file containing the fitness measure. The *wait* function monitored the completion status of the child processes. We found this method sufficient for most of our small to medium-sized applications.

To implement parallelism on distributed memory systems and/or heterogeneous environments, we wrote shell scripts which used *rcp/rsh* to distribute the parameter files, execute the simulations on remote machines, and collect the results. This method required that an executable copy of the simulation program was installed on every machine. In the interest of security, many system administrators disable remote commands to prevent password-free access to their machines. Therefore while we were able to access many departmental workstations, we could not access all the HPC machines to which we had access in this fashion. To maximize efficiency, we wrote load balancing routines to determine the CPU load on each machine before submitting jobs. Unfortunately, if other users submitted jobs after we started a simulation, we still had to wait for simulations on the loaded machines to complete. If the wait became too long, we started another version of the delayed simulation on the fastest machine that was currently idle. We found this method of obtaining parallelism to be a great way for researchers without access to machine time at a HPC center to steal extra CPU cycles from idle workstations overnight!

LoadLeveler is a software product developed by IBM for managing parallel applications. Versions of this product are available for IBM RS6000, Sun, Hewlett-Packard and Silicon Graphics architectures. This is reliable and easy to use software featuring machine specific configurations, load balancing, checkpointing, and a graphical interface for developing applications and monitoring the status of jobs currently executing. It schedules either serial or parallel (including MPI or PVM) jobs written as LoadLeveler or NQS scripts. The good news and bad news is that this is a commercial product; while it is technically well-supported by IBM, it is not free. As a low budget alternative, there are several shareware products available via anonymous ftp that claim similar functionality and features. We did not evaluate the claims of the shareware products, but we found LoadLeveler to be a useful and stable tool.

MPI (Message Passing Interface) and PVM (Parallel Virtual Machine) are libraries of routines for passing messages between processors that potentially represent a range of architectures. These libraries have quickly become the standard for the development of parallel codes. The libraries are available via anonymous ftp, but must be installed on every machine participating in the virtual network. They offer a wide variety of communication classes including *gather*, *scatter*, and *all-to-all* using communication modes such as *standard*, *synchronous*, *ready*, or *buffered* in *blocking* and *non-blocking* forms (Snir et al. 1995, Gest et al. 1994). At the time of this writing these libraries have limited error handling routines, do not support dynamic task spawning, and require the user to incorporate load

balancing algorithms. However, the widespread interest in these libraries by developers and the adoption of these libraries by multiple vendors have resulted in a number of third party applications now available to manage CPU resources when the libraries themselves are insufficient.

For many applications, parallelism implemented with file transfers is inefficient compared with the message passing model. This is because the communication time required for the data transfer is large relative to the computation time performed by each distributed CPU (Kumar et al. 1994). Most parallel architectures, (ex.. Thinking Machines Corporations CM-5, Cray's T3E) have dedicated hardware and parallel versions of standard programming languages (ex. Fortran 90, C*) to reduce communication overhead, in addition to their support of MPI/PVM. Our programs yielded approximately linear speedup[12] with both the message passing and file transfer implementations because of the relatively large amount of CPU time required per simulation. However, it has been our observation from the EA literature that most fitness functions do not require as much CPU time as we needed. For maximum portability, longevity, and efficiency, we recommend parallelization using a message passing library such as MPI/PVM in conjunction with third party CPU management software. However, depending on the application and the available architectures, process/file-based parallelism is easy to implement and may be just as efficient as other methods.

7. A success story. We judged our application to be a success by two criteria. First, the average fitness of the population quickly exceeded the fitness score achieved by Traub's parameter set. Second, the high fitness parameter sets produced time series that resembled those observed experimentally. Similar results were obtained in three separate experiments, each using different random initial conditions. As a bonus, we identified manifolds for our low dimensional descriptors (the total conductance and the excitatory/inhibitory conductance ratio) of the evolved high dimensional parameter space which will allow us to reduce the parameter space in future refining searches. (See Eichler West 1996 for additional results.)

The EA evolved a population of high fitness parameter strings from random initial conditions. The initial generation yielded an average fitness of 0.04. This was consistent with our observations that parameter sets composed of random numbers showed a 96% probability of scoring a fitness less than 0.20 ($n = 11,600$). The EA improved the overall fitness of the population, rapidly at first but more slowly at higher fitness. The average fitness per generation exceeded the fitness of Traub's parameter

[12]Speedup is calculated as the serial execution time divided by the parallel execution time. Linear speedup means that the ratio for a given algorithm is proportional to the number of processors. We have seen some applications where parallelization resulted in *worse* performance than the serial version. Such applications spend a large amount of time message passing relative to the computation time that is performed.

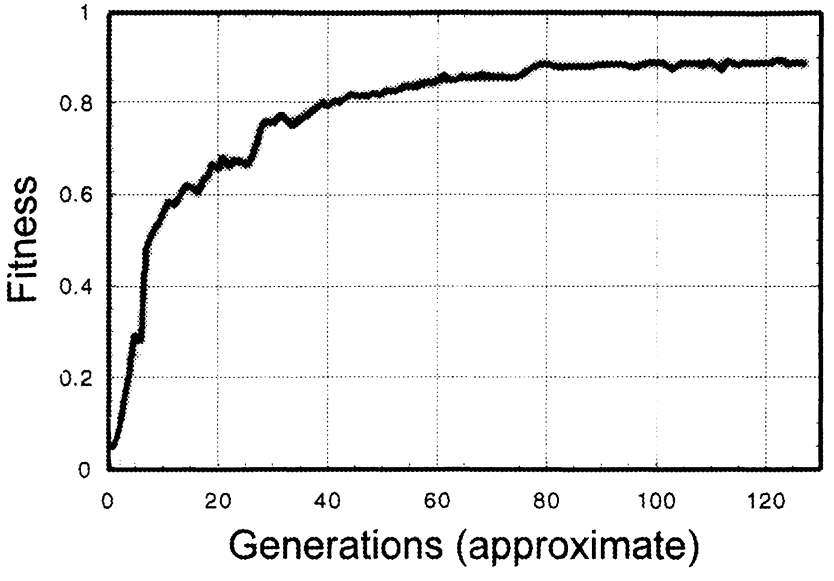


FIG. 6. *The average fitness per generation demonstrates rapid improvement. Because the population size is dynamic, the x-axis is only approximate.*

set (0.85) by generation 62. The best fitness overall (0.92) was obtained in generation 101.

The solution with the highest fitness demonstrates fast sodium- dependent bursting in the soma, low amplitude spiking in the basal dendrites, and slow calcium-dependent responses in the apical dendrites. Neither the EA-produced parameter sets nor Traub's parameter set satisfied all the fitness criteria, suggesting that additional channel kinetic equations may be necessary to achieve a higher correspondence to experimental results. A quantitative description is available elsewhere (Eichler West 1996).

We examined the manifolds for the total conductance and the excitatory/inhibitory conductance ratio. The total conductances of high fitness solutions (> 0.9) lay between 442 and 652 mS/cm^2 ($\overline{x} = 513$, s.d. $= 34$, $n = 47,484$), consistent with the total conductance of 527 mS/cm^2 proposed by Traub. The excitatory/inhibitory conductance ratio (EIR) for high fitness solutions (> 0.9) occurred in the range 0.78 to 1.31 ($\overline{x} = 0.96$, s.d. $= 0.12$, $n = 47,484$), consistent with the EIR of 0.97 proposed by Traub. Many low fitness solutions also fell within the ranges of these solutions. Thus, while selecting a parameter set within these ranges does not guarantee a high fitness solution, selection of any parameter set outside of this range yielded low fitness solutions.

8. Conclusions. This study has applied a new method for parameter optimization that promises to dramatically improve the robustness of

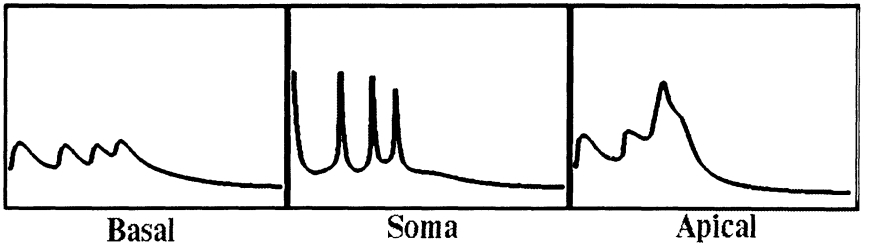


FIG. 7. *The solution with highest fitness yields spatiotemporal behaviors consistent with those observed experimentally. The y-axis range is* -85 *mV to* $+40$ *mV. The length of the x-axis is approximately* 80 *ms.*

neuronal simulations. Further, the automation aspects of the search process promises to improve the efficiency of model development. By robustness, we mean that the manifold of high fitness solutions produced by the EA application demonstrates that the model produces appropriate behaviors over a range of parameter values, reflecting a variability analogous to that observed in nature. The results of our proof-of-concept experiment yielded improved correspondence between simulated and experimental behaviors. The method should find general applicability in a broad range of HPC simulation endeavors outside of neurophysiological modeling.

The manifold approach suggests methods of analysis that would not have been possible with previous approaches to parameter fitting.

1. *The EA-generated manifold can be used to determine an area of parameter space to examine in future refining searches.* While we carefully designed our recombination operators to explore outside of the boundaries of our initial conditions, high fitness solutions confined themselves to a more restricted area of parameter space.

2. The interpretation and significance of the individual channel parameter manifolds require a much greater description of the biological system and are thus not within the scope of this chapter[13]. However, *all the channel distribution manifolds resembled noisy instances of simple*

[13]See Eichler West (1996) for a detailed description of the channel manifolds and their interpretation.

functions: linear (potassium A, calcium-dependent potassium); exponential (sodium, potassium delayed rectifier); or, sinusoidal (calcium, potassium AHP). This encouraged us to re-evaluate the potential of the function representation described in section 4.

3. *The manifolds were subsequently analyzed to reveal covariance relationships between parameters, thereby providing an indication of the sensitivity of parameters with respect to modulations in the other parameters.* The manifolds identified varying degrees of parameter sensitivity for the potassium A and the calcium-dependent potassium channels and a strong positive covariance between the sodium and potassium delayed rectifier channel distributions. The complementary colocalization of the calcium and potassium AHP distributions is suggestive of an experimentally testable role for calcium-dependent conductances in the control of spatiotemporal plasticity (Eichler West 1996).

One future goal made evident by these preliminary studies is the need to reduce the number of function evaluations required to sufficiently search parameter space by applying better EA selection strategies. The results in Figure 6 suggest that a reexamination of our termination criteria might be the first place to start. The search strategy we used required access to significant amounts of CPU time on supercomputer-class machines. We believe that our EA-based approach to neuronal modeling offers a powerful new set of methods and interpretive paradigms, but it will not be generally useful to all neural modelers until it is implementable by those who do not have access to high performance computers.

How do we know when we are including too much or too little information to the fitness measure? The concern is that additional criteria will increase the computational time required to evaluate the response of each simulation with little or no gain for parameter manifold refinement. One approach may be to divide criteria into "training and testing" sets. The set of high fitness solutions created from the "training" criteria can be evaluated for criteria for which there is no explicit selection (the "testing set"). Those testing criteria which fail must necessarily become members of the new training set.

We are first and foremost neuroscientists with collective intuition into our system gained by reviewing the experimental literature, performing experiments in the laboratory, and simulating models of neurons. In this first attempt at applying an EA, we tried to incorporate much domain-based intuition into the algorithm design. The application was quite successful. However, with respect to the NFL theorem, we are still left with two nagging questions. Would another method perform better, given a similar attentiveness to the incorporation of domain-based information? Did

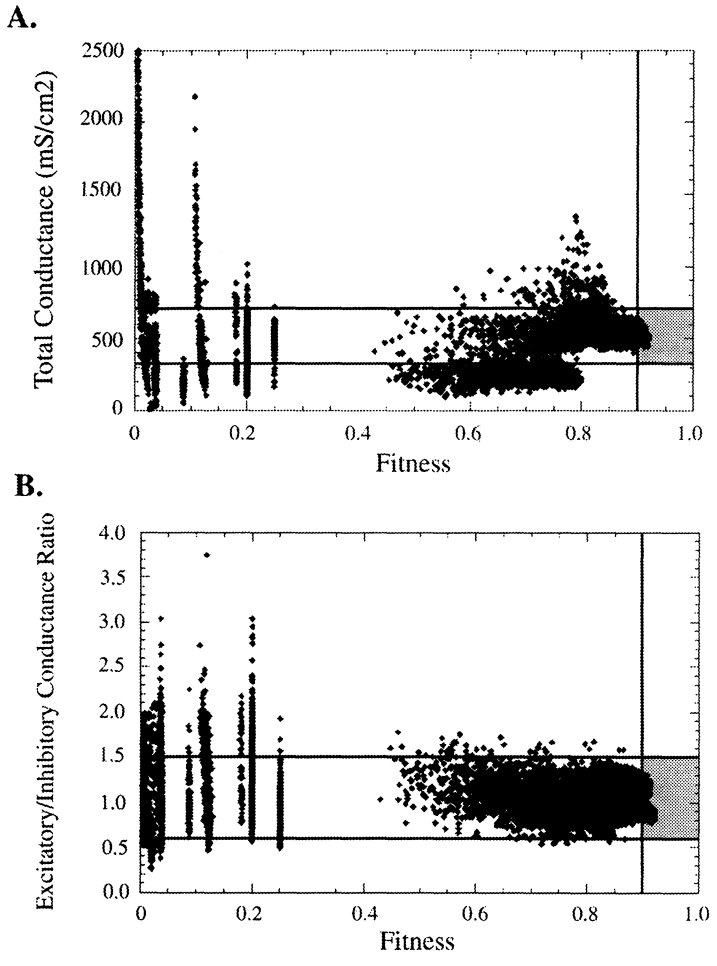


FIG. 8. *Solution manifolds for the low dimensional metrics as a function of fitness. a, Total conductance manifold as a function of fitness score for the search space explored by the EA. b, The excitatory/inhibitory conductance ratio manifold as a function of fitness score. Note that the range with respect to the y-axis decreases with increasing fitness.*

we get lucky? Answers to these questions will require significantly more empirical research.

Acknowledgements. This work was supported by grants from Cray Research/Minnesota Supercomputer Institute and NIMH R01-MH52903. The authors gratefully acknowledge generous access to supercomputing facilities provided by the Minnesota Supercomputer Institute, the IBM Shared University Research Project, and the Laboratory for Computational Sciences and Engineering at the University of Minnesota. RMEW

would like to thank the IMA and program committee for their invitation to present this work. RMEW would also like to thank Ihab Awad and Joe Haberman for their technical support with parallelization issues, and Jon Gottesman and David Yuen for thought-provoking discussions.

REFERENCES

[1] Abbott, L.F., Rolls, E.T., and Tovee, M.J., *Representational capacity of face coding in monkeys*, Cerebral Cortex 6: 498–505, 1996.

[2] Back, T., and Hoffmeister, F., *Extended selection mechanisms in genetic algorithms*, in *Fourth International Conference on Genetic Algorithms in University of California, San Diego*, edited by Belew, R.K., and Booker, L.B., Morgan Kaufmann, 92–99, 1991.

[3] Back, T., Hoffmeister, F., and Schwefel, H.-P., *A survey of evolution strategies*, in *Fourth International Conference on Genetic Algorithms in University of California, San Diego*, edited by Belew, R.K., and Booker, L.B., Morgan Kaufmann, 2–9, 1991.

[4] Bagchi, S. et al., *Exploring problem-specific recombination operators for job shop scheduling* in *Fourth International Conference on Genetic Algorithms* in University of California, San Diego, edited by Belew, R.K., and Booker, L.B., Morgan Kaufmann, 10–17, 1991.

[5] Bhalla, U.S., and Bower, J.M., *Exploring parameter space in detailed single neuron models: simulations of the mitral and granule cells of the olfactory bulb*, Journal of Neurophysiology 69: 1948–1965, 1993.

[6] Bramlette, M.F., *Initialization, mutation and selection methods in genetic algorithms* in *Fourth International Conference on Genetic Algorithms* in University of California, San Diego, edited by Belew, R.K., and Booker, L.B., Morgan Kauffman, 100–107, 1991.

[7] Chetkovich, D.M. et al., *N-Methyl-D-Aspartate receptor activation increases cAMP levels and voltage-gated Ca2+ channel activity in area CA1 of hippocampus*, Proceedings of the National Academy of Sciences of the USA 88: 6467–6471, 1991.

[8] Culberson, J., *On the futility of blind search*, University of Alberta Technical Report TR96-18, 1996.

[9] Davis, L., *Bit-climbing, representational bias, and test suite design*, in *Fourth International Conference on Genetic Algorithms* in University of California, San Diego, edited by Belew, R.K., and Booker, L.B., Morgan Kaufmann, 18–23, 1991.

[10] Davis, L. et al., *A genetic algorithm for survivable network design* in *Fifth International Conference on Genetic Algorithms* in University of Illinois at Urbana-Champaign, edited by Forrest, S., Morgan Kaufmann, 408–415,1993.

[11] De Jong, K.A., *An analysis of the behavior of a class of genetic adaptive algorithms*, Doctoral Thesis, University of Michigan, Ann Arbor, 1975.

[12] De Schutter, E., *A consumer guide to neuronal modeling software. Trends in Neurosciences* 15: 462–464, 1992.

[13] De Schutter, E., and Bower, J.M., *An active membrane model of the cerebellar purkinje cell. I. Simulation of current clamps in slice*, Journal of Neurophysiology 71: 375–400, 1994.

[14] Denk, W., Strickler, J.H., and Webb, W.W., *Photon laser scanning fluorescence microscopy*, Science 248: 73–76, 1990.

[15] Ebner, T.J., and Chen, G., *Use of voltage-sensitive dyes and optical recordings in the central-nervous-system*, Progress in Neurobiology 46: 463–506, 1995.

[16] Edmonds, B. et al., *Contributions of two types of calcium channels to synaptic transmission and plasticity*, Science 250: 1142–1146, 1990.

[17] Eichler West, R.M., *On the development and interpretation of parameter manifolds for biophysically robust compartmental models of CA3 hippocampal neurons*, Doctoral Thesis, University of Minnesota, 1996.

[18] Eichler West, R.M., and Wilcox, G.L., *A renumbering method to decrease matrix banding in equations describing branched neuron-like structures*, Journal of Neuroscience Methods 68: 15–19, 1996.

[19] Ferster, D., and Spruston, N., *Cracking the neural code*, Science 270: 756–757, 1995.

[20] Fitzgerald, K. et al., *Multiple forms of non-associative plasticity in Aplysia: A behavioral, cellular, and pharmacological analysis*, Philos Trans R Soc Lond 329: 171–178, 1990.

[21] Forrest, S., *Genetic algorithms: principles of natural selection applied to computation*, Science 261: 872–878, 1993.

[22] Foster, W.R., Ungar, L.H., and Schwaber, J.S,. *J. S. Significance of conductances in Hodgkin-Huxley models*, Journal of Neurophysiology 70: 2502–2518, 1993.

[23] Geist, A. et al., *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, Cambridge, MA: MIT Press, 1994.

[24] Georgopoulos, A.P., Taira, M., and Lukashin, A., *Cognitive neurophysiology of the motor cortex*, Science 260: 47–52, 1993.

[25] Goldberg, D.E., *Genetic algorithms in search, optimization, and machine learning*, Reading, MA: Addison-Wesley, 1989.

[26] Goldberg, D.E., *Real-coded genetic algorithms, virtual alphabets, and blocking*, Complex Systems 5: 139–168, 1991.

[27] Goldman, D.E., *Potential, impedance, and rectification in membranes*, The Journal of General Physiology 27: 37–60, 1943.

[28] Hart, W.E., and Belew, R.K., *Optimizing an arbitrary function is hard for Genetic Algorithms*, in *Fourth International Conference on Genetic Algorithms* in University of California, San Diego, edited by Belew, R.K., and Booker, L.B., Morgan Kaufmann, 190–195, 1991.

[29] Hille, B., *Ionic Channels of Excitable Membranes*, second ed., Sunderland, MA: Sinauer Associates Inc., 1992.

[30] Hines, M., *Efficient computation of branched nerve equations*, International Journal of Biomedical Computing 15: 69–76, 1984.

[31] Hodgkin, A.L., and Huxley, A.F., *A quantitative description of membrane current and its application to conduction and excitation in nerve*, Journal of Physiology 117: 500–544, 1952.

[32] Hodgkin, A.L., and Rushton, W.A.H., *The electrical constants of a crustacean nerve fibre*, Proceedings of the Royal Society of London Series B 133: 444–479, 1946.

[33] Holland, J.H., *Adaptation in natural and artificial systems*, Ann Arbor, MI: The University of Michigan Press, 1975.

[34] Holmes, R.W., and Rall, W., *Estimating the electrotonic structure of neurons with compartmental models*, Journal of Neurophysiology 68: 1438–1452, 1992.

[35] Jantsch, E., *The self-organizing universe: scientific and human implications of the emerging paradigm of evolution*, Elmsford, New York: Pergamon, 1980.

[36] Johnston, D. et al., *Active properties of neuronal dendrites*, Annual Review of Neuroscience 19: 165–186, 1996.

[37] Kallen, R.G., Cohen, S.A., and Barchi, R.L., *Structure, function, and expression of voltage-dependent sodium channels*, Molecular Neurobiology 7: 383–428, 1993.

[38] Keynes, R.D., *A new look at the mechanism of activation and inactivation of voltage-gated ion channels*, Proceedings of the Royal Society of London Series B 249: 107–112, 1992.

[39] Kido, M. et al., *Mantle viscosity derived by genetic algorithm using oceanic geoid and tomography for whole-mantle versus blocked-flow situations*, Phys. Earth Planet Int., 1998.

[40] Klein, M., and Kandel, E.R., *Presynaptic modulation of voltage-dependent Ca2+ current: Mechanism for behavioral sensitization in Aplysia californica*, Proceedings of the National Academy of Sciences of the USA 75: 3512–3516, 1978.

[41] Kuhar, M.J., and Unnerstall, J.R., *Quantitative receptor mapping by autoradiography: Some current technical problems*, Trends in Neurosciences 49–53, 1985.

[42] Kumar, V. et al., *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin-Cummings Addison-Wesley Publishing Company, 1994.

[43] Laurent, G., *Dynamical representation of odors by oscillating and evolving neural assemblies*, Trends in Neurosciences 19: 489–496, 1996.

[44] Li, M. et al., *Convergent regulation of sodium channels by protein kinase C and cAMP-dependent protein kinase*, Science 261: 1439–1442, 1993.

[45] Macready, W.G., Siapas, A.G., and Kauffman, S.A., *Criticality and parallelism in combinatorial optimization*, Science 261: 56–58, 1996.

[46] Maletic-Savatic, M., Lenn, N.J., and Trimmer, J.S., *Differential spatiotemporal expression of K+ channel polypeptides in rat hippocampal neurons developing in situ and in vitro*, Journal of Neuroscience 15: 3840–3851, 1995.

[47] Mascagni, M.V., *Numerical methods for neuronal modeling* in *Methods in Neuronal Modeling*, edited by Koch, C., and Segev, I., Cambridge, Ma: MIT Press, 439–483, 1989.

[48] Masukawa, L.M., Hansen, A.J., and Shepherd, G., *Distribution of single-channel conductances in cultured rat hippocampal neurons*, Cellular and Molecular Neurobiology 11: 231–243, 1991.

[49] Monster, A.W., and Chan, H., *Isometric force production by motor units of extensor digitorum communis in man*, Journal of Neurophysiology 40: 1432–1443, 1977.

[50] Numann, R., Caterall, W.A., and Scheuer, T., *Functional modification of brain sodium channels by protein kinase C phosphorylation*, Science 254: 115–118, 1991.

[51] Parsons, R.J., Forrest, S., and Burks, C., *Genetic algorithms, operators, and DNA fragment assembly*, Machine Learning 21: 11–33, 1995.

[52] Perezreyes, E., and Schneider, T., *Calcium channels - structure, function, and classification*, Drug Development Research 33: 295–318, 1994.

[53] Press, W.H. et al., *Numerical recipes*, The art of scientific computing, second ed., Cambridge: Cambridge University Press, 1992.

[54] Rall, W., *Theory of physiological properties of dendrites*, Annals New York Academy of Science 96: 1071–1092, 1962.

[55] Rall, W., *Cable theory for dendritic neurons*, in *Methods in Neuronal Modeling*, edited by Koch, C., and Segev, I., Cambridge, Mass: MIT Press, 9–62, 1989.

[56] Rhodes, K.J. et al., *Voltage-gated K+ channel beta-subunits - expression and distribution of KV-Beta-1 and KV-Beta-2 in adult rat brain*, Journal of Neuroscience 16: 4846–4860, 1996.

[57] Sakmann, B., and Neher, E., *Single Channel Recording*, New York: Plenum, 1983.

[58] Saravanan, N., Fogel, D.B., and Nelson, K.M., *A Comparison of Methods for Self-Adaptation in Evolutionary Algorithms*, BioSystems 36: 157–166, 1995.

[59] Segev, I., Fleshman, J.W., and Burke, R.E., *Compartmental models of complex neurons*, in *Methods in Neuronal Modeling*, edited by Koch, C., and Segev, I., Cambridge, Mass: MIT Press, 63–96, 1989.

[60] Snir, M. et al., *MPI: The Complete Reference*, Cambridge, MA: MIT Press, 1995.

[61] Spears, W.M., *Adapting Crossover in Evolutionary Algorithms*, in *Proceedings of the Fourth Annual Conference on Evolutionary Programming* in San Diego, CA, 1991.

[62] Spears, W.M., and De Jong, K.A., *An analysis of multi-point crossover*, in *Proceedings of the Foundations of Genetic Algorithms* Workshop in Bloomington, IN, 1990.

[63] Spruston, N., Jaffe, D.B., and Johnston, D., *Dendritic attenuation of synaptic potentials and currents - the role of passive membrane properties*, Trends in Neurosciences 17: 161–166, 1994.

[64] Spruston, N., and Johnston, D., *Perforated patch-clamp analysis of the passive membrane properties of three classes of hippocampal neurons*, Journal of Neurophysiology 67: 508–529, 1992.

[65] Stuart, G.J., and Sakmann, B., *Active propagation of somatic action potentials into neocortical pyramidal cell dendrites*, Nature 367: 69–72, 1994.

[66] Syswerda, G., *Uniform crossover in genetic algorithms*, in *Third International Conference on Genetic Algorithms*, edited by Shaffer, J.D., Morgan Kaufmann, 1989.

[67] Theunissen, F.E. et al., *Information theoretic analysis of dynamical encoding by four identified primary sensory interneurons in the cricket cercal system*, Journal of Neurophysiology 75: 1345–1364, 1996.

[68] Toro, L., and Stefani, E., *Calcium-activated K+ channels - metabolic regulation*, Journal of Bioengineering-B 23: 561–576, 1991.

[69] Traub, R.D. et al., *Analysis of gamma rhythms in the rat hippocampus in vitro and in vivo*, Journal of Physiology 493: 471–484, 1996.

[70] Traub, R.D. et al., *A model of a CA3 hippocampal pyramidal neuron incorporating voltage-clamp data on intrinsic conductances*, Journal of Neurophysiology 66: 635–650, 1991.

[71] Turner, D.A., *Segmental cable evaluation of somatic transients in hippocampal neurons (CA1, CA3, and Dentate)*, Biophysical Journal 46: 73–84, 1984.

[72] Turner, D.A., and Schwartzkroin, P.A., *Steady-state electrotonic analysis of intracellularly stained hippocampal neurons*, Journal of Neurophysiology 44: 184–199, 1980.

[73] Westenbroek, R.E., Ahlijanian, M.K., and Catterall, W.A., *Clustering of L-type calcium channels at the base of major dendrites in the hippocampal pyramidal neurons*, Nature 347: 281–284, 1990.

[74] Whitley, D. et al., *Comparing Heuristic, Evolutionary and Local Search Approaches to Scheduling*, in *Third Artificial Intelligence Planning Systems Conference*, 1995.

[75] Wolpert, D.H., and Macready, W.G., *No free lunch theorems for search*, Santa Fe Institute, 1995, 95-02-010.

[76] Wonderlin, W.F., French, R.J., and Arispe, N.J., *Recording and analysis of currents from single ion channels*, In *Neurophysiological Methods*, edited by Vanderwolf, C.H., Clifton, NJ: Humana Press, 35–142, 1990.

APPLYING GENETIC ALGORITHMS TO REAL-WORLD PROBLEMS

EMANUEL FALKENAUER*

Abstract. This paper outlines what the author perceives as crucial ingredients of a successful application of Genetic Algorithms (GAs) to real-world combinatorial problems. First, the importance of the Schema Theorem is stressed, pointing to crossover as the most potent force in a GA. Second, the importance of an encoding and operators adapted to the problem being solved is demonstrated, with two implications: the importance of the binary alphabet has been largely overstated in the past (in many problems it is not only unwarranted, it is detrimental), and practical GAs must be built to solve problems (i.e., sets of instances) rather than (arbitrary) functions.

The benefits of the above guidelines are illustrated by the Grouping GA (GGA), applied to three different grouping problems, namely Bin Packing and its variety Line Balancing, Equal Piles and Economies of Scale. The first application suggests a superiority of crossover-based search over a classic Branch & Bound, the second shows the superiority of the GGA over standard GAs applied to grouping problems, and the third illustrates the kind of industrial applications GAs can be called upon to solve.

1. Introduction. In this paper, we give an account of our experience in applying Genetic Algorithms (GAs) to real-world combinatorial optimization problems. First of all, we explain why we believe at all that GAs are a good optimization method for some problems. We identify a general strategy common to most iterative metaheuristics and argue that the GA implementation of that strategy, namely the crossover operator, has an appeal that is missing in the others. The crossover operator is the reason why we find the GA more appealing than other metaheuristics.

But there is of course a catch. Indeed, the crossover cannot function properly in an arbitrary setup. Since it combines parts of solutions into new solutions, the act of combination performed by the crossover must make sense with respect to the problem being solved. Indeed, we give examples of crossovers, not very different from what one can often read in the literature, which clearly make no sense: the probability of the recombinant to have any value at all is basically nil. This is due to the purely syntactical nature of the crossover, which does not "see" the meaning of the operation it performs.

One insight gained from the examples is that despite numerous arguments on the necessity of small alphabets, the importance of the binary representation has been largely overstated in the past. More importantly though, the examples lead us to the conclusion that the encoding and operators must be adapted to the particular optimization task on hand. In order to comply with that necessity in practice, we pledge to abandon *functions* as targets of GA optimization, in profit of optimization *problems*.

* Department of Applied Mechanics, Brussels University (ULB), CP 165, Av. F. D. Roosevelt 50, B-1050 Brussels, efalkena@ulb.ac.be

In order to support the above by practical results, we present the Grouping GA (GGA), a GA specifically designed for the class of grouping (partitioning, clustering) problems. The GGA follows the above guidelines by using a representation and operators that exploit the structure of those problems. That means in particular that the crossover constructs progeny by inheritance of properties of the parents that are meaningful in the context of the grouping problems. The price to pay is to abandon the simplicity of the standard GAs, even though it looks that they might be applied (and indeed have been in the past). The reward is a significant increase in performance.

We illustrate the reward gained by three applications of the GGA. A direct comparison of a GGA for the Equal Piles Problem with other GAs applied to the same data set shows that the GGA outperforms the standard GAs significantly. The Hybrid GGA for Bin Packing illustrates the potential of well-designed GAs: the HGGA outperforms a recent sophisticated Operations Research method for the problem, making it probably the current best method for Bin Packing. Finally, the GGA for Economies of Scale is an industrial application, illustrating the kind of problems GAs can be called upon to solve in the real world.

We terminate with the following conclusions. The bad news is that "the GA" is *not* a universal technique. Indeed, a poorly designed GA, be it a classic "standard GA", can perform nearly as badly as a random search. The good news is that a carefully designed GA has the potential to perform as well or better than any other method for the problem it targets. In short, the GAs are a *paradigm* that *must* be adapted to the task on hand.

2. Why a GA? Metaheuristics, such as Simulated Annealing (SA), Tabu Search (TS) and GAs, as already their name suggests, are methods that should be applicable to a very wide range of optimization tasks, possibly any. With the enormous diversity of those tasks, there is very little that could be exploited in addressing all of them. In fact, we believe that there is just one fully general rule that all metaheuristics follow to avoid a purely random search, namely "Stick to what you already know is good". In other words, in searching a vast space of possible solutions, in order to find a good one, one should use some sort of "inspiration" from the best solution(s) so far discovered.

Of course, it is impossible to follow the rule fully, because there would then be no search at all, so some reasonable digression must be adopted. The one usually adopted rests on the assumption that it is smarter to look in the vicinity of what is good than to look just anywhere (as in random search).

So both SA and TS explore the search space by iteratively looking in a usually small neighborhood of the current (best) solution. They use different mechanisms (probabilistic acceptance of deteriorations depending on temperature in the first, the tabu list in the second) to insure that a

distant point in the search space can be accessed anyway, should the current best solution turn out to be in fact a poor one.

The GA follows the general rule as well. The originality of the GA approach lies with the fact that, unlike in SA and TS, the definition of what is good is not the whole solution but its *parts*. Thus the basic assumption in a GA is that if a solution is good (with respect to the optimization task being performed), then it is probably because some of its parts are good. The important point is that this applies to *two* solutions as well: should two solutions be good, then it is probably because *each* of them contains parts that are good, although it is not necessarily (indeed, rarely) the same ones in both solutions.

Having adopted this assumption, the question is how it can be used in order to follow the metaheuristic "sticking rule" outlined above. It turns out to be relatively simple: since preserving parts of solutions is the aim, new solutions can be constructed by combining in one solution parts coming from two different solutions (it is impossible to preserve all parts of just one solution and yet construct a new one). This is of course nothing else than the idea of recombination, or *crossover*.

The problem is that we never really know which part(s) of a good solution are the ones that make it a good solution, because we only have a measure of worth of the whole of it (the objective function). The parts must thus be tested. But again, it is impossible to test a part of a solution, only whole solutions can be evaluated. It thus comes as an appealing idea to judge the quality of a part by the quality of *various* solutions that can contain it.

We thus come to the idea of *sampling* the set of all possible solutions that contain a given part, in order to estimate the worth of the part. The idea of course leads to the question of which parts should be tested (i.e., sampled) at what frequency in order to minimize the expected time spent on testing the parts that do not lead to good solutions, which is the question studied by Holland [7].

Now the work of Holland on the Two-armed Bandit that led to the Schema Theorem was done under strong simplifying assumptions, which are hardly true in a realistic GA. Indeed, we give below examples where they are almost certainly *not true*. Still, the idea of *combining* parts of solutions in order to create promising new solutions is extremely appealing (we often adopt that approach in daily life) and absent in the other metaheuristics. Moreover, we are convinced that by carefully designing a GA, one can come close to the ideal conditions of Holland's theory, which means that the search performed by the GA will come close to the optimal sampling rates sought by Holland. These are the reasons why we prefer the GA to other metaheuristics *wherever we can come reasonably close to the ideal conditions*. We will say more on this below.

3. Is the schema theorem useful? The well-known Schema Theorem of Holland [7] shows that the parts of solutions (schemata) that are observed to perform well (i.e., they are parts of good solutions) will be sampled with increasing frequency. The theorem takes into account the various losses of a schema in a generation and shows that under proportional selection, an above-average schema will receive an exponentially increasing number of copies over the generations. This approximates the ideal sampling rate given by the Bandit.

Nevertheless, the usefulness (not the validity) of the theorem has recently come under fire. It has been said that it is only a sort of tautology, only showing how good schemata are lost. We do not share that opinion.

Indeed, suppose that we would like to know whether this paper encounters a success by its readers. Rather than asking every person that read it whether or not they liked it (and how much), a more practical idea is to sample the readership, i.e., to perform a *poll* of a subset of the set of readers in order to estimate the overall success of the paper. Now asking several persons whether they liked it and averaging the results is very different from asking several times just *one* reader. While in the former case we might get a reasonable estimate of the average success with all readers, in the latter case we will just get the impression of that one reader, which is very far from a representative sample of the whole readership.

The same observation applies to the sampling performed by a GA. Although the Schema Theorem estimates the number of copies allocated to a schema, it is crucial to realize that in order to be of any use, those copies must be *different solutions*, i.e., different points of the search space belonging to the (same) schema.

In other words, mere reproduction is useless: a second copy of the same solution, i.e., a second sample performed in the same point of the search space, does not add any new knowledge on the expected quality of the schemata containing that point. It is equivalent to a second time we ask the same question to the same reader above. Schemata must be tested in *different* contexts, in different points of the search space. The instrument that meets the two criteria of (1) resampling with increasing frequency *the same* schemata that have been observed above-average, yet (2) *not* sampling the same points of the search space (i.e., sampling the schemata in new contexts), is the crossover operator[1].

As a result of these observations, we find the Schema Theorem quite useful, because it shows that, at least for the short low-order ones, old schemata are being tested in new contexts by the GA.

All of this is not to say that mutation is useless or "bad" in any sense.

[1]Mutation also creates a new point containing some of the old schemata. However, in ideal conditions, *all* of the schemata in a crossover child are inherited (i.e., are old schemata receiving an additional sample), while even in ideal conditions some of the schemata in a mutated individual are selected at random, violating the metaheuristic "sticking rule".

Indeed, in absence of diversity (a situation inevitable with crossover alone), crossover would again fall into the trap of nonrepresentative sampling, because the progeny would be identical to the parents as in reproduction. The Schema Theorem would become useless indeed. What we mean is that crossover gives the GA an advantage over other metaheuristics. Without crossover, given the fact that the GA lacks the additional instruments of SA or TS (temperature, tabu list), the GA would have little chance to perform better than the other metaheuristics. In this sense then, we consider crossover to be the driving force in the SA.

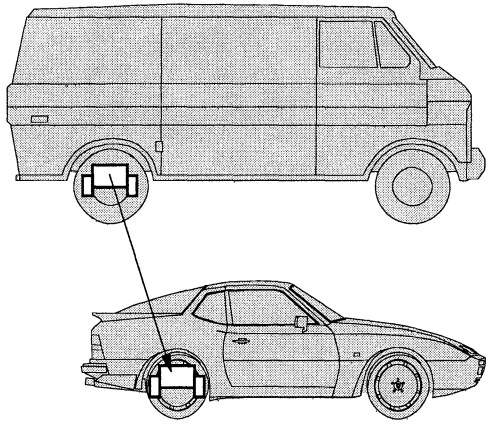


FIG. 1. *Inheriting brakes.*

4. The catch. So the reason why we prefer the GA to other heuristics is the concept of combining parts of good solutions to produce new ones. For instance, in designing a new car, this could correspond to taking the engine from a successful car and brakes from another successful car, like in figure 1, where the brakes from one car have been used in the design of a new one.

Now in the car example, we have supposed that crossover has at its disposal a description of the two cars that allows it to "extract" the brakes in a meaningful way. That is not necessarily the case. Indeed, suppose for instance that the cars are described by measurements of all their parts relative to the front left corner of the car. That is actually the way data (3D CAD models) are encoded by most of the automobile industry, so it could be considered as a sort of "standard" encoding. The same inheritance as in the previous figure[2] would this time yield figure 2.

Obviously, the inheritance performed in figure 2 makes no sense. Even if the brakes inherited were indeed very good, it would not show in the

[2]Under a standard (cut-and-paste) crossover.

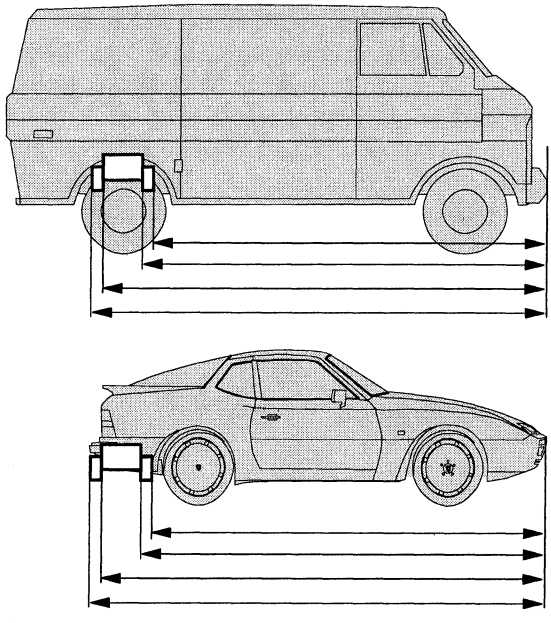


FIG. 2. *Inheritance under a "standard" encoding.*

new car, because the brakes could not function correctly. Yet the crossover cannot figure out that the position of the brakes is incorrect for their performance, because crossover is completely syntactical, i.e., it does not see the *semantics* of the data it manipulates, and will thus try to put the brakes where the wheels are not.

Now one of the objectives of inheritance is testing parts of solutions in different contexts, but the result of the test in the lower part of figure 2 is known in advance: "This car does not brake correctly." Given the fact that the result of the test is the same whatever the actual quality of the brakes, it follows that the inheritance *as performed in figure 2* makes no sense and will not make crossover a useful operator.

Admittedly, the example in figure 2 looks silly, but it is unfortunately not very far from many GAs in use today. But the approach we will describe now will be much more familiar. Suppose that brake models are numbered, and cars are described by numbers of their various parts, as illustrated in figure 3. The idea is to leave most of the handling of the semantic contents to the human operator, so that the various parts of the future car will be assembled appropriately[3].

[3]The enormous overhead associated with the human evaluation of the cost function is neglected here for the sake of the example. Besides, it is in principle conceivable to have an automatic evaluator incorporating the semantics, i.e., building a functional car from the various model numbers.

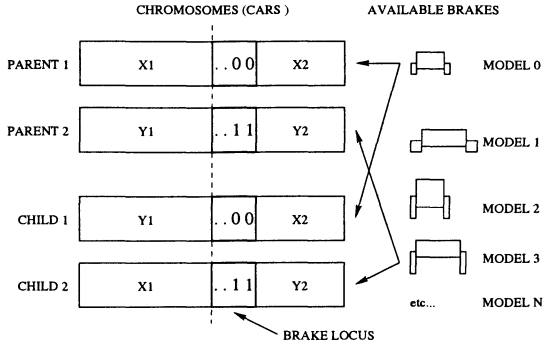


FIG. 3. *Inheriting model numbers in N-ary.*

As long as whole model numbers are transmitted as in figure 3, the sampling of the various brake models is appropriate. Indeed, as the figure shows, two successful cars having the brake models 0 and 3 respectively, transmit those models to their progeny[4]. However, it has often been argued that the alphabet used in GAs should be as small as possible, indeed binary.

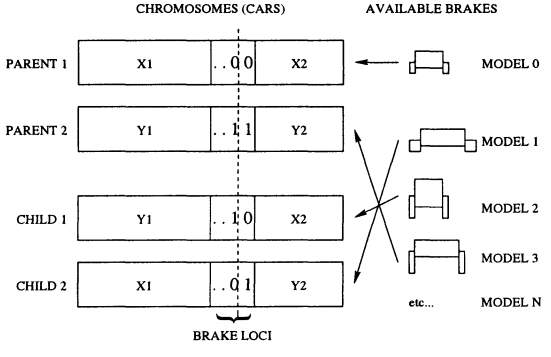


FIG. 4. *Inheriting model numbers in binary.*

Figure 4 shows the situation where the same brake model numbers are this time encoded in binary, and two children are produced from the same parents by the standard one-point crossover. Since the crossing point is no longer constrained to model number boundaries, it can (and most often will) fall inside the bits representing the number.

We see that in these conditions, the brake models of the parents were *not* inherited, while the progeny received brake models not present in either of the parents. That means that the brake models that should have been resampled (because they made it into the parental set) were not, while

[4]The standard one-point crossover was used in this example.

models that should not have been resampled (because they did not make it into the parental set) were.

In short, representing the brake model numbers in binary leads to a *complete malfunction* of the crossover. The implicit statistics that the crossover is performing have *no* sense in this example, because promising brake models[5] are not resampled, while the under-performing ones are.

Note that we have used the low-disrupting one-point crossover in figure 4, which allowed us to correctly inherit at least the rest of the cars, if not the brake models. Should we have used the all-too-popular Uniform Crossover of Syswerda [12] instead, the failure would have been complete: the progeny would most probably represent cars constructed with parts selected completely at random.

Now of course one could argue that if the numbering of the brake models is done correctly, it will reflect some inherent order which might be correctly exploited by the binary GA after all. One might think for example of sorting the brake models according to their size, their weight, or maybe their braking power. But it is not difficult to see that such an a priori ordering is not known in practice, for if it was, the problem would in fact be trivial (just choose the best of each part for the future car).

What are the conclusions of the car construction example? First, we believe that the importance of the binary alphabet has been largely overestimated in the past. A lot of ink has been consumed on arguments showing that under binary encoding the GA has an easier task combining the genes in various (supposedly useful) ways. Unfortunately, rarely (if ever) was the *meaning* of those genes taken into account. It is our conviction that figure 4 illustrates the point that if that question is not asked *first*, then the results can be very far from satisfactory.

Of course, there are functions for which binary is the natural expression, for instance Holland's Royal Road. Indeed, these functions have been created using bit building blocks from the start. However, we believe that *many* practical problems treated by GAs fall into the car construction category. For those, the use of binary encoding (especially in conjunction with the Uniform Crossover) is not only unwarranted, it is *detrimental.*

The second conclusion is that there *must* be some prior knowledge of the function being optimized put into a successful GA, otherwise the encoding/operators will most probably not make much sense with respect to the function. Now this contradicts the widely held belief that GAs can efficiently optimize *any* function. Indeed, we believe that we should refrain from presenting the GA as a universal optimization method. In this we fully agree with the conclusions of the No Free Lunch Theorem. We find it much more appropriate to present the GA as a high-level *paradigm*, which

[5] That is, brake models found in successful *cars*.

is made into a *method* by fitting the encoding and the operators to the function being optimized.

Now it may look that we have put into question the validity of the Schema Theorem, because schemata follow it however defined. Well, doing so would be foolish, because the Schema Theorem is indeed a theorem and not a conjecture. Rather, what we tried to show is that if the building blocks manipulated by the crossover do *not* represent meaningful regularities of the optimized function, then the *premises* of the Theorem do not hold. Indeed, the Theorem is only useful when the sampling of the hyperplanes the GA performs makes sense, that is, when a sample of points in a hyperplane represents a reliable estimator of the performance of *other* points in that hyperplane. If that is not the case, the Schema Theorem is not applicable (albeit still valid), because the implicit statistics it performs do not make sense: while schemata *are* sampled according to the Theorem, there is little that can be learned from them about the function being optimized.

Already Holland [7] acknowledges the crucial importance of adequate encodings. Indeed, when discussing some special genetic operators (dominance, segregation, translocation, intrachromosomal duplication, etc.), he writes (p.117)

> "Moreover they do *not* compensate the major shortcoming of genetic plans[6] which use just the first three operators [i.e., crossover, mutation and inversion] described. That shortcoming is the **complete dependence of such plans upon the detectors**[7] determining **the representation**[8]. If the set of detectors is inadequate, in any way, the plan must operate within that constraint."

(his italics, though our bold).

Yet curiously, that "complete dependence" never explicitly shows up in the theoretical treatment of the GA given in the book. It thus follows that the conclusions concerning the (near-)optimal sampling rates of schemata are based on premises that are not always satisfied, i.e., the GA does not always perform an optimal sampling of the search space. Note that this could have been expected, because otherwise the GA would be "the best" algorithm for all functions, thus contradicting the No Free Lunch Theorem of Wolpert and Macready [14].

The major problem causing a divergence from the (near-)optimal sampling rates seems to be *epistasis*, strong mutual interaction among genes[9]. Of course, in *any* nontrivial function there is at least a mild degree of

[6] He uses in his book this term for what has since then become known as the Genetic Algorithms.

[7] He uses this term for the genes.

[8] *I.e.* the encoding.

[9] The dynamic (i.e., nonstationary) nature of any practical GA is also a problem.

epistasis, because otherwise the problem would be trivial: a hill-climber setting the optimal value for the genes one by one would solve the problem to optimality[10].

However, as the above examples illustrate, many forms of epistasis are in fact *artificial*: they are not inherent to the function, they are simply caused by inadequate encoding/operators. The whole message of the above, and indeed of the rest of the paper as well, is that although epistasis cannot be avoided completely, it can and *must* be minimized in a well-designed GA, by a careful design of the encoding and the operators. We are convinced that *only then* can a GA live up to its potential and compete with other optimization techniques.

5. Let's be practical. We have seen that the GA approach only becomes reliable when knowledge of the function being optimized is used in constructing meaningful encoding and operators. However, suppose we want to use a GA for daily scheduling of operations in a factory. Surely, our remarks seem to indicate that a *new* GA should be designed each day, because the function to optimize (as defined by the tasks to be scheduled, the availability of resources, and so on) changes every day. Writing a new GA every day would be impractical, to say the least. So what can be done? We believe that a good approach is to abandon the concept of *function* being optimized and use instead the concept of *problem*, as it is used in classic complexity. In short, we should write GAs for problems and not for functions. The reason is that while each problem defines an infinity of functions (one for each instance of the problem), they all share the *structure* of the problem, and that structure is what we can (and *must*) exploit in designing efficient GAs.

6. The grouping GA.

6.1. The grouping problems. We have applied the above insights in the design of what we call the Grouping GA, or GGA. Simply put, the GGA is a GA for grouping problems. Grouping (or partitioning or clustering) problems are those where the aim is to *group* together members of a set (items) into disjoined subset in some optimal way or, equivalently, find a good *partition* of the set. Some well-known grouping problems are Bin Packing, Graph Coloring, Load Balancing (also called Equal Piles) and Machine Cell Formation in Group Technology.

What is essential about grouping problems is that the objective function to optimize is defined on the set of all (valid[11]) groupings, i.e., it depends on the *composition of the groups* (subsets) of the items.

[10] This observation has been made before. The author apologizes to its author for not having the primary reference handy.

[11] Many practical grouping problems feature hard constraints which make some groupings invalid.

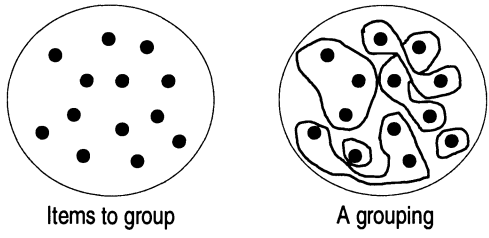


FIG. 5. *The grouping problems.*

The idea behind the GGA is extremely simple, once we agree with the arguments put forth in the previous sections. Since the objective function in a grouping problem depends on the composition of the groups in a solution, it follows that the composition of a group has a natural meaning in a grouping problem. In other words, if we want to follow the above reasoning, a successful GA for grouping problems should feature a crossover operator that transmits *groups* from parents to progeny.

The idea is arguably quite trivial, but before explaining how it is actually implemented in the GGA, let us see how have been grouping problems tackled by GAs in the past.

6.2. Previous GAs for grouping problems. Apart from one notable exception, grouping problems have been tackled by GAs in two ways. One used the standard Holland-style encoding and operators, while the other used the ordering GA.

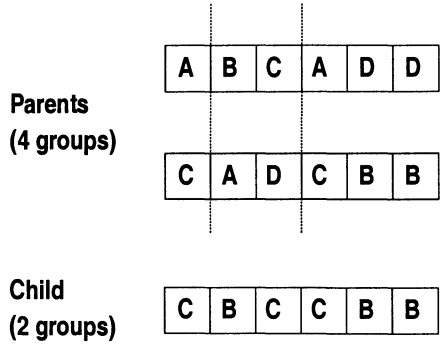


FIG. 6. *Context insensitivity of standard crossover.*

Group-numbers and standard GA. The standard GA for grouping problems uses chromosomes composed from group-numbers. There is one gene per item, the gene encoding the number of the group the item belongs to in the solution encoded by the chromosome. For instance, the

chromosome $ABCADD$ would encode[12] the solution where items 1 and 4 belong to the group A, item 2 to the group B, 3 to C and 5 and 6 to the group D. The appealing feature of that approach is that the number of items is a constant of the problem instance being optimized, which leads to the standard constant-length chromosomes. Appealing as it may seem, this encoding, when used in conjunction with standard operators (as has always been the case) suffers a number of drawbacks identified in [2]. Since it is the crossover operator that we concentrate on in this paper, let us see its effect.

The main problem with the standard crossover applied to the group-number chromosomes is what we have termed its *context insensitivity*: The crossover transmits genes that represent isolated items, but one isolated item has no meaning in a grouping problem, it only has a meaning in the context of the group of other items that are in the same group. This is because by definition of grouping problems, the objective function is defined on the groups, not isolated items.

One of the problems is illustrated in figure 6, where the standard two-point crossover was used to construct a child of two chromosomes. Under an appropriate crossover, two identical parents should recombine into progeny which is identical with the parents. The two parents in figure 6 *are* identical, because as the reader can check, they both encode the same grouping. However, the child is very far from being identical with the parents, because the solution it encodes features just two groups instead of the parents" four. It could be argued that a renaming of the groups in the example would alleviate the problem, but surprisingly, apart from the exception described in section 6.2 below, few authors have adopted that improvement. Note however that even that does not help in general, unless the problem has a special structure, like the one exploited below.

As in the car construction example above, the *meaning* of the genes is ignored in the crossover in figure 6. Parts of the parental solutions that have a meaning with respect to the problem being solved[13] and are thus supposed to be retested (resampled) through inheritance, are lost during the crossover, which means that the crossover does not perform the intended implicit statistics.

Ordering GA and a decoder. The other previous GA approach to grouping problems uses an ordering GA coupled with a decoder. The chromosome represents a *permutation* of the items and the decoder constructs the grouping in some way dependent on the permutation. The value of the objective function is then computed on the resulting grouping.

The decoders used are of course different for different problems. The overall idea is illustrated on an ordering GA for the Bin Packing problem in

[12] Under the convention that the genes correspond to the items according to the left-to-right order.

[13] Because they influence the value of the objective function.

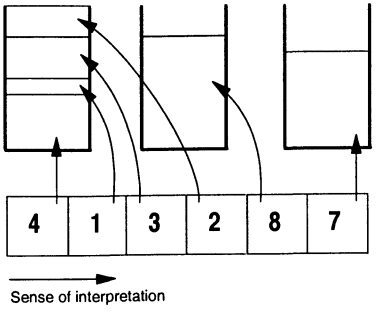


FIG. 7. *Ordering* GA *for Bin Packing.*

figure 7, where we labeled the genes with the sizes of the items they stand for. The value of the objective function is computed on the packing obtained by the following decoding: the items are taken in the order specified by the chromosome (from left to right), putting the current item into the current bin if possible, and requesting a new bin if not. The ordering GA is supposed to find a permutation that will allow the decoder to construct a good, maybe optimal solution to the grouping problem.

As for the GAs using the group-number encoding above, Falkenauer [2] identifies several drawbacks of this approach, but let us concentrate here on the effect of the crossover operator. Several ordering crossovers are in use today, but for the sake of the argument, we will use Goldberg's PMX, probably the most widely cited in the literature. The situation is illustrated in figure 8.

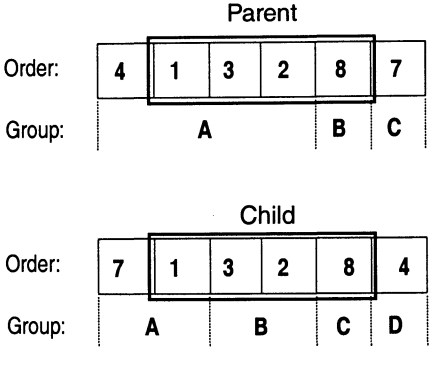


FIG. 8. *Effect of* PMX.

The PMX transmits the absolute positions of a crossing section taken from one parent, and the relative positions of the other genes taken from the other parent. Clearly, the chromosome in the lower part of the figure

is a valid child of the chromosome in the upper part[14].

Under an appropriate crossover operator, a child should inherit most of the meaningful information from one or the other parent. This is far from being the case in the example depicted. Indeed, Bin Packing is a grouping problem, but the groups obtained by decoding the child chromosome have very little in common with those obtained from the parental chromosome: the number and, more importantly, the contents of the groups are very different (only the group containing the item of size 8 made it into the child). Once again, the groups that should be resampled by inheritance through crossover are not, flawing the implicit statistics the GA is supposed to perform. Other ordering crossovers suffer similar problems. As for the group-number approach, context insensitivity is the cause. Indeed, the meaning of a gene, i.e., the composition of the group it is assigned to, only has sense in the *context* of the other genes in the chromosome: the groups obtained by the decoder strongly depend on the positions of a vast majority of the genes in the chromosome, and only little of that context can be transmitted by simple ordering crossovers.

A seemingly clever argument would object that an "intelligent" decoder would do a much better job than the one we used in the example. There are two answers to that objection. First, it is usually not difficult to construct examples where even a relatively smart decoder will do a poor job (i.e., construct a bad grouping).

The second answer is much more fundamental though. It consists of observing that the decoder does not participate in any way in the process of inheritance. Consequently, should indeed the decoder be so good that the solutions it produces are most of the time of high quality, then most of the optimization job would in fact be done by the decoder. Why bother with the GA then ? Indeed, in that case the crossover would be useless, and other techniques (like SA or TS, for instance) would most probably be more appropriate.

One notable exception. To our best knowledge, there has been in the past just one GA approach to grouping problems not following the two described above. Mühlenbein [10] and Von Laszewski [13] used the group-number encoding in solving the K-way Graph Partitioning problem, but realized that the standard crossover did not follow the structure of the problem: they rightly concluded that in order to make sense, the crossover had to transmit *whole groups.*

To remedy the drawback of the standard crossover, they took advantage of the particular structure of their grouping problem: all partitions (groups) were required to be of the same size, i.e., to have the same number of items. This allowed them to use an "intelligent" crossover instead of the standard one.

[14]The other parent, not shown in the figure, has the genes 4 and 7 permuted.

Their crossover exploits the renumbering "trick" suggested above. In the beginning of the crossover, a partition, say group A, is selected at random in one parent and the *most similar* one, as defined by the number of items in common, is identified in the future child (a copy of the second parent). That group is then renumbered, so that its items have the same group-number A as the group in the first parent.

The group in the first parent is then imposed on the child, so that the set of items having the group-number A be identical. Thanks to the property of equal-sized groups, this can be done by switching to A items that have not been its members, and distributing among the other groups items that were in A in the child but not in the first parent. At the end, the child has inherited the group A.

Apart from the fact that it seems to require the (rare) condition of equal-sized groups, and the apparent limitation of the crossover to transmit just one group, this approach certainly constitutes a more sensible approach to grouping problems than the standard ones described earlier. Indeed, this one is based on the principle of inheritance of whole groups, i.e., transmission of information meaningful to the problem being tackled.

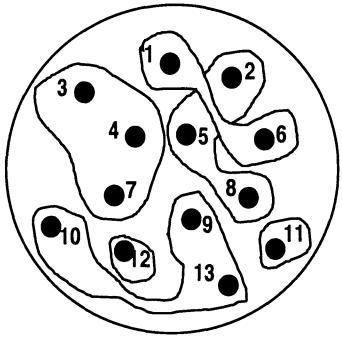
6.3. The algorithm. The Grouping GA (GGA) was born from an almost trivial observation: since in grouping problems it is the groups (i.e., their composition) that count, a GA appropriate for those problems must manipulate groups, rather than isolated items.

In designing the GGA, we have followed a simple strategy: make the regularities of the target problems into the *genes* of the chromosomes manipulated by the GA. If necessary, adapt subsequently the standard operators (crossover, mutation, inversion) in order to accommodate the non-standard chromosomes. In short, we first selected an *encoding* in such a way that meaningful regularities of the search space are explicit in the chromosomes, and then adapted the operators to that encoding.

The other possibility, illustrated by the above work of Mühlenbein and Von Laszewski, would consist in using a standard encoding and modifying the operators in a more extensive way, so that their operation would boil down to manipulating the regularities implicitly. The two approaches are largely equivalent [11]. We prefer the first approach for a simple reason: if we manage to (re-)define the operators in a way that follows the spirit of the standard operators to a reasonable extent, the large body of work already done on the standard operators (manipulating the standard chromosomes) will at least approximately apply to our operators as well.

The details of the GGA have already been presented elsewhere (*e.g.* [2, 3]), and we will not repeat them here. Instead, we will concentrate on its two essential features, namely the GGA encoding and the crossover operator.

The encoding. The encoding used in the GGA is illustrated in figure 9. Following our observation that the groups represent the meaningful regularities in grouping problems, there is one gene per group. The items in the set being partitioned are "inside" the genes.



2,5,8	9,13,10	12	3,4,7	11	1,6

FIG. 9. *The* GGA *encoding.*

The order of the items "inside" a gene has no relevance—only the composition counts. Also, the position of a gene on the chromosome is irrelevant.

The encoding has several peculiarities. The most visible one is that unless the constraints of the grouping problem being solved impose a fixed number of groups, the length of the chromosome is variable.

A more interesting property of the encoding is that there is no clear distinction between locus and allele: the meaning of a gene is solely defined by the composition of the group it encodes. If we take that composition for the locus, then there is no real notion of allele[15].

Finally, if we do insist that the composition of the group is in fact the allele (meaning that there is then no real notion of locus), then the alphabet of the encoding is not only exponentially large, but also dependent on the *instance* of the problem being solved.

One nice property of this encoding is that it is *nonredundant*: just as a solution to a grouping problem consists of a set of subsets of the items, the GGA chromosome is a set of genes, each representing one of the subsets in the solution. The more important nice property of the encoding is of course that it represents the regularities of the grouping problems as genes.

The GGA crossover. As we said above, the GGA crossover operator is basically the standard (two-point) crossover, adapted to work on the

[15] Or, if one insists, the allele would have only one possible value, namely "present".

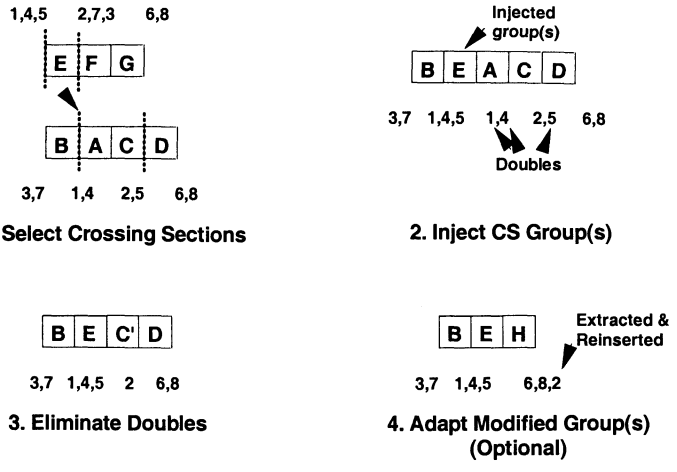


FIG. 10. *The* GGA *crossover.*

GGA chromosomes.

Since grouping problems are a rather diverse lot, the crossover is *not* the same for each of them. However, it always follows the pattern depicted in figure 10.

Given the particularity of the chromosomes it works with, the GGA crossover cannot proceed by a simple cut & paste as do the standard crossovers operating on chromosomes using a standard (orthogonal) encoding. It functions as follows.

First, a crossing section is selected at random in the first parent. In the case depicted in the figure, this is the gene *arbitrarily* labeled E. As in the standard crossovers, that crossing section is then "exported" into the other parent. Under a standard encoding, the genes in the crossing section would simply replace the genes in the other parent that have the same loci, but since there is no real notion of locus in the GGA encoding, a more sophisticated mechanism has to be adopted.

So our crossover *injects* the crossing section into the other parent. However, since each of the genes represents a group of items, this implies that in the solution represented by the child chromosome at this stage (stage 2 in the figure) contains some items twice.

Since double occurrences of items are not allowed in a grouping problem, the doubles are now eliminated by elimination of the "old" occurrences of the items just injected, as in stage 3 in figure 10. Thus the group membership of *some* of the items in the second parent gives way to the group membership defined by the first parent.

At this stage, the child chromosome represents a valid solution to a grouping problem and could be used as it is. However, in the process of elimination of double occurrences of items, some of the groups coming

from the second parent were modified yet not completely eliminated, like the gene C' in the figure. Depending on the objective function of the particular grouping problem being optimized, such groups are probably of poor quality (*those* groups did not make it into the parental set). This last problem can be fixed in the optional stage of *adaptation*: the modified groups are emptied and the items they contained are reinserted into the solution. This is usually done following a heuristic adapted to the problem.

The fundamental property of the GGA crossover is apparent in figure 10: the child has inherited the group labeled B from the second parent, and the group numbered E from the first parent. In other words, the child inherited information meaningful to the problem being solved from *both* parents. The GGA crossover thus carries out its intended mission.

6.4. Three GGA applications. The GGA has been successfully applied to a number of real-world problems. We describe here three of them in order to illustrate the possibilities and the performance of the algorithm. All of them have been detailed elsewhere, which will allow us to concentrate on the fundamentals and not excessively lengthen the paper.

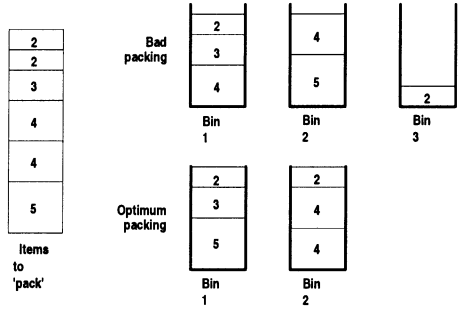


FIG. 11. *The Bin Packing problem.*

Bin packing and line balancing. The well-known Bin Packing problem is illustrated in figure 11: a set of items, each of a given size, must be "packed" into as few "bins" of a given capacity as possible. In other words, the set of items has to be partitioned into a minimal number of groups, under the constraint that the sum of item sizes in any group does not exceed the bin capacity.

Bin Packing has a generalization of great practical importance, the Line Balancing problem. With the items representing assembly operations, their sizes representing the durations of those operations, and the bin capacity corresponding to the cycle time of a paced assembly line, the problem becomes one of designing an assembly line that carries out all of the operations within the limit of the cycle time using the *least workforce*. The problem is usually generalized by imposing *precedence constraints* among the operations: the operations must be assigned in such a way that there is

no cycle between the groups of operations, so that a conveyor can connect all groups in one direction.

We have proposed a "straightforward" GGA for the problem in [5]. That algorithm was basically a "raw" GGA: only the simple First Fit and First Fit Descending heuristics exploited the structure of that particular problem, the former in the construction of the initial population, the latter in the stage 4 of the GGA crossover (see figure 10). Even with such "naive" algorithm, the performance was interesting. That paper also shows how compliance with precedence constraints can be achieved with little computational overhead, making the GGA applicable to the Line Balancing as well.

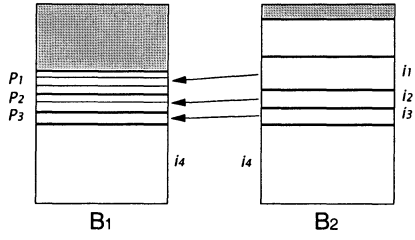


FIG. 12. *The Dominance criterion.*

The algorithm was later enhanced in [4] by hybridization with an elegant Operations Research technique, the Dominance Criterion of Martello and Toth [9].

The idea of dominance is illustrated in figure 12: given the contents of two bins B_1 and B_2, if there exists a subset $\{i_1, \ldots, i_n\}$ of items in B_2 and a partition $\{P_1, \ldots, P_n\}$ of items in B_1 such that for each item i_i there is a no bigger[16] corresponding P_i, then B_2 is said to *dominate* B_1, because a solution obtained with B_2 as one of the bins requires no more bins than one with B_1. So for instance in the case depicted in figure 12, among the two bins containing the item i_4, B_2 is preferable to B_1.

In the hybrid GGA, in the stage 4 of the crossover, we perform a *local optimization* inspired by these ideas in the following way. First, we empty the groups modified by the injection. Then, taking one by one the bins so far in the solution[17], we check whether it is possible to *replace* up to three items in the bin by one or two items from those currently missing in the solution, in such a way that the total size of the items in the bin increases without overflowing the bin. If so, we perform the replacement, which means that some of the previously unassigned items are assigned to the bin, while some of the items previously assigned to the bin become

[16]That is, the total size of items in P_i is less than or equal to the size of i_i.

[17]These are the bins coming from the first parent, as well as those which were not affected by the injection.

"unassigned". When no such replacement is possible anymore, the usual First Fit Descending heuristic is applied to complete the solution.

We have compared the hybrid GGA to the Branch&Bound method MTP of Martello and Toth [9], considered by many to be the best OR method for Bin Packing. The detailed results obtained on 160 difficult instances can be found in [4], so let us just spell here the main conclusion: as soon as the size and structure of an instance made it a truly difficult one, the hybrid GGA supplied a *substantially better* result in a *substantially shorter* time than the MTP. In fact, the GGA found a globally optimal solution in all but three cases out of the 160, while the MTP hardly ever did even when given more computational resources.

The performance of the hybrid GGA in comparison with the MTP is important in the following respect. Both the GA and the MTP use the same *local* search devices. Indeed, the GGA uses the Dominance Criterion and the First Fit Descending heuristic, and both of these are embedded in the MTP as well. So the main difference between the two algorithms lies with the *global* search strategy: like all GAs, the GGA uses the crossover operator[18], while the MTP follows a Branch&Bound tree. The superior results of the GA thus constitute a strong argument of viability of the GA approach in general, and GGA in particular.

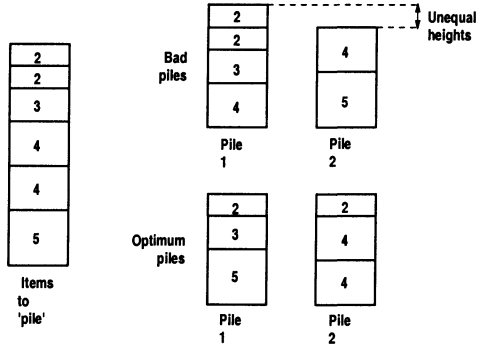


FIG. 13. *The Equal Piles problem.*

Equal piles (load balancing). The Equal Piles problem is illustrated in figure 13. It is similar to the Bin Packing problem, but this time the number of subsets is given as a constraint, and the objective is to obtain groups of sizes as equal as possible.

In the industrial realm, the Equal Piles is usually called Load Balancing. Indeed, under the same interpretation as for the Line Balancing above, the problem can be cast as one of even distribution of work load

[18]This is not to say that mutation was turned off. Simply, when the crossover was turned off, the GA got nowhere near the performance with crossover enabled.

among a given number of processors, a problem of considerable practical importance.

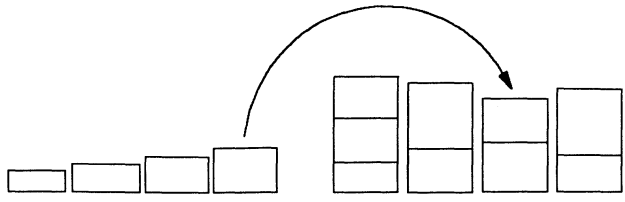


FIG. 14. *Equal Piles descending heuristic.*

The Equal Piles GGA presented in [3] is very simple. The only problem-dependent enhancement is the use of a simple heuristic depicted in figure 14. The heuristic is inspired by the First Fit Descending heuristic for Bin Packing. Having initialized the required number of groups at random, the remaining items are sorted in decreasing size, and successively added to the "smallest pile", i.e., the group that currently has the smallest sum of sizes of the items therein. As in the Bin Packing GGA above, the heuristic is used in particular during the initialization of the population and in stage 4 of the crossover.

The main objective of our writing of this GGA was a direct comparison with other GAs applied to this grouping problem. That comparison was possible thanks to the work of Jones and Beltramo [8], who not only applied nine different standard GAs to Equal Piles, but also made available the *instance* of the problem they were using in the comparison. Our algorithm could thus be compared on a fair basis. The instance used featured 34 items to be grouped into ten "piles". The instance was constructed with a known global optimum of all piles having the same size. The sizes of the items were relatively large with respect to the optimum "pile height", in order to make the instance reasonably difficult. Indeed, it was conjectured that the globally optimal solution is unique (apart from permutations of a few items of equal sizes).

Jones and Beltramo studied five different GA implementations using the group-numbers encoding, and four ordering GAs. The winner in their competition turned out to be an ordering GA using the PMX crossover of Goldberg [6] and a decoder based on a heuristic of a similar flavor as the one depicted in figure 14.

The details of the experimental results can be found in [3]. The GGA was not only significantly faster than the best among the nine standard GAs studied by Jones and Beltramo, it also found an optimal solution on every run, while the best standard GA never did. As a side effect, the GGA found several nontrivial equivalent combinations of the items, thus revealing that contrary to our impression, the instance actually admitted *many* global optima.

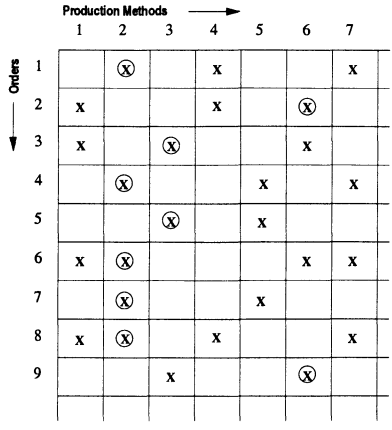


FIG. 15. *The Economies of Scale problem.*

Economies of scale. The last GGA application we present here is intended to illustrate the large variability of grouping problems the GGA can be applied to on one hand, and the kind of industrial problems GAs can be called upon to solve on the other. It has been presented in [1, 2], and is illustrated in figure 15.

The problem arises in a forge. In the beginning of the week, the list of production orders to execute is known, but each of the orders specifies *several* production methods that can be followed to produce the metal. In figure 15, each order corresponds to a line in the matrix, and the Xs in the line identify the possible production method for that order.

Organizing the week's production means selecting for each of the orders one method to follow, i.e., one of the Xs in the line. Two costs are incurred: *fixed* costs are intrinsic to each order/method couple, i.e., are associated with each X in the matrix. *Variable* costs are due to the cost of switching from one method to another (setup costs) and, more importantly, to the fact that metal can only be produced in multiples of a minimal quantity. This last cost means that whenever too little metal is produced by a method, the unsold metal must be stocked, at very high cost. The actual cost function used by the forge is extremely complicated, being composed of many terms induced by the various aspects of the industrial process. Besides, the function changes from time to time, reflecting changes in the process and the economic environment. What is constant though is the general *structure* of the problem: the orders have to be *grouped* into batches produced in a uniform way, while the sum of production cost of all batches plus the setup costs must be minimized. Note that this implies that an optimizing algorithm must find the proper tradeoff between the fixed and variable costs.

A rather simple[19] GGA for this problem is able to handle instances of up to one thousand orders and two hundred columns in a couple of hours on a low-powered workstation (4 MIPS). Used weekly in production since five years, it generates savings of the order of 330,000 US $ a year.

7. Good news, bad news, and a plea. In conclusion, we would say three things about GAs.

Do they work?
Yes. Correctly designed GAs work extremely well, sometimes significantly better than any other method. And in particular, yes, they *are* suited for real-world optimization, as illustrated by the examples above.

Is it universal?
No. "The GA" is not a universal optimization method. It is a *paradigm* that has to be fitted to each problem being solved to perform well. We believe that the job of a GA designer consists of studying the target problem and exploiting its structure in the encoding and the operators. Now this admittedly goes against the *very* often cited classification of the GA as a "strong weak method", meaning that it shows a strong performance without problem-dependent information. Personally, we do *not* subscribe to that classification. We believe that if we really want to insist on the weak aspect of the method, then it's exactly what we will get—a really *weak* method.

This finally leads us to the "big question". What if no meaningful encoding/operators come to mind for a given problem? In particular, what if we fail to see any interesting regularities that would allow us to *combine* two solutions to the problem at hand to produce a new solution in a reasonable crossover operator? Should we default to the "classics"?

No. Some other method should be tried. For in *most* of those cases, we would most probably produce an algorithm that would only damage the reputation of the GA paradigm.

Acknowledgment. Section 4 has profited enormously from discussions with Michael Vose, even though this author assumes full responsibility for its contents.

REFERENCES

[1] E. FALKENAUER, *The grouping genetic algorithms - widening the scope of the gas*, JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science, 33 (1993), pp. 79–102.

[2] ———, *A new representation and operators for GAs applied to grouping problems*, Evolutionary Computation, 2 (1994), pp. 123–144.

[19] In terms of domain-dependent knowledge exploited in the algorithm.

[3] ———, *Solving equal piles with a grouping genetic algorithm*, in Proceedings of the Sixth International Conference on Genetic Algorithms (IGGA95), L. J. Eshelman, ed., San Mateo, CA, July 1995, University of Pittsburg (Pennsylvania), Morgan Kaufman Publishers, pp. 492–497.

[4] ———, *A hybrid grouping genetic algorithm for bin packing*, Journal of Heuristics, 2 (1996), pp. 5–30.

[5] E. FALKENAUER AND A. DELCHAMBRE, *A genetic algorithm for bin packing and line balancing*, in Proceedings of the 1992 IEEE International Conference on Robotics and Automation, Los Alamitos, CA, May 1992, IEEE Computer Society Press, pp. 1186–1192.

[6] D. E. GOLDBERG, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison–Wesley Publishing Company Inc., 1989.

[7] J. H. HOLLAND, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[8] D. R. JONES AND M. A. BELTRAMO, *Solving partitioning problems with genetic algorithms*, in Proceedings of the Fourth International Conference on Genetic Algorithms, R. K. Belew and L. B. Booker, eds., San Mateo, CA, July 1991, University of California, Morgan Kaufmann Publishers.

[9] S. MARTELLO AND P. TOTH, *Lower bounds and reduction procedures for the bin packing problem*, Discrete Applied Mathematics, 22 (1990), pp. 59–70.

[10] H. MÜHLENBEIN, *Parallel genetic algorithms in combinatorial optimization*, in Computer Science and Operations Research - New Developments in Their Interfaces, O. Balci, R. Sharda, and S. A. Zenios, eds., Pergamon Press, 1992, pp. 441–453.

[11] N. RADCLIFFE AND P. SURRY, *Fundamental limitations on search algorithms: Evolutionary computing in perspective*, in LNCS1000, Springer-Verlag, 1995.

[12] G. SYSWERDA, *Uniform crossover in genetic algorithms*, in Proceedings of the Third International Conference on Genetic Algorithms, D. J. Shaffer, ed., San Mateo, CA, June 1989, George Mason University, Morgan Kaufman Publishers, pp. 2–9.

[13] G. VON LASZEWSKI, *Intelligent structural operators for the k-way graph partitioning problem*, in Proceedings of the Fourth International Conference on Genetic Algorithms, R. K. Belew and L. B. Booker, eds., San Mateo, CA, July 1991, University of California, Morgan Kaufmann Publishers.

[14] D. H. WOLPERT AND W. G. MACREADY, *No free lunch theorems for search*, Tech. Rep. SFI-TR-95-02-010, The Santa Fe Institute, Santa Fee, 1995.

AN OVERVIEW OF EVOLUTIONARY PROGRAMMING

DAVID B. FOGEL*

Abstract. Evolutionary programming is a method for simulating evolution that has been investigated for over 35 years. This paper offers an introduction to evolutionary programming, and indicates its relationship to other methods of evolutionary computation, specifically genetic algorithms and evolution strategies. The original efforts that evolved finite state machines for predicting arbitrary time series, as well as specific recent efforts in combinatorial and continuous optimization, are reviewed. Some areas of current investigation are mentioned, including assessing the optimization performance of the technique and extensions to include mechanisms of self-adaptation.

1. Introduction. The impact of evolutionary thinking on biology cannot be underestimated. But evolutionary thought extends beyond the study of life. Evolution is an optimization process that can be simulated on a computer and used for good engineering purpose. It also is essentially similar to the process of the scientific method, and as such it represents a procedure for generating machine intelligence.

There are three main lines of investigation within the current framework of evolutionary computation: (1) genetic algorithms, (2) evolution strategies, and (3) evolutionary programming. Reviews of these methods are offered in several recent books (Schwefel, 1995; Fogel, 1995a; Bäck, 1996; Mitchell, 1996; Michalewicz, 1996; and others). Each of these methods has developed over more than 30 years, and mostly independently of each other. Only recently have there been overt efforts to unify these very similar approaches to modeling the fundamental aspects of natural evolution. This paper provides an overview of one of the approaches: evolutionary programming. The scope is limited due to the nature of the submission in the context of the IMA workshop, as others will provide background and detailed information regarding alternative methods of evolutionary computation.

2. Perspectives of simulated evolution. When applied to optimization problems, all methods of evolutionary computation involve an iterative population-based search with random variation and selection. The methods differ with respect to the traditional choices of representation, as well as the procedures used for generating new solutions and for selecting which solutions to maintain into future generations. The choices that have been made historically have emerged from alternative perspectives of the complex process of natural evolution. Evolutionary mechanisms create a complex web of interacting effects that cannot be easily partitioned or disentangled (Hoffman, 1989, pp. 32–33), but the physics that governs the

* Natural Selection, Inc., 3333 N. Torrey Pines Ct., Suite 200, La Jolla, CA 92037, dfogel@natural-selection.com .

evolutionary process may be quite simple and expressible in a small set of rules. These rules indicate various approaches to modeling evolution.

Living organisms can be viewed as a duality of their genotype (the underlying genetic coding) and their phenotype (the manner of response contained in the behavior, physiology, and morphology of the organism). Lewontin (1974) illustrated this distinction by specifying two state spaces: a populational genotypic (informational) space $\mathbf{G}$ and a populational phenotypic (behavioral) space $\mathbf{P}$. Four functions map elements in $\mathbf{G}$ and $\mathbf{P}$ to each other (Figure 1). Atmar (1992) modified these function to be:

$$f_1 : \mathbf{I} \times \mathbf{G} \rightarrow \mathbf{P}$$
$$f_2 : \mathbf{P} \rightarrow \mathbf{P}$$
$$f_3 : \mathbf{P} \rightarrow \mathbf{G}$$
$$f_4 : \mathbf{G} \rightarrow \mathbf{G}.$$

The function f_1, *epigenesis*, maps the element $g_1 \in \mathbf{G}$ into the phenotypic space $\mathbf{P}$ as a particular collection of phenotypes p_1 whose development is modified by its environment, and indexed set of symbols $(i_1, ..., i_k) \in \mathbf{I}$, where $\mathbf{I}$ is the set of all such environment sequences. The function f_2, *selection*, maps phenotypes p_1 into p_2. As natural selection only operates on the phenotypic expressions of the genotype (Mayr 1960, p. 24, 1970, p. 131; Hartl and Clark, 1989, p. 431), the underlying coding g_1 is not involved in function f_2. The function f_3, *genotypic survival*, describes the effects of selection and migration processes on $\mathbf{G}$. Function f_4, *mutation*, maps the representative codings $g_2 \in \mathbf{G}$ to the point $g_1' \in \mathbf{G}$. This function represents the "rules" of mutation and recombination, and encompasses all genetic changes. With the creation of the new population of genotypes g_1', one generation is complete. Evolutionary adaptation occurs over successive iterations of these mappings.

The duality of the genotype and the phenotype suggest two main approaches for simulating evolution. In genotypic simulations, attention is focused on genetic structures. The candidate solutions to a task at hand are described as being analogous to chromosomes and genes. These data structures are then manipulated by genetic operators that model chromosomal transformations as observed in biota, such as crossing over, inversion, and point mutation. In phenotypic simulations, attention is focused on the behaviors of the candidate solutions in a population. Various techniques for modifying these behaviors may be applied, with an interest in generating the possibility for a nearly continuous distribution of new behaviors while maintaining a strong behavioral link between a parent (or multiple parents) and its offspring.

Evolutionary programming resides in the latter category of simulation. Attention is placed on variation operations that can be constructed for a given representation so as to usefully adjust the behavior of the solutions in light of the performance measure for the task at hand. No attempt is

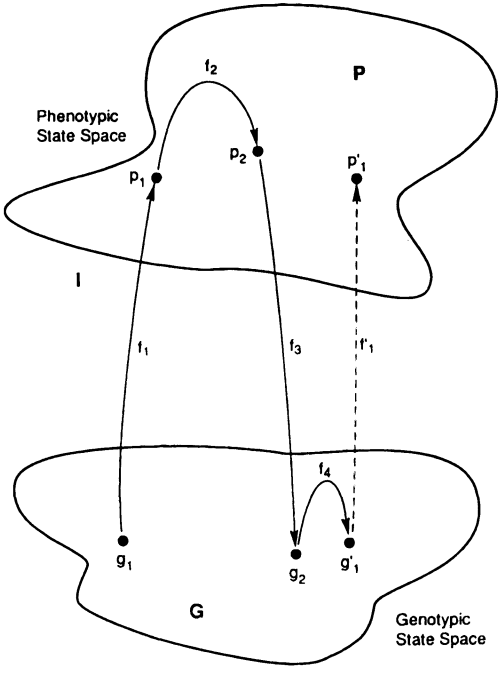


FIG. 1. *The evolution of a population within a single generation. Evolution can be viewed as occurring as a succession of four mapping functions (epigenesis, selection, genotypic survival, and mutation) relating the genotypic information state space and the phenotypic behavioral state space.*

made to model the overt mechanisms of genetics as found in nature. Instead, general mathematical transformations are applied to solutions. The underlying philosophy is one of asking about useful transformations from p_2 to p_1' in Figure 1, rather than asking about the application of heuristic genetic operators to g_2 so as to generate g_1'. There is no reason to believe that any particular mechanism of achieving a specified functionality will be productive in a simulation simply because it is observed in the natural world (e.g., modern aircraft are not ornithopters); the explicit and direct use of mechanisms such as allelic recombination, jumping genes, deletions, and so forth, may or may not be of any particular utility. Experience indicates that practitioners of evolutionary algorithms can often design more productive means for varying the solutions in a particular circumstance than is afforded by overtly mimicking natural genetic mechanisms (Fogel and Atmar, 1990; Rizki et al., 1993; Fogel and Stayton, 1994; Nettleton and Garigliano, 1994; Gehlhaar et al., 1995; and many others).

3. Early efforts in evolutionary programming. Evolutionary programming was devised by Lawrence J. Fogel in 1960 while serving at the National Science Foundation (NSF). Fogel was on leave from Convair,

tasked as special assistant to the associate director (research), Dr. Richard Bolt, to study and write a report on investing in basic research. With regard to artificial intelligence, at the time it was mainly concentrated around heuristics and the simulation of primitive neural networks. It was clear to Fogel that both these approaches were limited because they model humans rather than the essential process that produces creatures of increasing intellect: evolution. Fogel considered intelligence to be based on adapting behavior to meet goals in a range of environments. In turn, prediction was viewed the key ingredient to intelligent behavior and this suggested experiments on the use of simulated evolution of finite state machines to forecast nonstationary time series with respect to arbitrary criteria. These and other experiments were documented in a series of publications (e.g., Fogel, 1962; Fogel, 1964; Fogel et al., 1965, Fogel et al., 1966; and many others).

Intelligent behavior was viewed as requiring the composite ability to (1) predict one's environment, coupled with (2) a translation of the predictions into a suitable response in light of the given goal. For the sake of generality, the environment was described as a sequence of symbols taken from a finite alphabet. The evolutionary problem was defined as evolving an algorithm (essentially a program) that would operate on the sequence of symbols thus far observed in such a manner so as to produce an output symbol that is likely to maximize the algorithm's performance in light of both the next symbol to appear in the environment and a well-defined payoff function. Finite state machines provided a useful representation for the required behavior.

The proposal was as follows: A population of finite state machines is exposed to the environment, that is, the sequence of symbols that have been observed up to the current time. For each parent machine, as each input symbol is offered to the machine, each output symbol is compared with the next input symbol. The worth of this prediction is then measured with respect to the payoff function (e.g., all-none, absolute error, squared error, or any other expression of the meaning of the symbols). After the last prediction is made, a function of the payoff for each symbol (e.g., average payoff per symbol) indicates the fitness of the machine.

Offspring machines are created by randomly mutating each parent machine. Each parent produces offspring (this was originally implemented as only a single offspring per parent simply for convenience). There are five possible modes of random mutation that naturally result from the description of the machine: change an output symbol, change a state transition, add a state, delete a state, or change the initial state. The deletion of a state and change of the initial state are only allowed when the parent machine has more than one state. Mutations are chosen with respect to a probability distribution, which is typically uniform. The number of mutations per offspring is also chosen with respect to a probability distribution or may be fixed a priori. These offspring are then evaluated over the ex-

isting environment in the same manner as their parents. Other mutations, such as majority logic mating operating on three or more machines, were proposed in Fogel et al. (1966) but not implemented.

The machines that provide the greatest payoff are retained to become parents of the next generation. (Typically, half the total machines were saved so that the parent population remains at a constant size.) This process is iterated until an actual prediction of the next symbol (as yet unexperienced) in the environment is required. The best machine generates this prediction, the new symbol is added to the experienced environment, and the process is repeated. Fogel (1964; Fogel et al., 1966) used "nonregressive" evolution. To be retained, a machine had to rank in the best half of the population. Saving lesser-adapted machines was discussed as a possibility (Fogel et al., 1966, p. 21) but not incorporated.

For example, a nonstationary sequence of symbols was generated by classifying each of the increasing integers as being prime (symbol 1) or nonprime (symbol 0). Thus the environment consisted of the sequence 01101010001 . . ., where each symbol depicts the primeness of the positive integers $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...$, respectively. Testing for primeness is straightforward, but predicting whether or not the next integer will be prime based on the sequence of observed primes and nonprimes appeared nontrivial. The payoff function for prediction was all or none, that is, one point for each correct prediction, zero points for each error, modified by subtracting 0.01 multiplied by the number of states of the machine. This penalty for complexity was provided to maintain parsimonious machines in light of the limited memory of the available computer (IBM 704).

Figure 2 shows the cumulative percentage of correct predictions in the first 200 symbols. After the initial fluctuation (due to the small sample size), the prediction score increased to 78 percent at the 115th symbol and then essentially remained constant until the 200th prediction. At this point, the best machine possessed four states. At the 201st prediction, the best machine possessed three states, and at the 202nd prediction, the best machine possessed only one state with both output symbols being 0. After 719 symbols, the process was halted with the cumulative percentage of correct predictions reaching 81.9 percent. The asymptotic worth of this machine would be 100 percent correct because the prime number become increasingly infrequence and the machine continued to predict "nonprime."

The goal was then changed to offer a greater payoff for predicting a rare event. Correctly predicting a prime was worth one plus the number of nonprimes that preceded it. Similarly, correctly predicting a nonprime was worth one plus the number of primes that preceded that nonprime. During the first 150 symbols, there were 30 correct predictions of primes, 37 incorrect predictions (false alarms), and five missed primes. From the 151st symbol to the 547th there were 65 correct predictions of primes and 67 false alarms. That is, of the first 35 primes, five were missed; of the next 65 primes, none was missed. Fogel et al. (1966) indicated that the

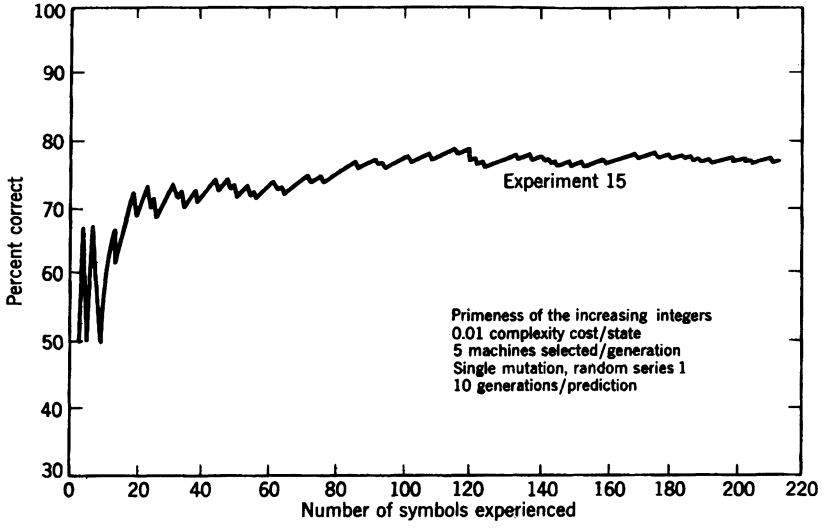


FIG. 2. *The cumulative percentage of correct predictions in the first 200 prime numbers (from Fogel et al., 1966, p. 37).*

evolutionary algorithm learned to recognize numbers that are divisible by two or three as being nonprime. Some recognition that numbers divisible by five are nonprime was also evidenced. Fogel (1968) later remarked that the evolutionary program had successively discovered "cyclic properties within the environment ... in essence, the equivalent of first recognizing that numbers divisible by two are not prime, etc. In other words, the program was synthesizing a definition of primeness without prior knowledge of the nature or primeness *or* an ability to divide. Clearly the evolutionary program would never be a perfect predictor, but that is to be expected." No finite state machine can represent the algorithm for generating prime numbers.

This general procedure was successfully applied to problems in prediction, identification, and automatic control (Fogel et al., 1964, 1966; Fogel, 1968) and was extended to simulate coevolving populations in Fogel and Burgin (1969). Additional experiments evolving finite state machines for sequence prediction, pattern recognition, and gaming can be found in Lutter and Huntsinger (1969), Burgin (1969), Atmar (1976), Dearholt (1976), Takeuchi (1980), and others.

4. Recent efforts in evolutionary programming.

Review

In the mid-1980s the general evolutionary programming procedure was

extended to alternative representations including ordered lists for the traveling salesman problem (Fogel and Fogel, 1986), and real-valued vectors for continuous function optimization (Fogel and Fogel, 1986). This led to other applications in route planning (Fogel, 1988; Fogel and Fogel, 1988), optimal subset selection (Fogel, 1989), training neural networks (Fogel et al., 1990), and well as comparisons to other methods of simulated evolution (Fogel and Atmar, 1990). Methods for extending evolutionary search to a two-step process including evolution of the mutation variance were offered in Fogel et al. (1991) and Fogel et al. (1992) (see section on self-adaptation below).

In the early 1990s efforts were made to organize annual conferences on evolutionary programming, these leading to the first conference in 1992 (Fogel and Atmar, 1992). This conference offered a variety of optimization applications of evolutionary programming in robotics (McDonnell et al., 1992; Andersen et al., 1992), path planning (Larsen and Herman, 1992; Page et al., 1992), neural network design and training (Sebald and Fogel, 1992; Porto, 1992; McDonnell, 1992), automatic control (Sebald et al., 1992), and others.

First contacts were made between the evolutionary programming and evolution strategies communities just before this conference, and the similar but independent paths that these two approaches had taken to simulating the process of evolution was clearly apparent. Members of the evolution strategies community have participated in all successive evolutionary programming conferences (Bäck et al., 1993; Sprave, 1994; Bäck and Schütz, 1995; Fogel et al., 1996). There is less similarity between evolutionary programming and genetic algorithms, as the latter emphasizes simulating specific mechanisms that apply to natural genetic systems, whereas evolutionary programming emphasizes the behavioral, rather than genetic, relationships between parents and their offspring. Members of the genetic algorithm and genetic programming communities have, however, also been invited to participate in the annual conferences, making for truly interdisciplinary interaction (e.g., Altenberg, 1994, Land and Belew, 1995; Koza and Andre, 1996).

Within the past five years, efforts in evolutionary programming have diversified in many directions. Applications in training neural networks have received considerable attention (English, 1994; Angeline et al., 1994; McDonnell and Waagen, 1994; Porto et al., 1995; and others), while relatively less attention has been devoted to evolving fuzzy systems (Haffner and Sebald, 1993; Kim et al., 1996). Image processing applications can be found in Bhattachajya and Roysam (1994), Brotherton et al. (1994), Rizki et al. (1995), and others. Recent efforts to use evolutionary programming in medicine (e.g., cancer diagnosis and pharmaceutical design) have been offered by Fogel et al. (1995), Gehlhaar et al. (1995), Luke (1994), and others. Efforts studying and comparing methods of self-adaptation can be found in Saravanan et al. (1995), Angeline et al. (1996), and others. Mathematical analyses of evolutionary programming have been summarized in Fogel (1995a).

Examples

The Traveling Salesman Problem

The traveling salesman problem (TSP) has received considerable attention from the evolutionary computation community. The problem is interesting because (1) it has become a benchmark test case, (2) it is easily stated yet difficult to solve, and (3) it is broadly applicable to a number of routing and networking problems. The task is to take n cities and construct a minimum length tour such that each city is visited once and only once, with a return to the original city completing the circuit. The problem is NP- hard and the number of possible solutions increases as $(n-1)!/2$. As such, it is often less of interest to find the optimum routing than to quickly find reasonably good solutions.

One natural representation of the problem is simply a listing of the cities to be visited in order. Possible solutions are then permutations of this listing. Under evolutionary programming, the construction of a suitable mutation operation should take into consideration two facets: (1) maintaining a strong behavioral link between each parent and its offspring, and (2) providing a (nearly) continuous range of potential new behaviors. One possible mutation operation that meets these requirements for the TSP is an inversion operation: Select two cities in the tour and reverse the order of the segment defined by those two cities. The mutation operation that would offer the generally smallest functional change would be to simply swap two adjacent cities. This is an inversion of length two. As the inversion length is increased, the expected functional change between a parent tour and its offspring is also increased, up to a maximum inversion length of $n/2$, due to the symmetry of the tour. Note that this inversion operation is not chosen because it has any analogous operator in biota; it simply provides a method of variation that will allow for suitable evolution in this problem.

Specifically, the method can be implemented as follows. A population of random tours are scored with respect to their Euclidean length. Each tour is mutated using the inversion operation defined above to generate an offspring. All of the offspring are then scored and a stochastic competition including all parents and offspring is conducted to determine which tours to purge from the population, and as a consequence which tours remain to generate the next succession of progeny.

This method has been implemented on several uniform TSPs (i.e., problems in which the cities are distributed in accordance with a uniform distribution) (Fogel 1993ab). Figure 3 shows an evolved solution to a 1,000-city uniform TSP after 4×10^7 function evaluations in Fogel (1993a). The optimization of the best tour in the population is quite rapid. The routing depicted was estimated to be within five to seven percent of the expected optimal tour length. Fogel (1993b) estimated that the number of tours that must be examined in order to achieve solutions which are on average

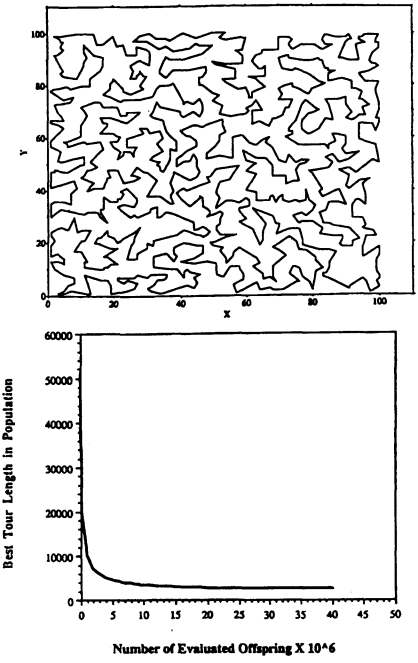


FIG. 3. *(a) The final best evolved tour for a 1,000-city TSP in which the cities were distributed at random in accordance with a uniform distribution in both the x and y dimensions. (b) The rate of optimization of the best solution in the population as a function of the number of tours generated. The figure is from Fogel (1993a).*

10 percent worse than the expected best in uniform TSPs increases as the square of the number of cities. Previous efforts by Ambati et al. (1991) with similar procedures estimated that solutions that are on average 25 percent worse than the expected best could be discovered in $O(n \log n)$.

In earlier research, Fogel (1988) implemented a different mutation operation which selected a city at random and replaced it in a random location. Ambati et al. (1991) demonstrated that more effective mutation operations could be constructed, yet the method of Fogel (1988) was observed to outperform some genetic algorithm techniques that relied on recombination (e.g., PMX) (Goldberg and Lingle 1985).

Evolving Neural Networks

Neural networks are general nonlinear mapping functions which are defined by a number of processing nodes and the manner in which these nodes are connected. Speculation that such networks could be optimized using simulated evolution goes back at least to Bremermann (1966). More recently there have been many efforts to apply evolutionary programming to the design and optimization of neural networks in pattern recognition

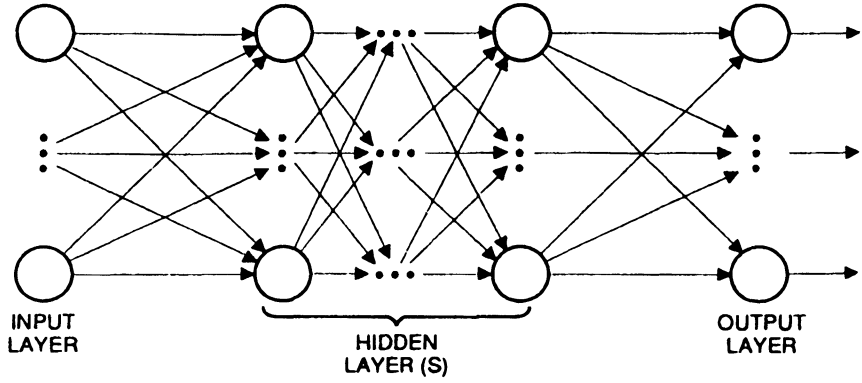


FIG. 4. *A multilayer feedforward perceptron consists of a number of input, output and hidden nodes. Each node typically performs a weighted sum of the inputs to that node, subtracts off a variable bias, and then passes the result through a nonlinear filter.*

and control systems.

The most common network involved in these studies has been the mutilayer perceptron (MLP) (Fig. 4). If the network architecture is predefined, the problem becomes one of trying to find optimal settings for the weight and bias terms. A natural representation for this is a real-valued vector where each component corresponds to a weight or bias. For such continuous-valued representations, the common method of mutation is to add a multi-variate zero mean Gaussian random variable to all components, thereby in this case changing all of the weights and biases simultaneously. The behavior of each offspring network is strongly related to its parent and there is a continuous range of possible new behaviors.

The variability between each parent and its offspring can be controlled directly by specifying the individual variances for each Gaussian perturbation (or covariances). One common method for accomplishing this is to set the step size (standard deviation) to be proportional to the mean squared error of the parent network. In this manner, as better solutions are discovered, the step size is reduced and the search effort is concentrated around the parent, and conversely, a larger variance is used for parents with relatively poor performance. As noted in the introduction, more sophisticated methods of self-adapting the variability of the permutations have been considered and are discussed below in the section on future efforts.

Some of the first efforts at applying evolutionary programming to optimizing neural networks can be found in Fogel et al. (1990) and Fogel (1991a). More recent research has involved simultaneously evolving both the structure and weights of feed forward and recurrent networks (McDon-

nell and Waagen 1993, 1994; Angeline et al. 1994). Some attention has also been given to evolving fuzzy neural networks in which classification of input patterns are made with respect to their fuzzy membership in evolved clusters (Brotherton and Simpson 1995).

Evolution and Games

One of the more interesting avenues of research in evolutionary computation involves the co-evolution of solutions in gaming, wherein the appropriateness of a solution depends on its interactions with other solutions in the population. Among these investigations are the *Tierra* simulations of Ray (1991), the competitive sorting research of Kinnear (1993), and many other simulations involving the iterated prisoner's dilemma (Axelrod 1987; Fogel 1991b, 1993c; Harrald and Fogel 1996).

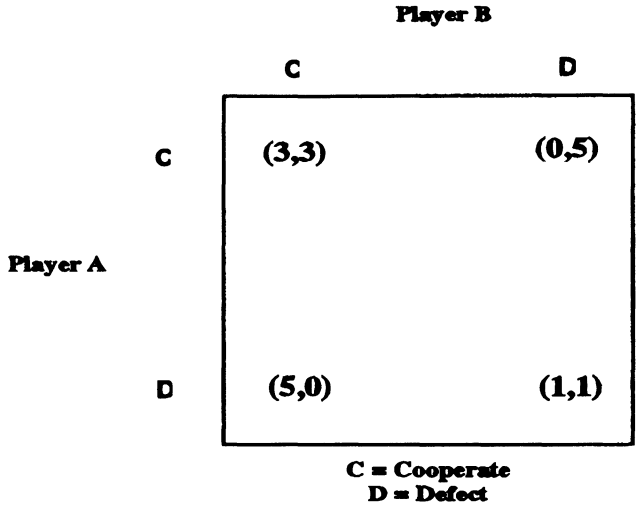
The prisoner's dilemma is a two-person game in which each of the players can adopt one of two behavioral policies: cooperate or defect. *Cooperation* implies increasing the reward of both participants, while *defecting* implies increasing ones own reward at the expense of the other player. A typical payoff matrix for the game is shown in Table 1 (see Fogel (1993c) for a general description of the game). If the game is played for a single iteration, the dominant move is to defect. Defection is also rational if the game is iterated over a series of plays under conditions in which both player's decisions are not affected by previous plays; the game degenerates into a series of independent plays. But if the players' strategies can depend on the results of previous iterations, mutual cooperation can become a rational policy. Evolutionary simulations have generally focused on the dynamics of a population of players interacting according to the payoff matrix in Table 1, or variants of this matrix.

Within the framework of evolutionary programming, the iterated prisoner's dilemma has been simulated by using both finite state machines (Fogel 1991b, 1993c, 1995b) and neural networks (Harrald and Fogel 1996). Fig. 5 shows a typical finite state machine used in such simulations, while Fig. 6 shows the design of a neural network. Finite state machines have been used for conditions in which there are discrete choices between cooperating and defecting, while neural networks have been used to offer a continuum of possible behaviors between these two extremes. In general, the results of experiments with these two constructions have indicated that mutual cooperation is more likely to occur when behaviors are limited to the extremes; when a continuum of behaviors exists, it becomes easier to slip toward mutual defection. Certainly, these results and conclusions are preliminary and readers interested in pursuing research on the prisoner's dilemma should also review Axelrod (1984), Lindgren (1991), Stanley et al. (1994), and others.

5. Evolutionary programming theory. When evolutionary programming is applied for problem solving, it is natural to enquire about its related mathematical properties as an optimization procedure. The most

TABLE 1

The payoff matrix for typical experiments with the iterated prisoner's dilemma. The matrix was used in Axelrod (1987), Fogel (1991b, 1993c), and others. The payoffs indicate the reward to each player for adopting a particular strategy in light of the other player's strategy. Mutual cooperation generates three points for each player. Mutual defection generates only one point for each player. Defecting against a cooperator yields five points for the defector and no points for the cooperator.



basic result is that the procedure will asymptotically converge to a global optimum of any well-posed problem (Fogel 1994). This result relies on a Markov chain analysis, and is rather straightforward. Analysis of the probability transition matrix that defines the evolutionary Markov chain also indicates the probability of an improvement over successive generations increases as a geometric series converging to 1.0, and is strictly a linear combination of the eigenvalues of the transition matrix.

These results are of limited utility, however, because they do not address the rate of improvement in the error, only the probability of obtaining any improvement. Convergence rate analysis in evolution strategies (Bäck et al., 1993) can be carried over to evolutionary programming due to the similarities in the approaches. These results indicate geometric rates of convergence on strongly convex functions, and moreover that the rate of convergence can be improved as the logarithm of the number of offspring λ in a $(1, \lambda)$ selection procedure. (In contrast, there are no known rates of convergence for canonical genetic algorithms.) Convergence rate information for nonconvex functions still remains for future effort.

6. Self-adaptation in evolutionary programming. As with most stochastic optimization algorithms, evolutionary programming relies on a hierarchy of parameters that control the behavior of the procedure. These include the population size, the selection pressure, and the mutation distri-

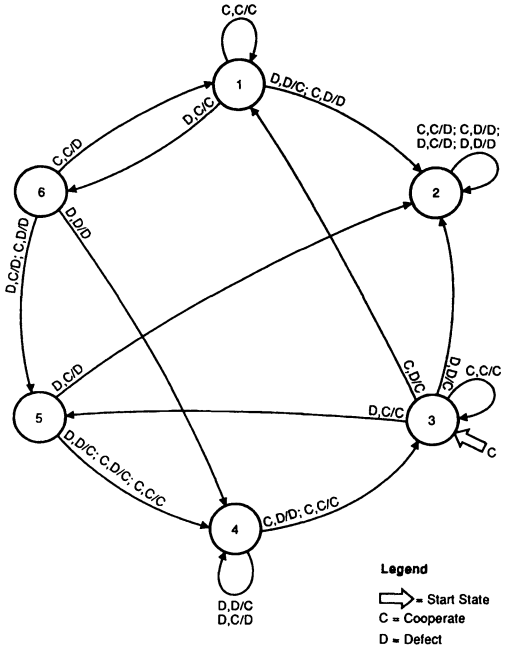


FIG. 5. *A typical finite state machine evolved in evolutionary programming experiments with the iterated prisoner's dilemma from Fogel (1993c). The input symbols are (C,C), (C,D), (D,C), (D,D), corresponding to the previous move (cooperate or defect) of the finite state machine and its opponent. The output symbols are C,D, corresponding to the possible behavior resulting from the previous pair of moves and the current state.*

bution. With regard to the latter, variation operators can be incorporated into each parent solution as additional information on how to generate offspring. This information can then be subject to mutation and selection in a manner analogous to the mutation and selection that is applied to the candidate solutions for the task at hand. Coincidentally, such methods arose in evolution strategies and evolutionary programming completely independently, and were quite similar.

Fogel et al. (1991) suggested that when applied to continuous function optimization, each solution could be represented as $(\mathbf{x}, \sigma)$ where $\mathbf{x}$ is a vector of parameters to be scored in light of the objective function, and σ is a vector of standard deviations used in generating an offspring as follows:

$$x_i' = x_i + N(0, \sigma_i)$$
$$\sigma_i' = \sigma_i + N(0, \alpha\sigma_i)$$

If the resultant values σ_i' are set less than zero, they are reset to a small positive value ϵ (often 10^{-3}). Fogel et al. (1992) extended the method to

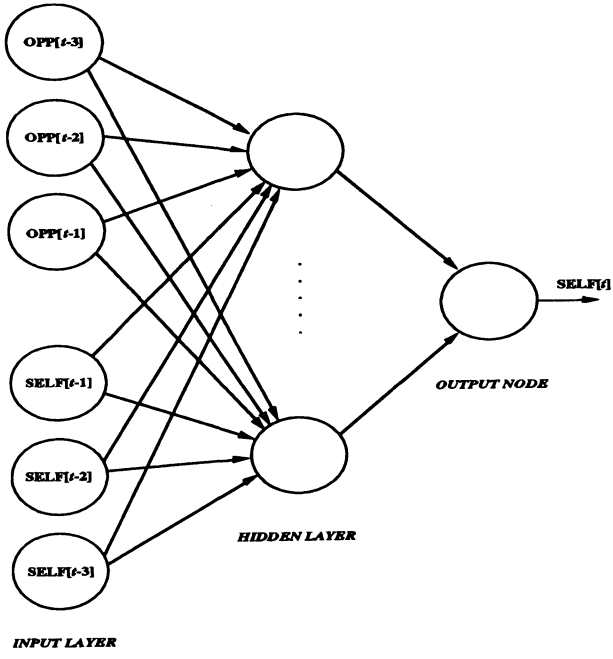


FIG. 6. *A multilayer perceptron used for evolving continuous behaviors in the iterated prisoner's dilemma. There are six input nodes corresponding to the three previous moves of the neural network and its opponent. The number of hidden nodes is variable. The output node is scaled to range from* $[-1, 1]$ *where* -1 *indicates complete defection and* $+1$ *indicates complete cooperation.*

operate on correlated zero mean Gaussian mutations, rather than independent steps in each dimension.

This procedure is very similar to one developed much earlier, as summarized in Schwefel (1981), within the evolution strategies community where the update procedure was implemented as:

$$\sigma_i' = \sigma_i \cdot \exp(\tau N(0,1) + \tau' N_i(0,1))$$
$$x_i' = x_i + N(0, \sigma_i')$$

where τ and τ' are proportional to $(2n)^{-0.5}$ and $(2\sqrt{n})^{-0.5}$, respectively, where there are n dimensions. The use of lognormal perturbations maintains positive values for the standard deviations, and also provides the property that it is equally like to increase the standard deviations by a factor of k as it is to decrease them by a factor of $1/k$. Methods for extending this procedure to include arbitrary rotation angles have also been offered (Schwefel, 1981) as has the suggestion that various forms of recombination applied to the standard deviations of different parents could be advantageous.

Some initial comparisons of the methods of Fogel et al. (1991) with those of Schwefel (1981) were offered in Saravanan and Fogel (1994) and Saravanan et al. (1995). Across a small suite of common benchmark functions, the experiments often indicated statistically significant evidence in favor of the lognormal approach (Schwefel, 1981). Angeline (1996) made a similar comparison using noisy test function but came to the obverse conclusion, thus the conditions for favoring one method of self-adaptation over the other is not clear. It is clear, however, that the use of recombination on the strategy parameters is not required for successful optimization across a broad range of function optimization problems (cf. Bäck, 1996).

Other recent efforts in self-adaptive evolutionary programming include the optimization of directed mutations in polar coordinates (Ghozeil and Fogel, 1996) and the combination of various mutation distributions (e.g., Cauchy and Gaussian in Saravanan and Fogel, 1997). Self-adaptation in evolutionary programming has also been applied to discrete optimization problems, including the adaptation of finite state machines (Fogel et al., 1995; Angeline et al., 1996), and circuits in traveling salesman problems (Chellapilla and Fogel, 1997).

7. Discussion. Evolutionary programming was originally proposed as a procedure for generating machine intelligence. Intelligence was viewed in the context of adaptive behavior, predicting the environment and taking suitable action in light of a given purpose. The same basic procedure can be extended and used for numerical optimization in pattern discovery, system identification, automatic control, and multiplayer gaming (control of an intelligent adversary). The representation for each problem follows from the task at hand; there is no specific requirement for using any particular coding (e.g., a finite state machine or a bit string). Similarly, rather than prescribe the use of any particular transformation operator (e.g., crossover), variations are applied to parent solutions using operators that follow naturally from the coding structure. If the codings are real-valued then Gaussian mutations (or Cauchy) often provide for an efficient search. If a number of elements must be altered, a Poisson random variable with an appropriately selected (or self-adaptive) rate may provide a suitable operation. The crucial element in constructing an appropriate mutation operation is that it must maintain a suitable behavioral link between each parent and its offspring.

The initial efforts of L.J. Fogel indicate some of the early attempts to (1) use simulated evolution to perform prediction, (2) include variable-length encodings, (3) use representations that take the form of a sequence of instructions, (4) incorporate a population of candidate solutions, and (5) to coevolve evolutionary programs. Moreover, Fogel (1963, 1964, Fogel et al., 1966) offered the early recognition that natural evolution and the human endeavor of the scientific method are essentially similar processes, a notion recently echoed in Gell-Mann (1994). The initial prescriptions for

operating on finite state machines have been extended to arbitrary representations, mutation operators, and selection methods, and techniques for self-adapting the evolutionary search have been proposed and implemented. The population size need not be kept constant and there can be a variable number of offspring per parent, much like the $(\mu + \lambda)$ methods offered in evolution strategies. In contrast to these methods, selection is often made probabilistic in evolutionary programming giving lesser-scoring solutions some probability of surviving as parents into the next generation. In contrast to genetic algorithms, no effort is made in evolutionary programming to maximize schemata processing, nor is the use of random variation constrained to emphasize specific mechanisms of genetic transfer, perhaps providing greater versatility to tackle specific problem domains that are unsuitable for genetic operators such as crossover.

Although evolutionary programming, evolution strategies, and genetic algorithms have some notable differences, their similarities are in many ways more important. Each of these general procedures, developed over the last 30–35 years, has demonstrated the utility of stochastic optimization using a population-based search. In some cases, similar procedures have been invented independently, such as the use of self-adaptive variances and covariances in real-valued function optimization in evolution strategies and evolutionary programming. As increased attention is focused on these optimization techniques, there will be greater synergy not only between alternative methods of evolutionary computation, but also between other research communities such as operations research, numerical analysis, optimization, and artificial intelligence.

8. Acknowledgments. The author thanks the workshop organizers for the opportunity to participate in the event.

REFERENCES

L. Altenberg (1994) "Emergent Phenomena in Genetic Programming," *Proceedings of the Third Annual Conference on Evolutionary Programming*, A.V. Sebald and L.J. Fogel (eds.), World Scientific, River Edge, NJ, pp. 233–241.

B.K. Ambati, J. Ambati, and M.M. Mokhtar (1991) "Heuristic Combinatorial Optimization by Simulated Darwinian Evolution: A Polynomial Time Algorithms for the Traveling Salesman Problem," *Biological Cybernetics*, Vol. 65, pp. 31–35.

B. Andersen, J. McDonnell, and W. Page (1992) "Configuration Optimization of Mobile Manipulators with Equality Constraints using Evolutionary Programming," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 71–79.

P.J. Angeline (1996) "The Effects of Noise on Self-Adaptive Evolutionary Optimization," *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, L.J. Fogel, P.J. Angeline, and T. Bäck (eds.), MIT Press, Cambridge, MA, pp. 433–439.

P.J. Angeline, G.M. Saunders, and J.B. Pollack (1994) "An Evolutionary Algorithm that Constructs Recurrent Neural Networks," *IEEE Trans. Neural Networks*, Vol. 5:1, pp. 54–65.

P.J. Angeline, D.B. Fogel, and L.J. Fogel (1996) "A Comparison of Self-Adaptation Methods for Finite State Machines in Dynamic Environments," *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, L.J. Fogel, P.J. Angeline, and T. Bäck (eds.), MIT Press, Cambridge, MA, pp. 441–449.

J.W. Atmar (1976) "Speculation on the Evolution of Intelligence and Its Possible Realization in Machine Form," Doctoral Dissertation, New Mexico State University, Las Cruces, NM.

W. Atmar (1992) "On the Rules and Nature of Simulated Evolutionary Programming," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 17–26.

R. Axelrod (1984) *The Evolution of Cooperation*, Basic Books, NY.

R. Axelrod (1987) "The Evolution of Strategies in the Iterated Prisoner's Dilemma," *Genetic Algorithms and Simulated Annealing*, L. Davis (ed.), Pitman, London, pp. 32–42.

T. Bäck (1996) *Evolutionary Algorithms in Theory and Practice*, Oxford, NY.

T. Bäck, G. Rudolph, and H.-P. Schwefel (1993) "Evolutionary Programming and Evolution Strategies: Similarities and Differences," *Proceedings of the Second Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 11–22.

T. Bäck and M. Schütz (1995) "Evolution Strategies for Mixed-Integer Optimization of Optical Multilayer Systems," *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, J.R. McDonnell, R.G. Reynolds, and D.B. Fogel (eds.), MIT Press, Cambridge, MA, pp. 33–51.

A.K. Bhattachajya and B. Roysam (1994) "Joint Solution of Low-, Intermediate-, and High-Level Vision Tasks by Evolutionary Optimization: Application to Computer Vision at Low SNR," *IEEE Trans. Neural Networks*, Vol. 5:1, pp. 83–95.

H.J. Bremermann (1966) "Numerical Optimization Procedures Derived from Biological Evolution Processes," *Cybernetic Problems in Bionics*, H.L. Oestreicher and D.R. Moore (eds.), Gordon and Breach, London, pp. 543–562.

T.W. Brotherton and P.K. Simpson (1995) "Dynamic Feature Set Training of Neural Nets for Classification," *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, J.R. McDonnell, R.G. Reynolds, and D.B. Fogel (eds.), MIT Press, Cambridge, MA, pp. 83–94.

T.W. Brotherton, P.K. Simpson, D.B. Fogel, and T. Pollard (1994) "Classifier Design using Evolutionary Programming," *Proceedings of the Third Annual Conference on Evolutionary Programming*, A.V. Sebald and L.J. Fogel (eds.), World Scientific, River Edge, NJ, pp. 68–75.

G.H. Burgin (1969) "On Playing Two-Person Zero-Sum Games against Nonminimax Players," *IEEE Trans. Systems Science and Cybernetics*, Vol. SSC-5:4, pp. 369–370.

K. Chellapilla and D.B. Fogel (1997) "Exploring Self-Adaptive Methods to Improve the Efficiency of Generating Approximate Solutions to Traveling Salesman Problems Using Evolutionary Programming," *Evolutionary Programming VI: Proceedings of the Sixth Annual Conference on Evolutionary Programming*, P.J. Angeline, R.C. Eberhart, R.G. Reynolds, and J.R. McDonnell (eds.), Springer, Berlin, in press.

D.W. Dearholt (1976) "Some Experiments on Generalization Using Evolving Automata," *Proceedings of the 9th Intern. Conf. on System Sciences*, Honolulu, pp. 131–133.

T.M. English (1994) "Generalization in Populations of Recurrent Neural Networks," *Proceedings of the Third Annual Conference on Evolutionary Programming*, A.V. Sebald and L.J. Fogel (eds.), World Scientific, River Edge, NJ, pp. 26–33.

D.B. Fogel (1988) "An Evolutionary Approach to the Traveling Salesman Problem," *Biological Cybernetics*, Vol. 60:2, pp. 139–144.

D.B. Fogel (1989) "Evolutionary Programming for Voice Feature Analysis," *Proc. of 23rd Asilomar Conference on Signals, Systems & Computers*, Pacific Grove, California, pp. 381–383.

D.B. Fogel (1991a) "An Information Criterion for Optimal Neural Network Selection," *IEEE Trans. Neural Networks*, Vol. 2, pp. 490–497.

D.B. Fogel (1991b) "The Evolution of Intelligence Decision Making in Gaming," *Cybernetics and Systems*, Vol. 22, pp. 223–236.

D.B. Fogel (1993a) "Applying Evolutionary Programming to Selected Traveling Salesman Problems," *Cybernetics and Systems*, Vol. 24, pp. 27–36.

D.B. Fogel (1993b) "Empirical Estimation of the Computation Required to Discover Approximate Solutions to the Traveling Salesman Problem using Evolutionary Programming," *Proceedings of the Second Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 56–61.

D.B. Fogel (1993c) "Evolving Behaviors in the Iterated Prisoner's Dilemma," *Evolutionary Computation*, Vol. 1:1, pp. 77–97.

D.B. Fogel (1994) "Asymptotic Convergence Properties of Genetic Algorithms and Evolutionary Programming: Analysis and Experiments," *Cybernetics and Systems*, Vol. 25:3, pp. 389–407.

D.B. Fogel (1995a) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, NY.

D.B. Fogel (1995b) "On the Relationship between the Duration of an Encounter and the Evolution of Cooperation in the Iterated Prisoner's Dilemma," *Evolutionary Computation*, Vol. 3:3, pp. 349–363.

D.B. Fogel and J.W. Atmar (1990) "Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes Using Linear Systems," *Biological Cybernetics*, Vol. 63, pp. 111–114.

D.B. Fogel and W. Atmar (eds.) (1992) *Proceedings of the First Annual Conference on Evolutionary Programming*, Evolutionary Programming Society, La Jolla, CA.

D.B. Fogel and L.J. Fogel (1988) "Route Optimization Through Evolutionary Programming," *Proc. of 22nd Asilomar Conference on Signals, Systems & Computers*, Pacific Grove, California, pp. 679–680.

D.B. Fogel, L.J. Fogel, and J.W. Atmar (1991) "Meta-Evolutionary Programming," *Proc. of 25th Asilomar Conference on Signals, Systems & Computers*, R.R. Chen (ed.), Maple Press, San Jose, CA, pp. 540–545.

D.B. Fogel, L.J. Fogel, W. Atmar, and G.B. Fogel (1992) "Hierarchic Methods of Evolutionary Programming," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 175–181.

D.B. Fogel, L.J. Fogel, and V.W. Porto (1990) "Evolving Neural Networks," *Biological Cybernetics*, Vol. 63:6, pp. 487–493.

D.B. Fogel and L.C. Stayton (1994) "On the Effectiveness of Crossover in Simulated Evolutionary Optimization," *BioSystems*, Vol. 32:3, pp. 171–182.

D.B. Fogel, E.C. Wasson, and E.M. Boughton (1995) "Evolving Neural Networks for Detecting Breast Cancer," *Cancer Letters*, Vol. 96, pp. 49–53.

L.J. Fogel (1962) "Autonomous Automata," *Industrial Research*, Vol. 4:2, pp. 14–19.

L.J. Fogel (1963) *Biotechnology: Concepts and Applications*, Prentice-Hall, Englewood Cliffs, NJ.

L.J. Fogel (1964) "On the Organization of Intellect," Ph.D. Dissertation, UCLA.

L.J. Fogel (1968) "Extending Communication and Control through Simulated Evolution," *Bioengineering – An Engineering View: Proceedings of the Symp. Engineering Significance of the Biological Sciences*, G. Bugliarello (ed.), San Francisco Press, San Francisco, pp. 286–304.

L.J. Fogel, P.J. Angeline, and T. Bäck (eds.) (1996) *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, MIT Press, Cambridge, MA.

L.J. Fogel, P.J. Angeline, and D.B. Fogel (1995) "An Evolutionary Programming Approach to Self-Adaptation on Finite State Machines," *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, MIT Press, Cambridge, MA, pp. 355–365.

L.J. Fogel and G.H. Burgin (1969) "Competitive Goal-Seeking through Evolutionary Programming," Final Report under Contract no. AF19(628)-5927, Air Force Cambridge Research Labs.

L.J. Fogel and D.B. Fogel (1986) "Artificial Intelligence through Evolutionary Programming," Final Report for U.S. Army Research Institute, Contract # PO-9-X56-1102C-1.

L.J. Fogel, A.J. Owens, and M.J. Walsh (1964) "On the Evolution of Artificial Intelligence," *Proc. of the 5th National Symp. on Human Factors in Electronics*, IEEE, San Diego, CA, pp. 63–76.

L.J. Fogel, A.J. Owens, and M.J. Walsh (1965) "Artificial Intelligence through a Simulation of Evolution," *Biophysics and Cybernetics Systems*, A. Callahan, M. Maxfield, and L.J. Fogel (eds.), Spartan Books, Washington DC, pp. 131–156.

L.J. Fogel, A.J. Owens, and M.J. Walsh (1966) *Artificial Intelligence through Simulated Evolution*, John Wiley, NY.

D.K. Gehlhaar, G.M. Verkhivker, P.A. Rejto, C.J. Sherman, D.B. Fogel, L.J. Fogel, and S.T. Freer (1995) "Molecular Recognition of the Inhibitor AG-1343 by HIV-1 Protease: Conformationally Flexible Docking by Evolutionary Programming," *Chemistry and Biology*, Vol 2:5, 317–324.

M. Gell-Mann (1994) *The Quark and the Jaguar*, Freeman Press, NY.

A. Ghozeil and D.B. Fogel (1996) "A Preliminary Investigation into Directed Mutation in Evolutionary Algorithms," *Parallel Problem Solving from Nature PPSN IV*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (eds.), Springer, Berlin, pp. 329–335.

D.E. Goldberg and R. Lingle (1985) "Alleles, Loci, and the Traveling Salesman Problem," *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, J.J. Grefenstette (ed.), Lawrence Erlbaum, Hillsdale, NJ, pp. 154–159.

S.B. Haffner and A.V. Sebald (1993) "Computer-Aided Design of Fuzzy HVAC Controllers using Evolutionary Programming," *Proceedings of the Second Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 98–107.

P.G. Harrald and D.B. Fogel (1996) "Evolving Continuous Behaviors in the Iterated Prisoner's Dilemma," *BioSystems: Special Issue on the Prisoner's Dilemma*, D.B. Fogel (ed.), Vol. 37:1-2, pp. 135–145.

D.L. Hartl and A.G. Clark (1989) *Principles of Population Genetics*, 2nd ed., Sinauer, Sunderland, MA.

J.-H. Kim and J.-Y. Jeon (1996) "Evolutionary Programming-Based High-Precision Controller Design," *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, L.J. Fogel, P.J. Angeline, and T. Bäck (eds.), MIT Press, Cambridge, MA, pp. 73–81.

K. Kinnear (1993) "Evolving a Sort: Lessons in Genetic Programming," *IEEE International Conference on Neural Networks 1993*, IEEE Press, Piscataway, NJ.

J.R. Koza and D. Andre (1996) "Evolution of Iteration in Genetic Programming," *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, L.J. Fogel, P.J. Angeline, and T. Bck (eds.), MIT Press, Cambridge, MA, pp. 469–479.

M. Land and R.K. Belew (1995) "Towards a Self-Replicating Language for Computation," *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, J.R. McDonnell, R.G. Reynolds, and D.B. Fogel (eds.), MIT Press, Cambridge, MA, pp. 403–413.

R.W. Larsen and J.S. Herman (1992) "A Comparison of Evolutionary Programming to Neural Networks and an Application of Evolutionary Programming to a Navy

Mission Planning Problem," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 127–133.

R.C. Lewontin (1974) *The Genetic Basis of Evolutionary Change*, Columbia University Press, NY.

K. Lindren (1991) "Evolutionary Phenomena in Simple Dynamics," *Artificial Life II*, C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussed (eds.), Addison-Wesley, Reading, MA, pp. 295–312.

B.T. Luke (1994) "Evolutionary Programming Applied to the Development of Quantitative Structure-Activity Relationships and Quantitative Structure-Property Relationships," *J. Chemical Information and Computer Sciences*, Vol. 34:6, pp. 1279–1287.

B.E. Lutter and R.C. Huntsinger (1969) "Engineering Applications of Finite Automata," *Simulation*, Vol. 13, pp. 5–11.

E. Mayr (1960) "The Evolution of Life," Panel in *Evolution after Darwin: Issues in Evolution*, Vol. 3, S. Tax and C. Callender (eds.), Univ. of Chicago Press, Chicago.

J.R. McDonnell (1992) "Training Neural Networks with Weight Constraints," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 111–119.

J.R. McDonnell, B.L. Andersen, W.C. Page, and F.G. Pin (1992) "Mobile Manipulator Configuration Optimization using Evolutionary Programming," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 52–62.

J.R. McDonnell and D. Waagen (1993) "Neural Network Structure Design by Evolutionary Programming," *Proceedings of the Second Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 79–89.

J.R. McDonnell and D. Waagen (1994) "Evolving Recurrect Perceptrons for Time-Series Modeling," *IEEE Trans. Neural Networks*, Vol. 5:1, pp. 24–38.

Z. Michalewicz (1996) *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed., Springer, Berlin.

M. Mitchell (1996) *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA.

D.J. Nettleton and R. Garigliano (1994) "Evolutionary Algorithms and a Fractal Inverse Problem," *BioSystems*, Vol. 33:3, pp. 221–232.

W.C. Page, J.R. McDonnell, and B. Anderson (1992) "An Evolutionary Programming Approach to Multi-Dimensional Path Planning," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 63–70.

V.W. Porto (1992) "Alternative Methods for Training Neural Networks," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 100–110.

V.W. Porto, D.B. Fogel, and L.J. Fogel (1995) "Alternative Neural Network Training Methods," *IEEE Expert*, Vol. 10:3, June, pp. 16–22.

T. Ray (1991) "An Approach to the Synthesis of Life," *Artificial Life II*, C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen (eds.), Addison-Wesley, Reading, MA, pp. 371–408.

M.M. Rizki, L.A. Tamburino, and M.A. Zmuda (1993) "Evolving Multi-Resolution Feature Detectors," *Proceedings of the Second Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 108–118.

M.M. Rizki, L.A. Tamburino, and M.A. Zmuda (1995) "Evolution of Morphological Recognition Systems," *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, J.R. McDonnell, R.G. Reynolds, and D.B. Fogel (eds.), MIT Press, Cambridge, MA, pp. 95–106.

N. Saravanan and D.B. Fogel (1994) "Learning Strategy Parameters in Evolutionary Programming: A Empirical Study," *Proceedings of theThird Ann. Conf. on Evo-*

lutionary Programming, A.V. Sebald and L.J. Fogel (eds.), World Scientific Publishers, River Edge, NJ, pp. 269–280.

N. Saravanan and D.B. Fogel (1997) "Multi-Operator Evolutionary Programming," *Evolutionary Programming VI: Proceedings of the Sixth Annual Conference on Evolutionary Programming*, P.J. Angeline, R.C. Eberhart, R.G. Reynolds, and J.R. McDonnell (eds.), Springer, Berlin, pp. 215–221.

N. Saravanan, D.B. Fogel, and K.M. Nelson (1995) "A Comparison of Methods for Self-Adaptation in Evolutionary Algorithms," *BioSystems*, Vol. 36, pp. 157–166.

H.-P. Schwefel (1981) *Numerical Optimization of Computer Models*, John Wiley & Sons, Chichester, U.K.

H.-P. Schwefel (1995) *Evolution and Optimum Seeking*, John Wiley & Sons, NY.

A.V. Sebald and D.B. Fogel (1992) "Design of Fault Tolerant Neural Networks for Pattern Classification," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 90–99.

A.V. Sebald, J. Schlenzig, and D.B. Fogel (1992) "Minimax Design of CMAC Encoded Neural Controllers for Systems with Variable Time Delay," *Proceedings of the First Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 120–126.

J. Sprave (1994) "Linear Neighborhood Evolution Strategy," *Proceedings of the Third Annual Conference on Evolutionary Programming*, A.V. Sebald and L.J. Fogel (eds.), World Scientific, River Edge, NJ, pp. 42–51.

E.A. Stanley, D. Ashlock, and L. Tesfatsion (1994) "Iterated Prisoner's Dilemma with Choice and Refusal of Partners," *Artificial Life III*, C.G. Langton (ed.), Addison-Wesley, Reading, MA, pp. 131–175.

A. Takeuchi (1980) "Evolutionary Automata Comparison of Automaton Behavior and Restle's Learning Model," *Information Science*, Vol. 20, pp. 91–99.

A HIERARCHICAL GENETIC ALGORITHM FOR SYSTEM IDENTIFICATION AND CURVE FITTING WITH A SUPERCOMPUTER IMPLEMENTATION

MEHMET GULSEN* AND ALICE E. SMITH†

Abstract. This paper describes a hierarchical genetic algorithm (GA) framework for identifying closed form functions for multi-variate data sets. The hierarchy begins with an upper GA that searches for appropriate functional forms given a user defined set of primitives and the candidate independent variables. Each functional form is encoded as a tree structure, where variables, coefficients and functional primitives are linked. The functional forms are sent to the second part of the hierarchy, the lower GA, that optimizes the coefficients of the function according to the data set and the chosen error metric. To avoid undue complication of the functional form identified by the upper GA, a penalty function is used in the calculation of fitness. Because of the computational effort required for this sequential optimization of each candidate function, the system has been implemented on a Cray supercomputer. The GA code was vectorized for parallel processing of 128 array elements, which greatly speeded the calculation of fitness. The system is demonstrated on five data sets from the literature. It is shown that this hierarchical GA framework identifies functions which fit the data extremely well, are reasonable in functional form, and interpolate and extrapolate well.

1. Introduction. In the broadest definition, system identification and curve fitting refer to the process of selecting an analytical expression that represents the relationship between input and output variables of a data set consisting of *iid* observations of independent variables ($x_1, x_2 \ldots, x_n$) and one dependent variable (y). The process involves three main decision steps: (1) selecting a pool of independent variables which are potentially related to the output, (2) selecting an analytical expression, usually a closed form function, and (3) optimizing the parameters of the selected analytical expression according to an error metric (e.g., minimization of sum of squared errors).

In most approaches, closed form functions are used to express the relationship among the variables, although the newer method of neural networks substitutes an empirical "black box" for a function. Fitting closed form equations to data is useful for analysis and interpretation of the observed quantities. It helps in judging the strength of the relationship between the independent (predictor) variables and the dependent (response) variables, and enables prediction of dependent variables for new values of independent variables. Although curve fitting problems were first introduced almost three centuries ago [25], there is still no single methodology that can be applied universally. This is partially due to the diversity of the problem areas, and particularly due to the computational limitations

*Department of Industrial Engineering, 1031 Benedum Hall, University of Pittsburgh, Pittsburgh, PA 15261.

†Department of Industrial Engineering, 1031 Benedum Hall, University of Pittsburgh, Pittsburgh, PA 15261, aesmith@engrng.pitt.edu

of the various approaches that deal with only subsets of this broad scope. Linear regression, spline fitting and autoregressive analysis are all solution methodologies to system identification and curve fitting problems.

2. Hierarchical genetic approach. In the approach of this paper, the primary objective is to identify an appropriate functional form and then estimate the values of the coefficients associated with the functional terms. In fact, the entire task is a continuous optimization process in which a selected error metric is minimized. A hierarchical approach was used to accomplish this sequentially as shown in Figure 1. The upper genetic algorithm (GA) selects candidate functional forms using a pool of the independent variables and user defined functional primitives. The functions are sent to the lower (subordinate) GA which selects the coefficients of the terms of each candidate function. This selection is an optimization task using the data set and difference between the actual and fitted y values.

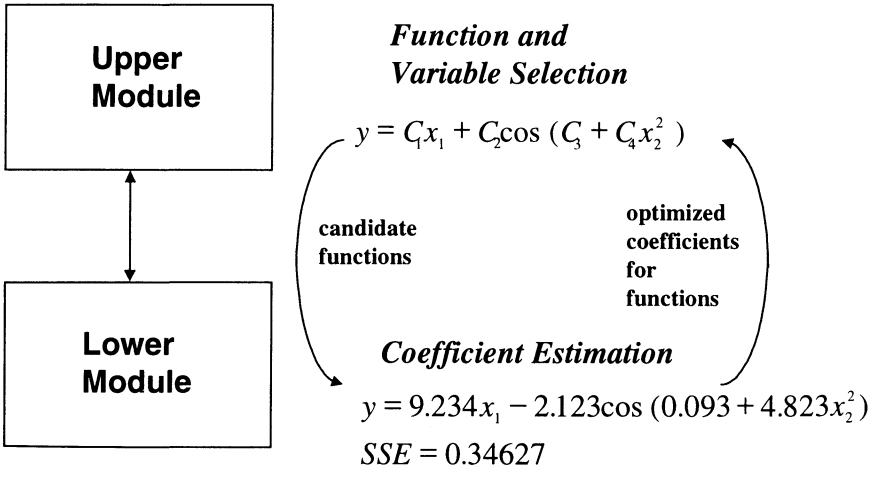


FIG. 1. *Hierarchical GA framework.*

GA has been applied to system identification and curve fitting by several researchers. The relevant work on these can be categorized into two groups. The first category includes direct adaptation of the classical GA approach to various curve fitting techniques [17, 23]. In these works GA replaces traditional techniques as a function optimization tool. The second category includes more comprehensive approaches where not only

parameters of the model, but the model itself, are search dimensions. One pioneering work on this area is Koza's adaptation of Genetic Programming (GP) to symbolic regression [18]. The idea of genetically evolving programs was first implemented by [8, 12]. However, GP has been largely developed by Koza who has done the most extensive study in this field [18, 19]. More recently, Angeline used the basic GP approach enhanced with adaptive crossover operators to select functions for a time series problem [1].

3. Coefficient estimation (lower GA). The approach consists of two independently developed modules which work in a single framework. The lower GA works as a function call from the upper one to measure how well the selected model fits the actual data. In the lower GA, the objective is to select functional term coefficients which minimize the total error over the set of data points being considered. A real value encoding is used. The next sections describe population generation, evolution, evaluation, selection, culling and parameter selection.

3.1. The initial population. Using the example of the family of curves given by equation 3.1, there are two variables, x_1 and x_2, and four coefficients. Therefore each solution (population member) has four real numbers, C_1, C_2, C_3, C_4.

$$y = C_1 + C_2 x_1^3 + C_3 \cos x_2 + C_4 x_1 x_2 \tag{3.1}$$

The initial estimates for each coefficient are generated randomly over a pre-specified range defined for each coefficient. The specified range need not include the true value of the coefficient (though this is beneficial). The uniform distribution is used to generate initial coefficient values for all test problems studied in this paper, although this is not a requirement either (see [15] for studies on diverse ranges and other distributions).

3.2. Evolution mechanism. A fitness (objective function) value which measures how well the proposed curve fits the actual data is calculated for each member of the population. Squared error is used here (see [15] for studies on absolute error and maximum error). Minimum squared error over all data points results in maximum fitness.

Parents are selected uniformly from the best half of the current population. This is a stochastic form of truncation selection originally used deterministically in evolution strategies for elitist selection (see [2] for a good discussion of selection in evolution strategies). Mühlenbein and Schlierkamp-Voosen later used the stochastic version of truncation selection in their Breeder Genetic Algorithm [20]. The values of the offspring's coefficients are determined by calculating the arithmetic mean of the corresponding coefficients of two parents. This type of crossover is known as extended intermediate recombination with $\alpha = 0.5$ [20].

For mutation, first the range for each coefficient is calculated by determining the maximum and minimum values for that particular coefficient in

the current population. Then the perturbation value is obtained by multiplying the range with a factor. This factor is a random variable which is uniformly distributed between $\pm k$ coefficient range. The value of k is set to 2 in all test problems in this paper. However, the range among the population generally becomes smaller and smaller during evolution. In order to prevent this value from becoming too small, and thus insignificant, a perturbation limit constant, Δ, is set. This type of mutation scheme is related to that used in Mühlenbein and Schlierkamp-Voosen's Breeder Genetic Algorithm [20]. However their range was static where the range in this approach adapts according to the extreme values found in the current population with a lower bound of Δ. Mutation is applied to M members of the population, which are selected uniformly. After mutation and crossover, the newly generated mutants and offspring are added to the population and the lowest ranked members are culled, maintaining the original population size, ps.

Cycle length, cl, is used as a basis for terminating the search process. It refers to the number of generations without any improvement in the fitness value prior to termination. If the algorithm fails to create offspring and mutants with better fitness values during this fixed span of generations, the search process is automatically terminated.

3.3. Parameter selection. The lower GA was examined at various population parameters, namely population size ps, number of mutants M, number of offspring O, perturbation limit constant Δ and cycle length cl, as shown in Table 1. The experimental design covered a wide range of values and a full factorial experiment was performed.

Based on experimental test results, $ps = 30, O = 10, M = 10, \quad \Delta = 10^{-5}$ and $cl = 100$ were selected for the lower GA. The strategy in selecting and cycle length is based on selecting the maximum Δ value and minimum cycle length provided that the algorithm satisfies convergence criteria for all cases. Increasing Δ or decreasing cl reduces computational effort, but it may also deteriorate the accuracy of the final results. An opposite strategy, decreasing Δ or increasing cl improves accuracy, but it will take the algorithm more effort to converge to the final results.

3.4. Test problems. Since the lower GA module was initially developed independently from the framework, it was tested on three test problems where the correct functional form is provided. Two of these three test problems were taken from previous GA studies on curve fitting.

The first test problem is to estimate the value of two coefficients (C_1 and C_2) of the straight line given in equation 3.2. This was studied by [17] with their GA approach.

$$y = C_1 x + C_2 \tag{3.2}$$

The second test problem, which is given in equation 3.3, has 10 independent variables and 9 coefficients to be estimated. This was studied for spline

TABLE 1
Evolution Parameter Levels for the Lower GA.

Level	*ps*	*O*	*M*	Δ	*cl*
Low	30	10	5	10^{-6}	10
				10^{-5}	50
					100
Medium	50	20	10	10^{-4}	500
				10^{-3}	1000
					2000
High	100	50	20	10^{-2}	5000

fitting using a GA by [23] and, before that, for conventional spline fitting by [11]. A set of 200 randomly selected data points was used, and the problem was designed so that the response depends only on the first five variables. The remaining variables, x_6 through x_{10}, have no effect on the response function, and they are included in the data set as misleading noise. Therefore the optimal values of C_5 through C_9 should be zero.

$$(3.3) \quad y = C_1 \sin(\pi x_1 x_2) + C_2(x_3 - 0.5)^2 + C_3 x_4 + C_4 x_5 + \sum_{n=6}^{10} C_{n-1} x_n$$

The third test problem is a general second level equation of three variables. The function, which is given in equation 3.4, has 10 coefficients to be estimated, and it is larger and more complex than any that can be found in the previous literature on using genetic algorithms for coefficient estimation.

$$(3.4) \quad y = C_1 + C_2 x_1 + C_3 x_2 + C_4 x_3 + C_5 x_1^2 + C_6 x_2^2 + C_7 x_3^2 + C_8 x_1 x_2 + C_9 x_1 x_3 + C_{10} x_2 x_3$$

The performance of the lower GA was tested with respect to data set size, population distribution and interval for sampling, error metric, initial coefficient range and random seeds. Robust performance was observed for all of the above parameters and performance exceeded previous approaches in terms of minimization of error (for details see [15]). The lower GA is

highly flexible, and it can be adopted to different problem instances and parameters with very minor modifications in the algorithm. The coefficient optimization module is used as a subroutine to the function identification module described in the next section.

4. Functional form selection (upper GA). Although the desire to develop a heuristic method to select the functional form was expressed in the literature as early as 1950s, it was suggested as a possible extension of variable selection in stepwise regression [7, 9, 28]. In the approach of this paper, a very liberal selection procedure is used so that the search domain is open to all possible functional forms, but can also be restricted by the user.

4.1. Encoding. In terms of GA parlance, each member of the population set is a closed form function. Since the elements of the population are not simple numeric values, a binary array representation is not applicable. Instead, a tree structure is used to represent each closed form function. As an example, the tree structure for the candidate solution given below (equation 4.1) is illustrated in figure 2.

$$y = 3.2 + 7.5x_1^3 + 1.6\cos(-4.1x_2) + 4.6x_1x_2 \tag{4.1}$$

The nodes in a tree structure can be categorized into two groups according to the number of outgoing branches: the "terminal node" is the last node in each branch without any outgoing branches. The nodes between the first and terminal nodes are called "intermediate nodes" and they have at least one outgoing branch. Each intermediate node has an operand (e.g., sin, cos, exp, ln, $+$, $\times$) and the terminal node carries a variable or a constant. The numbers beside nodes in figure 2 are real valued coefficients (obtained from the lower GA).

Each tree structure has a starting node where branching is initiated. The number of branches originating from a node is determined probabilistically, restricted to 0, 1 or 2. The only exception is the first node where there must be at least one branch in order to initiate the generation of the tree structure. The branches connecting the nodes also determine the "parent offspring relation" between the nodes. For a branch coming out of node a and going into node b, node a is named the parent of node b. In a similar manner, node b is called a child of node a. Each parent node can have one or two child nodes. It is also possible that the parent may not have a child. If a node has two children, they are respectively called the left-child and the right-child. However, if a node has only one child, it is still called the left-child.

For computational reasons, such as the fitness calculation, the nodes are indexed. The indexing process starts from the first node, which is indexed as 1. At this point the number of children are also calculated for the node. Then the process moves, first to the left-child of the current node (now the child becomes a parent), and indexes it as 2. This process

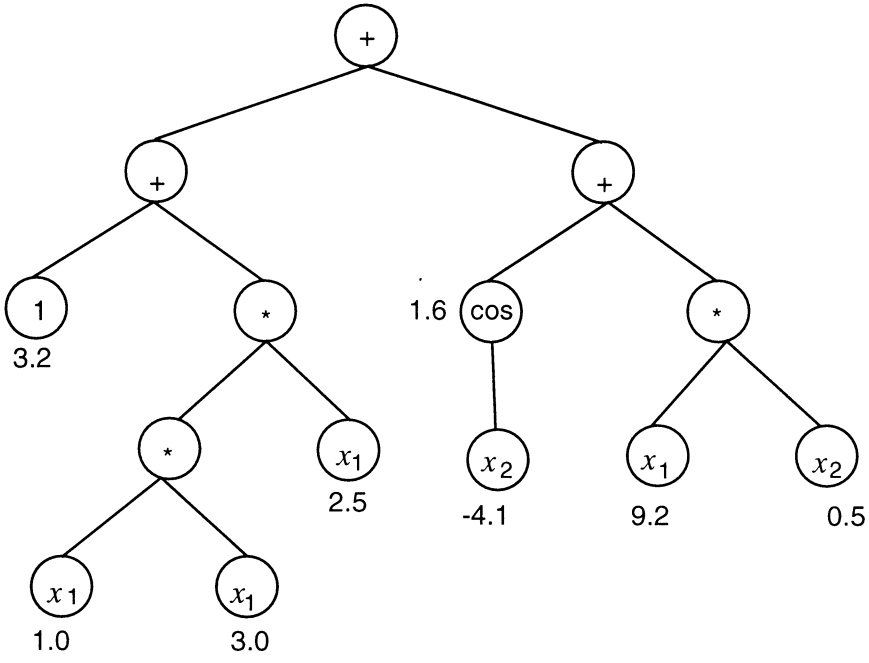


FIG. 2. *Tree structure representation of equation 4.1.*

continues until a node is reached with no left-child. In such a situation the process backups to the parent node and looks for a right-child. If parent has a right-child which is not indexed yet, then it is indexed. Otherwise, the procedure climbs up to next parent (or grandparent). This recursive process, which is analogous to a "depth first search" algorithm, stops when all the nodes are indexed. Figure 3 illustrates the tree structure of figure 2 which is indexed according to this procedure.

The number of branches from a node will be $\Pr(B_1, B_2, B_3)$ where B_1, B_2 and B_3 represent the probability of having 0, 1 or 2 offspring nodes respectively. Special attention should be paid when assigning values for B_1, B_2 and B_3. For example if B_1, B_2 and B_3 are selected as 0.2, 0.4 and 0.2, each node, on the average, will have $0.2 \times 0 + 0.4 \times 1 + 0.2 \times 2 = 1.2$ offspring nodes. With such a number for the mean number of offspring nodes, it is possible to have tree structures with less than 10 or 15 nodes. However, it is also possible that tree structures which have tens or hundreds of nodes will be created. To remedy this, B_1, B_2 could be increased and B_3 decreased. However, this will create deep structures which will reduce

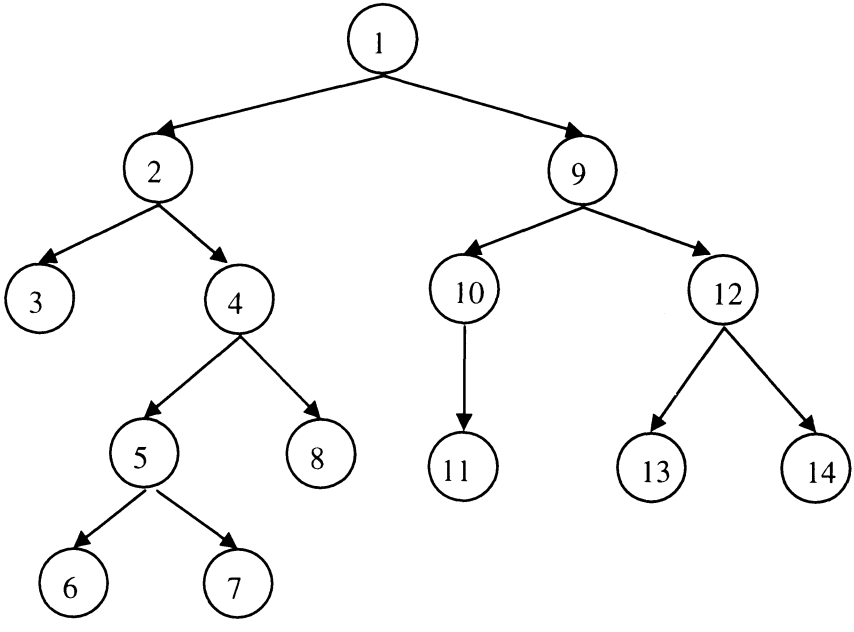


FIG. 3. *Indexing of a tree structure.*

the efficiency of representing the functional forms. As a remedy, the values of B_1, B_2 and B_3 change during the course of branching. At the early stages, there is a higher probability for double branching which diminishes gradually as the number of nodes increase. This is done by choosing a factor, δ, that is equal to $0.04 \times$ *index number*. The δ value is added to B_1, and it is subtracted from B_3 at each node. As the number of nodes becomes larger, the probability of double branching decreases with respect to the probability of no branching.

One dimensional arrays were used to store the tree structures. The following information is stored for each node: (1) index number, (2) number of offspring nodes, (3) index number of left child node, (4) index number of right child node, (5) index number of parent node, (6) type of functional primitive assigned to the node, and (7) whether the node carries a coefficient or not.

4.2. Functional primitives. The functional primitives are categorized into three hierarchical groups. The lowest level includes the independent variables of the data set and the constant 1. These are assigned

TABLE 2
Node primitives.

Levels	Primitives*
First Group	$x_1, \ldots, x_n$, 1
Second Group (unary)	sin, cos, exp, ln
Third Group (binary)	$+, \times, \div$

*n is the number of independent variables in the dataset.

only to terminal nodes and all carry a coefficient. If the variable x is assigned to a node and has coefficient value of 4.3, then it corresponds to $4.3x$. Similarly, a node with constant 1 and coefficient -2.3 corresponds to $-2.3 \times 1 = -2.3$. The second and third groups include unary and binary operators respectively. While operators such as sin, cos, ln, exp, are categorized as second group operators, multiplication "$\times$", division "$\div$", and addition "+" operators form the third group (table 2). Functional primitives can be customized by the user according to estimated characteristics of closed form representation and the desired restrictiveness of the search.

The assignment of primitives to the nodes is carried out in the following manner: the first group elements x_n and the constant 1 are assigned to terminal nodes where there is no outgoing branch. The unary operators are assigned to nodes with a single outgoing branch and the binary operators are assigned to nodes with two outgoing branches. Each of the first and second group operators is proceeded by a coefficient. The third group operators do not have a coefficient preceding them, because they simply combine two sub-functions which already have coefficients.

4.3. Initial population and fitness. The initial population is randomly created in three stages. In the first stage, the tree structure is developed for each member of the population. At the second stage, appropriate primitives are assigned to the nodes in a random manner. After the functional form representation is obtained for the initial population, the fitness value of each member is calculated. The coefficient estimation process is done by the subordinate GA program, which is used as a function call in the main program. Two metrics are used for fitness: the first one is the raw fitness, which is the sum of squared errors. The second fitness value is developed to penalize complex functional representations in order to prevent overfitting. This secondary fitness value, called "penalized fitness", is calculated by worsening the raw fitness in proportion to the complexity of the closed form function.

TABLE 3
Penalized fitness.

Raw Fitness	9.78
Number of Nodes	12
T	5
P	0.4
Complexity Coefficient	$\left(\frac{12}{5}\right)^{0.4} = 1.4193$
Penalized Fitness	$1.4193 \times 9.78 = 13.88$

The complexity of a closed form function is determined by the number of nodes it has. However complexity is not a absolute concept and the user needs to judge the level of complexity for each data set. The objective is to balance the bias and variance in a closed form representation, where bias results from smoothing out the relationship and variance indicates dependence of the functional form on the exact data sample used [14]. Ideally, both bias and variance will be small. A penalty function with user tunable parameters is used to balance bias and variance of the functional form in the GA approach.

The penalty is determined by dividing the number of nodes in the tree structure by a threshold, T, and then taking the $(P)^{th}$ power of the quotient (equation 4.2), where $0 < P < 1$. T corresponds to the minimum number of nodes in an unpenalized tree structure. In this paper $T = 5$ was used, thus every tree structure was subject to some level of penalty if there were more than 5 nodes. Conservatively, a small value of T can be used safely while letting P impose the severity level of the penalty. A sample calculation for penalized fitness is presented in table 3.

$$\text{(4.2)} \quad \textit{Penalized fitness} = \textit{raw fitness} \times \left(\frac{\textit{number of modes}}{T}\right)^{P}$$

4.4. Evolution mechanism. The evolution mechanism is carried out through crossover and mutation. In each generation, O offspring and M mutants are generated. Selection for crossover and mutation, the culling process and algorithm termination are done as in the lower GA.

Crossover is carried out by swapping sub-branches of two members at

randomly selected nodes (figures 4 and 5). Each crossover process produces two offspring. Mutation is performed by replacing a sub-branch (selected at any node randomly) with a new randomly generated one. One solution could produce more than one mutant provided that branching is performed at different nodes. The new branch used for replacement is a full function (like a member of the initial population) but smaller in size (figures 6 and 7).

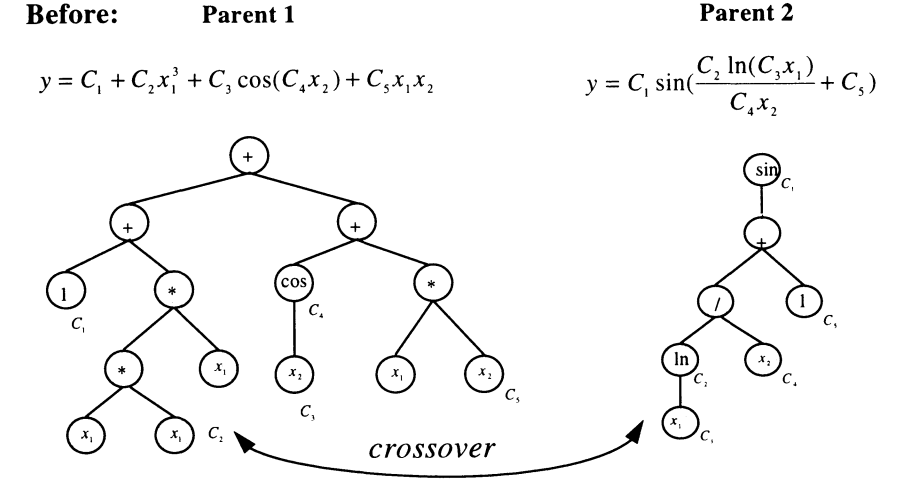


FIG. 4. *Two functions before crossover.*

4.5. Parameter setting. The experiments on the upper GA were performed using the selected parameters of the lower GA held constant and a cycle length of 100 generations for the upper GA. Since the algorithm converged to sufficiently accurate results for all population parameter combinations tested (Table 4), the population parameters were selected only on the basis of computational efficiency (total number of fitness calculations performed during the search process). Based on this criterion, $ps = 30, O = 10$, and $M = 20$ were selected for the upper GA. The penalty severity factor P was harder to choose. A low value enables better fit as measured by the sum of the squared errors (SSE) but encourages more complex functions. A high value has the opposite effect. In the test problems, small data sets were run with $P = 0.40$ and larger data sets were run at both $P = 0.05$ and $P = 0.40$. However, the user will probably want

After: **Offspring 1**

$$y = C_1 + \frac{C_2 \ln(C_3 x_1)}{C_4 x_2} + C_5 \cos(C_6 x_2) + C_7 x_1 x_2$$

Offspring 2

$$y = C_1 \sin(C_2 x_1^3 + C_3)$$

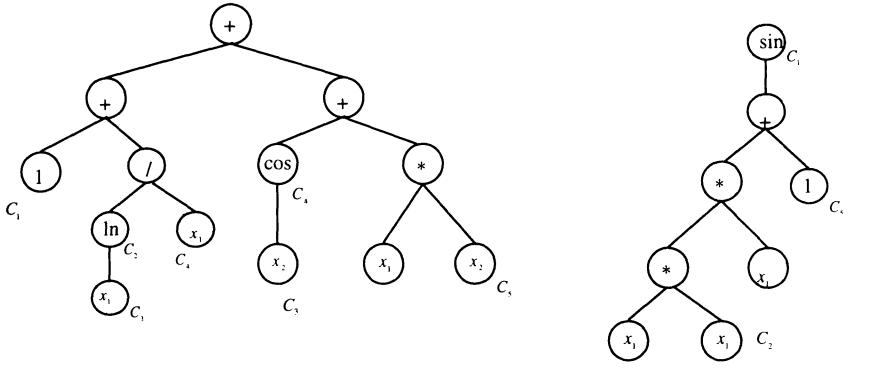


FIG. 5. *Two functions after crossover.*

TABLE 4
Evolution parameter levels for the upper GA.

Level	*ps*	*O*	*M*	*P*
Low	30	10	20	0.05
				0.10
Medium	50	20	40	0.20
				0.40
High	100	50	60	0.80

to run the GA at several values of P and compare the resulting error and functional forms.

5. Computational considerations. The framework includes two sequential search algorithms (the upper and lower GAs). For each closed form representation, the coefficients are determined by running a secondary

Before: **Parent 1**

$$y = C_1 + C_2 x_1^3 + C_3 \cos(C_4 x_2) + C_5 x_1 x_2$$

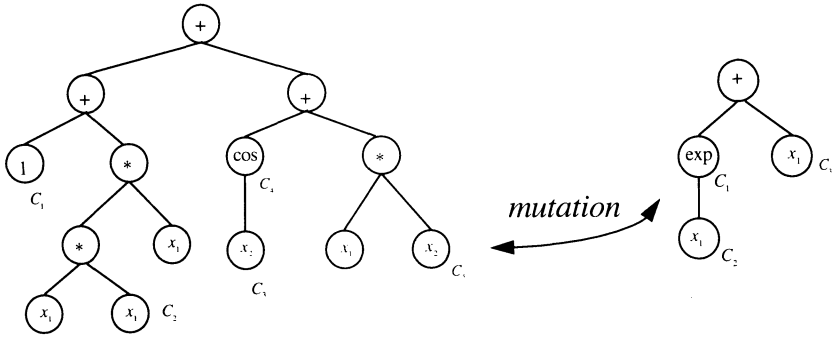


FIG. 6. *Function before mutation.*

search routine. In a similar manner, for each set of coefficient values, an error metric is calculated over the data points. The computational requirement expands exponentially from top to the bottom (figure 8).

Because of the exploding computational characteristic of the problem, maximum attention was paid to increase the efficiency of the code. First, early detection of pathological behavior at the higher levels was done. For example, anticipated floating point errors, ln, division or exp terms in sequence with themselves, and three or more unary terms in sequence all terminate the GA for that solution.

The search algorithm was coded on a C/Unix platform. The program includes five imbedded loops as shown in figure 9: a "do-while and for" loop combination for each search routine and an innermost "for" loop for fitness calculations. In both search routines, the "do-while" loops control the search process. The next level "for" loops control execution of the code for a set of candidate solutions generated within each "do-while" loop. The initial four imbedded loops are highly "fragile" with several control mechanisms which may skip loop execution for some cases or even break the loop.

Test problems 1, 2 and 5 were done on Sun/SPARC^{TM} workstations. For test problems 3 and 4, the program was run on a Cray C90^{TM} su-

After: Mutant

$$y = C_1 + C_2 x_1^3 + C_3 \cos(C_4 x_2) + C_5 x_1 \exp(C_6 x_1) + C_7 x_1^2$$

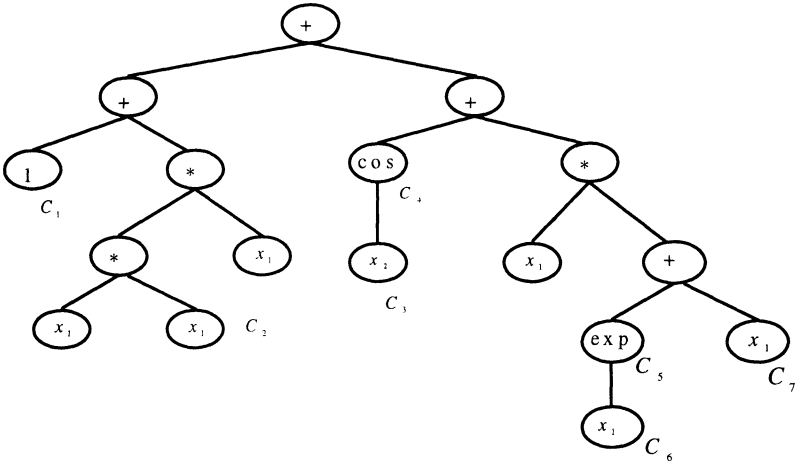


FIG. 7. *Function after mutation.*

percomputer. In order to use the "vectorization" utility, the innermost loop, where the error metric (fitness) is calculated over the data set, was reorganized to eliminate all data dependencies. The vector size was 128, which means 128 operations when calculating fitness can be done in parallel. With a fully vectorized innermost loop the program's flop rate ranged between 170 Mflops to 350 Mflops, depending on the data set size.

6. Empirical data sets. In order to evaluate the effectiveness of the GA framework, five benchmark problems from the literature were studied. The size of the test problems ranges from single variable/50 observations to 13 variables/1000 observations. Each of the five test problems addresses a specific issue in the curve fitting field. The first two problems are classic examples from the nonlinear regression literature. The third test problem is sunspot data, which is one of the benchmark problems in time series analysis. The fourth test problem includes 13 variables and approximately 1000 observations. This data set was originally used for neural network modeling of a ceramic slip casting process. The last problem with 5 variables was studied earlier for qualitative fuzzy modeling and system identification using neural networks.

The GA was run five times for each penalty factor with different random number seeds. For test problems 1 and 2 only one penalty factor,

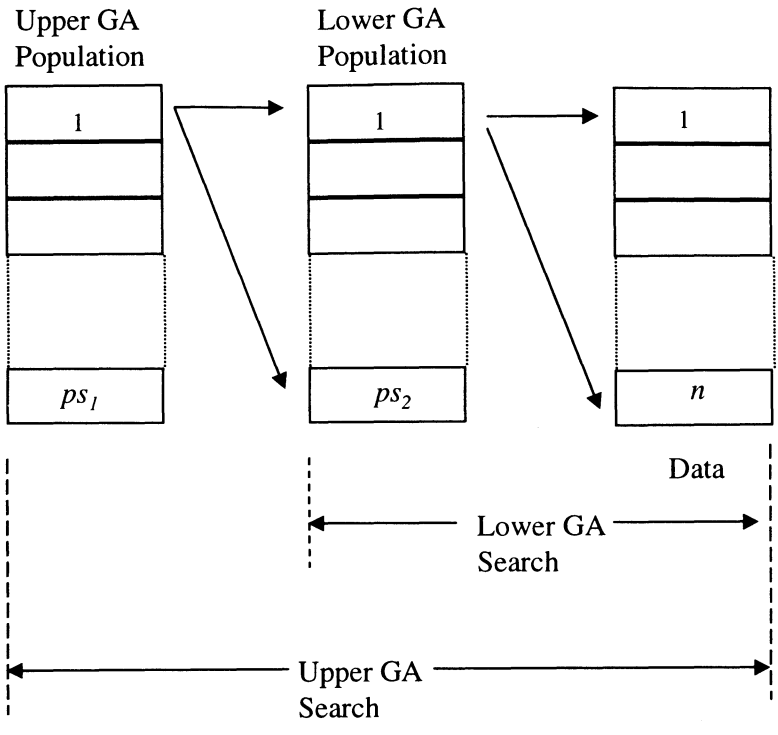


FIG. 8. *Problem layout.*

```
do {
   create O1 crossovers and M1 mutants
      for (i = 0; i < O1 + M1)
            do {
                  create O2 crossovers and M2 mutants
                     for (i = 0; i < O2 + M2)
                           for (i = 0; i < n_data)
                                 fitness calculation
               } while (lower GA search criteria is not satisfied);
   } while (upper GA search criteria is not satisfied);
```

FIG. 9. *Pseudo-code for search algorithm.*

0.4, was used. For test problems 3 through 5, two penalty factors, 0.4 and 0.05, were used. As expected with the higher penalty factor, the algorithm converged to more compact functions. When the penalty was relaxed, the final closed form representations became more complex, but with better fitness values.

6.1. Test problem 1. This example is taken from Nash [21], and it is one of the classic data sets of nonlinear regression that has been studied by many researchers. The data gives the growth of White Imperial Spanish Onions at Purnong Landing. It includes only one independent variable, density of plants (plants/meter squared), and a dependent variable, yield per plant. For the data set, Nash cites three potential models for the relationship between the independent variable, x (density), and the dependent variable, y (yield):
Bleasdale and Nelder [4]:

$$y = (b_1 + b_2 x)^{-1/b_3} \tag{6.1}$$

Holliday [16]:

$$y = (b_1 + b_2 + b_3 x^2)^{-1} \tag{6.2}$$

Farazdaghi and Harris [10]:

$$y = (b_1 + b_2 x^{b_3})^{-1} \tag{6.3}$$

Since density figures are in the order of hundreds and power terms are present in the models, the parameters are logarithmically scaled. Determination of model parameters of the above equations was based on minimizing a natural logarithm error metric.

The GA used the data directly, not in the logarithmic form. Using a penalty factor = 0.4, four runs were selected for further examination. As can be seen from the listing in table 5, the simplest closed form function has two imbedded logarithmic terms divided by the independent variable, x. The second representation is more complex than the first one, and it has an exponential term with a compound power argument. The third and fourth functions are complex versions of the second and first functions respectively. Apart from the first closed form function, all equations have sinusoidal components. Since the actual data is not cyclic in nature, presence of such terms implies overfitting the data. The plot of the first function is presented in figure 10. The comparison of SSE values of the GA models with respect to those cited by Nash is given in table 6. In order to establish a common base for comparison, SSE values were recalculated for the four GA models after a logarithmic transformation. Although the error metric for fitness calculation was different than those used in the models cited by Nash [21], all of the GA models gave superior results in terms of logarithmic SSE values.

6.2. Test problem 2. This example is taken from Galant, and it is one of the classic data sets of the nonlinear regression literature [13, 24]. The data set has 70 observations for two variables; age of preschool boys in months (independent variable) and weight/height ratio (dependent variable) (Figure 11). In terms of SSE value, the best fit using nonlinear

TABLE 5
Selected closed form functions for test problem 1.

Model	Equation	SSE
A	$2190.3680\ln(33.7043(0.0597x))/x$	5065.244
B	$17.6377\exp(-2.8443\sin(-0.3734\ln(-35.8300\cos(0.9751x)+8.7108x)))$	4021.240
C	$12.6422\exp(-3.0512\sin(-0.3957\ln(-28.811\cos(-5.2830\sin(12.3316x))+4.7735x)))$	3651.378
D	$176.2871\ln(116.0904\cos(-1.0273\sin(182.5961x)))$ $/(35.6118/x+0.0833x)$	3388.789

TABLE 6
Comparison of results for test problem 1.

Model	SSE
Bleasdale-Nelder	0. 2833
Holliday	0. 2842
Farazdaghi-Harris	0. 2829
Model A	0. 276
Model B	0. 238
Model C	0. 216
Model D	0. 214

regression for the data set is obtained by a piece-wise polynomial regression where the data set is divided into separate regions. The SSE values for a two region quadratic-linear and a three region quadratic-quadratic-linear models are 0.03790 and 0.03754 respectively [13, 24].

The closed form representations for four selected solutions are given in table 7. The first two functions include linear combination of two components; C_1x and $C_2e^{C_3f(x)}$. Since the power term of the exponential component has a negative coefficient (C_3), its contribution to the output diminishes as x becomes larger, and the closed form function behaves as a simple linear function. The third and fourth functions are more complex

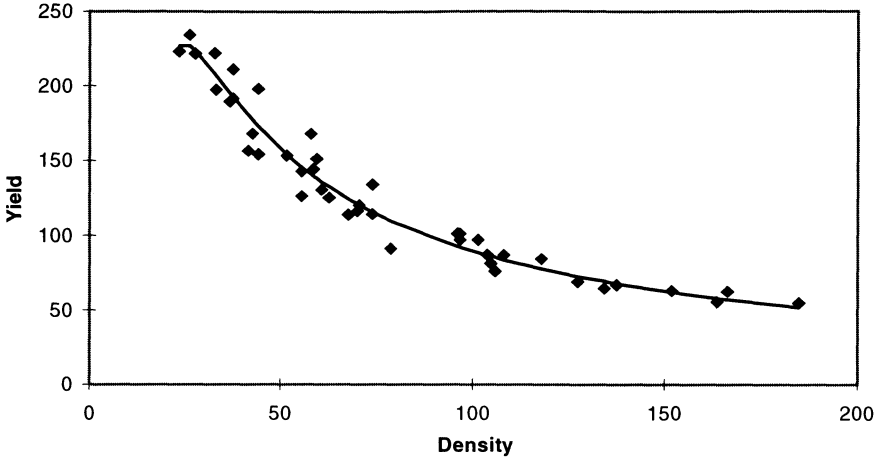


FIG. 10. *Fitted function and data set for test problem 1 (model A).*

TABLE 7
Selected closed form functions for test problem 2.

Model	Equation	SSE
A	$0.7357\exp(-0.6072\exp(-0.2837x)) + 0.0038x$	0.0383
B	$0.7324\exp(-0.4853\sin(1.7409\exp(-0.3532x))) + 0.0039x$	0.0373
C	$0.1545\ln(-44.8315\sin(6.4001x) + 12.3580\sin(-3.4739\sin(-5.4145x)) + 8.5875x)$	0.0355
D	$0.1648\ln(-9.7605\sin(-14.6836 + 3.3565x) + 27.4178\sin(6.1689x) + 9.0167 + 5.7035x)$	0.0320

and they both have the following representation: $C_1 \ln(C_2 f(x) + C_3 x)$. Since $f(x)$ consists only of sinusoidal terms, the function is bounded by $C_3 x$ for the large values of x, and the effects of $f(x)$ becomes marginal as x becomes larger.

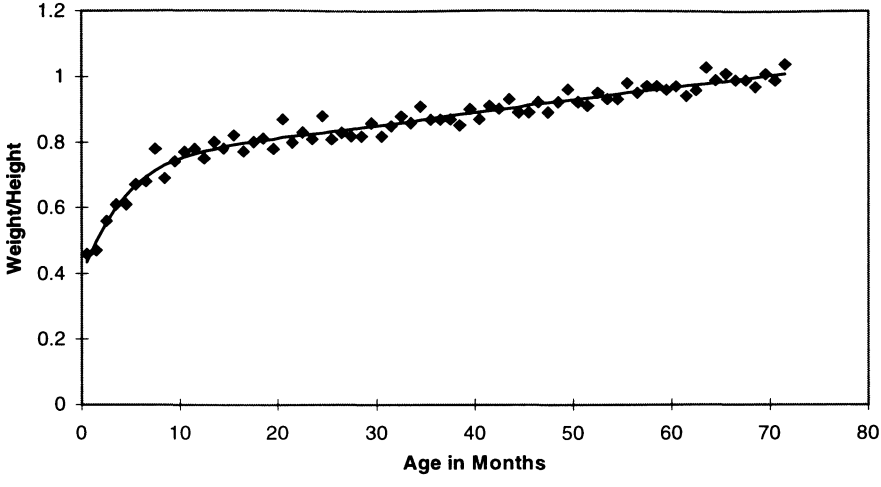


FIG. 11. *Fitted function for test problem 2 (Model A).*

6.3. Test problem 3. For time series analysis sunspot data from 1700 to 1995 was used. The sunspot number is computed by counting each individual spot and cluster of spots on the face of the sun. According to a technique developed by Johann Rudolph Wolf (1816-1893) of the Zurich Observatory, each cluster of spots is counted as 10 spots. Currently this number is referred to as the Zurich Sunspot Number, and it is the most commonly used measure for sunspot counting. Since results vary according to interpretation of the observer and the atmospheric conditions above the observation site, an international number is computed by averaging the measurements of 25 cooperating observatories throughout the world [1]. The data is highly cyclic with peak and bottom values approximately in every 11.1 years. However, the cycle is not symmetric, and the number of counts reaches a maximum value faster than it drops to a minimum.

The original data which had only one input and the output value (time (t) and sunspot number $Y(t)$) was reorganized to include 15 additional input variables, sunspot numbers $Y(t-15)$ to $Y(t-1)$ (equation 6.4). The observations start from 1715 and the first observation includes the following variables: time $(t = 15)$ and 15 sunspots numbers from 1700 to 1714. The data was divided into two groups. The first 265 observations (until 1979) were used to develop the models. In order to evaluate the model performance beyond the training data, sunspot data from 1980 to 1995 was used.

$$Y(t) = f(t, Y(t-1), Y(t-2), \ldots, Y(t-15)) \tag{6.4}$$

[1]For more information about sunspots and computation techniques see the homepage of the National Oceanic and Atmospheric Administration at http://www.ngdc.noaa.gov.stp.SOLAR.SSN.ssn.html.

TABLE 8
Selected closed form functions for test problem 3.

Model	Equation	SSE
A	$1.1965(t-1)-0.4585(t-2)+0.2471(t-9)$	61964
B	$0.8337(t-1)-1.1989\exp(-0.3512(t-9))$ $+15.7476\exp(-2.7260\exp(-0.3263(t-1))-0.6271(t-2))$	45533
C	$1.2410\exp(-0.6099\cos(1.4097(t-4)+0.4282(t-1))$ $-0.8446(t-2))(t-1)+0.8064(t-1)-0.1316(t-4)+0.1148(t-9)$	40341
D	$1.6258\exp(-0.5564\cos(-1.4893(t-4)$ $+0.6979\exp(-3.3442\cos(0.2561(t-4))$ $+2.8807(t-2))-3.1756(t-2))-0.7485(t-2))-0.9362(t-2)(t-1)$ $+0.8253(t-1)-0.1413(t-4)+0.1046(t-9)$	38715

The closed form representations for the selected solutions are given in table 8. The simplest model (model A) has only linear components of, $(t-1), (t-2), (t-9)$, and is similar to a Box Jenkins model [5] which is one of the early models of sunspot data. However, the Box Jenkins model consists only of $(t-1)$ and $(t-2)$, whereas the GA model adds one extra term $(t-9)$. The second model includes the same components but its nonlinear representation is more complex than the first. The third and fourth functions have one additional term $(t-4)$ and they are both more complex than first two models (A and B). The plot of the last model (D) is presented in figure 12. The extrapolation of this model on a forecasting range of 1980 to 1996 is given in figure 13 when forecasting one period ahead.

Sunspot data has been analyzed by many researchers [1, 22, 27]. In a recent study by Park [22], different neural network structures were employed to model sunspot data from 1770 to 1869. His study reports mean squared error values (MSE) ranging from 233.8 to 151.9. The MSE values in the GA models ranges from 234.7 (model A) to 146.7 (model D). For forecasting data MSE ranges from 291.1 (model A) to 145.5 (model C). Angeline [1] also studied this data set with a GP approach. He used values of $(t-1), (t-2), (t-4)$ and $(t-8)$ only, and reports SSE over the period

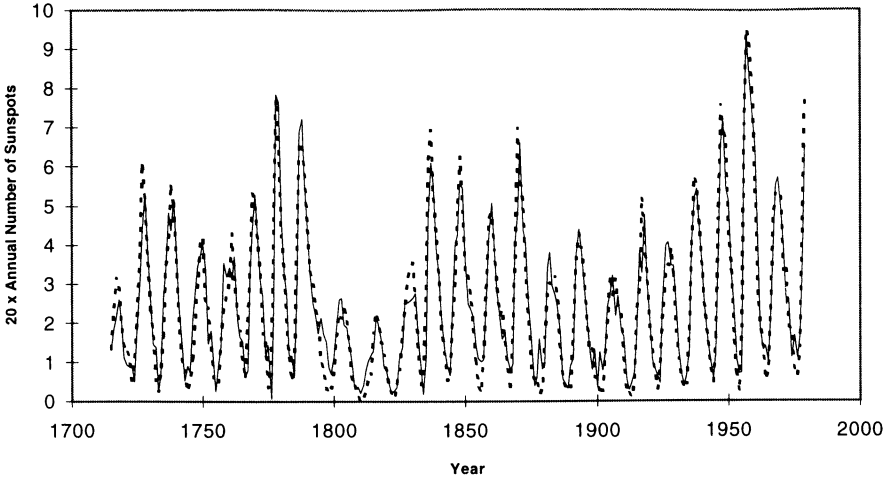


FIG. 12. *Fitted function for test problem 3 (Model D).*

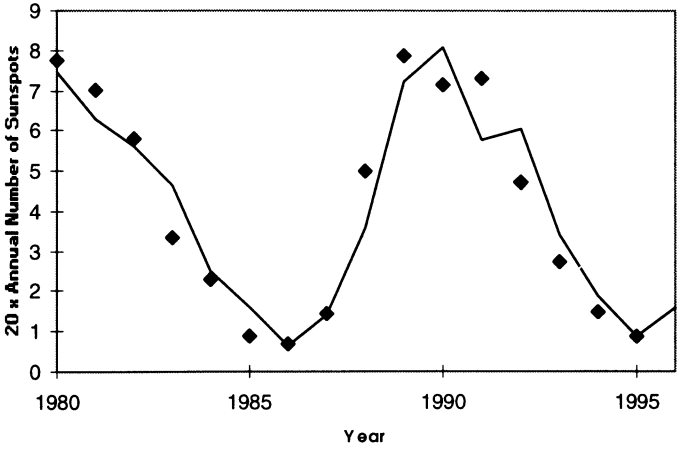


FIG. 13. *Extrapolation of models beyond training data range.*

1708 to 1993 of 80000 and average percent error of 8%. This compares with the GA SSE of about 62000 to 39000, equating to roughly 5% to 6% error.

6.4. Test problem 4. This example with 13 independent variables and approximately 1000 observations was the largest test problem used in this research. The data comes from a ceramic casting process where the relationship between the casting rate and 13 operational input parameters was investigated [6]. In figure 14, the output is plotted for each observation. As can be seen from this figure, because of high dimensionality of the data set, it is impossible to visually detect any pattern in the data.

Selected closed form representations are given in table 9. Three of

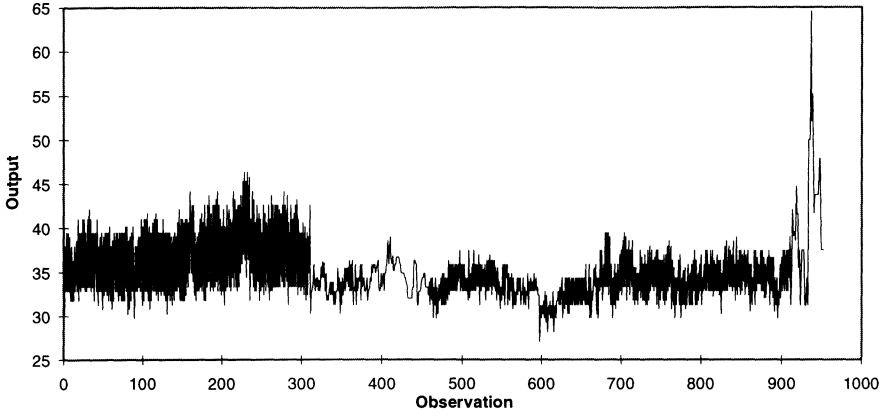


FIG. 14. *Multi-variable casting data.*

the first four functions include linear combination of two variables, x_3 and x_{10}, which seem to be the most important variables in the data set. Interestingly, the coefficients associated with these variables are very close to each other in all three functions. This strongly suggests that the additional components in the third and fourth equations have a lesser effect on the output. The plot of the difference between the fitted model and the actual data points (error) is illustrated in figure 15. The results of model E roughly correspond to the same error rate obtained by the neural network of [6], however the GA approach has the advantage of a discernible form.

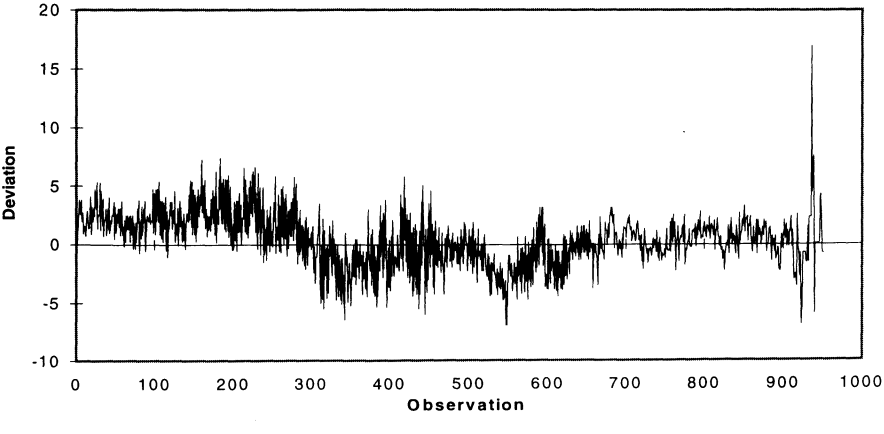


FIG. 15. *Difference between the data and the fitted function (Model D).*

TABLE 9
Selected closed form functions for test problem 4.

Model	Equation	SSE
A	$3.2142x_{10} + 0.2184x_3$	6861.575
B	$-46.8821\sin(22.5976x_3) + 10.4534\cos(6.3422x_5)$	6818.134
C	$3.2526x_{10} + 0.1438x_4 + 0.2221x_3$	6070.081
D	$0.2192x_3 + 3.1044x_{10}$ $+ 0.1705\exp(-4.4522\cos(6.1689x_5))$	5948.197
E	$3.0454\ln(-2.7639x_{11} + 19.0614x_{12}x_9)$ $+ 0.5194(((x_3(7.8126\cos(-4.8424x_3)$ $+ 2.5578x_{10}))/x_5)/(-6.9509\cos(6.0312\ln(5.6902x_4))$ $+ 1.5221x_{12}))((x_1x_{10})/x_5)$	3994.057

6.5. Test problem 5. This example is taken from [26] where a qualitative fuzzy model for an operator's contol of a hemical plant is built. The chemical process has five candidate input variables which may be manually adjusted to control the output, set value for monomer flow rate. The candidate variables are (1) monomer concentration (x_1), (2) change of concentration (x_2), (3) actual monomer flow rate (x_3), (4) temperature 1 inside the plant (x_4) and (5) temperature 2 inside the plant (x_5). The data set includes 70 observations of the above six variables (figure 16). The authors first generated six data clusters by using fuzzy clustering and identified (x_1), (x_2) and (x_3) as the most relevant parameters.

The same data set was also studied by [3]. Their neural network model identified x_3 as the single most significant input parameter followed by x_1. Their model also includes x_5, which has a very marginal effect on the accuracy of the model.

There are some similarities between the GA results and the previous research. All five solutions given in table 10 contain a linear component of x_3, which was identified as the most significant parameter by both [3] and [26]. This shows that there is one to one relationship between x_3 and the output. In the actual system, the output is the set point value for monomer flow rate, and x_3 represents the actual flow rate. Thus, such a relationship between these two parameters could be anticipated. The exponential term with a negative coefficient of three of the functions reflects that the deviation between the output and x_3 is diminishing in later observations.

TABLE 10
Selected closed form functions for test problem 5.

Model	Equation	SSE
A	$0.1877\exp(-0.9109\sin(0.8913x_3)) + 0.9694x_3$	0.6057
B	$-0.2033\exp(-0.8714\cos(3.2885\exp(-3.8097\sin(1.0549x_3)))) + 0.9903x_3$	0.3864
C	$1.8257\cos(2.0012x_5 + 0.2486x_2) - 2.3442x_1 + 0.2013\sin(-0.8969x_3) + 0.8915x_3$	0.3553
D	$0.0505\sin(7.9847x_5\cos(-5.8208x_3 - 2.7400\sin(-3.0262x_3)) + 1.5718x_3)x_3 + 1.046464x_3$	0.3321
E	$0.1318\exp(-1.4083\sin(3.0478\cos(1.6935 - 2.7229x_3) - 1.8731\sin(3.3575 + 3.8175x_5 + 3.0451 - 6.9550x_3) + 0.4712x_3)) + 0.9968x_3$	0.2753

Although there is not any information about the data collection procedure, the nature of the closed form functions suggests that the data comes from a start up process. In figure 16, one of the closed form functions (model E) is plotted with respect to actual data points. As can be seen from this figure, the fitted model represents the actual data points very well.

7. Concluding remarks. For classical nonlinear regression problems the genetic framework gave comparable or better numerical results (i.e., lower SSE values) than models cited in the literature. It should be noted that the final population always contains complex functions with better numerical results than those cited in the literature. The simpler functions in the final population have less accurate results compared to those obtained by alternative techniques such as neural networks. However, unlike neural networks, this approach provides a function which can be used directly for both intuitive analysis and mathematical computation.

One distinguishing feature of this approach is that it gives a set of solutions to the user. The user can study those solutions and choose the model that satisfies both complexity and numerical accuracy constraints. The user can also review the set of good functions for similarity in terms (independent variables and primitive functions) so that the relationship can

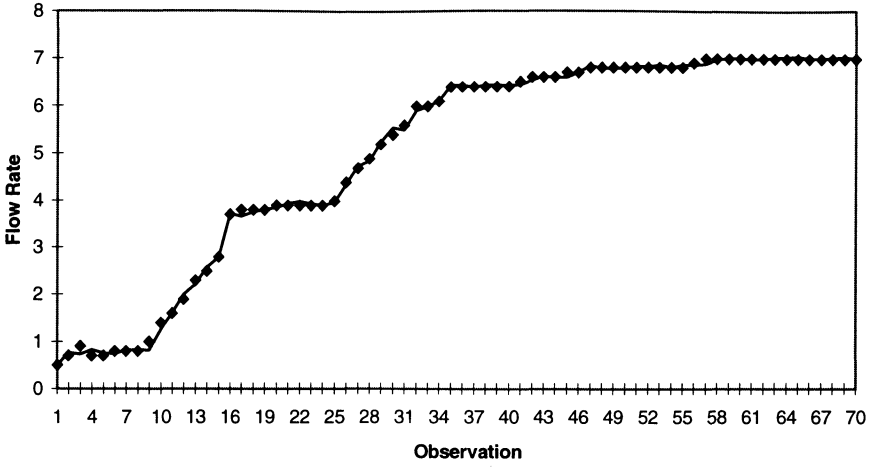


FIG. 16. *Fitted function for chemical plant problem (Model E).*

be better understood. Furthermore, it is very easy for the user to specify different penalty severities, different sets of primitives, and different error metrics to explore the variety of functions, their form and fit with the data. This distinguishes the hierarchical GA framework as an effective global approach for system identification and curve fitting. The only significant drawback is the magnitude of computational effort required. However, since system identification and curve fitting are usually done infrequently and off-line, there is no imperative for extreme computational speed.

8. Acknowledgement. Alice E. Smith gratefully acknowledges the support of the U.S. National Science Foundation CAREER grant DMI 95-02134.

REFERENCES

[1] ANGELINE, PETER J., *Two self-adaptive crossover operators for genetic programming*, Advances in Genetic Programming Volume II (editors: Peter J. Angeline and Kenneth E. Kinnear, Jr.), MIT Press, Cambridge MA, 89–109, 1996.

[2] BÄCK, T. AND SCHWEFEL, H. P., *An overview of evolutionary algorithms for parameter optimization*, Evolutionary Computation, **1**, 1–23, 1993.

[3] BASTIAN, A. AND GASOS, J., *A type I structure identification approach using feed forward neural networks*, IEEE International Conference on Neural Networks, **5** (of 7), 3256–3260, 1994.

[4] BLEASDALE, J. K. A. AND NELDER, J. A., *Plant population andcrop yield*, Nature, 188, 342, 1960.

[5] BOX, G. E. AND JENKINS, G. M., *Time Series Analysis: Forecasting and Control*, Holden-Day, San Francisco, CA, 1970.

[6] COIT, D. W. AND TURNER-JACKSON, B. AND SMITH, A. E., *Static neural network process models: considerations and case studies*, International Journal of Production Research, **36**, in print, 1998.

[7] COLLINS, J. S., *A regression analysis program incorporating heuristic term selection*, Machine Intelligence, Donald Michi, Editor, American Elsevier Company, New York NY, 1968.

[8] CRAMER, N. L., *A representation for the adaptive generation of simple sequential programs*, Proceedings of an International Conference on Genetic Algorithms and Their Applications, 183–187, 1985.

[9] DALLEMAND, J. E., *Steepwise regression program on the IBM 70*, Technical Report of General Motors Research Laboratories, 1958.

[10] FARAZDAGHI, H. AND HARRIS, P. M., *Plant competition and crop yield*, Nature, **271**, 289–290, 1968.

[11] FRIEDMAN, J., *Multivariate adaptive regression splines*, Department of Statistics Technical Report 102, Stanford University, 1988 (revised 1990).

[12] FUJIKO, C. AND DICKINSON, J., *Using the genetic algorithm to generate LISP source code to solve prisoner's dilemma*, Genetic Algorithm and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, 236–240, 1987.

[13] GALLANT, A. R., *Nonlinear Statistical Models*, John Wiley and Sons, New York, 1987.

[14] GEMAN S. AND BIENENSTOCK, E., *Neural networks and the bias/variance dilemma*, Computation, **4**, 1–58, 1992.

[15] GULSEN, M., SMITH, A. E. AND TATE, D. M., *Genetic algorithm approach to curve fitting*, International Journal of Production Research, **33**, 1911–1923, 1995.

[16] HOLLIDAY, R., *Plant population and crop yield*, Field Crop Abstracts, **13**, 159–167, 247–254, 1960.

[17] KARR, C. L., STANLEY D. A., AND SCHEINER B. J., *Genetic algorithm applied to least squares curve fitting*, U.S. Bureau of Mines Report of Investigations 9339, 1991.

[18] KOZA, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge MA, 162–169, 1992.

[19] KOZA, J. R., *Genetic Programming: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge MA, 1994.

[20] MÜHLENBEIN, H. AND SCHLIERKAMP-VOOSEN, D., *Predictive models for the breeder genetic algorithm. I. Continuous parameter optimization*, Evolutionary Computation, **1**, 25–49, 1993.

[21] NASH, J. C. AND WALKER-SMITH, M., *Nonlinear Parameter Estimation: An Integrated System in BASIC*, Marcel Dekker, New York, 1987.

[22] PARK, Y. R., MURRAY, T. J. AND CHEN, C., *Predicting sun spots using a layered perceptron neural network*, IEEE Transactions on Neural Networks, **7**, 501–505, 1996.

[23] ROGERS, D., *G/SPLINES: A hybrid of Friedman's multivariate adaptive regression splines (MARS) algorithms with Holland's genetic algorithm*, Proceedings of the Fourth International Conference on Genetic Algorithms, 384–391, 1991.

[24] SEBER, G. A. F. AND WILD, C. J., *Nonlinear Regression*, John Wiley and Sons, New York, 1989.

[25] STIGLER, S. M., *History of Statistics: The Measurement of Uncertainty Before 1900*, Harvard University Press, Cambridge MA, 1986.

[26] SUGENO, M., AND YASUKAWA, T., *A fuzzy-logic-based approach to qualitative modeling*, IEEE Transactions on Fuzzy Systems, **1**, 7–31, 1993.

[27] VILLAREAL, J. AND BAFFES, P., *Sunspot prediction using neural networks*, University of Alabama Neural Network Workshop Report, 1990.

[28] WESTERVELT, F., *Automatic System Simulation Programming*, unpublished Ph.D. Dissertation, University of Michigan, 1960.

EXPERIENCES WITH THE PGAPack PARALLEL GENETIC ALGORITHM LIBRARY

DAVID LEVINE*, PHILIP HALLSTROM†, DAVID NOELLE‡, AND BRIAN WALENZ§

Abstract. PGAPack is the first widely-distributed parallel genetic algorithm library. Since its release, several thousand copies have been distributed worldwide to interested users. In this paper we discuss the key components of the PGAPack design philosophy and present a number of application examples that use PGAPack.

1. Introduction. PGAPack is the first widely-distributed parallel genetic algorithm library. Since its release, several thousand copies have been distributed worldwide to interested users. Key features in PGAPack are support for multiple data types, parallel portability across uniprocessors, multiprocessors, and workstation networks, Fortran and C interfaces, a simple interface for novice and application users, multiple levels of access for expert users, object-oriented design, extensibility, and multiple GA operators and parameter choices.

PGAPack supports parallel and sequential implementations of the single population model. The population may be updated using either generational or steady-state replacement schemes, or any of their parameterized variants. The supported crossover operators are one-point, two-point, and uniform crossover. Selection may be done using proportional, stochastic universal, binary tournament, or probabilistic binary tournament selection. Different mutation operators are used with each different data type. A restart operator is available that reseeds a population using random variants of the best string.

Options allow the user to specify the population size, stopping criteria, whether duplicate strings should be allowed in the population, and whether to mutate *or* crossover strings, or to mutate *and* crossover strings. In all cases, defaults are provided if the user does not explicitly specify a choice. PGAPack system calls provide random number generation, output control, error reporting, and debugging capabilities.

In simplest form, a parallel (or sequential) PGAPack program can be written using only four PGAPack functions and a string evaluation function. Figure 1 shows such a minimal program and evaluation function for the Maxbit problem. `PGACreate` initializes PGAPack and returns the address of the context variable (see Section 2). The parameters to `PGACreate`

*Current address: Boeing Information & Support Services, PO Box 3707, MS 7L-20, Seattle, WA 98124-2207.

†Current address: Dunn Systems, Inc., Lincolnwood, Illinois.

‡Current address: Computer Science Department, UCLA, Los Angeles CA, 90024.

§Current address: Sandia National Laboratories, Org. 9223, MS 1110, Albuquerque, NM 87185-1110.

```
#include "pgapack.h"
double evaluate (PGAContext *ctx, int p, int pop);
int main(int argc, char **argv)
{
 PGAContext *ctx;
 ctx = PGACreate (&argc, argv, PGA_DATATYPE_BINARY, 100,
        PGA_MAXIMIZE                                        );
 PGASetUp        (ctx                                       );
 PGARun          (ctx, evaluate                             );
 PGADestroy      (ctx                                       );
 return;
}
double evaluate (PGAContext *ctx, int p, int pop)
{
 int i, nbits=0, stringlen;
 stringlen = PGAGetStringLength(ctx);
 for (i=0; i<stringlen; i++)
     if (PGAGetBinaryAllele(ctx, p, pop, i))
         nbits++;
 return((double) nbits);
}
```

FIG. 1. *PGAPack C Program for the Maxbit Example*

are the program arguments, the data type (`PGA_DATATYPE_BINARY`), the string length (100), and the direction of optimization (`PGA_MAXIMIZE`). `PGASetUp` initializes all parameters and function pointers that have not been explicitly set (none in this example) to default values. `PGARun` executes the genetic algorithm. Its second argument is the name of a user-defined function (`evaluate`) that will be called whenever a string evaluation is required. `PGADestroy` releases all memory allocated by PGAPack.

The evaluation function (`evaluate`) must be written by the user. `PGAGetStringLength` returns the string length. `PGAGetBinaryAllele` returns the value of the `i`th bit of string `p` in population `pop`. The values returned by this function, sometimes called "raw fitness," are automatically mapped into nonnegative values according to whether any of the raw fitness values are negative, the direction of optimization, and the type of fitness function (identity, ranking, or linear normalization) used.

PGAPack provides functions to encode and decode real and integer values in a binary string. The string representation may be either binary or Gray coded. This capability allows the use of existing real- and integer-valued functions with no modification required to the function source. For example, suppose the user has a real-valued function f of three real variables x_1, x_2, and x_3, each constrained on the interval $[-10, 10]$, and wishes to use 10 bits for each and a Gray code encoding. This may be done as show

```
#include "pgapack.h"
double grayfunc (PGAContext *ctx, int p, int pop);
double f        (double x1, double x2, double x3);
 :   :
double grayfunc (PGAContext *ctx, int p, int pop)
{
double x1, x2, x3, v;
x1 = PGAGetRealFromGrayCode (ctx, p, pop,  0,  9, -10., 10.);
x2 = PGAGetRealFromGrayCode (ctx, p, pop, 10, 19, -10., 10.);
x3 = PGAGetRealFromGrayCode (ctx, p, pop, 20, 29, -10., 10.);
v  = f(x1,x2,x3);
return(v);
}
```

FIG. 2. *Using Legacy Code for Function Evaluation*

in Figure 2. Note that the function f *need not be modified.* The function `grayfunc` is used as a "wrapper" to decode the real values from the Gray coded string, pass them as real values to f, and return the corresponding function value.

2. Design and implementation. PGAPack supports four *native* data types: binary-valued, integer-valued, real-valued, and character-valued strings. A data-hiding capability provides the full functionality of the library to the user, in a transparent manner, irrespective of the data type used. The *context variable* is the data structure that provides the data hiding capability. The context variable is a pointer to a C language structure, which is itself a collection of other structures. These (sub)structures contain all the information necessary to run the genetic algorithm, including the data type to use, parameter values, which functions to call, operating system parameters, debugging flags, initialization choices, and internal scratch arrays.

All the functionality of PGAPack is provided through function calls. Typically, users call high-level, data-type-*neutral* functions which themselves call data-type-*specific* functions that correspond to the data type used. The data-type-specific functions use addresses and offsets of the population data structures. The user-level routines, however, provide the abstraction of data type neutrality and an integer indexing scheme and can be called independent of the data type.

PGAPack maintains two populations: an *old* one and a *new* one. Formally, string p in population *pop* is referred to by the 2-tuple `(p,pop)` and the value of gene i in that string by the 3-tuple `(p,pop,i)`. To aid the user abstractions, two symbolic constants, `PGA_OLDPOP` and `PGA_NEWPOP`, always refer to the last generation and new generation, respectively.

PGAPack provides multiple levels of control to support the require-

ments of different users. The simplest level, previously shown in Figure 1, encapsulates the genetic algorithm "machinery" within the single function `PGARun`. The user need specify only three parameters: the data type, the string length, and the direction of optimization. All other parameters have default values.

At the next level, the user calls data-type-neutral functions explicitly (e.g., `PGASelect`, `PGACrossover`, `PGAMutation`). This mode is useful when the user wishes more explicit control over the steps of the genetic algorithm, or wishes to hybridize the genetic algorithm with a hill-climbing heuristic.

At the third level, the user can customize the genetic algorithm by supplying function(s) to customize a particular operator(s) while still using one of the native data types. Finally, at the lowest-level of usage, the user can define a *new* data type, write the data-type-specific low-level GA functions (i.e., crossover, mutation, etc.), and have these functions executed by the high-level data-type-neutral functions.

PGAPack is written in ANSI C. A set of interface functions allows user-level PGAPack functions to be called from Fortran. Message-passing calls follow the MPI (see Section 3) standard. Nonoperative versions of the MPI functions are supplied if the user does not have an MPI implementation for their machine. These allow the PGAPack library to be built (for sequential use) in the absence of an MPI implementation.

3. Parallel Computing. The first release of PGAPack was primarily targeted at application developers and supports a parallel (and sequential) implementation of the single population model. This initial choice was made because in most real applications, the dominant computational cost is executing function evaluations. Being able to execute these in parallel should lead to a significant reduction in the elapsed computing time.

The single population model may be parallelized by executing in parallel the loop iterations that create generation $t + 1$ from generation t. Most steps in this loop (crossover, mutation, evaluation) can be executed in parallel. The execution efficiency, however, depends upon the computer architecture and parallel execution overhead, the number of new population members created each generation (the degree of parallelism), and the computational cost of the steps being executed in parallel (the granularity).

The parallel implementation in PGAPack uses a master/slave algorithm in which one process, the *master*, executes all steps of the genetic algorithm except the function evaluations, which are executed by *slave* processes. A master/slave implementation is shown in Figure 3.

We chose a master/slave algorithm for two reasons. First, since function evaluation time is the dominant cost in most GA runs, the performance benefits from parallel execution may be achieved by parallelizing only this step. Second, since we use a message-passing programming model to implement the master/slave algorithm, there may be a significant parallel

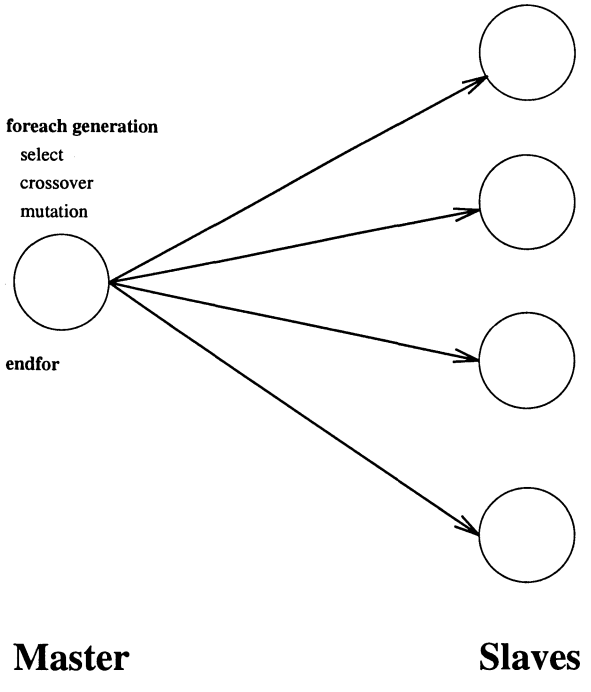


FIG. 3. *PGAPack Master-Slave model*

execution overhead. Focusing only on parallelizing the function evaluations allows for modest data distribution requirements (just the strings to be evaluated) and minimal synchronization requirements.

PGAPack is implemented using the message-passing interface (MPI) standard [5]. MPI is a *specification* of a message-passing library for parallel computers and workstation networks—it defines a set of functions and their behavior. Implementations of MPI exist for both sequential (uniprocessors) and parallel (multiprocessors, multicomputers, and workstation networks) computer hardware, thereby allowing PGAPack to run on all these machines without any code changes. MPI offers a number of useful features that were used in PGAPack including collective communication operations, routines to configure the logical topology of the processors, barriers, a unique message namespace for library messages, and the ability to send and receive arbitrary structures.

The parallel implementation will produce the *same* result as the sequential implementation, usually faster. The choice of sequential or parallel execution depends on the number of processes specified when the program is started. If one process is specified, a sequential implementation is used. If two processes are used, both the master process and the slave process will compute the function evaluations. If more than two processes are used, the master executes all GA steps except the function evaluations, and the

slaves execute the function evaluations.

There are two primary considerations in determining the performance advantage of using the master/slave model. First, the speedup will vary according to the amount of computation associated with a function evaluation and the computational overhead of distributing and collecting information to and from the slave processes. Second, the number of function evaluations that can be executed in parallel will limit the speedup. This number depends on the population size and the number of new strings created each generation. In a generational replacement model, the entire population may be evaluated in parallel. In the more popular steady-state model, however, typically only one or two new strings are produced, and the degree of parallelism is minimal. By default, PGAPack replaces 10% of the population. In our experience this percentage usually provides an acceptable degree of parallelism, while retaining the superior performance characteristics of the steady-state model.

4. Application experiences. In this section we present examples of several projects done using PGAPack. Our focus is on parallel execution and custom extensions to PGAPack.

4.1. Molecular docking. STALK [7] is a system for molecular docking that uses PGAPack. The goal in molecular docking is to predict the conformation (location and orientation) of a ligand (a small molecule) in a protein active site (the part of the protein that the ligand binds to).

Molecular docking may be formulated as a nonlinear optimization problem, where the goal is to maximize the intermolecular interaction energy. Typically, the solution space has a large number of possible conformations and many local minima. The energy function is highly nonlinear and computationally expensive to evaluate; being able to compute these in parallel is highly desirable for solving realistic problems.

STALK uses a rigid backbone model. The protein is fixed in space, and the position of the ligand determined by the GA. The six degrees of freedom of the ligand are translation and rotation about the protein's center of mass.

Table 1 contains performance results for a test problem with approximately 300 ligand atoms and 1500 protein atoms. The *Compute Proc.* column is the number of IBM SP processors that execute function evaluations. The *Time* column is the average over six runs of the total time spent by the master process (executing the GA, packing and sending data to the slave processes, and waiting for results). The *Speedup* column is the ratio of the time to execute the one-processor case to the time to execute with that number of processors. The speedup achieved is fairly constant, although not ideal. Several solutions with better energy values than the x-ray crystallographic solution were found.

A novel feature of STALK is the use of virtual-reality (VR) technology to provide an interactive computational steering capability. In the CAVE

TABLE 1
Solution Time vs. No. of Processors

Compute Proc.	Time (sec.)	Speedup
1	263581	1.0
2	148666	1.8
3	87208	3.0
6	46950	5.6
13	22150	11.9
25	12831	20.5
50	7193	36.6
100	4181	63.0

[4], a room-sized VR environment that communicates with the IBM SP through an SGI Onyx workstation front-end, the best conformation in the population and associated energy are displayed. Using a pointing device in the CAVE, the user may translate and/or rotate the ligand. The updated ligand coordinates are sent to the IBM SP running the GA which uses the GA's evaluation function to calculate and return the corresponding intermolecular energy. The user then has the option of using the "hand-docked" solution to replace the worst conformation in the population, to reseed the entire population using random perturbations of the hand-docked solution, or to make no changes at all.

4.2. Quantum chemistry parameter optimization. In [3] PGAPack was used to study chemical reactions in the condensed phase. The goal was to develop a quantum mechanical (QM) simulation code that could determine accurate values for certain molecular properties (e.g., the heat of formation, dipole moments, atomic bond lengths, dihedral angles).

To calibrate the parameters of the QM simulation code, a GA was used to find values for these parameters that resulted in calculated molecular properties that were in close agreement with the experimentally determined results. Formally, if $Y_p(exp)$ is the value for property p known from the experimentally determined results and $Y_p(calc)$ the value for property p to be calculated, than the evaluation function is to minimize the sum of weighted errors given by

$$\sum_M \sum_p |Y_p(calc) - Y_p(exp)| \, w_p$$

where w_p is a weighting factor.

As a test case, the energetics of a proton transfer reaction in gas-phase and aqueous solution were studied. The basis set of molecules consisted of methanol, imidazole, methoxide, and imidazolium. A real-valued GA was run for 15,000 steady-state iterations. Parallelism was applied *within* each

individual function evaluation by determining $Y_p(calc)$ for each molecule independently. Message-passing was used to distribute and collect the function evaluation components. Using the parameter values determined by the GA, more accurate values for the molecular properties of the proton transfer reaction were calculated with the QM simulation code than had been previously determined by other methods.

4.3. Timber harvest scheduling. The goal in adjacency constrained timber harvest scheduling (ACTHS) is to find near optimal tree harvesting schedules, subject to constraints on the clear-cut opening size and the level of timber harvests from schedule period to period. ACTHS is a difficult combinatorial optimization problem; a problem with N stands and M planning periods has M^N integer solutions (including infeasible solutions). A typical operational scheduling problem has 10–20 periods, and 500–1000 stands.

In [8] the authors compare a traditional GA, that represents solutions directly on the chromosomes, and an order-based GA, that represents permutations of the stand identification numbers on the chromosomes and uses each permutation as an order list for scheduling stands, with Monte Carlo integer programming (MCIP). PGAPack was used to implement both the traditional and order-based GAs. For the order-based GA, custom PGAPack operators were developed for order-based crossover, position-based crossover, partially matched crossover, order-based mutation, position-based mutation, and scramble mutation.

Using test data that ranged from 42 to 849 stands and 10 to 15 time periods, the results showed the order-based GA was superior to the other methods, averaging 2.2% better than the traditional GA, and 3.5% better than the MCIP. This project was so successful, that the order-based GA is now the in house production planning system for ACTHS at Rayonier Corp.

4.4. Vehicle clustering. In [9] the authors describe the application of PGAPack to the Multiple-Depot Vehicle Routing Problem (MDVRP). The MDVRP is an extension of the vehicle routing problem, with the customers being served from multiple depots instead of a single, central depot. The MDVRP is solved using a genetic clustering method that clusters customers using route primitives.

The genetic clustering is done using PGAPack. Once the clusters for the customers are obtained, the clusters are improved using a branch-exchange procedure. The final solution obtained by the branch exchange procedure serves as the fitness value for the string. Due to the computationally expensive nature of the branch exchange procedure, A hash function was written for PGAPack that prevents previously evaluated strings from being evaluated again. The addition of the hash function reduced the processing time approximately 30%.

4.5. Evolutionary robotics. The work in [2] describes the use of genetic algorithms to design neural network controllers for a simulated, box-pushing robot. The world of the robot is a small grid with blocks randomly placed on it. The goal is to have the robot push blocks into the corners, placing blocks near the corner blocks when the corners are filled.

The evaluation function for this problem is very expensive. Each robot must be evaluated in more than one initial configuration, and each configuration requires several hundred time steps. A parallel version of PGAPack was installed on an SGI workstation network and used to corroborate previous results.

In addition, a version of niching based on the work in [6] was added to PGAPack to support multi-objective optimization. A custom function was written that performed selection by choosing two strings randomly from the population and returning the one which dominates a random sample of the population. If neither string dominates, the string with the fewest neighbors is returned.

4.6. Finite element mesh optimization. In finite-element analysis, structures are modeled as meshes of elements and nodes which match the geometry, rigidity, and loading of each structure. While a very fine uniform mesh will give accurate results, it usually leads to unacceptably high computational loads. Therefore, a nonuniform mesh with higher resolution in parts of the structure where the stress gradients are high, and lower resolution where the stress gradients are low is desired.

In the work in [1], a GA is used to search for an optimal mesh. Initially, a uniform mesh is imposed on a loaded structure with a small number of degrees of freedom that are not computationally burdensome. An energy-based error norm is calculated and used as an objective function to be minimized. In the main loop, randomly perturbed node positions are generated, the finite-element mesh is regenerated using the new node positions, and the objective function is recalculated. The mesh regeneration is stopped when the objective function has reached a stationary (assumed to be minimum) value.

This approach has been implemented on simple beams and plates using PGAPack on a cluster of eight Sun workstations. The results appear promising for the simple cases tried. A real-coded GA is used that has a specialized function to regenerate elements around the perturbed positions of the nodes. This function eliminates all meshes that result in malformed (e.g., high aspect ratio, concave or physically impossible) elements before mutation and crossover are attempted.

5. Conclusions and future work. Judging by the number of users, and the applications they have implemented, the release of PGAPack has met with widespread acceptance. We attribute this success to four key factors. First is ease-of-use. Many users use PGAPack as a black-box; parameter and operator choices have robust default values and their details

are encapsulated in a few simple function calls. The object-oriented interface, coupled with the user-level abstractions, allows the user to concentrate on functionality and not data structure details.

The second important attribute is portability. PGAPack has successfully installed on most workstations, workstation clusters, and parallel computers. The ANSI C language PGAPack is written in is standardized and fully portable. A small set of link options provides a compatible Fortran interface. From a parallel computing perspective, the message passing programming model maps easily onto both shared- and distributed-memory hardware. Additionally, MPI is now a fully accepted message-passing standard, with versions available for all current workstations and parallel computers.

The third reason for PGAPack's success is the task parallel master/slave model. All real applications we have worked on, and most others we are aware of, are dominated by the computational cost of the function evaluations. The performance improvement from executing these evaluations simultaneously can provide significantly improved turnaround. In some cases the improved performance is critical to even being able to apply GAs.

Finally, the fourth reason for PGAPack's success is extensibility. Although PGAPack supports multiple data types and their common operators, problem-specific functionality is sometimes needed. PGAPack provides a simple means to replace any operator with a user function. Also, users may define a new data type by writing the low-level data-structure-specific functions, but still take advantage of the high-level data-structure-neutral functions in PGAPack. Finally, by passing the context variable as a parameter to a user function, the user has complete access to solution and parameter values, and may develop any custom functionality desired.

Prototype implementations of several new features have been developed for future incorporation into PGAPack. These include an island model GA, a genetic programming implementation, and a meta-GA for optimizing parameter choices. In addition, we plan to incorporate the additional functionality that has been developed and contributed by our users.

PGAPack is freely available and may be obtained by anonymous ftp from `info.mcs.anl.gov` in file `pub/pgapack/pgapack.tar.Z`, or via the World Wide Web at the following URL.

`http://www.mcs.anl.gov/home/levine/PGAPACK/index.html`

Acknowledgments. We thank Ara Arabyan, Karthik Balakrishnan, Ralph Butler, David Mullen, and Sam Thangiah for discussions of their PGAPack applications, Greg Reeder for writing prototype MPI communication routines, and Bill Gropp, Lois Curfman McInnes, and Barry Smith for many helpful discussions about their PETSc library. This work was supported by the Mathematical, Information, and Computational Sciences

Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

REFERENCES

[1] A. ARABYAN, S. CHEMISHKIAN, AND A. TONOYAN, *Finite-element mesh optimization using genetic algortihms*, Univeristy of Arizona, 1997.

[2] K. BALAKRISHNAN AND V. HONAVAR, *Analysis of neurocontrollers designed by simulated evolution*, In Proceedings of IEEE International Conference on Neural Networks ICNN '96, 1996.

[3] P. BASH, L. HO, A. MACKERELL, P. HALLSTROM, AND D. LEVINE, *Progress toward chemical accuracy in the computer simulation of condensed phase reactions*, Proceedings of the National Academy of Sciences, 93:3698–3703, 1996.

[4] C. CRUZ-NEIRA, D. SANDIN, AND T. DEFANTI, *Surround-screen projection-based virtual reality: The design and implementation of the CAVE*, In ACM SIGGRAPH '93 Proceedings, pages 135–142, Lawrence Erlbaum Associates, 1993.

[5] W. GROPP, E. LUSK, AND A. SKJELLUM, *USING MPI Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, Cambridge, 1994.

[6] J. HORN AND N. NAFPLIOTIS, *Multiobjective optimization using the niched pareto genetic algorithm*, In Proceedings of the First IEEE Conference on Evolutionary Computation, **volume 1**, Piscataway, 1994, Lawrence Erlbaum Associates.

[7] D. LEVINE, M. FACELLO, P. HALLSTROM, G. REEDER, B. WALENZ, AND F. STEVENS, *STALK: An Interactive Virtual Molecular Docking System*, 1996, Accepted for publication in IEEE Computational Science & Engineering.

[8] D. MULLEN, *A comparison of genetic algorithms and monte carlo integer programming for optimization of adjacency constrained timber harvest scheduling problems*, Master's thesis, University of North Florida, 1996.

[9] S. SALHI, S. THANGIAH, AND F. RAHMAN, *A genetic clustering method for the multi-depot vehicle routing problem*, In Proceedings of the Third International Conference on Neural Networks and Genetic Algorithms, 1997.

THE SIGNIFICANCE OF THE EVALUATION FUNCTION IN EVOLUTIONARY ALGORITHMS

ZBIGNIEW MICHALEWICZ*

Abstract. The major component of any evolutionary algorithm is its evaluation function, which serves as a major link between the algorithm and the problem being solved. The evaluation function is used to distinguish between better and worse individuals in the population, hence it provides an important feedback for the search process. In this paper we survey a few typical methods for constructing an evaluation function for constrained optimization problems.

Key words. Constrained optimization, evolutionary algorithms, evaluation function, infeasible individuals.

1. Introduction. It is generally accepted that any evolutionary algorithm to solve a problem must have five basic components (Davis, 1987):

- a genetic representation of solutions to the problem,
- a way to create an initial population of solutions,
- an evaluation function (i.e., the environment), rating solutions in terms of their 'fitness',
- 'genetic' operators that alter the genetic composition of children during reproduction, and
- values for the parameters (population size, probabilities of applying genetic operators, etc.)

Most successful implementations of an evolutionary technique for a particular real-world problem require some additional heuristics (problem-specific knowledge) which are incorporated into the basic components listed above. These heuristics apply to genetic representation of solutions, to 'genetic' operators that alter their composition, to values of various parameters, and to methods for creating an initial population. The evaluation function (as the only item out of the above list of five basic components of evolutionary algorithm) usually is taken "for granted"; most researchers did not consider any modifications of evaluation function, since it is often implicitly defined by the problem. However, for most real-world problems (e.g., constrained problems), the process of selection of an evaluation function might be quite complex by itself, especially when we deal with feasible and infeasible solutions to the problem; several heuristics usually are incorporated in this process. In this paper we examine some of these heuristics and discuss their merits and drawbacks.

The paper is organized as follows. Section 2 states the problem by defining feasible and infeasible individuals and Section 3 provides a dis-

*Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA, and Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland.

cussion on evaluation methods for evolutionary techniques. Section 4 concludes the paper.

2. Feasible and infeasible solutions. The evaluation function serves as the major link between the problem and the evolutionary algorithm. The evaluation function rates individuals in the population: better individuals have better chances for survival and reproduction. Hence it is essential to define an evaluation function which characterizes the problem in a "perfect way". For constrained optimization problems, the issue of handling feasible and infeasible individuals should be addressed very carefully: very often a population contains infeasible individuals but a *feasible* optimal is required. Finding proper evaluation measures for feasible and infeasible individuals is of great importance; it directly influences the outcome (success or failure) of the algorithm.

The issue of processing infeasible individuals is very important for solving constrained optimization problems using evolutionary techniques. For example, in continuous domains, the general nonlinear programming problem[1] is to find $\overline{X}$ so as to

$$optimize\ f(\overline{X}), \overline{X} = (x_1, \ldots, x_n) \in R^n,$$

where $\overline{X} \in \mathcal{F} \subseteq \mathcal{S}$. The set $\mathcal{S} \subseteq R^n$ defines the search space and the set $\mathcal{F} \subseteq \mathcal{S}$ defines a *feasible* search space. Usually, the search space $\mathcal{S}$ is defined as a n-dimensional rectangle in R^n (domains of variables defined by their lower and upper bounds):

$$l(i) \leq x_i \leq u(i), \quad 1 \leq i \leq n,$$

whereas the feasible set $\mathcal{F}$ is defined by an intersection of $\mathcal{S}$ and a set of additional $m \geq 0$ constraints:

$$g_j(\overline{X}) \leq 0, \ for\ j = 1, \ldots, q, \ \ and \ \ h_j(\overline{X}) = 0, \ for\ j = q+1, \ldots, m.$$

Most research on applications of evolutionary computation techniques to nonlinear programming problems was concerned with complex objective functions with $\mathcal{F} = \mathcal{S}$. Several test functions used by various researchers during the last 20 years consider only domains of n variables; this was the case with five test functions F1–F5 proposed by De Jong (1975), as well as with many other test cases proposed since then.

In discrete domains the problem of constraints was acknowledged much earlier. The knapsack problem, the set covering problem, and all types of scheduling and timetabling problems are constrained. Several heuristic methods emerged to handle constraints; however, these methods have not been studied in a systematic way.

In general, a search space $\mathcal{S}$ consists of two disjoint subsets of feasible and infeasible subspaces, $\mathcal{F}$ and $\mathcal{U}$, respectively (see Figure 1). We do not

[1] We consider here only continuous variables.

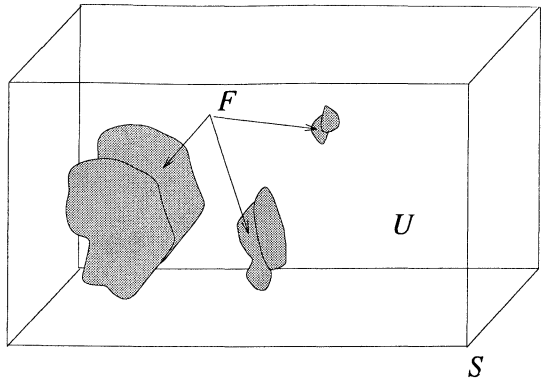


FIG. 1. *A search space and its feasible part*

make any assumptions about these subspaces; in particular, they need not be convex and they need not be connected (e.g., as is the case in the example in Figure 1 where the feasible part $\mathcal{F}$ of the search space consists of three disjoined subsets). The fundamental question (which must be addressed by any practitioner) is

> Should we consider infeasible individuals harmful and eliminate them from the population?

The answer 'yes' implies a "death penalty" heuristic, which is a popular option in many evolutionary techniques. Note that rejection of infeasible individuals offers a significant simplification of the algorithm: there is no further need to consider infeasible individuals! There is no need to evaluate them, penalize them, repair them; there is no need to compare them with feasible ones.

The method of eliminating infeasible solutions from a population may work reasonably well when the feasible search space is convex and it constitutes a reasonable part of the whole search space (e.g., evolution strategies do not allow equality constraints since with such constraints the ratio between the sizes of feasible and infeasible search spaces is zero). Otherwise such an approach has serious limitations. For example, for many search problems where the initial population consists of infeasible individuals only, it might be essential to improve them (as opposed to rejecting them). Moreover, quite often the system can reach the optimum solution easier if it is possible to "cross" an infeasible region (especially in non-convex feasible search spaces). On the top of that, for many constrained problems, the optimum solution lies on the boundary of feasible and infeasible parts of the search space (i.e., the boundary between $\mathcal{F}$ and $\mathcal{U}$); because of that infeasible individuals allow the system to approach the optimum from various directions (as opposed to the case, when a "narrow" feasible corridor leads to the optimum solution). For all these reasons, in most real-world

applications, it is essential to process infeasible individuals, so the answer for the question posed in the previous paragraph is simply 'no'!

Consequently, during the search process we have to deal with various feasible and infeasible individuals. The presence of feasible and infeasible individuals in the population influences other parts of the evolutionary algorithm; for example, should the elitist selection method consider a possibility of preserving the best feasible individual, or just the best individual overall? Further, some operators might be applicable to feasible individuals only. However, the major aspect of such a scenario is the need for evaluation of feasible and infeasible individuals. The problem of how to evaluate individuals in the population is far from trivial. In general, we have to design two evaluation functions, $eval_f$ and $eval_i$, for feasible and infeasible domains, respectively.

Several trends for handling infeasible solutions have emerged in the area of evolutionary computation. We discuss them in the following section using examples from discrete and continuous domains.

3. Heuristics for evaluating individuals. In this section we discuss several methods for handling feasible and infeasible solutions in a population; most of these methods emerged quite recently. Only a few years ago Richardson et al. (1989) claimed: "Attempts to apply GA's with constrained optimization problems follow two different paradigms (1) modification of the genetic operators; and (2) penalizing strings which fail to satisfy all the constraints." This is no longer the case as a variety of heuristics have been proposed. Even the category of penalty functions consists of several methods which differ in many important details on how the penalty function is designed and applied to infeasible solutions. Other methods maintain the feasibility of the individuals in the population by means of specialized operators or decoders, impose a restriction that any feasible solution is 'better' than any infeasible solution, consider constraints one at the time in a particular linear order, repair infeasible solutions, use multiobjective optimization techniques, are based on cultural algorithms, or rate solutions using a particular co-evolutionary model.

Before we discuss several constraint-handling techniques, it is worthwhile to point out that (for some problems) even the construction of the evaluation function $eval_f$ for feasible solutions might be far from trivial. For example, for many design problems there are no clear formulae for comparing two feasible designs. Moreover, some problems require optimization of several (possibly conflicting) goals (multi-objective optimization cases), and often problem-dependent heuristics are necessary in such cases to provide with a numerical measure $eval_f(x)$ of a feasible individual x.

One example to illustrate the problem of evaluating feasible individuals is the satisfiability (SAT) problem. For a given conjunctive normal form formula, say

$$F(x) = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee x_3),$$

it is hard to compare two feasible individuals $p = (0, 0, 0)$ and $q = (1, 0, 0)$ (in both cases $F(p) = F(q) = 0$). De Jong and Spears (1989) examined a few options. For example, it is possible to define $eval_i$ to be a ratio of the number of conjuncts which evaluate to true; in that case

$$eval_f(p) = 0.666 \quad and \quad eval_f(q) = 0.333.$$

It is also possible (Pardalos 1994) to change the Boolean variables x_i into floating point numbers y_i and to assign, for example,

$$eval'_f(y) = (y_1-1)^2(y_2+1)^2(y_3-1)^2+(y_1+1)^2(y_3+1)^2+(y_2-1)^2(y_3-1)^2.$$

In the above case the solution to the SAT problem corresponds to a set of global minimum points of the objective function: the *true* value of $F(x)$ is equivalent to the global minimum value 0 of $eval_i(y)$.

It may seem, though, that for most optimization problems the evaluation function $eval_f$ for feasible solutions is given. This is the case for numerical optimization problems and for most operation research problems (knapsack problems, traveling salesman problems, set covering problems, etc.) However, it need not be always the case. For example, Falkenauer (1994) rejected the idea of constructing a straightforward $eval_f$ for the bin packing problem (BBP):

> "Let's us define a suitable cost function for the BPP. The objective being to find the minimum number of bins required, the first cost function that comes to mind is simply the number of bins used to 'pack' all the objects. This is correct from a strictly mathematical point of view, but it is unusable in practice. Indeed, such a cost function leads to an extremely unfriendly landscape of the search space: a very small number of optimal points in the space are lost in an exponential number of points where this purported cost function is just one unit above the optimum. Worse, those slightly suboptimal points yield the same cost. The trouble is that such a cost function lacks any capacity of guiding an algorithm in the search, making the problem a 'needle in a haystack'.
> We thus settled for the following cost function for the BPP [...]: maximize
>
> $$f_{BPP} = \frac{\sum_{i=1}^{N}(F_i/C)^k}{N},$$
>
> with N being the number of bins actually used in the solution being evaluated, F_i the sum of sizes of the objects in (the fill of) the bin i, C is the bin capacity, k a constant, $k > 1$.

> The constant k expresses our concentration on the 'extremist' bins in comparison to the less filled ones. The larger k is, the more we prefer well-filled 'elite' groups as opposed to a collection of about equally filled bins. In fact, the value of k gives us the possibility to vary the 'ruggedness' of the function to optimize, from the 'needle in a haystack' ($k = 1$, $f_{BPP} = 1/N$) up to the 'best-filled bin' ($k \to \infty$, $f_{BPP} \to \max_i[(F_i/C)^k]$)."

Clearly, the problem of selecting a "perfect" $eval_f$ is far from trivial.

The main three approaches for constructing evaluation function $eval_i$ for infeasible individuals are based on

- penalty functions,
- repair algorithms, and
- special data structures and operators.

We discuss these in turn in the following subsections.

3.1. Penalty functions. It is possible to extend the domain of function $eval_f$ to handle infeasible individuals in the following way:

$$eval_i(x) = eval_f(x) \pm Q(x),$$

where $Q(x)$ represents a penalty for infeasible individual x. The major question is, how should such a penalty function $Q(x)$ be designed? The intuition is simple: the penalty should be kept as low as possible, just above the limit below which infeasible solutions are optimal (the so-called *minimal penalty rule*, see Le Riche et al. 1995). However, it is difficult to implement this rule effectively.

The relationship between infeasible individual 'x' and the feasible part $\mathcal{F}$ of the search space $\mathcal{S}$ plays a significant role in penalizing such individuals: an individual might be penalized just for being infeasible, the 'amount' of its infeasibility is measured to determine the penalty value, or the effort of 'repairing' the individual might be taken into account. However, in such cases a penalty function should consider the "easiness of repairing" an individual as well as the quality of its repaired version; designing such penalty functions is problem-dependent and, in general, quite hard.

Several researchers studied heuristics on design of penalty functions. Some hypotheses were formulated (Richardson et al. 1989):

- "penalties which are functions of the distance from feasibility are better performers than those which are merely functions of the number of violated constraints,
- for a problem having few constraints, and few full solutions, penalties which are solely functions of the number of violated constraints are not likely to find solutions,
- good penalty functions can be constructed from two quantities, the *maximum completion cost* and the *expected completion cost*,

- penalties should be close to the *expected completion cost*, but should not frequently fall below it. The more accurate the penalty, the better will be the solutions found. When penalty often underestimates the completion cost, then the search may not find a solution."

and (Siedlecki and Sklanski 1989):

- "the genetic algorithm with a variable penalty coefficient outperforms the fixed penalty factor algorithm,"

where a variability of penalty coefficient was determined by a heuristic rule.

This last observation was further investigated by Smith and Tate (1993). In their work they experimented with dynamic penalties, where the penalty measure depends on the number of violated constraints, the best feasible objective function found, and the best objective function value found.

For numerical optimization problems penalties usually incorporate degrees of constraint violations. Most of these methods use constraint violation measures f_j (for the j-th constraint) for the construction of the $eval_i$; these functions are defined as

$$f_j(\overline{X}) = \begin{cases} \max\{0, g_j(\overline{X})\}, & if\ 1 \le j \le q \\ |h_j(\overline{X})|, & if\ q+1 \le j \le m \end{cases}$$

For example, Homaifar et al. (1994) assume that for every constraint we establish a family of intervals that determines appropriate penalty values. The method works as follows:

- for each constraint, create several (ℓ) levels of violation,
- for each level of violation and for each constraint, create a penalty coefficient R_{ij} ($i = 1, 2, \ldots, \ell$, $j = 1, 2, \ldots, m$); higher levels of violation require larger values of this coefficient.
- start with a random population of individuals (i.e., these individuals are feasible or infeasible),
- evaluate individuals using the following formula

$$eval(\overline{X}) = f(\overline{X}) + \sum_{j=1}^{m} R_{ij} f_j^2(\overline{X}),$$

where R_{ij} is a penalty coefficient for the i-th level of violation and the j-th constraint.

Note that the function *eval* is defined on $\mathcal{S}$, i.e., it serves both feasible and infeasible solutions.

It is also possible (as suggested in Siedlecki and Sklanski 1989) to adjust penalties in a dynamic way, taking into account the current state of the search or the generation number. For example, Joines and Houck (1994) assumed dynamic penalties; individuals are evaluated (at the iteration t)

by the following formula:

$$eval(\overline{X}) = f(\overline{X}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\overline{X}),$$

where C, α and β are constants. As in Homaifar et al. (1994), the function *eval* evaluates both feasible and infeasible solutions.

The method is quite similar to Homaifar et al. (1994), but it requires many fewer parameters (C, α and β), and this is independent of the total number of constraints. Also, the penalty component is not constant but changes with the generation number. Instead of defining several levels of violation, the pressure on infeasible solutions is increased due to the $(C \times t)^{\alpha}$ component of the penalty term: towards the end of the process (for high values of t), this component assumes large values.

Michalewicz and Attia (1994) considered the following method:

- divide all constraints into four subsets: linear equations, linear inequalities, nonlinear equations, and nonlinear inequalities,
- select a random single point as a starting point (the initial population consists of copies of this single individual). This initial point satisfies all linear constraints,
- create a set of active constraints A; include there all nonlinear equations and all violated nonlinear inequalities.
- set the initial temperature $\tau = \tau_0$,
- evolve the population using the following formula:

$$eval(\overline{X}, \tau) = f(\overline{X}) + \frac{1}{2\tau} \sum_{j \in A} f_j^2(\overline{X}),$$

  (only active constraints are considered),
- if $\tau < \tau_f$, stop, otherwise
  - decrease temperature τ,
  - the best solution serves as a starting point of the next iteration,
  - update the set of active constraints A,
  - repeat the previous step of the main part.

Note that the algorithm maintains the feasibility of all linear constraints using a set of closed operators (see Michalewicz and Janikow (1991)). At every iteration the algorithm considers active constraints only, and so the pressure on infeasible solutions is increased due to the decreasing values of temperature τ. The method requires "starting" and "freezing" temperatures, τ_0 and τ_f, respectively, and the cooling scheme to decrease temperature τ.

A method of adapting penalties was developed by Bean and Hadj-Alouane (1992). As in the previous method, it uses a penalty function, however, one component of the penalty function takes feedback from the

search process. Each individual is evaluated by the formula:

$$eval(\overline{X}) = f(\overline{X}) + \lambda(t) \sum_{j=1}^{m} f_j^2(\overline{X}),$$

where $\lambda(t)$ is updated every generation t in the following way:

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), & if\ \overline{B}(i) \in \mathcal{F}\ for\ all\ t-k+1 \leq i \leq t \\ \beta_2 \cdot \lambda(t), & if\ \overline{B}(i) \in \mathcal{S}-\mathcal{F}\ for\ all\ t-k+1 \leq i \leq t \\ \lambda(t), & otherwise, \end{cases}$$

where $\overline{B}(i)$ denotes the best individual, in terms of function *eval*, in generation i, $\beta_1, \beta_2 > 1$ and $\beta_1 \neq \beta_2$ (to avoid cycling). In other words, the method (1) decreases the penalty component $\lambda(t+1)$ for the generation $t+1$, if all best individuals in the last k generations were feasible, and (2) increases penalties, if all best individuals in the last k generations were unfeasible. If there are some feasible and unfeasible individuals as best individuals in the last k generations, $\lambda(t+1)$ remains without change.

Some researchers (Powell and Skolnick 1993, Michalewicz and Xiao 1995) reported good results of their evolutionary algorithms, which worked under the assumption that any feasible individual was better than any infeasible one. Powell and Skolnick (1993) applied this heuristic rule for the numerical optimization problems: evaluations of feasible solutions were mapped into the interval $(-\infty, 1)$ and infeasible solutions—into the interval $(1, \infty)$ (for minimization problems). Michalewicz and Xiao (1995) experimented with the path planning problem and used two separate evaluation functions for feasible and infeasible individuals. The values of $eval_i$ were increased (i.e., made less attractive) by adding such a constant, so that the best infeasible individual was worse that the worst feasible one. This approach can be viewed as 'adaptive' penalty, which is a function of the values of individuals in the current population.

Yet another approach was proposed recently by Le Riche et al. 1995. The authors designed a (segregated) genetic algorithm which uses two values of penalty parameters (for each constraint) instead of one; these two values aim at achieving a balance between heavy and moderate penalties by maintaining two subpopulations of individuals. The population is split into two cooperating groups, where individuals in each group are evaluated using either one of the two penalty parameters.

It seems that the appropriate choice of the penalty method may depend on (1) the ratio between sizes of the feasible and the whole search space, (2) the topological properties of the feasible search space, (3) the type of the objective function, (4) the number of variables, (5) number of constraints, (6) types of constraints, and (7) number of active constraints at the optimum. Thus the use of penalty functions is not trivial and only some partial analysis of their properties is available. Also, a promising direction

for applying penalty functions is the use of adaptive penalties: penalty factors can be incorporated in the chromosome structures in a similar way as some control parameters are represented in the structures of evolution strategies and evolutionary programming.

3.2. Repair methods. Repair algorithms enjoy a particular popularity in the evolutionary computation community: for many combinatorial optimization problems (e.g., traveling salesman problem, knapsack problem, set covering problem, etc.) it is relatively easy to 'repair' an infeasible individual. Such a repaired version can be used either for evaluation only, i.e.,

$$eval_i(y) = eval_f(x),$$

where x is a repaired (i.e., feasible) version of y, or it can also replace (with some probability) the original individual in the population.

The process of repairing infeasible individuals is related to a combination of learning and evolution (the so-called *Baldwin effect*, Whitley et al. 1994). Learning (as local search in general, and local search for the closest feasible solution, in particular) and evolution interact with each other: the fitness value of the improvement is transferred to the individual. In that way a local search is analogous to learning that occurs during one generation of a particular string.

The weakness of these methods is in their problem dependence. For each particular problem a specific repair algorithm should be designed. Moreover, there are no standard heuristics on design of such algorithms: usually it is possible to use a greedy repair, random repair, or any other heuristic which would guide the repair process. Also, for some problems the process of repairing infeasible individuals might be as complex as solving the original problem. This is the case for the nonlinear transportation problem (see Michalewicz 1993), most scheduling and timetable problems, and many others.

On the other hand, the recently completed Genocop III system (Michalewicz and Nazhiyath 1995) for constrained numerical optimization (nonlinear constraints) is based on repair algorithms. Genocop III incorporates the original Genocop system (which handles linear constraints only; see section H), but also extends it by maintaining two separate populations, where a development in one population influences evaluations of individuals in the other population. The first population P_s consists of so-called search points which satisfy linear constraints of the problem. As in Genocop, the feasibility (in the sense of linear constraints) of these points is maintained by specialized operators. The second population P_r consists of so-called reference points from $\mathcal{F}$; these points are fully feasible, i.e., they satisfy *all* constraints. Reference points $\vec{r}$ from P_r, being feasible, are evaluated directly by the objective function (i.e., $eval_f(\vec{r}) = f(\vec{r})$). On the other hand, search points from P_s are "repaired" for evaluation

and the repair process works as follows. Assume, there is a search point $\vec{s} \in P_s$. If $\vec{s} \in \mathcal{F}$, then $eval_f(\vec{s}) = f(\vec{s})$, since $\vec{s}$ is fully feasible. Otherwise (i.e., $\vec{s} \notin \mathcal{F}$), the system selects one of the reference points, say $\vec{r}$ from P_r and creates a sequence of points $\vec{z}$ from a segment between $\vec{s}$ and $\vec{r}$: $\vec{z} = a\vec{s} + (1 - a)\vec{r}$. This can be done either (1) in a random way by generating random numbers a from the range $\langle 0, 1\rangle$, or (2) in a deterministic way by setting $a_i = 1/2, 1/4, 1/8, \ldots$ until a feasible point is found. Once a fully feasible $\vec{z}$ is found, $eval_i(\vec{s}) = eval_f(\vec{z}) = f(\vec{z})$. Clearly, in different generations the same search point $\mathcal{S}$ can evaluate to different values due to the random nature of the repair process.

The question of replacing repaired individuals is related to so-called *Lamarckian evolution* (Whitley et al. 1994), which assumes that an individual improves during its lifetime and that the resulting improvements are coded back into the chromosome. As stated in Whitley et al. 1994:

> "Our analytical and empirical results indicate that Lamarckian strategies are often an extremely fast form of search. However, functions exist where both the simple genetic algorithm without learning and the Lamarckian strategy used [...] converge to local optima while the simple genetic algorithm exploiting the Baldwin effect converges to a global optimum."

This is why it is necessary to use the replacement strategy very carefully.

Recently (see Orvosh and Davis 1993) a so-called 5%-rule was reported: this heuristic rule states that in many combinatorial optimization problems, an evolutionary computation technique with a repair algorithm provides the best results when 5% of repaired individuals replace their infeasible originals. However, many recent experiments (e.g., Michalewicz 1994) indicated that for many combinatorial optimization problems this rule did not apply. Either a different percentage gives better results, or there is no significant difference in the performance of the algorithm for various probabilities of replacement.

In continuous domains, a new replacement rule is emerging. The Genocop III system (see section E) for constrained numerical optimization problems with nonlinear constraints is based on the repair approach. The first experiments (based on 10 test cases which have various numbers of variables, constraints, types of constraints, numbers of active constraints at the optimum, etc.) indicate that the 15% replacement rule is a clear winner: the results of the system are much better than with either lower or higher values of the replacement rate.

At present, it seems that the 'optimal' probability of replacement is problem-dependent and it may change over the evolution process as well. Further research is required for comparing different heuristics for setting this parameter, which is of great importance for all repair-based methods.

3.3. Specialized data structures and operators. There are some other possibilities for handling constraints by evolutionary algorithms. One of the popular directions is based on maintenance of feasible populations by special representations and genetic operators.

During the last decade several specialized systems were developed for particular optimization problems; these systems use a unique chromosomal representations and specialized 'genetic' operators which alter their composition. Some of such systems were described in Davis (1991); other examples include various systems developed for the traveling salesman problem (Michalewicz, 1996), as well as Genocop (Michalewicz and Janikow 1991) for optimizing numerical functions with linear constraints and Genetic-2N (Michalewicz et al. 1991) for the nonlinear transportation problem. For example, Genocop assumes linear constraints only and a feasible starting point (or feasible initial population). A closed set of operators maintains feasibility of solutions. For example, when a particular component x_i of a solution vector $\overline{X}$ is mutated, the system determines its current domain $dom(x_i)$ (which is a function of linear constraints and remaining values of the solution vector $\overline{X}$) and the new value of x_i is taken from this domain (either with flat probability distribution for uniform mutation, or other probability distributions for non-uniform and boundary mutations). In any case the offspring solution vector is always feasible. Similarly, arithmetic crossover[2]

$$a\overline{X} + (1-a)\overline{Y}$$

of two feasible solution vectors $\overline{X}$ and $\overline{Y}$ yields always a feasible solution (for $0 \leq a \leq 1$) in convex search spaces (the system assumes linear constraints only which imply convexity of the feasible search space $\mathcal{F}$). Consequently, there is no need to define the function $eval_i$; the function $eval_f$ is (as usual) the objective function f.

Such systems are much more reliable than any other evolutionary techniques based on the penalty approach (Michalewicz 1994). This is a quite popular trend. Many practitioners use problem-specific representations and specialized operators in building very successful evolutionary algorithms in many areas; these include numerical optimization, machine learning, optimal control, cognitive modeling, classic operation research problems (traveling salesman problem, knapsack problems, transportation problems, assignment problems, bin packing, scheduling, partitioning, etc.), engineering design, system integration, iterated games, robotics, signal processing, and many others.

Also, it is interesting to note, that original evolutionary programming techniques (Fogel et al. 1966) and genetic programming techniques (Koza

[2]The arithmetical crossover operator generate offspring by linear combinations of the parents. As noted, such a strategy of generating a set of diverse trial points by linear and convex combinations (and allowing the offspring to influence the search) was proposed some years ago in the scatter search approach by Glover (1977).

1992) fall into this category of evolutionary algorithms: these techniques maintain feasibility of finite state machines or hierarchically structured programs by means of specialized representations and operators.

Another possibility for restricting the search to feasible individuals only is based on the idea of decoders. In these techniques a chromosome "gives instructions" on how to build a feasible solution. For example, a sequence of items for the knapsack problem can be interpreted as: "take an item if possible"—such interpretation would lead always to feasible solutions. Let us consider the following scenario: we try to solve the 0–1 knapsack problem with n items; the profit and weight of the i-th item are p_i and w_i, respectively. We can sort all items in decreasing order of p_i/w_i's and interpret the binary string

$$(1100110001001110101001010111010101...0010)$$

in the following way: take the first item from the list (i.e., the item with the largest ration of profit to weight) if the item fits in the knapsack. Continue with the second, fifth, sixth, tenth, etc. items from the sorted list, until the knapsack is full or there are no more items available. Note that the sequence of all 1's corresponds to a greedy solution. Any sequence of bits would translate into a feasible solution, every feasible solution may have many possible codes. We can apply classical binary operators (crossover and mutation): any offspring is clearly feasible.

However, it is important to point out that several factors should be taken into account while using decoders. Each decoder imposes a relationship T between a feasible solution and decoded solution. It is important that several conditions are satisfied: (1) for each solution $s \in \mathcal{F}$ there is a decoded solution d, (2) each decoded solution d corresponds to a feasible solution s, and (3) all solutions in $\mathcal{F}$ should be represented by the same number of decodings d.[3]

Additionally, it is reasonable to request that (4) the transformation T is computationally fast and (5) it has locality feature in the sense that small changes in the decoded solution result in small changes in the solution itself. An interesting study on coding trees in genetic algorithm was reported by Palmer and Kershenbaum (1994), where the above conditions were formulated.

[3]However, as observed by Davis (1997), the requirement that all solutions in $\mathcal{F}$ should be represented by the same number of decodings seems overly strong: there are cases in which this requirement might be suboptimal. For example, suppose we have a decoding and encoding procedure which makes it impossible to represent suboptimal solutions, and which encodes the optimal one: this might be a good thing. (An example would be a graph coloring order-based chromosome, with a decoding procedure that gives each node its first legal color. This representation could not encode solutions where some nodes that could be colored were not colored, but this is a good thing!)

The third option for maintaining feasible individuals only is based on the idea of exploring boundaries between feasible and infeasible parts of the search space. Some other heuristic methods recognized the need for searching areas close to the boundary of the feasible region. For example, one of the most recently developed approaches for constrained optimization is strategic oscillation. Strategic oscillation was originally proposed in accompaniment with the strategy of scatter search (Glover, 1977), and more recently has been applied to a variety of problem settings in combinatorial and nonlinear optimization (see, for example, the review of Glover, 1995). The approach is based on identifying a critical level, which represents a boundary between feasibility and infeasibility. The basic strategy is to approach and cross the feasibility boundary, by a design that is implemented either by adaptive penalties and inducements (which are progressively relaxed or tightened according to whether the current direction of search is to move deeper into a particular region or to move back toward the boundary) or by simply employing modified gradients or sub-gradients to progress in the desired direction.

It seems that the evolutionary computation techniques have a huge potential in incorporating specialized operators which search the boundary of feasible and infeasible regions in an efficient way. The first results were reported in Michalewicz et al. (1996); for resent results, see Schoenauer and Michalewicz (1996, 1997).

4. Conclusions. This paper surveys several methods which support the most important step of any evolutionary technique: evaluation of the population. It is clear that further studies in this area are necessary: different problems require different "treatment". It is also possible to mix different strategies described in this paper; for example, Paechter et al. 1994 built a successful evolutionary system for a timetable problem, where "each chromosome in the population gives instructions on how to build a timetable. These instruction may or may not result in a feasible timetable", thus allowing other heuristics to be added to the proposed decoder. The author is not aware of any results which provide heuristics on relationships between categories of optimization problems and evaluation techniques in the presence of infeasible individuals; this is an important area of future research.

5. Acknowledgments. This material is based upon work supported by the National Science Foundation under Grant IRI-9322400.

REFERENCES

[1] BEAN, J.C. AND HADJ-ALOUANE, A.B. (1992), *A Dual Genetic Algorithm for Bounded Integer Programs*, Department of Industrial and Operations Engineering, The University of Michigan, TR 92-53.

[2] DAVIS, L. (1987), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Los Altos, CA, 1987.

[3] DAVIS, L. (1991), *Handbook of Genetic Algorithms*, New York, Van Nostrand Reinhold.

[4] DAVIS, L. (1997), *Private communication.*

[5] DE JONG, K.A. (1975), *An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Doctoral dissertation*, University of Michigan, Dissertation Abstract International, 36(10), 5140B, (University Microfilms No 76-9381).

[6] DE JONG K.A. AND W.M. SPEARS (1989), *Using Genetic Algorithms to Solve NP-Complete Problems*, In Proceedings of the Third International Conference on Genetic Algorithms, Los Altos, CA, Morgan Kaufmann Publishers, 124–132.

[7] FALKENAUER, E. (1994), *A New Representation and Operators for GAs Applied to Grouping Problems*, Evolutionary Computation, **Vol. 2, No.2**, 123–144.

[8] FOGEL, L.J., A.J. OWENS AND M.J. WALSH (1966), *Artificial Intelligence through Simulated Evolution*, New York, Wiley.

[9] GLOVER, F. (1977), *Heuristics for Integer Programming Using Surrogate Constraints*, Decision Sciences, **Vol.8, No.1**, 156–166.

[10] GLOVER, F. (1995), *Tabu Search Fundamentals and Uses*, Graduate School of Business, University of Colorado.

[11] HOMAIFAR, A., S. H.-Y. LAI AND X. QI (1994), *Constrained Optimization via Genetic Algorithms*, Simulation, **Vol.62**, 242–254.

[12] JOINES, J.A. AND C.R. HOUCK (1994), *On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems With GAs*, In Proceedings of the Evolutionary Computation Conference-Poster Sessions, part of the IEEE World Congress on Computational Intelligence, Orlando, 27–29, June 1994, 579–584.

[13] KOZA, J. R. (1992), *Genetic Programming*, Cambridge, MA, MIT Press.

[14] LE RICHE, R., C. VAYSSADE, R. T. HAFTKA (1995), *A Segregated Genetic Algorithm for Constrained Optimization in Structural Mechanics*, Technical Report, Universite de Technologie de Compiegne, France.

[15] MICHALEWICZ, Z. (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 3rd edition, New York.

[16] MICHALEWICZ, Z. AND N. ATTIA (1994), *In Evolutionary Optimization of Constrained Problems*, Proceedings of the 3rd Annual Conference on Evolutionary Programming, eds. A. V. Sebald and L. J. Fogel, River Edge, NJ, World Scientific Publishing, 98–108.

[17] MICHALEWICZ, Z. AND C. JANIKOW (1991), *Handling Constraints in Genetic Algorithms*, In Proceedings of the Fourth International Conference on Genetic Algorithms, Los Altos, CA, Morgan Kaufmann Publishers, 151–157.

[18] MICHALEWICZ, Z. AND NAZHIYATH, G. (1995), *Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints*, In Proceedings of the 2nd IEEE International Conference on Evolutionary Computation, Perth, 29 November - 1 December 1995.

[19] MICHALEWICZ, Z., NAZHIYATH, G., AND MICHALEWICZ, M. (1996), *A Note on Usefulness of Geometrical Crossover for Numerical Optimization Problems*, In Proceedings of the 5th Annual Conference on Evolutionary Programming, L. J. Fogel, P. J. Angeline, and T. Baeck (eds.), MIT Press, Cambridge, MA, 1996, 305–312.

[20] MICHALEWICZ, Z., G.A. VIGNAUX, AND M. HOBBS (1991), *A Non-Standard Genetic Algorithm for the Nonlinear Transportation Problem*, ORSA Journal on Computing, **Vol.3, No.4**, 1991, 307–316.

[21] MICHALEWICZ, Z. AND J. XIAO (1995), *Evaluation of Paths in Evolutionary Planner/ Navigator*, In Proceedings of the 1995 International Workshop on Biologically Inspired Evolutionary Systems, Tokyo, Japan, May 30–31, 1995, 45–52.

[22] ORVOSH, D. AND L. DAVIS (1993), *Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints*, In Proceedings of the Fifth International Conference on Genetic Algorithms, Los Altos, CA, Morgan Kaufmann Publishers, 650.

[23] PAECHTER, B., A. CUMMING, H. LUCHIAN, AND M. PETRIUC (1994), *Two Solutions to the General Timetable Problem Using Evolutionary Methods*, In Proceedings of the IEEE International Conference on Evolutionary Computation, 27–29 June 1994, 300–305.

[24] PALMER, C. C. AND A. KERSHENBAUM (1994), *Representing Trees in Genetic Algorithms*, In Proceedings of the IEEE International Conference on Evolutionary Computation, 27–29 June 1994, 379–384.

[25] PARDALOS, P. (1994), *On the Passage from Local to Global in Optimization*, In Mathematical Programming, J.R. Birge and K.G. Murty (Editors), The University of Michigan, 1994.

[26] POWELL, D. AND M.M. SKOLNICK (1993), *Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints*, In Proceedings of the Fifth International Conference on Genetic Algorithms, Los Altos, CA, Morgan Kaufmann Publishers, 424–430.

[27] RICHARDSON, J. T., M. R. PALMER, G. LIEPINS AND M. HILLIARD (1989), *Some Guidelines for Genetic Algorithms with Penalty Functions*, In Proceedings of the Third International Conference on Genetic Algorithms, Los Altos, CA, Morgan Kaufmann Publishers, 191–197.

[28] SCHOENAUER, M. AND MICHALEWICZ, Z. (1996), *Evolutionary Computation at the Edge of Feasibility*, In Proceedings of the 4th Parallel Problem Solving from Nature, H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel (Editors), Berlin, September 22–27, 1996, Springer-Verlag, Lecture Notes in Computer Science, **Vol. 1141**, 245–254.

[29] SCHOENAUER, M. AND MICHALEWICZ, Z. (1997), *Boundary Operators for Constrained Parameter Optimization Problems*, In Proceedings of the 7th International Conference on Genetic Algorithms, East Lansing, Michigan, July 19–23, 1997.

[30] SIEDLECKI, W. AND J. SKLANSKI (1989), *Constrained Genetic Optimization via Dynamic Reward-Penalty Balancing and Its Use in Pattern Recognition*, In Proceedings of the Third International Conference on Genetic Algorithms, Los Altos, CA, Morgan Kaufmann Publishers, 141–150.

[31] SMITH, A.E. AND D.M. TATE (1993), *Genetic Optimization Using a Penalty Function*, In Proceedings of the Fifth International Conference on Genetic Algorithms, 499–503, Urbana-Champaign, CA: Morgan Kaufmann.

[32] WHITLEY, D., V.S. GORDON, AND K. MATHIAS (1994), *Lamarckian Evolution, the Baldwin Effect and function Optimization*, In Proceedings of the Parallel Problem Solving from Nature, 3, Springer-Verlag, New York, 6–15.

GENETIC ALGORITHM OPTIMIZATION OF ATOMIC CLUSTERS

J.R. MORRIS[†*], D.M. DEAVEN[†], K.M. HO[†], C.Z. WANG[†], B.C. PAN[†], J.G. WACKER[†], AND D.E. TURNER[†]

Abstract. We have been using genetic algorithms to study the structures of atomic clusters and related problems. This is a problem where local minima are easy to locate, but barriers between the many minima are large, and the number of minima prohibit a systematic search. We use a novel mating algorithm that preserves some of the geometrical relationship between atoms, in order to ensure that the resultant structures are likely to inherit the best features of the parent clusters. Using this approach, we have been able to find lower energy structures than had been previously obtained. Most recently, we have been able to turn around the "building block" idea, using optimized structures from the GA to learn about systematic structural trends. We believe that an effective GA can help provide such heuristic information, and (conversely) that such information can be introduced back into the algorithm to assist in the search process.

1. Introduction. One of the main interests in genetic algorithms (GA's) is their application to difficult optimization problems. In many such problems, there are many possible solutions, with no clear way to conduct a systematic search. The problem is compounded when there are many optimization parameters, especially if these parameters are strongly coupled. We have recently applied genetic algorithms to one type of these problems, the determination of the lowest energy configurations of a collection of atoms. This is a particularly difficult problem because the number of metastable local energy minima grows exponentially with the number of degrees of freedom available to the system [1, 2]. In many cases, especially in systems with strong directional covalent bonds like carbon or silicon, metastable energy states are separated by large barriers reflecting the high energy cost of breaking bonds to rearrange a cluster's structure. Attempts to use traditional simulated annealing methods [3] to find the global energy minimum usually fail, leaving the system trapped in one of the numerous metastable configurations. Thus, unless some of the rules are known for constructing the ground state structure of the atomic cluster, we do not have reliable predictions for the global energy minimum for clusters once their sizes go beyond 10 to 20 atoms.

It is clear that an algorithm is needed which can 'hop' from one minimum to another and permit an efficient sampling of phase space. Genetic algorithms, in principle, can do this using an optimization strategy modeled on the Darwinian evolution process. The fitness of a population of candidate solutions to the problem is improved by selecting a group of best-performing candidates to act as parents. A "mating" operation is

*jrmorris@ameslab.gov

[†]Ames Laboratory and Department of Physics, Iowa State University, Ames, Iowa 50011-3020.

then performed among the parents to produce children to form the next generation of candidates. This process is repeated until the best solution is located [4, 5].

A key ingredient for the success of a genetic algorithm is the efficiency of the mating operation in producing good solutions to the problem. Traditional implementations of the genetic algorithm, using various forms of string recombination as the mating operator, have been tried previously on the present problem of molecular geometry optimization, but without much success [6].

We feel that the problem of the traditional mating operation is that it does not efficiently transfer favorable properties from the parents to the child. With this in mind, we have constructed a mating operation based upon the physical structure of the parent clusters, in such a way that resulting structures are much more likely to inherit promising characteristics of the parent structures. We have applied our approach to a number of problems, including carbon clusters [7], the Thomson problem of point charges on a sphere [8], and Lennard-Jones clusters [9]. In each case, we have found the genetic algorithm to be helpful in locating optimal solutions. In the most recent (and challenging) application of locating low energy silicon clusters, the genetic algorithm has led to a new understanding of the structural trends - potentially more useful than knowing the lowest energy states of particular clusters.

2. Method. Before discussing the genetic algorithm approach, we should first point out some of the challenges in accurately determining the structures of atomic clusters. (We only discuss theoretical problems; there are a number of experimental difficulties as well.) The atomic arrangement will usually be a low energy one; therefore, our searches are focused on finding the lowest energy configurations. However, calculating the energy of a cluster is not trivial. First-principles total energy calculations are quite accurate, but are computationally intensive. Therefore, searching many different configurations using this approach is not practical. In some cases, reasonably accurate empirical models exist, which produce fairly reliable energies and structures. However, if there are several structures that are close in energy (which often happens), then these empirical predictions must be supported by more accurate techniques. Even when the empirical potentials are reliable, as appears to be the case for carbon [10], the calculations of the energies and the local optimization of the structures still require significant computational resources. Given that there are a large number of local optima (as discussed in the results section below), it is therefore important to explore possible low energy structures as efficiently as possible.

To achieve this goal, we devised the following genetic algorithm approach. Starting with a population of candidate structures, we relax these candidates to the nearest local minimum using a conjugate-gradient mini-

mization or molecular dynamics quenching. Using the relaxed energies as the criteria of fitness, a fraction of the population is selected as parents. The mating operation is performed by a "cut-and-paste" procedure: First, we choose a random plane passing through the center of mass of each parent cluster. We then cut the parent clusters in this plane, and assemble the child from the atoms of one parent which lie above the plane, and the atoms of the other parent which lie below the plane. If the child generated in this manner does not contain the correct number of atoms, the parent clusters are translated an equal distance in opposing directions normal to the cut plane so as to produce a child which contains the correct number of atoms. The resulting child structure is then relaxed to the nearest local minimum.

3. Results.

3.1. Carbon clusters. We first illustrate the above algorithm by an application to the case of C_{60}. In this simulation, the interatomic interaction in the carbon clusters is described by a tight-binding potential previously shown to be very accurate for description of fullerene structures [10, 11]. Mating operations are performed every 30 time steps: 4 structures with the lowest energies are chosen as parents from a population of 16 candidates structures. We adopted an 'elitist' strategy in which the 4 parents are included in the next generation together with 12 new structures formed by mating operations between different parents. Each new structure is relaxed using molecular dynamics (MD) quenching. After 30 time steps, the energy of the new structure is examined. If the new structure is not within a certain energy range ($\sim$ 0.2 eV) of the energy of the parent structures, it is discarded and replaced by another child. Otherwise, it is fully relaxed to its lowest energy state (this usually takes about 200 MD steps). If the energy of the fully relaxed structure is lower than any of the parent structures, it is incorporated into the parent population and the highest energy parent structure discarded.

Figure 1 shows the results of the C_{60} simulations. The energy per atom is plotted for the lowest energy (solid line) and highest energy (dashed line) structure in the population as a function of the number of MD steps expended by the simulation. Several generic features of the algorithm are apparent in these results: In the initial stage, the energy drops very quickly and the population soon consists of reasonable candidates. A sampling of the structures of the population during the initial period indicate a sequence of structures of unfinished cages very similar to that observed in simulated annealing. The initial stage usually occupies only a small fraction of the total time spent by the algorithm. The results from the initial stage resembles what would be obtained by the simulated annealing method, except that the genetic algorithm obtains it more efficiently by quenching without going through long annealing procedures. The rest of the time is spent in an end game, where the remaining defects in the structure are

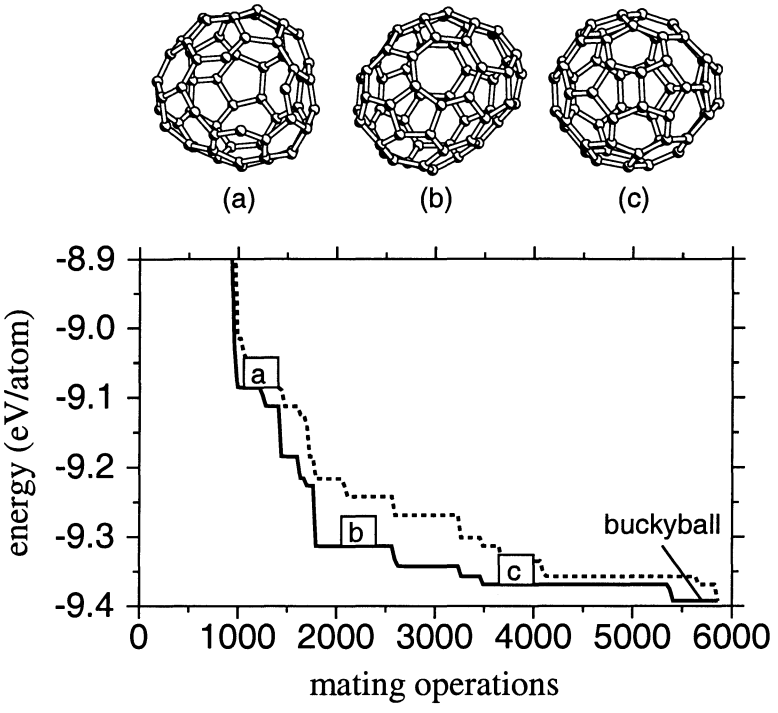


FIG. 1. *Running the genetic algorithm on* C_{60}.

removed until the ground state structure is found.

3.2. Thomson problem. In view of the success of the genetic algorithm in solving the structure of carbon clusters, we decided to test the robustness of the scheme in applications to other kinds of cluster problems. The first is the well-studied historical Thomson problem of finding the arrangement of N charges constrained on a sphere [2]. We studied this problem to observe the effect of long range forces on our algorithm. The problem is also of interest in the study of symmetry-breaking in the ground state and metastable structures due to the constrained finite number of particles. The similarity of the spherical geometry to the cage structures of the C_{60} clusters also encouraged us to apply our approach to this problem.

This problem has been used as a test case for optimization, and there are a number of results for systems less than 100 particles [2, 16, 17]. From these studies, the most reliable results had come from *random* searches, rather than simulated annealing approaches.[2] The difficulty apparently arises in the fact that most simulated annealing studies would use one starting structure and try to optimize that, rather than using multiple

initial structures.

Our approach was quickly able to locate the known results (for $N \leq 132$), and to extend these results up to 200 particles [8]. For less than about 80 particles, the problem is fairly trivial, as there are only a handful of possible solutions. However, the number of solutions rapidly grows, with an estimated 270 possible local minima for $N = 132$ and nearly 8000 for $N = 200$ [2]. The fact that we (apparently) could locate the minimum for most of these cases demonstrates the efficiency of our approach.

3.3. Lennard-Jones clusters. We have also studied the case of Lennard-Jones clusters to observe the behavior of the algorithm on compact clusters (as opposed to the cage-like structures that occur in the previous cases). This is a simple interaction, modeling the interaction between noble gas atoms such as argon. There are many studies of these clusters for $N \leq 100$ [18, 19, 20], and the number of known local minima grows much more rapidly than that of the Thomson problem – for $N = 55$, the estimate is on the order of 10^{21} minima. Again, we find that our algorithm efficiently reproduces most of the ground state structures reported (with exceptions for the $N = 75$, 76 and 77 Lennard-Jones clusters; see Ref. [20]). In several cases, we were even able to locate structures with lower energies than those previously reported. (In one case, we found two different structures with lower energies than had been found previously.)

3.4. Silicon clusters. Most recently, we have been studying Si clusters [21]. Unlike the previous cases, there are few known results except for the smallest of clusters ($N \leq 10$), and even structural trends are not well understood. Further, there have not been any empirical potentials whose results were confirmed by more accurate calculations. Even well-accepted potentials which describe bulk behavior reasonably well produce non-physical results for clusters. Most empirical models predict that the $N = 20$ cluster would be higher in energy than two $N = 10$ clusters, an unphysical result. The lack of success in determining the structures of these clusters demonstrates the challenge of this problem.

Recently, our group has produced a more accurate Si potential, and we have begun applying it to Si clusters. The accuracy of the potential is very important: if the potential leads to unphysical clusters, or overestimates the energies of clusters that are actually low energy, then even a sophisticated search algorithm will not lead to important results. On the other hand, if the potential can be used to identify low energy candidates correctly, then these clusters may be checked using more accurate calculations. Rather than use the genetic algorithm to find the lowest energy structure (within our empirical model), we have used it to generate a set of low energy solutions, producing a smaller search space that was then studied using more accurate treatments.

Using a combination of a new Si potential and the genetic algorithm, we have identified new low-energy structures in the range from $N = 10$ to

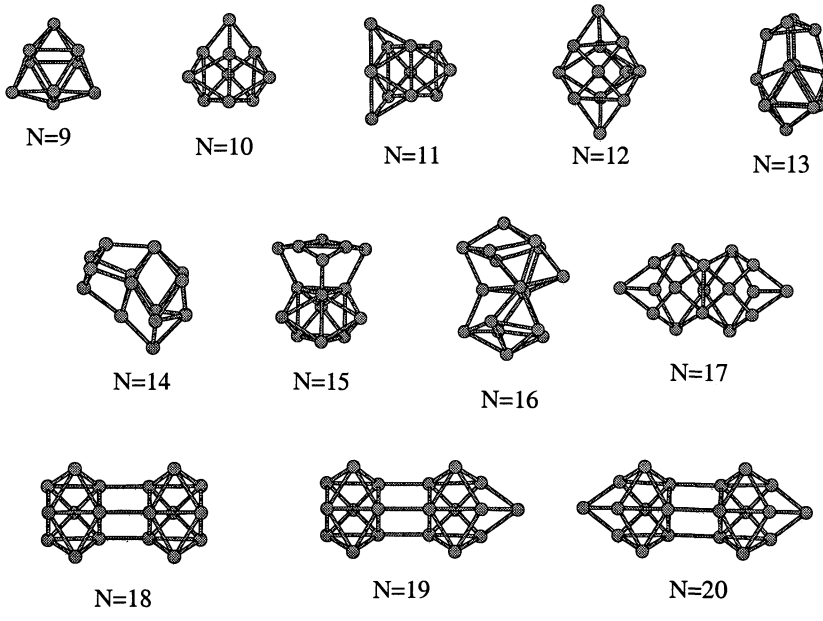


FIG. 2. *Structures based upon the $N = 9$ silicon sub-unit.*

$N = 20$. The genetic algorithm approach is essentially the same as that of the carbon clusters. To ensure a good global search, 10–20 ecologies were run for each value of N. Also, we performed simulated annealing to produce other candidate structures, and to compare the effectiveness of this approach with that of the GA. We found that for clusters larger than 13 atoms, the GA was able to produce lower energy candidate structures, indicating that this is a better search strategy.

For $N < 13$, our results are consistent with previous predictions. However, the $N = 13$ structure that we found is significantly lower in energy than previous results. Unlike previous calculations, we find that this structure is built upon that of a nine-atom sub-unit. Although this sub-unit (shown in fig. 2) is not the lowest energy $N = 9$ structure, there is a family of clusters based upon this sub-unit (or several such units, for $N \geq 18$) for all clusters in this study. These structures are shown in fig. 2; with the exceptions of $N = 9$, 17, 19 and 20, these are the lowest energy structures that we have found.

For $N \geq 17$, we have found another family of structures that are more cage-like than the previous structures. These are the lowest energy structures that we have found for $N = 17$, 19 and 20. These structures, shown in fig. 3, have one central atom, surrounded by a shell of atoms. Such "stuffed fullerene" clusters have been suggested previously for larger clusters [22, 23, 24, 25].

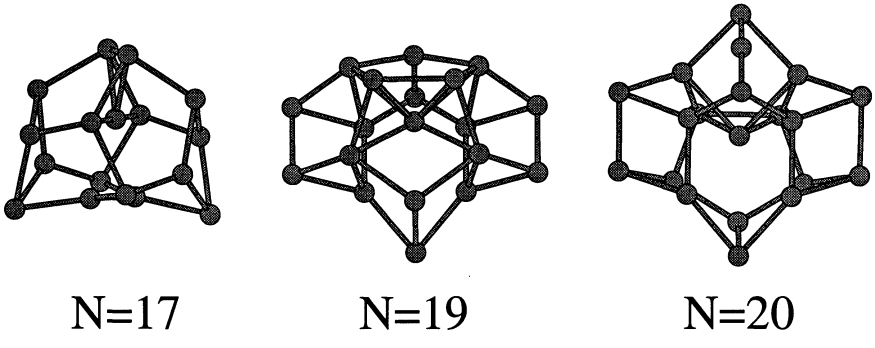


FIG. 3. *Cage-like low energy structures for Si_{17}, Si_{19} and Si_{20}.*

4. Discussion. The work presented here demonstrates that our approach is successful for a number of applications. The fact that one mating approach is applicable to these reasonably disparate problems is striking, especially given the strong differences in geometries. These are problems that other approaches or traditional GA's have not been able to address. The difficulty is not only that we must rapidly explore many minima; with the large number of possible structures available, we must also ensure that favorable "traits" of the parents be preserved through the generations. In our case, that corresponds to good local structures. Using string operations to generate new structures will not do this.

In most applications, a successful genetic algorithm must be able to preserve the relationship between strongly coupled parameters. This will only be successful for traditional linear operations (such as crossover) if the coupling is extremely simple and is also known in advance. We feel that any GA application using such operations will be inefficient, if the parameters are strongly interdependent. (Of course, if the search space is sufficiently simple, then such operations combined with local optimization may be successful. In such cases, however, a genetic algorithm will not be preferable to a random search or to simulated annealing.) It is clear that mating algorithms that respect important relationships between related parameters may be much more efficient. For our applications, the approach we described preserves most local atomic environments, while searching the complicated solution space in an unbiased way. The arbitrary numbering or ordering of the atoms is not taken into account in the "mating" process; rather, it is the physical arrangement that is important, and that is used to generate new structures.

In many applications, searching for optimal solutions is aided by using "heuristic" information. If some factor is known about good solutions, then building this into the search can greatly speed up the search. However, if this "known factor" is not necessarily a part of the solution, forcing

solutions to include such information may miss other good solutions. In the case of the Lennard-Jones particles, most results have been obtained by growth rules, searching for a structure based upon the results of smaller structures. By avoiding such biases, we were able to locate the optimal structure for $N = 38$, which is not similar to any other optimal clusters for $N < 100$.

In the case of the Si clusters, we were able to turn around the "building block" idea. The genetic algorithm generated a number of structures which contained a "structural unit" in the optimal structure. It is not surprising that some core arrangement of atoms exist, but in our case, it was the genetic algorithm that led to the particular common sub-unit. By knowing what sub-unit may appear, we may also try to extend results by building structures by hand, using the identified sub-units. This might be able to extend our predictive abilities, even when the problem is sufficiently complex that our GA breaks down. Furthermore, this suggests the idea of "seeding" the GA with some possible sub-units. If we put likely sub-units into the initial population, then the search may more efficiently locate low energy structures. If there are several possible sub-units, then such seeding may help broaden the search, preventing one class of structures from dominating the population.

In general, we believe that searches through complex solution spaces may be aided by genetic algorithms. However, blind application of bit-string mating operations seems unlikely to efficiently improve solutions, unless the parameters may be optimized more or less independently. Our application demonstrates that genetic algorithms may be constructed in such a way that complex interrelationships between parameters may be preserved by the mating process, leading to new solutions that might otherwise be missed. Our approach might be made more optimal in a number of ways; some variations and additions are currently being explored. We feel, however, that these applications are an interesting and useful demonstration of an alternative approach to genetic algorithms, and may provide some insight towards other challenging optimization problems.

5. Acknowledgments. Ames Laboratory is operated for the U.S. Department of Energy by Iowa State University under contract no. W-7405-Eng-82. This work was supported by the Director for Energy Research, Office of Basic Energy Sciences, and the High Performance Computing and Communications Initiative. Part of this work was made possible by the Scalable Computing Laboratory, which is funded by Iowa State University and Ames Laboratory.

REFERENCES

[1] L. T. Wille and J. Vennik, J. Phys. A **18**, L419 (1985).
[2] T. Erber and G. M. Hockney, Phys. Rev. Lett. **74**, 1482 (1995); review article in Adv. Chem. Phys. **98**, 495 (1997).

[3] S. KIRKPATRICK, Science **220**, 671 (1983); D. VANDERBILT, J. Comput. Phys. **56**, 259 (1984).
[4] J. H. HOLLAND, *Adaptation in natural and artificial systems* (Ann Arbor: The University of Michigan Press, ©1975.
[5] D. E. GOLDBERG, in *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley ©1989.
[6] For some examples, see Y. XIAO AND D. E. WILLIAMS, Chem. Phys. Lett. **215**, 17 (1993); T. BRODMEIER AND E. PRETSCH, J. Comput. Chem, **15**, 588 (1994); D. B. MCGARRAH AND R. S. JUDSON, J. Comput. Chem. **14**, 1385 (1993); J. A. NIESSE AND H. R. MAYNE, Chem. Phys. Lett. **261**, 576 (1996).
[7] D. M. DEAVEN AND K. M. HO, Phys. Rev. Lett. **75**, 288 (1995).
[8] J. R. MORRIS, D. M. DEAVEN AND K. M. HO, Phys. Rev. B **53**, 1740 (1996). Recent updates may be found at `http://cmp.ameslab.gov/~jrmorris/thomson.html`.
[9] D. M. DEAVEN, N. TIT, J. R. MORRIS AND K. M. HO, Chem. Phys. Letters **256**, 195 (1996).
[10] C. H. XU, C. Z. WANG, C. T. CHAN AND K. M. HO, J. Phys. Cond. Mat. **4**, 6047 (1992).
[11] B. L. ZHANG, C. Z. WANG, AND K. M. HO, Chem. Phys. Lett. **193**, 225 (1992); C. Z. WANG, B. L. ZHANG, K. M. HO, AND X. Q. WANG, Intn. J. Mod. Phys. B **7**, 4305 (1993).
[12] J. C. GROSSMAN, L. MITAS, AND K. RAGHAVACHARI, Phys. Rev. Lett. **75**, 3870 (1995).
[13] E. A. ROHLFING, D. M. COX, AND A. KALDOR, J. Chem. Phys. **81**, 3322 (1984).
[14] D. TOMÁNEK AND M. A. SCHLUTER, Phys. Rev. Lett. **67**, 2331 (1991).
[15] See for example, P. BALLONE AND P. MILANI, Phys. Rev. B **42**, 3201 (1990).
[16] J. T. WILLE, Nature **324**, 46 (1986).
[17] E. L. ALTSCHULER, T. J. WILLIAMS, E. R. RATNER, F. DOWLA AND F. WOOTEN, Phys. Rev. Lett. **72**, 2671 (1994); **74**, 1483 (1995).
[18] J. A. NORTHBY, J. Chem. Phys. **87**, 6166 (1987).
[19] N. J. A. SLOANE, R. H. HARDIN, T. D. S. DUFF, AND J. H. CONWAY, Discrete Comput. Geom. **14**, 237 (1995).
[20] J. P. K. DOYE AND D. J. WALES, J. Chem. Phys. **103**, 4324 (1995).
[21] K. M. HO ET AL, to appear in Nature.
[22] E. KAXIRAS, Chem. Phys. Lett. **163**, 323 (1989).
[23] J. L. ELKIND, J. M. ALFORD, F. D. WEISS, R. T. LAAKSONEN, AND R. E. SMALLEY, J. Chem. Phys. **87**, 2397 (1987); L. R. ANDERSON, S. MARUYAMA, AND R. E. SMALLEY, Chem. Phys. Lett. **176**, 348 (1991).
[24] U. ROTHLISBERGER, W. ANDREONI, AND M. PARRINERRO, Phys. Rev. Lett., **72**, 665 (1994).
[25] J. PAN AND M. V. RAMAKRISHNA, Phys. Rev. B **50**, 15431 (1994); M. V. Ramakrishna and A. Bahel, J. Chem. Phys., **104**, 9833 (1996).

SEARCH, BINARY REPRESENTATIONS AND COUNTING OPTIMA

SORAYA RANA* AND L. DARRELL WHITLEY†

Abstract. Choosing a good representation is a vital component of solving any search problem. However, choosing a good representation for a problem is as difficult as choosing a good search algorithm for a problem. Wolpert and Macready's No Free Lunch theorem proves that no search algorithm is better than any other over all possible discrete functions. We elaborate on the No Free Lunch theorem by proving that there tend to be a small set of points that occur as local optima under almost all representations. Along with the analytical results, we provide some empirical evaluation of two representations commonly used in genetic algorithms: Binary Reflected Gray coding and standard Binary encoding.

Key words. genetic algorithms, evolutionary computing, search, mathematical foundations.

1. Introduction. Wolpert and Macready's No Free Lunch theorem [5] proves that no search algorithm is better on average over the set of all possible functions than any other search algorithm. While this theorem is simple, it makes it clear (subject to certain assumptions) that there is no general purpose search algorithm that can solve all search problems better than any other algorithm.

We present a No Free Lunch result proving, for any function of fixed size N and any local search operator with a fixed neighborhood size k, that "good" local optima remain local optima under almost all representations of that function. We compute the expected number of local optima that will occur for any function and fixed neighborhood search operator under all possible representations. We also look at the practical limitations of this No Free Lunch result by illustrating how commonly used encodings measure up to the expected number of local optima for arbitrary functions. In practice, commonly used encodings for sample test problems reveal dramatically fewer local optima than expected for arbitrary representations. We also introduce a new technique for exploiting multiple binary representations of a search space.

2. No free lunch. There are few theoretical results that apply in a general fashion to search. Recently, there has been a good deal of interest in the "No Free Lunch" results as they apply to discrete function optimization and search. To understand these results, we first outline some of the simple assumptions behind this theorem. First, assume the optimization problem is discrete; this describes all combinatorial optimization

*Computer Science Department, Colorado State University, Fort Collins, CO 80523 email: rana@cs.colostate.edu

†Computer Science Department, Colorado State University, Fort Collins, CO 80523 email: whitley@cs.colostate.edu

problems–and really all optimization problems being solved on computers since computers have finite precision. Second, we ignore the fact that we can resample points in the space.

The "No Free Lunch" result can be stated as follows:

> The performance of all possible search algorithms is exactly the same when averaged over all possible functions.

Abstractly, we can represent an algorithm as an ordering, or permutation, over the points in the search space. We can also view all search algorithms as being deterministic; "stochastic" algorithms are, in practice, deterministic and can be modeled as a stochastic search operator coupled with a specific random seed. Thus, for any particular problem with a fixed search space of size N, an algorithm is just one of N! ordering of points in the search space. Furthermore, from this operational point of view, all possible representations of a search space is also the same set of N! orderings. Thus, there is an isomorphic relationship between all possible search algorithms over this collection of points, all possible functions that can be constructed from this set of points and even all possible representations over this set of points.

Note that if we fix the search algorithm, then these permutations represent all possible functions (or representations) over this set of points. On the other hand, if we fix the function (representation), then the permutations represent all possible search algorithms over this set of points. If we vary both search algorithms and functions, then every search algorithm is searching over the same set of permutations and the performance of all search algorithms are identical. Since this is true for any discrete set of points we might wish to pick, all search algorithms are the same over all possible discrete functions [3].

3. Representation and local optima. Given any discrete function of size N, one can compute the average number of optima that will occur over all local search operators with a fixed neighborhood of size k. We can also compute the number of points that remain local optima with a specific probability p.

Suppose we have N unique points in our search space and a search operator that explores k points before making its next move. (We will deal with points with duplicate evaluation in a later section.) A point is considered a local optimum from a steepest ascent perspective if its evaluation is better than all of its k-neighbors. Further suppose that the N points in our search space each have unique values. We can sort those points to create a ranking, $R = r_1, r_2, ..., r_n$, in terms of their function values (where r_1 is the best point in the space and r_n is the worst point in the space). Using this ranking, we can compute the probability that a point ranked in the i-th position in r is a local optimum under an arbitrary representation of the search space. This probability is given by the formula:

$$P(i) = \frac{\binom{N-i}{k}}{\binom{N-1}{k}} \qquad [1 \leq i \leq (N-k)] \tag{1}$$

Proof. For any point in the search space, there are $\binom{N-1}{k}$ possible neighbors for that point. If the point is ranked in position r_1, then there are $\binom{N-1}{k}$ sets of neighbors that do not contain a point of higher evaluation than the point r_1. Therefore, the point ranked in position r_1 will always be a local optimum under all representations of the function. In the general case, a point in position r_i has only $\binom{N-i}{k}$ sets of neighbors that do not contain a point of higher evaluation than the point r_i. Therefore the probability that the point in position r_i remains a local optimum under an arbitrary representation is $\binom{N-i}{k}/\binom{N-1}{k}$. □

As N increases, the probabilities quickly become difficult to compute. For implementation purposes, it is easier to represent the probabilities as a recurrence relation. The recurrence relation is given as:

$$P(i) = P(i-1)\frac{N-k-(i-1)}{N-(i-1)} \; [2 \leq i \leq (N-k)], \quad P(1) = 1.0 \tag{2}$$

These probabilities enable us to count the expected number of local optima that should occur in any function of size N. The formula for computing the expected number of times a particular point will be a local optima is simply $N! \times P(i)$. Therefore the expected number of optima over the set of all representations is:

$$\mathcal{E}(N,k) = \sum_{i=1}^{N-k} P(i) \times N! \tag{3}$$

If we want to find the expected number of local optima for a single representation instance, we divide $\mathcal{E}(N,k)$ by $N!$, which yields:

$$\mu(N,k) = \sum_{i=1}^{N-k} P(i) = \frac{N}{k+1} \tag{4}$$

In addition to computing the expected number of local optima for an arbitrary function, we can compute the ranking index i for r_i for any specific probability. This information can be used to indicate how many points are likely to occur as optima in most representations. Suppose we

are interested in knowing which r_i has a probability $P(i) \geq p$. Let $p = 1/X$ and assume $P(i) = p$. We now need to solve the following equation:

$$\frac{\binom{N-i}{k}}{\binom{N-1}{k}} = 1/X$$

It follows that:

$$X = \frac{\binom{N-1}{k}}{\binom{N-i}{k}} = \prod_{j=1}^{i-1} \frac{N-j}{N-k-j}$$

Note that we can compute bounds for this equation.

$$\left(\frac{N}{N-k}\right)^{(i-1)} \leq X \leq \left(\frac{N-(i-1)}{N-k-(i-1)}\right)^{(i-1)} \tag{5}$$

Let the lower bound estimate of i be represented by i_{LB}, which is computed as follows.

$$i_{LB} = \frac{ln(X)}{ln(\frac{N}{N-k})} - 1 \tag{6}$$

We next compute i_{UB} using the following:

$$i_{UB} = \left(\frac{N-(i_{LB}-1)}{N-k-(i_{LB}-1)}\right)^{(i_{LB}-1)} \tag{7}$$

Using these bounds, a search between the bounds can quickly produce the exact point i at which $P(i) \geq p$ even for very large search spaces.

3.1. Functions with reoccurring values. Most real-world functions are not restricted to producing unique function values. Therefore, we adapt the calculations in the previous section to the case where there are duplicate function evaluations.

Once again, we start with the ranking function R where all function evaluations are sorted (with duplicates). We then create a set of disjoint subsets, G. Each subset $g \in G$ contains indices into R such that for all $i \in g$, $r_i = y$ where y is a particular function evaluation. In other words, all elements of a subset index into R where all of those points have an equal evaluation. The groups that make up G can then be ordered based on evaluation, with the ranking of elements in g_k being better (i.e., "less than" for a minimization problem) than the evaluation of members of g_{k+i} for all positive integers i. The function G serves as a meta-ranking of the function R.

We must first tackle the problem of computing the probability that any set of points occurs as a local optima. The function P that was defined

in the previous section is a one-to-one function with R. We can use P to define a new set of probabilities $\mathcal{P}$ that can be associated with the subsets in G. Using those new probabilities, we can compute the expected number of optima that will occur over the set of all functions (denoted $\mu(N,k)$). Finally, we can adapt the calculation that a point in a given position in R will occur as a local optimum with probability of at least p.

Case 1: We define "locally optimal" points as points that are better than and not equal to any of their neighbors (i.e. we assume that points that occur on ridges and plateaus are not considered locally optimal).

The probability, $\mathcal{P}(k)$, for any point indexed by subset g_k to occur as a local optima is:

$$\mathcal{P}(k) = P(max(g_k)) \tag{8}$$

where the function $max(g)$ returns the maximum index contained in the set g. We can then define $\mu(N,k) = \sum_{k=1}^{m} |g_k| \times \mathcal{P}(k)$ where $m = |G|$.

Now we are interested in finding the index i' such that the point $r_{i'}$ occurs as a local optima with at least probability p. We already know how to compute the value i where $P(i) \geq p$ when there are no duplicates in R. We can use G and i to compute the new value i' that takes into account the duplicate function evaluations. First we locate g_k such that $i \in g_k$. Then we know the following information about $P(i)$: either $P(i) = \mathcal{P}(k)$ or $P(i) > \mathcal{P}(k)$.

$$i' = \begin{cases} i & if \;\; \mathcal{P}(k) = P(i) \\ max(g_{k-1}) & otherwise \end{cases} \tag{9}$$

Case 2: We now define locally optimal to mean that a point must be better than or equal to any of its k-neighbors. This is the case where a ridge or plateau is treated as locally optimal.

The probability $\mathcal{P}(k)$ of a subset of points indexed by the subset g_k occurring as local optima is:

$$\mathcal{P}(k) = P(min(g_k)) \tag{10}$$

where the function $min(g)$ returns the minimum index contained in the set g. The average number of local optima that one would expect to find in an arbitrary representation of the function is still $\mu(N,k) = \sum_{k=1}^{m} |g_k| \times \mathcal{P}(k)$ where $m = |G|$.

Once again we need to find the position i' in R such that the point $r_{i'}$ occurs as a local optima with at least probability p. Starting with the value i computed assuming no duplicates, we can generate the new index i'. First we locate g_k such that $i \in g_k$. Then we know that $P(i) \leq \mathcal{P}(k)$ by definition. So the new index i' must be:

$$i' = min(g_k) \tag{11}$$

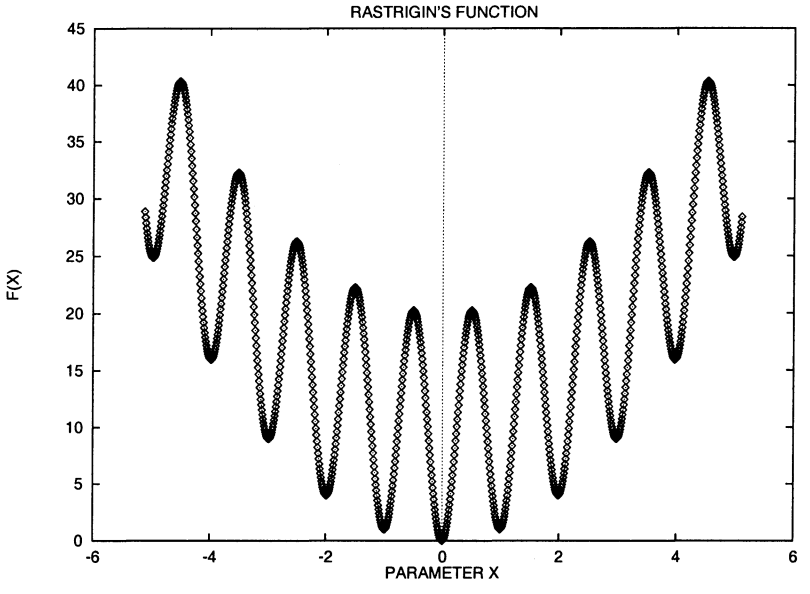


FIG. 1. *Rastrigin's function encoded using 1024 data points on a single dimension.*

4. Analysis of a sample function. The calculations in the previous section can be of use to genetic algorithm researchers in several ways. Since genetic algorithms have traditionally been used with binary encodings, these calculations can provide useful insight into the relative quality of different encoding techniques. Secondly, these calculations can also provide insight into the complexity of the underlying function landscape given some fixed binary encoding.

We looked at two possible binary encodings: Binary Reflected Gray Coding [1] and standard binary encoding. There are a large family of Gray codes (we speculate there are at least $L * 2^L$ such codes; the exact number is an open question).

Throughout this section we will analyze a specific multimodal function (Eq.12) known as Rastrigin's function [2] and illustrated in Figure 4.

$$f(x_i \mid_{i=1,N}) = (10N) + \left[\sum_{i=1}^{n} \left(x_i^2 - 10cos(2\pi x_i)\right)\right] \qquad (12)$$

$$x_i \in [-5.12, 5.11]$$

This function is typically minimized over multiple dimensions with each parameter remaining independent of all others. If we look at Rastrigin's function in numeric space with a neighborhood size of two (looking only at a point on either side of our current point), then there are 11 local minima along any dimension. This also means that there are 11^N possible local minima in an N-dimensional version of the function.

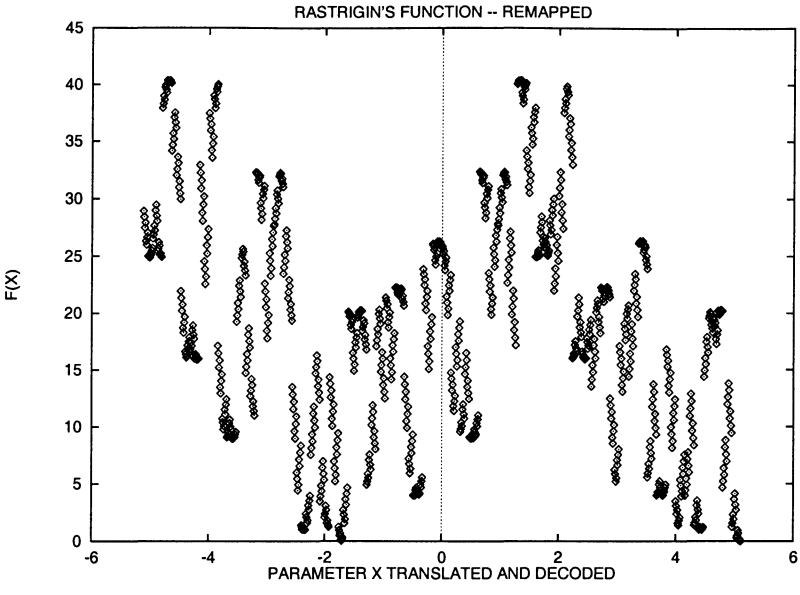


FIG. 2. *Function resulting from decoding a binary representation of Rastrigin's function as a gray code.*

When we impose a binary encoding on the function, however, the number of locally optimal points that occur will change. The function is discretized using a 10-bit encoding. The probabilities of any point occurring as a local optima in a space of 1024 data points with a neighborhood size of 10 can be quickly computed using the recurrence relation. Since this function does contain duplicates, it is necessary to adjust the probabilities to account for the duplicates. Once this is done, the average number of local optima expected to occur in any random reordering of this function can be computed by summing the probabilities. That value is 93.5885. This value was empirically verified using 1000 random orderings of the function values yielding an average of 93.3450 locally optimal points.

Using Reflected Gray coding there are only 5 locally optimal points in Rastrigin's function. While there were 11 optima in numeric space, under standard Binary encoding there are 19 locally optimal points. Using our previous results, we note that given the expected number of local optima for an arbitrary binary encoding is nearly 93: the standard Binary representation is still much better than most other possible encodings. Taking these values to the context of an N-dimensional problem, it is clear that having 5^N is much more desirable than having 11^N or 19^N local optima.

Another interesting way to visualize the effects of the standard binary encoding on this problem is as follows. Let $\mathcal{R}_b$ be the representation obtained by converting from integers representing the numeric input space

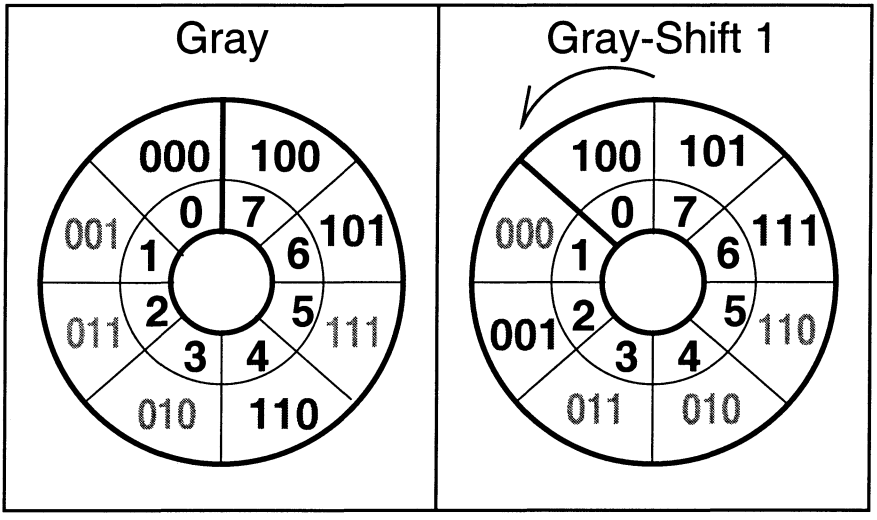


FIG. 3. *Example of shifting using a Gray encoding.*

for Rastrigin's function to bits using a standard Binary encoding. Now assume we have a function $G(x)$ that when encoded in Binary and Grayed (using Binary Reflected Gray coding) also results in $\mathcal{R}_b$. A Gray encoding $\mathcal{R}_b$ preserves the connectivity of the numeric function $G(x)$ so that all points adjacent in numeric space are adjacent in Hamming space [4]. The resulting function $G(x)$ is shown in Figure 2. This image illustrates the additional roughness introduced into the topology of the function when standard Binary encoding is used.

4.1. Changing representation by shifting. It is well understood that arbitrary recodings of the problem will probably not reduce the number of local optima in a function and in the case of a well-behaved function like Rastrigin's function, it is likely to induce more local optima into the function. However, it is possible to generate small modifications to the representation of a function in a controlled fashion so that the change to the landscape is small. This type of change could be effectively used when running many search algorithms in parallel using different representations. To this end, we introduce **shifting**.

To understand shifting consider figure 3 which illustrates shifting Gray coded 3-bit strings. Consider the outer circle as a dial that can be turned counterclockwise. The inner circle contains the number to which each string is mapped. Shifting is done by turning the dial to create a new mapping. The right half of Figure 3 shows the Gray coded 3-bit strings with a single shift applied.

A shift in any Gray representation just changes the representation to a different Gray code, since the adjacency of the numbers is preserved.

Applying a shift to an encoding does not randomly shuffle the points underlying the bit encoding. However, it alters the structure of the neighborhood of a given point. Take the value 7 in Figure 3, for example. The neighbors of the value 7 have changed with a simple shift of one. By changing the neighborhood structure of a given encoding, one should expect changes to the number of local optima in the new function mapping.

The next section illustrates the effects of shifting under binary and Gray encodings for 3 multimodal test problems. We look at the previously defined Rastrigin's function as well as functions developed by Schwefel (Eq.13) and Griewangk (Eq.14) [2].

$$f(x_i \mid_{i=1,N}) = \sum_{i=1}^{N} -x_i sin(\sqrt{\mid x_i \mid}) \tag{13}$$

$$f(x_i \mid_{i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{10} (cos(x_i/\sqrt{i})) \tag{14}$$

$$x_i \in [-512, 511]$$

All three functions were analyzed for a single parameter represented using 10 bits.

4.2. Effects of shifting. In order to examine what kinds of changes can be induced by shifting the binary encoding of a function, we applied the 1024 possible shifts (for a 10-bit encoding) to all three test functions. Figure 4 shows the histograms of number of optima (assuming minimization) generated over 1024 shifts of both Binary and Gray encodings where the black bars represent the Gray encodings. Table 1 provides some statistics for the histograms in Figure 4. For all three test functions, the shifting generates fairly tight distributions which indicates that the absolute number of local optima is not radically changed. In all cases, the number of local optima for all shifts is still far from approaching the expected number of local optima under arbitrary representations. It is clear from the histograms that shifting the space does vary the number of local optima in a controlled manner. However, it says nothing about which local optima occur.

Table 1

	Numeric	Binary		Gray	
Function	#Opt	μ	σ	μ	σ
Rastrigin	11	19.69	1.21	5.26	1.25
Schwefel	8	11.38	2.57	2.93	0.91
Greiwangk	163	29.73	6.21	17.58	3.23

Table 1: Mean (μ) and standard deviations (σ) for number of optima found for 1024 shifts of three test functions.

Rastrigin's function contains a squared term that causes duplicate parameter evaluations. This means that a locally optimal value can be in-

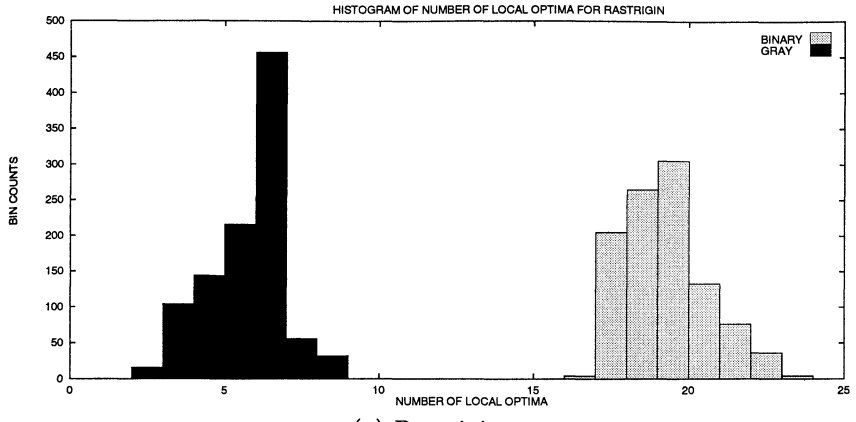
duced by two different parameter values with the exception of the parameter value zero. Rather than looking only at which locally optimal values occur under different representations, it is more informative to examine which locally optimal parameters occur under different representations. Figure 5 contains three graphs of probabilities that any parameter will be locally optimal. The first graph (a) presents the probability that any parameter will be a local optima under all possible representations. The graphs (b) and (c) represent the probability that any parameter will be an optima under any shift of Binary or Gray coding, respectively. For graphs (b) and (c), a vertical line was drawn to indicate how often each parameter occurred as a local optimum. In the Binary case, notice that 5 parameters that will occur as local optima with probability 1. However, around each of those 5 parameters is a blackened area that is due to a large number of parameters that occasionally occur as local optima. The actual number of parameters that occur as local optima in Binary space over all shifts is 346 (more than one third of the space). In Gray space, there is only 1 out of only 9 possible local optima that will always occur under all shifts. In both cases, the implication is that if multiple shifts were effectively used together, the search space could become easier.

5. Conclusion. This paper has presented several new ideas that can be used to help genetic algorithm researchers understand how encoding can affect search performance. While it has already been shown that all functions of a particular discretization have the same number of local optima [3], we can compute how many local optima will occur on average for that set of functions. We also introduced methods for determining which points are likely to be optima under all representations.

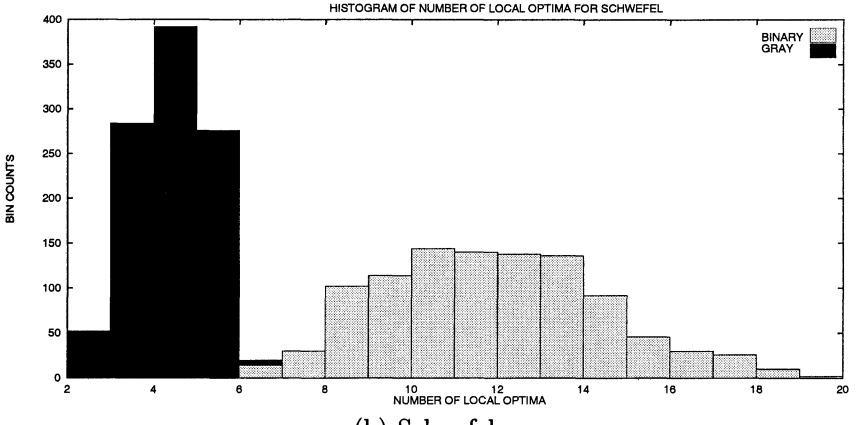
In practice, researchers do not deal with random functions. There is usually some continuity or gradient information in the function landscape. In other words, optima are often surrounded by relatively good points. When genetic algorithm researchers use a binary encoding, the complexity of the remapped function can be quite different than its numeric counterpart. We have illustrated how the landscape changes in a simple problem using two commonly used binary encodings, Standard Binary and Reflected Gray encodings.

We also introduced a new and simple mechanism for altering a problem encoding. Shifting does not drastically alter the number of local optima under an encoding but alter which points occur as local optima. We believe that using multiple shifts simultaneously in parallel may offer performance benefits.

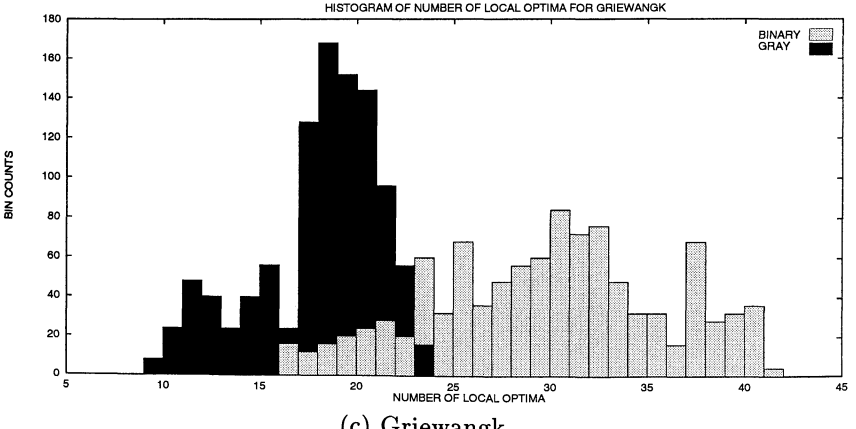
6. Acknowledgements. This research was supported by NSF grants IRI-9312748 and IRI-9503366. Soraya Rana was also supported by a National Physical Science Consortium fellowship.



(a) Rastrigin

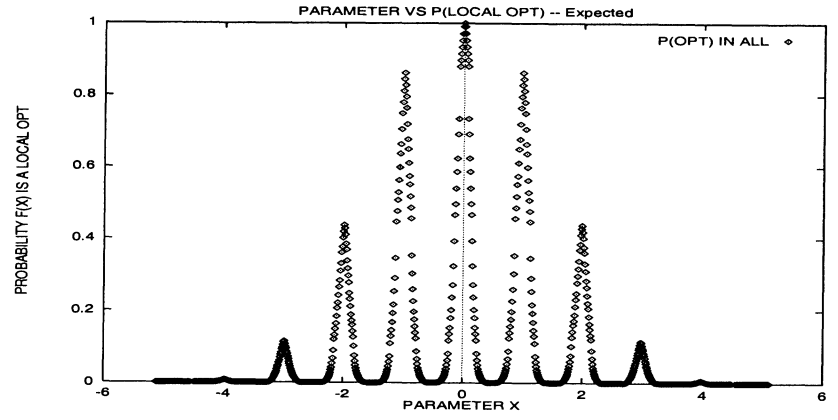


(b) Schwefel

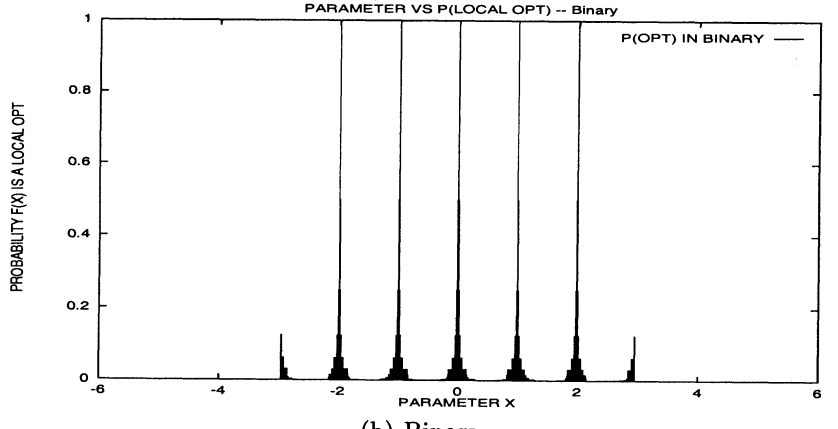


(c) Griewangk

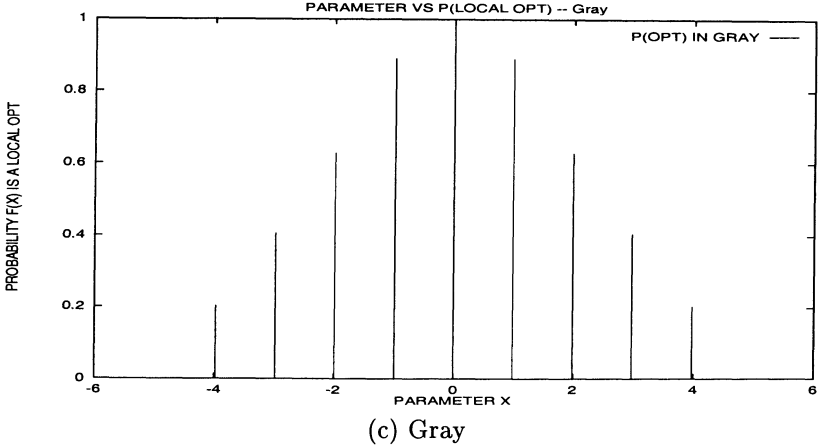
FIG. 4. *Histograms of the number of local optima found in Binary and Gray spaces*



(a) Expected



(b) Binary



(c) Gray

FIG. 5. *Probability that a parameter occurs as a local optima under an all shifts.*

REFERENCES

[1] KEITH E. MATHIAS AND L. DARRELL WHITLEY, *Changing representations during search: A comparative study of delta coding*, Journal of Evolutionary Computation, **2 (3)** pp. 249–278, 1994.

[2] H. MÜHLENBEIN, *Evolution in time and space: The parallel genetic algorithm*, In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pp. 316–337, Morgan Kaufmann, 1991.

[3] NICHOLAS J. RADCLIFFE AND PATRICK D. SURRY, *Fundamental limitations on search algorithms: Evolutionary computing in perspective*, In Jan van Leeuwen, editor, *Lecture Notes in Computer Science*, **volume 1000**, Springer-Verlag, 1995.

[4] DARRELL WHITLEY, KEITH MATHIAS, SORAYA RANA, AND JOHN DZUBERA, *Evaluating evolutionary algorithms*, Artificial Intelligence Journal, **85**, August 1996.

[5] DAVID H. WOLPERT AND WILLIAM G. MACREADY, *No free lunch theorems for search*, Technical Report SFI-TR-95-02-010, Santa Fe Institute, July 1995.

AN INVESTIGATION OF GA PERFORMANCE RESULTS FOR DIFFERENT CARDINALITY ALPHABETS

JACKIE REES* AND GARY J. KOEHLER*

Abstract. Theoretical and empirical results give mixed advice for choosing the cardinality for GA representation. Using GA models that capture the exact expected behavior of both the binary and higher cardinality cases, the determination of which representation is best for a given GA can be made. De Jong et al. and Spears and De Jong presented how the exact model for the binary genetic algorithm can give important insights to transient GA behavior. This paper uses a similar approach to study the impact of different cardinalities using the Koehler-Bhattacharyya-Vose general cardinality model.

1. Introduction. In choosing a representation for Genetic Algorithms (GAs), most practitioners use binary alphabets. This is motivated by the Principle of Minimal Alphabets [8] which states that given two possible encodings, the one with the lower cardinality alphabet gives a greater number of schema sampled by a given population. To practitioners, this means selecting the smallest alphabet that permits a natural expression of the problem.

There have been mixed result with different encodings. Some studies have suggested better performance with binary encodings while others have shown higher cardinality alphabets provide better results [3]. Antoinsse [1] argued that higher cardinality alphabets are theoretically preferable to binary encodings. Since most real world problems do not lend themselves to binary encoding, higher cardinality alphabets might prove to be necessary for these complex problems. Using GA models that capture the exact expected behavior of both the binary case and higher cardinality strings, the determination of which representation is best for a given GA can be made.

In [15], Vose presented an infinite-sized population model for simple binary Genetic Algorithms (GAs) that give their exact expected behavior over time. Similar models were provided later by T. Davis [4] and Vose and Liepins [17]. This model was extended to finite-sized populations by Nix and Vose in 1992 [12]. However all of these papers restrict GA strings to the binary case.

Some attempts have been made to generalize the Vose models to higher cardinality encodings. Bhattacharyya and Koehler [2] considered GA strings composed of components drawn from 2^v cardinality alphabets and extended the Vose model to cover this case. Their method of analysis follows the binary case fairly closely, as might be expected. Recently an exact model for the general cardinality case has been developed by Koehler, Bhattacharyya and Vose [10].

*Decision and Information Sciences, University of Florida, 351 BUS, Gainesville, FL 32611; jrees@groves.ufl.edu, koehler@nervm.nerdc.ufl.edu.

De Jong et al. [6] and Spears and De Jong [14] presented how the exact model for the binary genetic algorithm can give important insights to transient GA behavior. This paper will use a similar approach to study the impact of different cardinalities using the Koehler-Bhattacharyya-Vose general cardinality model.

In Section 2 we provide details of the form of GA model we employ, the general cardinality model that captures the exact behavior of these GAs and formulas for the various waiting-time properties of these models. (De Jong et al. [6] give detailed formulas for many waiting time properties of Markov Chains for Vose binary GA models. In section 2.3 we provide matrix-level formulas and cover the general higher cardinality GA models.)

In Section 3 we discuss our goals and various issues that impact the study of cross-cardinality comparisons of GA performance. In Section 4 we describe our experimental approach. Results are given in Section 5. We discuss these results in Section 6 and give conclusions and future directions in Section 7.

2. Notation and models.

2.1. Simple GA algorithm. There are many variants of the simple GA algorithm. For this paper we use the algorithm shown in Table 1. The models given in this paper apply to this algorithm.

2.2. Markov model of expected GA behavior. Nix and Vose [12] have derived a Markov Chain (MC) model for the binary GA which directly carries to the higher cardinality case. The MC has states that represent the possible populations. Each state $i, i \in \{0, ..., N-1\}$, indexes a vector, ϕ_i, of length $r = c^\ell$. Let e be a vector of ones of appropriate length and e' its transpose. The ϕ_i vectors are defined by

$$e'\phi_i = n,$$
$$(\phi_i)_j \in \{0, 1, ..., n\} \quad j = 0, ..., r-1$$

and

$$\phi_i \succ \phi_j \quad \text{if} \quad i < j.$$

Here $\succ$ means "lexicographically greater than." (Notation used throughout the paper is summarized in Table 2.) Nix and Vose [12] showed that

$$N = \binom{n+r-1}{r-1}$$

Let $Z' = (\phi_0, \phi_1, ...\phi_{N-1})$ be the matrix of the ϕ vectors and let B be the matrix formed from Z by

$$B_{i,j} = \delta(Z_{i,j} \neq 0)$$

where

$$\delta(x) = \begin{cases} 0 & x \text{ false} \\ 1 & x \text{ true} \end{cases}.$$

The probability of transition from population i to population j is given by [12]:

$$P_{i,j} = n! \prod_{g=0}^{r-1} \frac{q_{i,g}^{Z_{j,g}}}{Z_{j,g}!}$$

where

$$q_{i,g} = \mathcal{M}\left(\frac{F\phi_i}{e'F\phi_i}\right)_g$$

and F is a diagonal matrix having values $f(0), f(1), \ldots f(r-1)$ along the diagonal. Operator $\mathcal{M}()$ of an r dimensional vector is defined by (see [17])

$$\mathcal{M}(x) = \begin{pmatrix} (\sigma_0 x)' M \sigma_0 x \\ \cdot \\ \cdot \\ \cdot \\ (\sigma_{r-1} x)' M \sigma_{r-1} x \end{pmatrix}$$

where the permutation matrix σ_i is defined by $(\sigma_i)_{j,k} = \delta(k = i \oplus j)$. Finally, matrix M is defined by (see [16]).

$$M_{y,z} \equiv 0.5 \sum_{p \in \Omega_2} \sum_{j \in \Omega} \mu_j (\chi_p + \chi_{\overline{p}})[y \otimes p \oplus \overline{p} \otimes z \oplus j = 0]$$

where

$$\chi_p \equiv \begin{cases} \dfrac{\chi}{\ell - 1} & \text{if } p = 2^h - 1 \text{ for some } h \in (0, \ell) \\ 1 - \chi & \text{if } p = 0 \\ 0 & \text{otherwise} \end{cases}$$

and

$$\mu_i \equiv \left(\frac{\mu}{c-1}\right)^{m(i)} (1-\mu)^{\ell - m(i)}.$$

The latter two quantities are crossover and mutation masks (see [16]). Under our assumptions on μ, χ, and $f()$, the transition matrix, P, is completely ergodic.

2.3. Expected waiting times. We start our GA with an initial population, say population h. The probability that h contains string i is merely $e_h' B e_i$. Either string i is in population h or not. If state h does not contain i, then the probability of first seeing i after one transition is equal to the sum of transition probabilities from state h to states containing string i. Likewise, the probability of first seeing string i starting in state h after t transitions (i.e., after t new generations) is equal to the sum of probabilities going through $t-1$ states not containing string i to a state containing string i. Let e_i be the i^{th} unit vector and $d(x)$ be the diagonal matrix formed from any vector x. In matrix notation, the vector of first passage probabilities to string i in $t > 0$ transitions is

$$FP_{i,t} = d(e - Be_i)[d(e - Be_i)Pd(e - Be_i)]^{t-1}PBe_i. \tag{1}$$

The first term on the right has zeros on the diagonal for states containing string i and ones otherwise. The second term gives $t-1$ step transition probabilities through states not containing string i. The final term gives one-step transition probabilities to states containing string i. Clearly $FP_{i,0} = Be_i$. Let $D_i = d(e - Be_i)$. Then Equation (1) can be written more succinctly as

$$FP_{i,t} = [D_i P]^t Be_i.$$

Thus, the vector of expected first passage probabilities to states containing string i in exactly t or fewer generations is

$$\begin{aligned} SFP_{i,t} &= \sum_{k=0}^{t} [D_i P]^k Be_i \\ &= (I - [D_i P]^{t+1})(I - [D_i P])^{-1} Be_i \\ &= (I - [D_i P]^{t+1})e \end{aligned}$$

since

$$(I - [D_i P])^{-1} Be_i = e$$

and

$$\sum_{k=0}^{t} [D_i P]^k = (I - [D_i P]^{t+1})(I - [D_i P])^{-1}.$$

The expected time till string i is seen can be computed by

$$ET_i = \sum_{i=0}^{\infty} tFP_{i,t} = \sum_{i=0}^{\infty} t(D_i P)^t Be_i = (I - D_i P)^{-1} e - e.$$

Let π represent the prior probabilities for choosing population zero. Then the probability of expected first passage to states containing i in t generations or less is

$$EFP_{i,t} = \pi' SFP_{i,t} \tag{2}$$

and expected waiting time to see string i is

$$EWT_i = \pi' ET_i$$

where π is a vector of prior probabilities of choosing each state for population zero. Since the initial population is typically chosen randomly from Ω with replacement, we can use

$$\pi_j = \frac{n!}{r^n \prod_{g=0}^{r-1} (\phi_j)_g!} .$$

If the initial population is chosen randomly without replacement we have

$$\pi_j = \frac{\delta(\phi_j \leq e)}{\binom{r}{n}} .$$

3. Goals and issues. Our overall goal is to explore the theoretical expected performance of a Simple GA using different cardinality representations of the original problem. This objective is made more explicit in the following subsections.

3.1. Comparing performance. To determine the effect of cardinality on performance, we will compare the expected waiting time to an optimal string under various cardinality representations of some chosen problems.

Other measures of performance are possible. In practice, one measure often used is the number of populations before an optimal solution first arises. Theoretically, this statistic is completely characterized by the distribution of first passage probabilities to optimal strings. The weighted time to the first appearance of an optimal string using this distribution is just the expected waiting-time measure we propose.

Another approach used in practice is to compare the average population fitnesses of two (or more) approaches. However, this measure doesn't give clear guidelines to performance since these curves seldom dominate each other.

An advantage of using expected waiting time is that this is a scalar measure of a GA's performance and is easy to use in comparisons. In any case, there are several issues we need to address when varying the cardinality of a GA representation and these are covered below.

3.2. Representation. Given a search problem, the representation of the search strings may not lend itself to arbitrary cardinalities. For example, suppose we have an objective function $g(x_1, x_2)$ where $x_i \in \{0, 1, ..., 80\}$. This problem can be naturally represented by using

- $c = 3$ and $\ell = 6$ or by
- $c = 9$ and $\ell = 4$ or by
- $c = 81$ and $\ell = 2$.

(Using $c = 6561$ would not be permitted under our GA since its string length would be 1.) In each case, the original x values can be retrieved from the integer value of a string, s, by

$$x_2 = \left\lfloor \frac{s}{81} \right\rfloor$$

(where $\lfloor x \rfloor$ and $\lceil x \rceil$ are the floor and ceiling of x, respectively) and

$$x_1 = s - 81x_2.$$

A natural fitness function might be

$$f(s) = g(x_1, x_2) = g\left(s - 81\left\lfloor \frac{s}{81} \right\rfloor, \left\lfloor \frac{s}{81} \right\rfloor\right).$$

This problem cannot be represented "naturally" using binary or any other cardinality. To represent this problem using a binary representation, a number of strategies can be invoked.

One strategy is to penalize the fitness values of infeasible strings. For example, if we use a binary representation of $x_i \in \{0, 1, ..., 80\}$ (say with $\ell = 13$) then $1,631$ of the $8,192$ possible strings don't conform to any valid string in the original problem range. These could receive a very small fitness value in comparison to valid values of $g(x_1, x_2)$. Possible drawbacks are the distortion in selection pressures brought about by different penalty structures or the different ways that the $1,631$ invalid strings can be chosen and their impact on reproduction.

Another strategy is to stretch the original fitness function to cover the whole range of the representation. For example, using a binary representation with $\ell = 14$ given the original ranges of $x_i \in \{0, 1, ..., 80\}$, we might have

$$x_2 = \left\lceil \frac{80}{127} \left\lfloor \frac{s}{128} \right\rfloor \right\rceil$$

and

$$x_1 = \left\lceil \frac{80}{127} \left(s - 128 \left\lfloor \frac{s}{128} \right\rfloor \right) \right\rceil.$$

However, this method has drawbacks. Some (x_1, x_2) points will be mapped into more frequently than others. For example, $s = 13$ and 14 both map to $(0, 9)$ but only $s = 15$ maps to $(0, 10)$.

In this paper we will choose problems that can be represented by a number of natural cardinalities.

3.3. Mutation and crossover rates. If we wish to compare the performance impact of using different cardinalities, we need to control the impact of other parameters on the performance measure. Two parameters under our control are the mutation and crossover rates.

It seems reasonable to assume that a mutation rate used at one cardinality may not be appropriate for a different cardinality. For example, say we use a mutation rate of 0.01 for a binary problem of string length 4 and for a problem with cardinality $c = 4$ and string length 2. The probabilities of mutating to different domain values would be different. For example, the probabilities of mutating to various strings starting from string s having an integer value of 13 (1101 in binary and 31 with $c = 4$) would be:

Mutate $s = 13$ to	Binary Probability	$c = 4$ Probability
0	0.00000099	1.11111E-05
1	0.00009801	0.0033
2	0.00000001	1.11111E-05
3	0.00000099	1.11111E-05
4	0.00009801	1.11111E-05
5	0.00970299	0.0033
6	0.00000099	1.11111E-05
7	0.00009801	1.11111E-05
8	0.00009801	0.0033
9	0.00970299	0.9801
10	0.00000099	0.0033
11	0.00009801	0.0033
12	0.00970299	1.11111E-05
13	0.96059601	0.0033
14	0.00009801	1.11111E-05
15	0.00970299	1.11111E-05

There is no simple way to keep these probabilities equal.

Now consider crossover. The crossover rate merely specifies the probability that two strings will mate. The same rate used for one cardinality appears to have the same impact as another cardinality. Any two strings will face the same probability of mating. However, given two strings are to mate, there will be fewer crossover points for the higher cardinality string, which also impacts the overall reproduction probabilities.

In order to balance these factors, we will use a strategy given by De Jong et al. [6] and compare the expected waiting times of the different cardinalities evaluated at the mutation and crossover rates that minimize this value. In other words, we will compare the best realization of expected waiting times.

3.4. Sampling of the initial population. The initial population of a GA is typically drawn randomly from Ω with replacement. However, more diversity (at a slightly higher initial computation cost) can be obtained by

sampling without replacement. We will compute expected waiting times under both initial conditions.

3.5. Multiple optima. If there are multiple optimal strings, the expected waiting time to any of them can be computed by a simple variation of the formulas given in Section 2.3. However, we choose functions having a unique global optimum (with, perhaps, local optima) to push our analysis to the extreme.

3.6. Computational limitations. Computing expected waiting times requires computing and using the transition matrix P. Unfortunately, P is N by N and N can be very large. For example, the De Jong test function F1 [5] has three variables each with a range of 1024 values. A binary encoding would require a 30 bit string resulting in

$$N = \begin{pmatrix} n + 2^{30} - 1 \\ 2^{30} - 1 \end{pmatrix}$$

which is over 10^{17}. At this point, practical considerations limit us to exploring problem/population sizes having small r and n values.

4. Experimental design.

4.1. Hypotheses. The main hypothesis we seek to test is the Principle of Minimal Alphabets. An implication of this principle is the following:

Hypothesis 1: *(Principle of Minimal Alphabets)*
The expected waiting time to see the optimal string should be minimal for the smallest natural cardinality representation.

A slightly stronger hypothesis will also be tested:

Hypothesis 2:
The expected waiting time to see the optimal string should be non-increasing with decreasing cardinality.

We also seek to test a few other hypotheses. Since choosing an initial population by randomly sampling strings without replacement gives more diversity, one might expect faster discovery of an optimal string. This leads to:

Hypothesis 3:
The expected waiting time to see the optimal string under sampling with replacement should not be lower than that obtained with sampling without replacement.

Along similar lines, increasing the population size should also decrease the expected time to see an optimal solution. This gives:

Hypothesis 4:

The expected waiting time to see the optimal string should be non-increasing in increasing population size.

4.2. Test bank. We have chosen four test functions to explore (see Table 3). The first two are restricted versions of De Jong's F1 and F8 functions [5]. The third is a fully deceptive trap function for binary problems [7] and the fourth is a higher cardinality deceptive function [11].

4.2.1. De Jong's F1 (Modified). De Jong's F1 is simplified to one variable and converted to a maximization form having all positive values. The original function is

$$f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 \qquad -5.12 \leq x_i \leq 5.11.$$

We first restrict the function to one variable with a range of $100x \in \left[-\frac{r}{2}, \frac{r}{2}\right)$. To convert it to a maximization problem we subtract it from its maximum (plus a small number to keep it positive). Thus we get

$$f(x) = \frac{4 + r^2}{40000} - \frac{x^2}{10000}.$$

Converting this to a function of a string value requires shifting s from a range of $s \in [0, r)$ to x's range. Thus, our modified F1 function is

$$f(x) = \frac{4 + r^2}{40000} - \frac{\left(s - \frac{r}{2}\right)^2}{10000} = \frac{1 - s^2 + rs}{10000}.$$

This function has a unique maximum at $s = r/2$ and no local maxima.

4.2.2. De Jong's F8 (modified). De Jong's F8 is simplified to one variable and converted to a maximization form having all positive values. The original function is

$$f(x_1, ..., x_k) = 1 + \sum_{i=1}^{k} \frac{x_i^2}{4000} - \prod_{i=1}^{k} \cos\left(x_i / \sqrt{i}\right) \qquad -512 \leq x_i \leq 511.$$

Performing the various steps yield

$$f(s) = 1 - \frac{s^2 - rx}{4000} + \cos\left(s - r/2\right).$$

This fitness function has a unique maximum at $s = r/2$ and many local maxima.

4.2.3. Trap function. Our third function is a trap function [7]. These are defined for binary problems and are given by

$$f(s) = 1 + \begin{cases} \frac{a}{z}(z - m(s)) & m(s) \leq z \\ \frac{b}{\ell - z}(m(s) - z) & \text{otherwise} \end{cases}$$

where a and b are positive and z is chosen under various criteria. We use $z = \ell - 1$, $a = 10$ and $b = 12$. These choices make $f(s)$ fully deceptive [7].

4.2.4. Mason's higher cardinality deceptive function. Mason [11] presented a method for constructing higher cardinality deceptive functions of any order. One chooses the maximal point and order of deception desired and then works through an algorithm constructing fitnesses of successively higher order schemata. We constructed this function for $c = 4$, $\ell = 2$ and $i^* = 13$. This same function was used over the binary strings having the same integer value.

4.3. Experiments. For each test function, we explore two natural cardinalities. For each cardinality we ran 2 experiments by having the population size vary from 1 to 2. For each of these we will obtain two expected times - one with the initial population generated randomly without replacement and one with the initial population generated randomly with replacement. For each data point, we performed the following enumerative search to determine the best expected waiting time:

Vary χ from 0.01 to 1.0 by step sizes of 0.01 and vary μ from 0.01 to $1/c$ in step sizes of 0.01 and then compute the expected waiting time to the optimal string to find the minimum expected waiting time.

5. Results. Tables 4, 5, 6 and 7 contain the best expected waiting times from each of our runs. The blocks marked by "with (w/o)" mean expected waiting times where the initial population is generated with (with out) replacement.

6. Discussion of results.

6.1. Fitness invariance. Notice in Tables 4-7 that the expected waiting times for populations of size one are independent of the fitness function, cardinality and string size. The explanation is as follows. When $n = 1$

$$q_{i,g} = \mathcal{M}\left(\frac{F\phi_i}{e'F\phi_i}\right)_g = \mathcal{M}(e_i)_g$$

and no fitness information remains.

Also, expected waiting times for binary cardinality problems with populations of size 2 seem invariant with the fitness function. A partial explanation is as follows. In all cases, the minimum expected waiting time was found at $\chi = 1$ and $\mu = 0.5$. At these values

$$M = \frac{1}{2^\ell} ee'$$

and

$$q_{i,g} = \mathcal{M}\left(\frac{F\phi_i}{e'F\phi_i}\right)_g = \frac{1}{2^\ell}\frac{\phi_i F}{e'F\phi_i}\sigma_g ee'\sigma_g\frac{F\phi_i}{e'F\phi_i} = \frac{1}{2^\ell}.$$

So here also all fitness information is removed. Why the minimum expected waiting time was found at $\chi = 1$ and $\mu = 0.5$ remains unknown.

6.2. Hypotheses. Expected waiting times all increased with increased cardinality so we cannot reject Hypothesis 1, the principle of minimal alphabets. The same conclusion hold for the stronger statement in Hypothesis 2.

In all cases, the expected waiting times for sampling of the initial population without replacement were less than or equal to that sampled with replacement so we cannot reject Hypothesis 3. Sampling without replacement gives a greater initial diversity which, at least in these small population sizes, appears to give a slight edge to the search.

In all cases, the expected waiting time to see the optimal solution decreased with increased population size, so we cannot reject Hypothesis 4.

In sum, all our hypotheses seem supported by the experiments.

6.3. Other observations. Two additional points were observed in our runs. The first was that the results for population size one were invariant with respects to the crossover rate, as one might suspect. Second, the expected waiting times were decreasing in increasing crossover and mutation rates. This latter result was not expected.

7. Summary and future directions. In this paper we computed the expected waiting time to see an optimal string for four different test functions under varying cardinalities and population sizes. In all cases, our original hypotheses were supported. In particular, the observed expected waiting times increased with cardinality (consistent with the principle of minimal alphabets) and decreased with population size. Furthermore, the expected waiting times were lower when the initial population was sampled without replacement. Finally, the expected waiting times were all decreasing in increasing crossover and mutation rates.

This study can be extended in many ways. First, larger population sizes, larger string sizes and more families of natural cardinalities should be studied. All of these extensions yield heavy computational demands which need to be solved. One possible solution is to use iterative procedures (see [9] for a start on this problem) to compute the expected waiting times. Also, knowing the optimal mutation and crossover rates can reduce the overall computational effort dramatically. Even using some sort of gradient search instead of an enumeration may yield reasonable computational demands.

Another area that could be explored would be how to handle problems not naturally represented by an alphabet. We outlined two procedures to deal with this issue but haven't explored their impact on expected waiting times.

All of our test problems had a unique optimal solution. Another extension would be to study problems with multiple optimal strings.

TABLE 1
The Simple Genetic Algorithm with One-Point Crossover and Uniform Mutation.

Algorithm 1: *(Genetic Algorithm)*

Given:

String length $\ell \geq 2$, cardinality $c \geq 2$, fitness function $f()$, mutation rate $\mu \in (0, 1/c)$, one-point crossover rate $\chi \in (0, 1]$ and population size $n \geq 1$.

Initialization: Generate an initial population 0. This is usually done by randomly drawing n strings from Ω with (or without) replacement.

Step 1: Form a new population as follows. Repeat the following steps until the new population has n members.

(A) Randomly choose two members from the old population according to their fitness values relative to the sum of all fitness values of the current population. These are called parent strings.

(B) Let RAN(0,1) give a uniform random number between zero and one. If RAN(0,1) $\leq \chi$, derive two new strings by a random single-point crossover. Randomly choose one to add to the new population. Otherwise, randomly choose one of the parents to add to the new population.

(C) Perform mutation on the new population member. For each of the ℓ digits, if RAN(0,1) $\leq \mu$ randomly replace the digit by one of the remaining c-1 digits.

Step 2: If stopping conditions are not met, return to Step 1.

TABLE 2
Summary of Notation.

c	cardinality of string alphabet
$\oplus$	$a \oplus b = (a + b) \bmod c$. When c is 2, $\oplus$ is the bitwise exclusive-or operator.
$\otimes$	$a \otimes b = (ab) \bmod c$. When c is 2 $\otimes$ is the bitwise and operator.
$\overline{k}$	complement of k defined by $k \oplus \overline{k} = 1$
$m(x)$	number of nonzero digits of string x.
μ	uniform mutation rate with $\chi \in (0, 1/c)$.
χ	one-point crossover rate with $\chi \in (0, 1]$.
μ_i, χ_p	mutation and crossover masks, respectively.
ℓ	string length.
r	number of strings, $r\ c'$.
Ω	set of all strings, $\Omega = \{0, 1, ..., r-1\}$. $\Omega_2 = \{0, 1, ..., 2^\ell - 1\}$
n	population size.
x'	transpose of vector/matrix x.
ϕ_i	r-vector of string frequencies in population i.
N	number of possible populations.
Z	$N \times r$ matrix of ϕ_i vectors. $Z' = [\phi_0, \phi_1, ..., \phi_{N-1}]$.
B	$N \times r$ matrix formed by setting nonzero components of Z to one.
M	$r \times r$ mixing matrix.
$\delta(x)$	is 0 if x is false, 1 otherwise.
σ_i	i^{th} permutation matrix with $(\sigma_i)_{j,k} = \delta(k = i \oplus j)$.
$\mathcal{M}()$	r-vector recombination operator.
$f(i)$	fitness of string i.
F	focusing matrix, a diagonal matrix with $F_{i,i} = f(i)$.
P	transition matrix over state space of populations.
$d(x)$	a diagonal matrix formed using vector x as the diagonal.
e, e_i	vector of ones and i^{th} unit vector, respectively.
D_i	$d(e - Be_i)$.
π	N-vector of prior probabilities over the population states.
$FP_{i,t}$	expected first passage probabilities to string i in exactly t generations.
$SFP_{i,t}$	probability of expected first passage to states containing i in t generations.
ET_t	vector of expected times to see string i.
EWT_t	expected waiting time to see string i.

TABLE 3
Test Functions.

Function		Optimal String	r	Cardinalities
De Jong $F1$	$f(x) = \frac{1 - s^2 + rs}{10000}$	$r/2$	16	2, 4
De Jong $F8$	$f(s) = 1 - \frac{s^2 - rs}{4000} + \cos(s - r/2)$	$r/2$	16	2, 4
Trap Function	$f(s) = 1 + \begin{cases} \frac{a}{z}(z - m(s)) & m(s) \leq z \\ \frac{b}{\ell - z}(m(s) - z) & otherwise \end{cases}$	$r - 1$	16	2, 4
Mason Function	$c = 4$ $\ell = 2$ $i^* = 13$	13	16	2, 4

TABLE 4
Expected Waiting Times for De Jong's F1.

n	$\ell = 4, c = 2$	$\ell = 2, c = 4$
1	with 15.00	with 34.20
	w/0 15.00	w/0 34.20
2	with 7.25806	with 14.9161
	w/o 7.22581	w/o 14.8313

TABLE 5
Expected Waiting Times for De Jong's F8.

n	$\ell = 4, c = 2$	$\ell = 2, c = 4$
1	with 15.00	with 34.20
	w/o 15.00	w/o 34.20
2	with 7.25806	w/o 21.9628
	w/o 7.22581	w/o 21.8864

TABLE 6
Expected Waiting Times for the Trap Function.

n	$\ell = 4, c = 2$	$\ell = 2, c = 4$
1	with 15.00	with 34.20
	w/o 15.00	w/o 34.20
2	with 7.25806	with 10.8265
	w/o 7.22581	w/o 10.7381

TABLE 7
Expected Waiting Times for the Mason Function.

n	$\ell = 4, c = 2$	$\ell = 2, c = 4$
1	with 15.00	with 34.20
	w/o 15.00	w/o 34.20
2	with 7.25806	with 35.1026
	w/o 7.22581	w/o 35.0329

REFERENCES

[1] ANTONISSE, J., *A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint*, Proceedings of the Third International Conference on Genetic Algorithms, Ed. J.D. Schaffer, Morgan Kaufmann, San Francisco, 1989, pp. 86–97.

[2] BHATTACHARYYA, S. AND G.J. KOEHLER, *An Analysis of Genetic Algorithms of Cardinality* 2^V, Complex Systems, 8, 1994, pp. 227–256.

[3] DAVIS, L., *Handbook of Genetic Algorithms*, 1991, Van Nostrand Reinhold, New York.

[4] DAVIS, T., *Toward an Extrapolation of the Simulated Annealing Convergence Theory onto the Simple Genetic Algorithm*, 1991, Dissertation, University of Florida.

[5] DE JONG, K., *An analysis of the Behavior of a Class of Genetic Adaptive Sytems*, PhD Thesis, 1975, University of Michigan, Department of Computer and Communication Sciences, Ann Arbor, MI.

[6] DE JONG, K.A., W.M. SPEARS AND D.F. GORDON, *Using Markov Chains to Analyze GAFOs*, in *Foundations of Genetic Algorithms*, 3, 1995, Morgan Kaufmann, San Francisco, pp. 115–137.

[7] DEB, K. AND D.E. GOLDBERG, *Analyzing Deception in Trap Functions*, Foundations of Genetic Algorithms 2, Ed. L. Darrell Whitley, 1993, Morgan Kaufman, San Francisco, pp. 93–108.

[8] HOLLAND, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[9] KOEHLER, G.J., *Computing Simple GA Expected Waiting Times*, in review, January, 1997, College of Business, BUS 351, University of Florida, Gainesville, FL 32611.

[10] KOEHLER, G.J., S. BHATTACHARYYA, AND M. VOSE, *General Cardinality Genetic Algorithms*, in review, No. 95-06-101, College of Business, BUS 351, University of Florida, Gainesville, FL 32611.

[11] MASON, A.J., *Partition Coefficients, Static Deception and Deceptive Problems for Non-Binary Alphabets*, Proceedings of the Fourth International Conference on Genetic Algorithms, Ed. R.K. Belew and L.B. Booker, Morgan Kaufmann, San Francisco, 1991, pp. 210–214.

[12] NIX, A. AND M.D. VOSE, *Modeling Genetic Algorithms with Markov Chains*, Annals of Mathematics and Artificial Intelligence, 5, 1992, pp. 79–88.

[13] SHAFFER, J.D., *Some Experiments in Machine Learning using Vector Evaluated Genetic Algorithms*, Dissertation, 1984, CS Department, Vanderbilt University, Nashville, TN.

[14] SPEARS, W.M. AND K.A. DE JONG, *Analyzing GAs using Markov Models with Semantically Ordered and Lumped States*, forthcoming Foundations of Genetic Algorithms 4, 1996.

[15] VOSE, M.D., *Formalizing Genetic Algorithms*, Proceedings of the IEEE Workshop on Genetic Algorithms, Neural Networks, and Simulated Annealing Applied to Signal and Image Processing, Glasgow, Scotland, May, 1990.

[16] VOSE, M.D., *The Simple Genetic Algorithm: Foundations and Theory*, forthcoming, MIT Press, Cambridge, MA.

[17] VOSE, M.D. AND G.E. LIEPINS, *Punctuated Equilibria in Genetic Search*, Complex Systems, 5, 1991, pp. 31–44.

GENETIC ALGORITHMS AND THE DESIGN OF EXPERIMENTS

COLIN R. REEVES* AND CHRISTINE C. WRIGHT†

Abstract. The genetic algorithm (GA) has most often been viewed from a biological perspective. The metaphors of natural selection, cross-breeding and mutation have been helpful in providing a framework in which to explain how and why they work. However, most practical applications of GAs are in the context of optimization, where alternative approaches may prove more effective. In attempting to understand how GAs function as optimizers, several alternative viewpoints have been suggested. In this paper we discuss one of these in some detail—one in which GAs are regarded as a form of sequential experimental design.

The first section of this paper is expository, where we review some of the basic concepts of experimental design as introduced in earlier papers [1, 2] and show how they relate to GAs. We then show how these ideas can be used to investigate the question of how epistasis can be measured, and demonstrate the fact that not all epistasis in necessarily harmful. Using ED concepts we are also able to show that in general it is not possible to determine *a priori* that a given problem instance is likely to be hard or easy to optimize by means of a GA. Finally, we show how the concept of an *orthogonal* ED can be used to define a lower limit in terms of the necessary amount of computation required to solve an optimization problem.

In the second part, we describe some empirical work in which GAs are compared with ED-based methods. In a typical designed experiment, a fraction of the Universe of all possible solutions is chosen in a way that is thought to enable the identification of important factors (genes in traditional GA terminology) and their associated levels (alleles) in a candidate optimal solution. Generally, this is only the first stage; further tests must be made in order to confirm or reject these tentative conclusions. Sometimes, further experiments are also carried out to refine the values of some of the less important factors.

However, one of the problems with the use of ED for optimization is the sheer amount of human effort needed in order to apply such techniques in this context. For this reason, ED has usually been restricted to fairly small problems (by GA standards). Recently, some suggestions have been made for increasing the range of problems to which ED ideas can be applied by automating some of the decisions customarily made by the experimenter. A GA is also a means for conducting experiments, testing hypotheses (at least implicitly), and identifying candidate optimal solutions. It thus seems appropriate to compare the performance of a GA with ED-based approaches.

We examine an engineering design problem in which one version of the ED-based approach outperformed a GA, and also describe some experiments on artificial test problems of bounded epistasis in which a GA appears much better able to cope with higher-order interactions than ED-based methods.

In conclusion, we discuss the similarities and differences between GAs and ED revealed in the course of this research. One difference in focus is that ED has traditionally been concerned, not merely in finding the best solution, but also in explaining this result in terms of a model. Statisticians have previously seen this as a rather intractable problem for a GA, but recent developments in computer algebra may provide a means for resolving this issue.

*School of Mathematical and Information Sciences, Coventry University, UK. Email: C.Reeves@coventry.ac.uk

†School of Mathematical and Information Sciences, Coventry University, UK.

1. Introduction. ¿From their introduction by Holland [3] and popularization by Goldberg [4], most accounts of genetic algorithms (GAs) explain their behaviour by making reference to the biological metaphor of evolutionary adaptation. While this is helpful in providing an understanding of how a GA works in a qualitative way, in the context of optimization-which is probably the main context for most applications of GAs-it creates something of a puzzle. DeJong, whose original work [5] initiated the application of GAs to optimization problems, has recently [6] argued that for the purposes of optimization, the original GA has been changed to a point where it is almost unrecognizable, and should not be given the same name. He prefers the term GA-based function optimizer (GAFO). However, the GA community in general has not adopted this terminology.

It is also fair to ask, in the particular context of optimization, why it should be thought useful to rely on so inefficient a process as evolution. Evolution is supposed to have needed millions of years to produce its purported effects-a timescale which does not sound encouraging for the development of efficient optimizers! Further, some scientists (see, for example Behe [7], Denton [8, 9]) are starting to doubt whether evolutionary theory on its own is an adequate explanation for the world around us.

It is clear that Holland recognized the need for 'biological' explanations to be reinforced with something which was more firmly grounded. The approach he followed was founded on the now well-known concept of a *schema*, which focuses not on the individual strings, but on particular subsets of strings-the *schemata* or hyperplanes.

The first of Holland's ideas is that of *intrinsic* (or *implicit*) parallelism-the notion that information on many schemata can be processed in parallel. Under certain conditions on population size and schema characteristics, Holland estimated that a population of size M contains information on $\mathcal{O}(M^3)$ schemata. The second concept is expressed by the so-called *Schema Theorem*, in which Holland showed that if there are $N(S,t)$ instances of a given schema S in the current population, then in the next population (following reproduction), the expected number of instances can be bounded by

$$E[N(S,t+1)] \geq \frac{F(S,t)}{\bar{F}(t)} N(S,t)\{1-\epsilon(S,t)\}$$

where $F(S,t)$ is the fitness of schema S, $\bar{F}(t)$ is the average fitness of the population, and $\epsilon(S,t)$ is a term which reflects the potential for genetic operators to destroy instances of schema S.

It follows from this theorem that certain schemata that are resistant to destruction by crossover and/or mutation, and that are fitter than the average of the current population, are likely to increase their presence in the next population. Any attempt to extrapolate this result for more than one generation is doomed to failure because the terms in the theorem are then not all independent of what is happening in the rest of the population.

Holland also attempted to model schema processing (or hyperplane competitions) by means of an analogy to stochastic two-armed bandit problems. This is a well-known statistical problem: we are given two 'levers' which if pulled give 'payoff' values according to different probability distributions. The problem is to use the results of previous pulls in order to maximize the overall future expected payoff. In [3] it is argued that a GA approximates an 'optimal' strategy which allocates an increasing number of trials to the observed better lever; this is then used to contend for the supposed efficiency of a GA in distinguishing between competing schemata or hyperplanes.

The third assumption implicit in the explanation of GA performance is that the re-combination of small pieces of the genotype ('good' schemata) into bigger pieces is indeed a sensible method of finding optimal solutions. Goldberg [4] calls this the 'building-block' hypothesis (BBH), and a substantial part of recent theoretical research into GAs has centred on the question of the circumstances under which this hypothesis fails.

1.1. Criticisms of schema-based theory. Several powerful criticisms have been made of schema-based theory. Mühlenbein [10], for example, echoing criticisms of the neo-Darwinian theory of evolution itself, has argued that

> ...the Schema Theorem is almost a tautology, only describing proportional selection...

while Radcliffe [11] and others have pointed out that the Theorem extends to arbitrary subsets of the search space. Intrinsic parallelism is recognized to be of strictly limited application; it merely describes the number of schemata which are likely to be present in some numbers given certain assumptions about string length, population size and (most importantly) the way in which the population has been generated-the last assumption being one which is unlikely to be true except at a very early stage of the search. The two-armed bandit analogy also breaks down in at least two ways: firstly, Macready and Wolpert [12] have recently argued that there is no reason to believe that the strategy described by Holland as approximated by a GA is an optimal one. (They also believe there is in any case a flaw in Holland's mathematics.) Secondly, as mentioned in [2] and expanded upon later in this paper, the assumption that hyperplane competitions can be isolated and 'solved' independently is false, unless the population has a very particular composition.

The BBH has also come in for some examination. Goldberg [13] introduced the idea of a 'deceptive' function-one which would lead the GA in the 'wrong' direction. In other words, the 'building blocks' needed for a successful optimization were not present, and, as expected, GAs can find such problems very hard to solve [14], although the extent of the difficulty also depends on the amount of deception [15]. This is however mainly 'negative' evidence for the BBH; Mitchell *et al.* [16] described some experiences with

the so-called 'Royal Road' function-a function that was meant to demonstrate the positive side of the BBH, in that it provided an environment for a GA to build an optimizing solution step by step, in the manner in which a GA is thought to behave. It was somewhat disconcerting to find that the GA was comprehensively outperformed by a simple hill-climber-on a function which should have been ideal for the supposed *modus operandi* of a genetic algorithm.

2. Alternative viewpoints. In the light of some of the above observations, it is not surprizing that several authors have suggested alternative ways in which to examine the behaviour of a genetic algorithm. Given the stochastic nature of a GA, it is also not surprizing that many of these perspectives are statistical in nature.

Whitley [17] has described an 'executable' model based on iterating an exact version of the Schema Theorem which predicts exactly the expected representation of a given string in a population. This considerably eases the problem of experimentally investigating how the behaviour of a GA depends on different GA parameters, but is only computationally practicable for short strings and comparatively small populations. Vose [18] has described a Markov chain approach that uses a similar model; some interesting steady-state results have been obtained relating, for instance, to the influence of mutation [19], and to the stability of fixed points (i.e. populations) in an infinite population GA [20].

For the case of finite population GAs, De Jong *et al.* [21] have examined absorption probabilities, waiting-times and other traditional Markov chain performance measures. However, once again, computational requirements mean that only very small populations of very short strings can be analyzed by these means.

Another perspective is that adopted by Davidor [22, 23] who tried to identify statistical properties of *functions* that would make them suitable (or otherwise) for optimization by a GA, assuming the usual concepts of schema-processing. Reeves and Wright [1, 2] have shown this is equivalent to viewing GAs as a form of statistical experimental design, and it is this viewpoint which will be explained in what follows.

3. Experimental design. The starting point for this perspective on GAs was Davidor's [22, 23] attempt to analyse epistasis in GAs. (Epistasis is a term used in genetics to denote the fact that the expression of a chromosome is not merely a linear function of the effects of its individual alleles.) Subsequently, it was realized that there is also a close connection to the concept of Walsh function analysis, as promoted in papers by Goldberg [13, 24]. A more complete version of the ideas we now develop is contained in Reeves and Wright [1].

The central idea here is the assumption of an underlying linear model (defined on the bits) for the fitness of each string. In conventional experimental design this model would be written in parametric form, in terms of

'main effects' and 'interactions'. For instance, the model for a string of 3 components could be written as follows:

$$v_{pqrs} = \mu + \alpha_p + \beta_q + \gamma_r + (\alpha\beta)_{pq} + (\alpha\gamma)_{pr} + (\beta\gamma)_{qr} + (\alpha\beta\gamma)_{pqr} + \varepsilon_{pqrs} \tag{3.1}$$

where v_{pqrs} is the fitness of the string (p, q, r), and the subscript s denotes the replication number (i.e. the s^{th} occurrence of the string). If there is no random error, we can of course drop the final term, and indeed the subscript s altogether. The parameters on the right-hand side are as follows:

μ average fitness
α_p effect of allele p at gene 1
β_q effect of allele q at gene 2
γ_r effect of allele r at gene 3
$(\alpha\beta)_{pq}$ joint effect of allele p at gene 1 and allele q at gene 2
$(\alpha\gamma)_{pr}$ joint effect of allele p at gene 1 and allele r at gene 3
$(\beta\gamma)_{qr}$ joint effect of allele q at gene 2 and allele r at gene 3
$(\alpha\beta\gamma)_{pqr}$ joint effect of allele p at gene 1, allele q at gene 2 and allele r at gene 3
ε_{pqrs} random error for replication s of string (p, q, r)

It is common in most GA applications to assume zero random error, although this is not essential. More importantly, a full analysis assumes knowledge of the fitness of every string in the Universe. In practice this is unrealistic-in reality we only know the fitness of strings in a subset of the Universe (i.e. our 'population', to use the conventional GA terminology, is in statistical terms merely a sample). Of course, in the case where the Universe is known, there is in one sense no problem: the optimal combination is obvious! Nevertheless, in order to understand some principles, it is necessary to assume complete knowledge of the Universe.

3.1. An example. Suppose we have a 3-bit string, and the fitness of every string in the Universe is known. There are of course $2^3 = 8$ strings, and therefore 8 fitness values, but the experimental design model above has 27 parameters. It is thus essential to impose some side conditions if these parameters are to be estimated; the usual ones are the obvious constraints that at every order of interaction, the parameters sum to zero for each subscript. This results in an additional 19 independent relationships and thus allows the 'solution' of the above model-in the sense that all the parameter values can be determined if we have observed every one of the 8 possible strings. For example, we find that

$$\begin{aligned} \mu &= v_{***} \\ \mu + \alpha_p &= v_{p**} \quad \text{for } p = 0, 1 \\ \mu + \beta_q &= v_{*q*} \quad \text{for } q = 0, 1 \\ \mu + \gamma_r &= v_{**r} \quad \text{for } r = 0, 1 \end{aligned}$$

where the notation v_{p**}, for instance, means averaging over subscripts q and r. The effects are identical to Davidor's proposed 'excess allele values' [22, 23] for measuring epistasis.

3.2. Connections to Walsh functions. Davidor's linear decomposition leads to a set of coefficients which are equivalent to the standard linear model of experimental design. Another linear decomposition which is often used in the analysis of GAs in a binary representation is the *Walsh transform*, first introduced by Bethke [25], and made more widely known by Goldberg [24, 13]. More recently, Mason [26] has defined the concept of a *partition coefficient* as a generalization of the Walsh coefficients for non-binary strings. From Mason's definition it is clear that these coefficients are just the 'effects' defined in the ED context, while his subsequent results are simply a derivation of the side constraints as outlined above. It further follows from this that the Walsh transform decomposition is also equivalent to that of experimental design.

However, it is instructive to examine the relationship between Walsh transform analysis and experimental design rather more closely. In Walsh transform analysis, the bits are usually numbered from right to left, so for the moment we shall adopt the same convention. The Walsh monomials are defined on the string positions $\{y_i\}$ coded for convenience as $+1$ or -1 rather than the usual 0 or 1:

$$\psi_j(y) = \Pi_{i=1}^{l}(y_i)^{j_i}$$

where j_i is the i^{th} bit (counting from the right) in the binary representation of the number j. The Walsh function representation of the fitness v is

$$v(y) = \sum_{j=0}^{2^l-1} w_j \psi_j(y)$$

where y encodes the bit positions as above. There are clearly the same number of independent coefficients in the ED decomposition as there are Walsh coefficients, so it is natural to ask how they are related.

The relationship is clearly illustrated in a 3-bit example. The Walsh coefficients can be found from the fitness averages for different schemata:

$$\begin{aligned} v_{***} &= w_0 \\ v_{**0} &= w_0 + w_1 \\ v_{**1} &= w_0 - w_1 \quad \text{etc} \end{aligned}$$

whereas from the experimental design viewpoint, we have

$$\begin{aligned} v_{***} &= \mu \\ v_{**0} &= \mu + \alpha_0 \\ v_{**1} &= \mu + \alpha_1 \quad \text{etc.} \end{aligned}$$

If we write out the full set of equations, we find that

$$\mu = w_0$$

$$\begin{aligned}
\alpha_i &= (-1)^i w_1 \\
\beta_j &= (-1)^j w_2 \\
(\alpha\beta)_{ij} &= (-1)^{i+j} w_3 \\
\gamma_k &= (-1)^k w_4 \\
(\alpha\gamma)_{ik} &= (-1)^{i+k} w_5 \\
(\beta\gamma)_{jk} &= (-1)^{j+k} w_6 \\
(\alpha\beta\gamma)_{ijk} &= (-1)^{i+j+k} w_7
\end{aligned}$$

The 'mapping' from the Walsh coefficient numbers to the appropriate 'effect' is given by writing the effects in what is known in experimental design as *standard order*: in this case

$$\{\mu, \alpha, \beta, \alpha\beta, \gamma, \alpha\gamma, \beta\gamma, \alpha\beta\gamma\}.$$

The general pattern is fairly obvious-on adding another factor the next set of effects is obtained by 'combining' the new factor with the effects already listed, in the same order. Thus in the case of a 4-bit problem, for example, the next 8 in order will be

$$\{\delta, \alpha\delta, \beta\delta, \alpha\beta\delta, \gamma\delta, \alpha\gamma\delta, \beta\gamma\delta, \alpha\beta\gamma\delta\}.$$

It is also fairly obvious that this order is a consequence of the definition of the Walsh monomials.

Thus in general, to convert from the Walsh representation to the ED coefficients, we first identify the appropriate coefficient as above, and its associated indices, and then multiply by $(-1)^{\sum indices}$.

Some consequences of this for the understanding of deception and other matters are explained in more detail in [1].

4. Epistasis measurement. Davidor's idea [22, 23] was to use the decomposition into linear (main effects) and non-linear (interaction) components to measure the degree of epistasis in a given problem using his 'excess allele values'. In fact, traditional statistical methods of Analysis of Variance provide a more straightforward route to the measurement of epistasis, as is shown in detail in [1], where we further note how these can apply to the measurement of the effect of adopting different codings on the ease or difficulty of solving a particular problem.

Analysis of Variance (Anova) is a technique whereby the variability of the fitness values (measured by sums of squared deviations from mean fitness, and denoted by SS) is partitioned into orthogonal components from identifiable sources. Associated with these SS are the *degrees of freedom*- the number of independent elements in the associated SS.

It is well-known (and easy to prove) that

$$\text{Total SS} = \text{Main effects SS} + \text{Interaction SS}.$$

Given the partitioning of the variances in the above way, an obvious measure for the degree of epistasis in a problem is the ratio

$$\eta = \frac{\text{Interaction SS}}{\text{Total SS}}.$$

The larger this value, the greater the degree of epistasis. As shown in [2], this quantity does appear to be meaningful in the sense that, for certain well-defined problems of known hardness, η correlates well with the degree of difficulty of solution by means of a GA.

However, this is not true for all functions. It is entirely possible to have two functions that are quite different in regard to their ease of solution, yet they have identical η values. If the SS are expressed in terms of the main effects and interactions, it becomes clear that it is the *square* of the effect that is important, so that positive interactions cannot be distinguished from negative ones. This is important, because some interactions have the effect of reinforcing the main effects, while others work in opposition. In [2] these are called benign and malign interactions respectively, and Figure 1 illustrates this phenomenon.

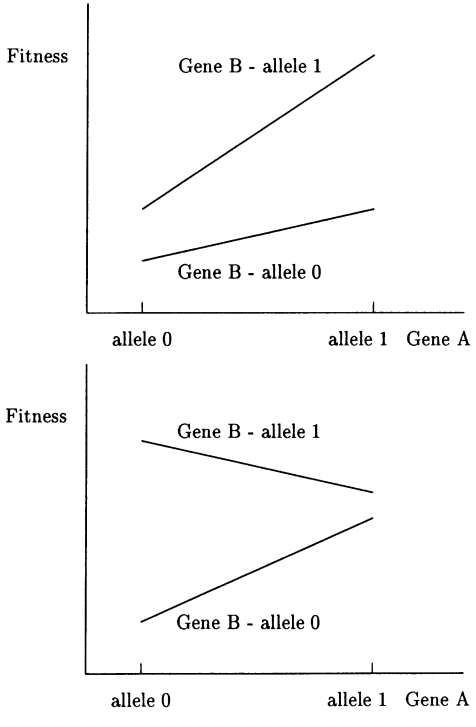


FIG. 1. *Benign and malign interactions.*

In the upper diagram, the best level of each factor is level 1, and while

there is epistasis, in that the joint effect of having both A and B at level 1 exceeds the sum of the individual main effects, the interaction reinforces the main effects. However, in the lower diagram, the interaction has a malign influence: the best level of both A and B is level 1, but overall it is better to set factor A to be at level 0. Thus the main effects and their associated interaction effect are 'pointing' in opposite directions. From a traditional GA perspective, this reflects what is often called *deception* [13, 14].

In terms of the actual values of the effects, the first situation corresponds to the case where the interaction effect $(\alpha\beta)_{01}$ has the same sign as the main effect α_0, and the second to the case where the signs are different. Thus the sign of the interaction effect clearly influences the epistasis, yet η cannot distinguish the two cases. While the sign is important, [2] also shows that the magnitude of the interaction effect, relative to its associated main effects, is also relevant. If the interaction effect has the opposite sign, but its magnitude is small, the main effects still 'point' in the right direction. This is illustrated in Figure 2.

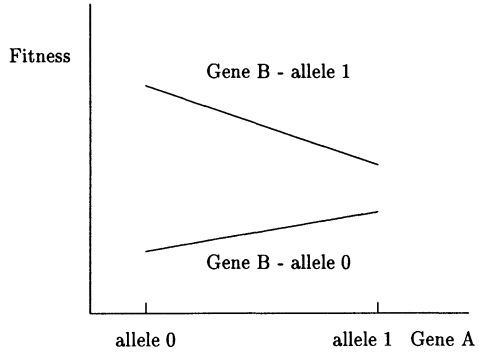


FIG. 2. *Relatively small interaction.*

We could sum this up in the following:

PROPOSITION 4.1. *Not all epistasis is harmful to the prospects of a GA finding a global optimum.*

The problem is that the simple epistasis measure suggested by [22] and clarified in [2] cannot distinguish between cases where it is harmful and cases where it is not. This property is also true of other summary measures that have been suggested based on Walsh transform coefficients. In order to uncover the nature of the epistasis in a problem, it is necessary to have the auxiliary information concerning the nature of the interaction effects-information that can be obtained from a decomposition model whose components can be estimated from the Universe. The question that now arises is whether this information is obtainable from a *sample* of the Universe.

5. Sample information. Suppose, using Davidor's 3-bit examples again, that the population is actually composed of the strings in table 1:

TABLE 1
Some half-fractions of the Universe.

fitness value	String			
	Universe	F_1	F_2	F_3
v_1	0 0 0	0 0 0	0 0 0	
v_2	0 0 1	0 0 1		0 0 1
v_3	0 1 0		0 1 0	0 1 0
v_4	0 1 1			0 1 1
v_5	1 0 0		1 0 0	
v_6	1 0 1			1 0 1
v_7	1 1 0	1 1 0	1 1 0	
v_8	1 1 1	1 1 1		

These are *half-fractions* of the Universe; the first one (F_1) is *balanced* but F_2 and F_3 are not—we can see that there are 2 occurrences of each allele in the first case, but in the second case only allele 0 is instantiated at gene 3, while F_3 has a different frequency of occurrences of alleles 0 and 1 in both gene 1 and 3.

A useful experimental design concept here is that of a *contrast*, usually denoted by upper case Roman letters. For example, the contrast

$$A = \alpha_1 - \alpha_0$$

(where α_p is as previously defined), expresses the average fitness value when allele 1 is instantiated at gene 1, compared to the instantiation of allele 0. In terms of the vector of fitness values $\mathbf{v}$ in the Universe,

$$A = \frac{1}{4}(-1,-1,-1,-1,1,1,1,1)\mathbf{v}$$

$$B = \frac{1}{4}(-1,-1,1,1,-1,-1,1,1)\mathbf{v}$$

$$C = \frac{1}{4}(-1,1,-1,1,-1,1,-1,1)\mathbf{v}.$$

Similarly, we can define contrasts relating to the interaction effects, so that

$$AB = \frac{1}{4}(1,1,-1,-1,-1,-1,1,1)\mathbf{v}$$

expresses the average fitness value for cases where the instantiated alleles at genes 1 and 2 are the same, compared to those where they are different. The other contrasts are as follows:

$$AC = \frac{1}{4}(1,-1,1,-1,-1,1,-1,1)\mathbf{v}$$

$$BC = \frac{1}{4}(1,-1,-1,1,1,-1,-1,1)\mathbf{v}$$

$$ABC = \frac{1}{4}(-1,1,1,-1,1,-1,-1,1)\mathbf{v}.$$

These 7 contrasts constitute an orthogonal partitioning of the information available in the 8 fitness values.

When the Universe is known, all these can be computed and the existence (or otherwise) of epistasis identified. (In fact, this is usually still possible even when there is some random error in the fitness evaluation function.) However, when only a fraction is available, some interesting problems arise.

Suppose we have the first fraction F_1 as shown above, so that in terms of the fitness values, we know only (v_1, v_2, v_7, v_8). Using only the *available* information to calculate estimates of the contrasts, we find that

$$\widehat{A} = \frac{1}{2}(-1,-1,0,0,0,0,1,1)\mathbf{v}$$

$$\widehat{B} = \frac{1}{2}(-1,-1,0,0,0,0,1,1)\mathbf{v}$$

$$\widehat{C} = \frac{1}{2}(-1,1,0,0,0,0,-1,1)\mathbf{v}.$$

$$\widehat{AB} = \frac{1}{4}(1,1,0,0,0,0,1,1)\mathbf{v}$$

$$\widehat{AC} = \frac{1}{2}(1,-1,0,0,0,0,-1,1)\mathbf{v}$$

$$\widehat{BC} = \frac{1}{2}(1,-1,0,0,0,0,-1,1)\mathbf{v}$$

$$\widehat{ABC} = \frac{1}{2}(-1,1,0,0,0,0,-1,1)\mathbf{v}.$$

It is clear that

$$\widehat{A} = \widehat{B}; \widehat{C} = \widehat{ABC}; \widehat{AC} = \widehat{BC}; \text{ and } \widehat{AB} = \widehat{\mu}.$$

An alternative viewpoint is in terms of the original contrasts, whence we see that (for example)

$$(A+B) = \frac{1}{2}(-1,-1,0,0,0,0,1,1)\mathbf{v} = \widehat{A} = \widehat{B};$$

in other words, we cannot estimate A and B separately, but only their sum. It can easily be confirmed that the same is true of the other pairs of estimates in the above table.

These pairs of indistinguishable contrasts are known as *alias sets*, and each set is associated with 1 degree of freedom, corresponding to the 3 degrees of freedom associated with having 4 fitness values. The contrast which is aliased with μ is known as the *defining contrast* for its half-fraction. By choosing a different defining contrast it is possible to generate any half-fraction.

It may now be impossible to discern from a given fraction whether there are any epistasis effects, since ABC—the 2^{nd} order interaction term—is indistinguishable from the main effect C. Thus, if an Anova table indicates a significant source of variation due to C, we cannot tell whether this is because the fitness is a linear function with a high contribution from gene 3, or whether it is because the function is epistatic with a large 2^{nd} order interaction. Conversely, if the table suggests that there is no variation due to C, it might be because the effect of ABC is in the opposite direction from C.

As an example, we present below the Anova table for Davidor's functions f_1, f_2, f_3, f_4.

TABLE 2
Anova results on F_1 for Davidor's functions.

	f_1		f_2		f_3		f_4	
Source	df	SS	df	SS	df	SS	df	SS
$(A+B)$	1	36.00	1	196.00	1	100.00	1	4.00
$(C+ABC)$	1	1.00	1	196.00	1	56.25	1	9.00
$(AB+BC)$	1	0.00	1	196.00	1	49.00	1	25.00
Total	3	37.00	3	588.00	3	205.25	3	38.00

It would still seem reasonable to conclude that f_1 is *not* epistatic, and that f_4 *is*, although it is necessary to bear in mind the possibility that C and ABC have cancelled each other out. However, it is much harder to make a positive statement about f_2 and f_3.

We can also now see why Davidor's attempt to estimate epistasis variance from a fraction is fatally flawed, precisely because of these alias sets. As we noted earlier, for example, using the fraction F_1, he obtained negative values in Table 8 of [23]. But it is not possible to compute his 'genic values' for *both* genes 1 and 2 (i.e. the contrasts A and B) simultaneously for this fraction.

Other fractions will give rise to different aliasing structures; for instance if we use the fraction F_2, we find that the estimable combinations are

$$(A - AC)/2; (B - BC)/2; (AB - ABC)/2; (-C + \mu)/2.$$

5.1. Some implications. In such situations it is clearly impossible to determine whether an apparent effect is really caused by the factor (or interaction of factors) with which it is ostensibly related. Thus, any epistasis measure that implicitly assumes a decomposition into main effects and interactions must be treated with great caution. In principle therefore, the amount of information in a sample can never be sufficient to enable us to decide on the nature of the epistasis in a problem. In practice, it may be sufficient: for a problem of bounded epistasis (for example, where all interactions of more than p factors are zero, for some value p), there will be an orthogonal fraction that would enable all the non-zero effects to be estimated independently and precisely. However, the catch is that even if we had such a case, we couldn't *know* that we had done so.

This is also of relevance to ideas of schema processing or hyperplane sampling. The 'traditional' GA interpretation uses the idea of hyperplane competitions to explain its operation, as in [14], for instance. It should be evident from the above that a hyperplane competition is related to the concept of a contrast—e.g. the competition between the hyperplanes $(1**)$ and $(0**)$ can be expressed by the contrast

$$A = \alpha_1 - \alpha_0.$$

Higher-order hyperplane competitions can also be expressed in terms of contrasts between components of the decomposition model; for instance, the competition between $(11*)$ and $(10*)$ can be expressed as

$$[\beta_1 - \beta_0] + [(\alpha\beta)_{11} - (\alpha\beta)_{10}] = B + \frac{AB}{2}.$$

When we have partial information, our estimates of these contrasts are confounded with many others. So the hyperplane $(1**)$ could 'win' the competition with $(0**)$ (i.e. $\widehat{A} > 0$) simply because in a particular population A is aliased with BC—i.e. if there is a large interaction between genes 2 and 3. In practice GA populations are not chosen in any systematic way, so that there will not even be a clear-cut choice between an alias pair, but rather a spectacular 'mess' of alternative explanations for the phenomenon observed.

This has some unpleasant implications for the arguments concerning implicit parallelism and the k-armed bandit analogy. If we grant the validity of Holland's estimate of the number of schemata or hyperplanes in a population being $\mathcal{O}(M^3)$, it still does not follow that they can all be usefully *processed.* In the light of the above argument, it is clear that the fitness of one hyperplane cannot be estimated independently of those with which it is aliased. The k-armed bandit analogy also runs into trouble here: the 'payoff' for a given 'arm' of the bandit is not a good estimate of the true average fitness of a particular hyperplane, unless all its aliases happen to have zero fitness—an unlikely circumstance!

In classical ED, balanced orthogonal fractions are defined which enable the aliasing to be controlled. Typically an *a priori* decision is made that certain interactions are negligible, perhaps using problem-specific information. Then a fraction of the Universe is chosen in a way that ensures that the remaining factors that are potentially important in explaining the data are aliased, not with each other, but with the 'unimportant' interactions. In a 'population' of M orthogonal points, assuming that the hypothesis concerning the negligible interactions is valid, it would then be possible to estimate M components independently. This is the *maximum* information that can be obtained from such a population; in GA terminology, only $\mathcal{O}(M)$ hyperplane competitions at best can be settled, even in the case of a carefully chosen orthogonal subset.

We could thus interpret the concept of an *orthogonal* ED as defining a lower bound on the necessary amount of computation required to solve an optimization problem. For a problem of bounded epistasis, the 'right' orthogonal design will identify the correct levels of all the important factors. ED texts (such as Montgomery [27]) frequently give lists of designs of different 'resolutions' that will enable the estimation of all effects up to and including p-factor interactions for some integer p. In practice, of course, the epistatic nature of a given problem can only be guessed, so that if the problem is actually *more* epistatic than assumed, such designs may give misleading results.

6. Experimental comparison. The question of searching for an optimal (or at least a 'good') point in a relatively large search space is not one that has concerned statisticians until fairly recently. Firstly, as already mentioned, classical ED has stressed the need for 'understanding' the problem, and for establishing cause-and-effect relationships involving the factors that affect the performance of a given system. Secondly, with their heavy reliance on algebraic properties such as orthogonality, finding suitable designs for real large-scale problems is difficult, at least for more than 2 levels of each factor.

However, two recent influences have stimulated new research into ways in which ED can be applied to such problems. The first is the increased emphasis given to ED concepts in achieving quality in manufacturing [28]. The second is the use of high-fidelity computer simulations of physical phenomena in science and engineering [29, 30]. These developments have both stimulated interest in less human-intensive experimentation methods.

One such technique was introduced by Wu *et al.* [31]. Known as *sequential elimination of levels* (SEL), it attempts to narrow the focus of the search by successively reducing the size of the search space in a series of stages. At each stage a small subset of points (chosen on the basis of ED principles of orthogonality) is evaluated and those levels (alleles) of each factor (gene) that appear to produce the worst performance are eliminated. Different elimination criteria have been proposed: SEL-*mean*

compares levels on the basis of mean performance, while SEL-*max* uses best performance as criterion. We also formulated a modified version of SEL-*max* (called SEL-*mod*) that we hoped would prove to be an improvement. An example of SEL in action is given in [32].

SEL-*mean* in particular is reminiscent of schema processing explanations of GA performance, at least for a generational GA running without mutation. However, in contrast to a GA, SEL works in an explicit, deterministic and somewhat myopic way. Nevertheless, a comparison would be interesting, so a series of experiments was carried out.

6.1. The problem. The test problem used was an engineering design problem reported by Carlson *et al.* [33]—a problem of designing a hydraulic system by selecting from a set of components those which gave the best performance. The system in question had 6 basic components (labelled A,B,...,F here for convenience of reference), each of which came in 5 types, so that the size of the total search space was $5^6 = 15625$. To enumerate a search space of this size in an experimental design would not normally be feasible, although for the purposes of the experiment the enumeration was in fact carried out. In [33] several GAs were tried, and the best one found the optimal solution in about 60% of 500 trials, on average enumerating 4170 points—rather more than one quarter of the search space. It should be noted that this frequency of success is considerably higher than would be expected from a purely random search.

In experimental design, it would be abnormal to use more than about 100 points, and there would be a less than 1% chance of selecting the optimum purely by chance. However, if the response surface has low epistasis, ED methods might be able to identify the optimum. A traditional design for this problem would be an orthogonal $\frac{1}{5^k}$ fraction, for some k. For example, with $k = 3$ we would have 125 points. As is shown in [32], this provides little useful information for this response surface, which has too much epistasis for such a small fraction to be useful.

However, in practice we might well settle for a high-quality solution that was reasonably close to the optimum. Here we were able to identify 85 such solutions, and in assessing the effectiveness of a search we counted the frequency with which one of these solutions was discovered. The requirements of SEL meant that 116 points were evaluated in the course of an experiment. This was also set as the total number of evaluations for the GA (which used a simple incremental reproduction/deletion strategy, uniform crossover and a small amount of mutation). Full details of the methods and the results of these experiments are reported in [32]. Here, Table 3 summarises the success rate of each method in finding one of the élite solutions (here separated into groups I and II) over the course of 100 experiments.

It should be realized that the probability of selecting at least one of these top 85 solutions in a random sample of 116 drawn without replace-

TABLE 3

Frequency of identification of at least one of the élite solutions (out of 100 trials): Latin Square initial populations.

Group	SEL *-mean*	SEL *-max*	SEL *-mod*	GA
I	1	12	93	27
II	27	25	7	25
Total	28	37	100	52

ment is, by the hypergeometric distribution, approximately 47%. It is thus immediately obvious from Table 3 that both the *max* and *mean* forms of SEL fail even to reach this level of performance. The GA is able to improve on a random search, although the difference is not statistically significant. (However, when they did find one of the top 85 solutions, all methods, especially the GA, usually found several. A random search has only a 13% chance of doing this.) In contrast, SEL-*mod* always found at least one of the élite group.

The next question is whether these methods are sensitive to the need for orthogonal subsets. Even for this problem it was not simple to find such subsets at every stage, and with the need to automate as much as possible, simpler alternatives were also investigated. One approach is to select subsets randomly, but in a way that ensures they are *balanced* with respect to the main effects—i.e. there is the same number of occurrences of each level for each factor. (The Latin Square of 25 points used in stage 1 of these experiments also ensures balance across each combination of levels for each *pair* of factors.) In the case of the SEL variants, the same policy was used at each stage of the search. In the case of the GA, this only affects the first stage, i.e., the initial population.

Normally, genetic algorithms generates initial populations completely at random, with no attempt at maintaining a balance. In the case of non-binary strings, as here, this could be dangerous, since there is a non-trivial probability that at least one allele will not be represented at all unless the population is fairly large [34]. It would thus be expected that all methods would be adversely affected by using completely random sets of strings at each stage.

The results of starting with different initial populations in place of the Latin Square are shown in Table 4.

The table shows that orthogonality is certainly important for the performance of ED-based methods—as might be expected, given the emphasis put on orthogonality in traditional ED. The reduction in performance is most marked for SEL-*mod* and SEL-*mean*. The effect on the GA's performance is, by comparison, fairly small.

We were able to carry out some larger experiments with orthogonal fractions of 625 points. These nearly always succeeded in predicting the

TABLE 4

Frequency of identification of at least one of the élite solutions (out of 100 trials): non-orthogonal initial populations.

Balanced random initial population				
Group	SEL *-mean*	SEL *-max*	SEL *-mod*	GA
I	0	17	25	24
II	20	21	27	33
Total	20	38	52	57
Unbalanced random initial population				
Group	SEL *-mean*	SEL *-max*	SEL *-mod*	GA
I	4	8	9	20
II	9	18	29	26
Total	13	26	38	46

correct levels of the most important factors. However, it should be noted that in a sample of 625 points, we would expect to find at least one of the top 85 solutions with probability 97% anyway, so this is perhaps of little practical relevance.

Whether this problem has any general lessons cannot be stated with certainty. However, an analysis of the epistasis in the problem was carried out using the methods outlined in section 3. This revealed that the problem contains sufficient epistasis to be misleading (deceptive) in some parts of the search space, and thus to be a reasonable challenge to any algorithm. It is not surprising that SEL-*mean* found it difficult, since averaging operations are precisely those which are prone to deception.

Some recent work by Haslam [35] supports this conclusion. In [35], artificial problems of up to 9 factors with up to 5 levels were constructed with different amounts of epistasis. SEL implementations always found the optimum for linear problems (no epistasis), but their performance degraded substantially as the amount of epistasis was increased. The performance of a GA also degraded with increasing epistasis, but much more slowly. SEL-*mean* was generally worse than either of the other versions, except for the linear case.

The poor performance of SEL-*mean* is intriguing since, as already remarked, this approach is closest in spirit to the idea of schema processing. Using random populations for this method (as in a traditional GA) produced the worst results, and while using orthogonal designs helps to some extent, SEL-*mean* does not perform as well as the GA. This adds weight to the feeling that there is rather more to GAs than schema-processing.

While in principle it is possible to imagine an algorithm that estimates the various components (main effects and interactions) from a sample and

hence identifies the optimal string, in practice it is hard to realize such an algorithm. The size of a true orthogonal fraction that would allow such an approach may be too large to be practicable. Other methods have been suggested (in, for example, [29]) which try to 'cover' the search space in some optimal sense. However, the approach taken by SEL, and by a GA, is to focus the search more and more narrowly, and not attempt to cover the space in such a way. However, on the evidence of SEL-*mean*, even in a search space that has been reduced by the algorithm, measuring averages (means) comes unstuck in the presence of epistasis. Using the best points, as in SEL-*max* and SEL-*mod*, appears to increase the chance of finding good solutions, and suggests that the GA, which was even more successful, also benefits from such a strategy, and in fact implements it better.

7. Conclusion. We have shown how the design of experiments can give a different perspective on GAs. In particular, they can provide some useful insights into the nature of epistasis and its measurement, and show that not all epistasis is harmful. The ED concept of aliasing has been described and some of its implications for traditional schema-processing arguments have been discussed. This also helps to show that in principle the amount of epistasis in a problem cannot be decided without complete knowledge of the Universe.

A simple incremental GA has been compared with three versions of a statistical method proposed in the ED literature for solving the sort of problems for which it is expected GAs would also be suitable. Experimental results show that in general the SEL approach was inferior to this GA, even when orthogonal fractional designs were employed. One of these methods (SEL-*mod*) performed extremely well when the orthogonal designs of ED theory were supplemented by the principle of élitism. However, it proved to be substantially less robust to departures from orthogonality than the GA. The SEL approach that worked least well was SEL-*mean*, which functions rather like an explicit schema-processing method. The fact that a GA did so much better also suggests that schema-processing is only part of the GA story.

Finally, we should record that the presupposition of one of us, before carrying out this work, was that methods based on orthogonal designs would outperform GAs. In fact, these experiments suggest that the reverse is the case. This may be a common presupposition in the statistics community, since very little of the ED literature seems even to be aware that GAs exist. This may be because ED usually has as its main aim that of understanding and explaining system performance, rather than merely finding the best region of the search space. Without orthogonal subsets, providing such an explanation is difficult, and using the highly non-orthogonal subsets generated by a typical GA run would seem to be problematic. However, recent work by Pistone and Wynn [36] has shown that computer algebraic methods can be used to identify and estimate ED effects from

non-orthogonal subsets. It is not yet clear how easy this would be as a matter of routine, but it is an advance that suggests the way might eventually be cleared to search for optimal points with a GA, and still deduce additional information of the sort that statisticians like to infer.

REFERENCES

[1] C. R. REEVES AND C. C. WRIGHT, (1995), *An experimental design perspective on genetic algorithms*, In D. Whitley and M. Vose (Eds.), (1995) Foundations of Genetic Algorithms 3, Morgan Kaufmann, San Mateo, CA, 7-22.

[2] C. R. REEVES AND C. C. WRIGHT, (1995), *Epistasis in genetic algorithms: an experimental design perspective*, In L. J. Eshelman (Ed.), (1995), Proceedings of the 6^{th} International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 217-224.

[3] J. H. HOLLAND, (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

[4] D. E. GOLDBERG, (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Mass.

[5] K. A. DE JONG, (1975), *An analysis of the behavior of a class of genetic adaptive systems*, Doctoral dissertation, University of Michigan.

[6] K. A. DE JONG, (1993), *Genetic algorithms are NOT function optimizers*, In L. D. Whitley (Ed.), (1993), Foundations of Genetic Algorithms 2, Morgan Kaufmann, San Mateo, CA, 5-17.

[7] M. BEHE, (1996), *Darwin's Black Box: The Biochemical Challenge to Evolution*, Free Press, New York, NY.

[8] M. DENTON, (1984), *Evolution: A Theory in Crisis*. Burnett Books, London.

[9] M. DENTON, (1996), *Biology: The Anthropic Perspective*, Libraire Artheme Fayard, Paris.

[10] H. MÜHLENBEIN, (1991), *Evolution in time and space—the parallel genetic algorithm*, In G. J. E. Rawlins (Ed.), (1991), Foundations of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 316-337.

[11] N. J. RADCLIFFE, (1991), *Forma analysis and random respectful recombination*, In R. K. Belew and L. B. Booker (Eds.), (1991), Proceedings of 4^{th} International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 222-229.

[12] W. G. MACREADY AND D. H. WOLPERT, (1996), *On 2-armed Gaussian Bandits and Optimization*, Technical Report SFI-TR-96-03-009, Santa Fe Institute, Santa Fe, New Mexico.

[13] D. E. GOLDBERG, (1989), *Genetic algorithms and Walsh functions: part II, deception and its analysis*, Complex Systems, **3**, 153-171.

[14] D. WHITLEY, (1991), *Fundamental principles of deception in genetic search*, In G. J. E. Rawlins (Ed.), (1991), Foundations of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 221-241.

[15] J. J. GREFENSTETTE, (1993), *Deception considered harmful*, In L. D. Whitley (Ed.), (1993), Foundations of Genetic Algorithms 2, Morgan Kaufmann, San Mateo, CA, 75-91.

[16] M. MITCHELL, J. H. HOLLAND AND S. FORREST, (1994), *When will a genetic algorithm outperform hill climbing?*, In J. D. Cowan, G. Tesauro and J. Alspector (Eds.), (1994), Advances in Neural Information Processing Systems 6, Morgan Kaufmann, San Mateo, CA.

[17] D. WHITLEY, (1993), *An executable model of a simple genetic algorithm*, In L. D. Whitley (Ed.), (1993), Foundations of Genetic Algorithms 2, Morgan Kaufmann, San Mateo, CA, 45-62.

[18] M. D. VOSE, (1993), *Modeling simple genetic algorithms*, In L.D.Whitley (Ed.),

(1993), Foundations of Genetic Algorithms 2, Morgan Kaufmann, San Mateo, CA, 63-73.

[19] M. D. VOSE, (1994), *A closer look at mutation in genetic algorithms*, Annals of Maths and AI, **10**, 423-434.

[20] M. D. VOSE AND A. H. WRIGHT, (1995), *Stability of vertex fixed points and applications*, In D. Whitley and M. Vose (Eds.), (1995), Foundations of Genetic Algorithms 3, Morgan Kaufmann, San Mateo, CA, 103-113.

[21] K. A. DE JONG, W. M. SPEARS AND D. F. GORDON, (1995), *Using Markov chains to analyze GAFOs*, In D. Whitley and M. Vose (Eds.), (1995), Foundations of Genetic Algorithms 3, Morgan Kaufmann, San Mateo, CA, 115-137.

[22] Y. DAVIDOR, (1990), *Epistasis variance: suitability of a representation to genetic algorithms*, Complex Systems, **4**, 369-383.

[23] Y. DAVIDOR, (1991), *Epistasis variance: a viewpoint on GA-hardness*, In G. J. E. Rawlins (Ed.), (1991), Foundations of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 23-35.

[24] D. E. GOLDBERG, (1989), *Genetic algorithms and Walsh functions: part I, a gentle introduction*, Complex Systems, **3**, 129-152.

[25] A. D. BETHKE, (1981), *Genetic algorithms as function optimizers*, Doctoral dissertation, University of Michigan.

[26] A. J. MASON, (1991), *Partition coefficients, static deception and deceptive problems for non-binary alphabets*, In R. K. Belew and L. B. Booker (Eds.), (1991), Proceedings of 4^{th} International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 210-214.

[27] D. C. MONTGOMERY, (1991), *Design and Analysis of Experiments*, Wiley, New York.

[28] S. GHOSH, (1990), (Ed.), *Statistical Design and Analysis of Industrial Experiments*, Marcel Dekker, New York.

[29] M. D. MORRIS AND T. J. MITCHELL, (1995), *Exploratory designs for computational experiments*, J. Statistical Planning and Inference, **43**, 381-402.

[30] R. A. BATES, R. J. BUCK, E. RICCOMAGNO AND H. P. WYNN, (1996), *Experimental design and observation for large systems*, J. R. Statist. Soc. B, **58**, 77-94.

[31] C. F. J. WU, S. S. MAO AND F. S. MA, (1990), *SEL: A search method based on orthogonal arrays*, In S. Ghosh (1990), (Ed.), Statistical Design and Analysis of Industrial Experiments, Marcel Dekker Inc., New York, 279-310.

[32] C. R. REEVES AND C. C. WRIGHT, (1997), *Genetic algorithms versus experimental methods: a case study*, In Th.Bäck (Ed.), (1997), Proceedings of 7^{th} International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 214-220.

[33] S. E. CARLSON, R. SHONKWILER AND M. INGRIM, (1993), *A comparative evaluation of search methods applied to catalog selection*, In S. Forrest (Ed.) ,(1993), Proceedings of 5^{th} International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 630.

[34] C. R. REEVES, (1993), *Using genetic algorithms with small populations*, In S. Forrest (Ed.), (1993), Proc. of 5^{th} International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 92-99.

[35] T. HASLAM, (1996), *An Investigation of the Effectiveness of the SEL Search Technique to Locate Optima over Different Response Surfaces*, BSc Dissertation, School of Mathematical and Information Sciences, Coventry University.

[36] G. PISTONE AND H. P. WYNN, (1996), *Generalised confounding with Gröbner bases*, Biometrika, **83**, 653-666.

EFFICIENT PARAMETER OPTIMIZATION BASED ON COMBINATION OF DIRECT GLOBAL AND LOCAL SEARCH METHODS

MICHAEL SYRJAKOW* AND HELENA SZCZERBICKA†

Abstract. In this paper we focus on direct optimization methods which require nothing but goal function values for orientation (blind search). On the one hand this property ensures robustness and universal applicability. On the other hand direct optimization usually requires a lot of computational effort (goal function evaluations) to ensure optimization success (convergence towards a globally-optimal region of the search space) and an acceptable quality of the optimization result (small approximation error). These fundamental drawbacks of direct optimization are due to the fact that no auxiliary information like derivatives or other problem specific knowledge is exploited to accelerate the optimization process. In order to substantially improve the performance of direct optimization we propose a combination of probabilistic global and deterministic local optimization methods. The resulting combined 2-phase optimization strategy has been proven to be both powerful and efficient. The excellent heuristic properties of this hybrid method allows to use it as the basic component of a multiple-stage optimization strategy. The goal of multiple-stage optimization is to perform a systematic analysis of the most important extreme points of a given optimization problem. This paper presents the structure of our developed optimization algorithms. Beyond that, results of an extensive optimization experiment are presented showing the potentialities but also the limits of our developed methods.

1. Introduction. In recent years direct optimization methods have gained considerably in importance. They have been employed successfully in various fields of application, especially in domains where classical mathematical methods do not work any more. The most common and powerful direct methods for global optimization are Genetic Algorithms [2], Evolution Strategies [7], and Simulated Annealing [1]. All these methods apply sophisticated probabilistic search operators which imitate principles of nature. Although these operators have been proven to be well-suited for global search the required computational effort (number of goal function evaluations) and the quality of the generated optimization results still remain a big problem.

In the following, an approach to further improvement of direct methods for global parameter optimization is presented. In order to make direct optimization more efficient and to achieve high quality solutions we propose to combine existing global and local optimization strategies. The resulting hybrid optimization algorithm is called combined 2-phase strategy. This strategy is based on the splitting of the optimization process into two phases: pre-optimization with a probabilistic global optimization method and fine-optimization performed by a deterministic local Hill-Climber. The

*Institute for Computer Design and Fault Tolerance (Prof. D. Schmid), University of Karlsruhe, 76128 Karlsruhe, P.O. Box 6980, GERMANY; Email: syrjakow@ira.uka.de.

†Department of Computer Science (Fb3 Informatik), University of Bremen, 28334 Bremen, P.O. Box 330440, GERMANY; Email: helena@informatik.uni-bremen.de.

task of pre-optimization is to explore the search space in order to find promising regions where globally-optimal solutions might be located. Outgoing from the result of pre-optimization the task of fine-optimization is to localize the optimal solution in a promising region efficiently and exactly. So, pre-optimization is predominantly responsible for optimization success, whereas fine-optimization has to ensure optimization quality. The considerable gain in efficiency makes it possible to employ combined 2-phase strategies for multiple-stage optimization. Here, a combined 2-phase strategy is executed several times in order to find the most prominent extreme values of a given optimization problem.

This paper gives an insight into the basic structure and the performance of our developed optimization algorithms. Section 2 starts with a brief introduction into direct parameter optimization. Subsequently in Section 3 existing methods for global and local search are presented. These so-called atomar strategies represent the basic components of combined 2-phase optimization described in Section 4. In Section 5 a combined 2-phase strategy is extended to multiple-stage optimization. In Section 6 some supplementary methods to reduce goal function evaluations are presented. Section 7 contains a brief description of the model optimization tool REMO (REsearch Model Optimization Package) in which our developed optimization algorithms are integrated. In Section 8 experimental results are presented, showing both the potentialities but also the limits of our approach. Finally, in Section 9 we summarize and draw some conclusions.

2. Direct parameter optimization. This Section briefly introduces into real parameter optimization with direct search methods. Beyond that, the terminology used in this paper is determined.

Each instance of a real parameter optimization problem can be formalized as a pair (S, F). The solution space S denotes the set of all possible problem solutions. Every solution from S is valued by a goal function $F : S \to R$. For unconstrained parameter optimization problems $S = R^n$, otherwise:

$$S = \left\{ \begin{array}{l} \vec{x} \in R^n | h_i(\vec{x}) = 0, i \in \{1, ..., p\}; \\ g_j(\vec{x}) \leq 0, j \in \{1, ..., q\}; p, q \in N \end{array} \right\},$$

$h_i : R^n \to R$ equality constraints

$g_j : R^n \to R$ unequality constraints

The overall goal of global parameter optimization is to find a vector $\vec{x} \in S$ which satisfies:

$$\forall \vec{x} \in S : F(\vec{x}) \circ F(\vec{x}^*) = F^*, \quad \text{with } \circ \epsilon \{\leq, \geq\}.$$

A solution $\vec{x}^*$ is called global optimum point. The function value $F(\vec{x}^*) = F^*$ is referred to as global optimum of F. Beside global optimum points

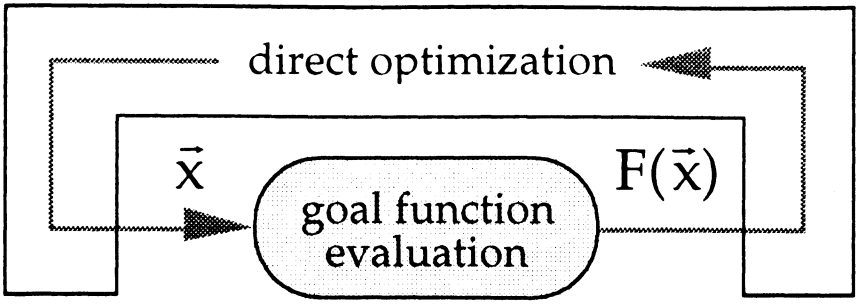


FIG. 1. *Principle of direct parameter optimization.*

there may exist local optimum points $\vec{x}^{\wedge}$, having the property that all neighbouring solutions have the equal or a worse goal function value. A local optimum $F^{\wedge} = F(\vec{x}^{\wedge})$ is defined as follows:

$$\exists \varepsilon > 0, \forall \vec{x} \in S : \|\vec{x} - \vec{x}^{\wedge}\| < \varepsilon \Rightarrow F(\vec{x}) \circ F(\vec{x}^{\wedge}) = F^{\wedge},$$
$$\text{with } \circ \in \{\leq, \geq\}.$$

Goal functions with several global and/or local optimum points are called multimodal functions. An optimization problem is either a minimization ($\circ = \geq$) or a maximization ($\circ = \leq$) problem. Minimization problems can be easily transformed into maximization problems and vice versa, because

$$\min\{F(\vec{x})\} = -\max\{-F(\vec{x})\}.$$

Our developed parameter optimization strategies presented in the following belong to the class of direct optimization methods. As shown in Fig. 1, direct optimization methods work iteratively only using goal function values to guide the search process.

3. Atomar optimization strategies. Table 1 shows some well known direct optimization methods, which roughly can be classified into strategies for global and local search. Because all these methods work homogeneously based on only one optimization principle they are referred to as atomar optimization strategies (aos) in the following.

TABLE 1
Atomar optimization strategies aos.

♦	global	local
atomar optimization strategies aos	Genetic Algorithm GA Simulated Annealing SA	Pattern Search PS

The goal of atomar global optimization strategies (aos_g) is to localize at least one global optimum point of a given optimization problem. Atomar global optimization strategies like GA or SA are mainly based on probabilistic search operators, allowing random jumps all over the search space. Exactly this property is a basic condition for global optimization because it enables to escape from sub-optimal search space regions. The great disadvantage of probabilistic global search however is, that usually an unreasonable number of goal function evaluations is required to produce high-quality optimization results (extreme low convergence speed close by an optimal solution).

Outgoing from a start point $\vec{x}_{\text{start}}$ the goal of atomar local optimization strategies (aos_1) is to find an optimum point lying in the direct neighbourhood of $\vec{x}_{\text{start}}$. Atomar local optimization methods like PS are guided predominantly by deterministic or quasi-deterministic search operators, which enable to efficiently approximate optimum points with an user specified accuracy ε. The use of those deterministic local search operators however makes it impossible to escape from sub-optimal regions of the search space. In this case the optimization result is strictly determined by the start point.

Summing up, the one and only optimization principle of atomar optimization strategies makes them specialized either for global or local search. For that reason neither the global nor the local optimization strategies presented in Table 1 can be considered as completely satisfactory. What we need is a strategy which combines the advantages of probabilistic global and deterministic local search and avoids their disadvantages. In the following Section the basic structure of such a hybrid optimization algorithm is described. As elementary components we use the atomar optimization strategies summarized in Table 1. For global search we employ a standard Genetic Algorithm [2] as well as homogeneous Simulated Annealing [1]. Local search is performed by the powerful and efficient Pattern Search algorithm of Hooke and Jeeves [3].

4. Combined 2-phase optimization. Fig. 2 shows the basic structure of a combined 2-phase optimization strategy os_{2P}, which is composed of an atomar global optimization method aos_g and an atomar local optimization method aos_1. The two atomar optimization methods are coupled by means of an interface, comprising a method to derive control parameter values from optimization trajectories (dcp) as well as a method to select start points (ssp). Outgoing from the result of aos_g the task of dcp and ssp is to compute a favourable starting condition for aos_1.

As already mentioned in Section 1, the basic idea of combined 2-phase optimization is to subdivide the optimization process into two phases: pre-optimization with aos_g and fine-optimization performed by aos_1. The task of pre-optimization is to explore the search space in order to find a promising region where a global optimum point might be located. Outgoing from the promising region found by pre-optimization the task of

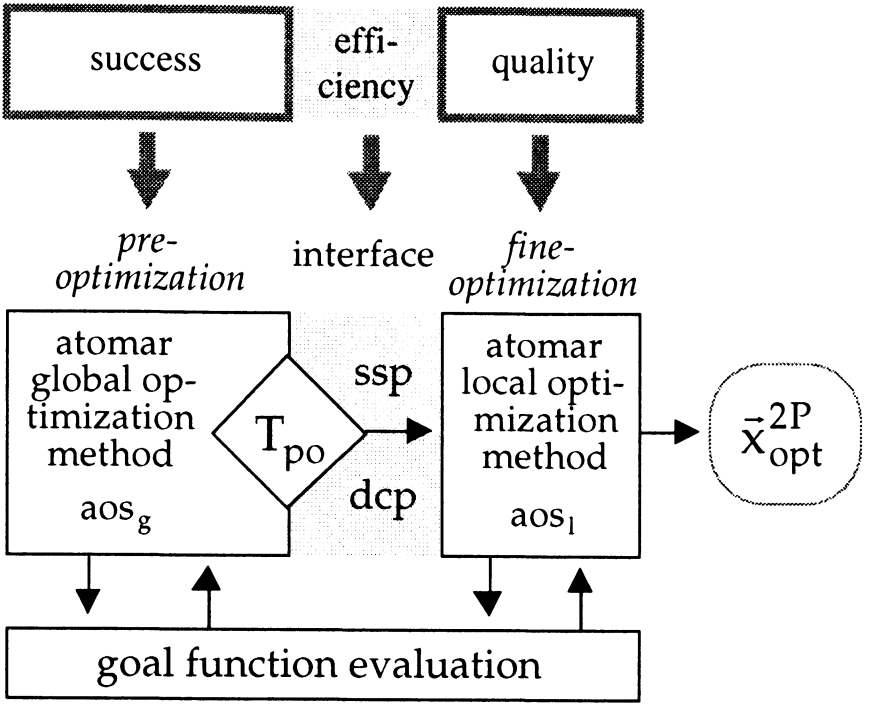


FIG. 2. *Basic structure of a combined 2-phase optimization strategy* os_{2P}.

fine-optimization is to efficiently and exactly localize the corresponding optimum point in it. Thus, pre-optimization is predominantly responsible for optimization success, whereas fine-optimization has to ensure high optimization quality (small approximation error).

The final result of a combined 2-phase optimization strategy as well as the required computational effort mainly depend on the specific capabilities of the employed atomar optimization methods but also on their co-operation. To realize a good co-operation the following problems have to be solved:

- choice of suitable control parameter values for aos_g

  Here, control parameter values have to be used rather forcing exploration of the search space than convergence towards a search space region. This can be achieved by emphasizing the probabilistic search operators of aos_g.

- choice of an advantageous switch-over point from pre- to fine-optimization

This problem affects the specification of a suitable termination condition T_{po} for aos_g in order to stop pre-optimization in time. Here, the main difficulty is to find a good compromise between two contrary goals: On the one hand a thorough exploration of the search space is required, because we do not want to get trapped into suboptimal regions. On the other hand the usually high computational effort (number of goal function evaluations) for pre-optimization should be kept as small as possible. For realization of T_{po} we have formulated several switch-over criteria based on

- the number of generated search points

  With this criterion the maximum number of goal function evaluations spent for pre-optimization can be specified.

- search point constellations

  Specific search point constellations (regional accumulations of search points in the search space) indicate convergence of aos_g towards a search space region. Applying standard cluster analysis methods, this property can be exploited profitably to compute switch-over points of good quality.

- the improvement of the goal function

  This criterion has been proven to be the most powerful one. Here, pre-optimization is stopped, if the goal function could not be improved p%, $p \in R^+$ over a specified number $n, n \in N$ of iteration steps.

- a priori knowledge about the goal function

  A priori knowledge about the goal function usually provides advantageous hints to improve efficiency. Hence, if available, it should be exploited in any case.

Of course, it is also possible to specify more complex switch-over criteria being combinations of the criteria above.

- choice of suitable control parameter values for aos_1

During pre-optimization the goal function has been evaluated many times. The computed goal function values (optimization trajectory), representing valuable knowledge about the goal function, can be used profitably to calculate suitable control parameter values for aos_1. For this purpose a method to derive control pa-

rameter values from optimization trajectories (dcp) has been developed. It is based on analyzing the optimization trajectory of aos_g by application of cluster analysis methods. From the size and form of the found clusters step sizes of high quality for aos_1 can be derived. Appropriate initial step sizes are very important to keep the required number of goal function evaluations for local search small.

- choice of a favourable start point $\vec{x}_{\text{start}}$ for aos_1 (ssp)

  The best solution found during pre-optimization is used as the start point $\vec{x}_{\text{start}}$ for aos_1.

5. Multiple-stage optimization. All optimization strategies presented so far localize only one optimum point when executed. Outgoing from these so called single-stage optimization strategies, we want to present an optimization algorithm which is able to detect several optimum points of a given multimodal optimization problem. The basic structure of such a multiple-stage optimization algorithm is shown in Fig. 3.

Central component of a multiple-stage optimization strategy os_{ms} is a combined 2-phase strategy os_{2P}. os_{2P} is embedded in an exterior iteration process, which generates step-by-step a sequence of optimum points $\vec{x}^1_{\text{opt}}, \vec{x}^2_{\text{opt}}, \ldots, \vec{x}^k_{\text{opt}}, k \in N$. An iteration step of a multiple-stage optimization strategy is called optimization stage.

os_{ms} stops, if the termination condition T_{ms} is fulfilled. A good termination criterion has been proven to be: stop, if a new optimum point could not be located over a specified number of optimization stages. If T_{ms} is not fulfilled, a method called avoidance of reexploration (AR) is applied. The task of AR is to avoid, that previously found optimum points are located again in subsequent optimization stages. This is done by making already explored regions of the search space unattractive for the global optimization method aos_g. For that purpose, in addition to the goal function values, attractiveness values are introduced and related to the generated search points. Attractiveness values are computed by means of an attractiveness function

$$av(\vec{x}) = \prod_{i=1}^{k} [1 - (1 + \alpha \cdot d_i)^{-\beta}],$$

with $d_i = \sqrt{(\vec{x} - \vec{x}^i_{\text{opt}})^2}$; α, β : scaling factors;

k : number of already found optimum points.

For a further description of multiple-stage optimization and AR we refer to [10,11].

6. Methods to reduce goal function evaluations. Especially if optimization problems with expensive to evaluate goal functions (simula-

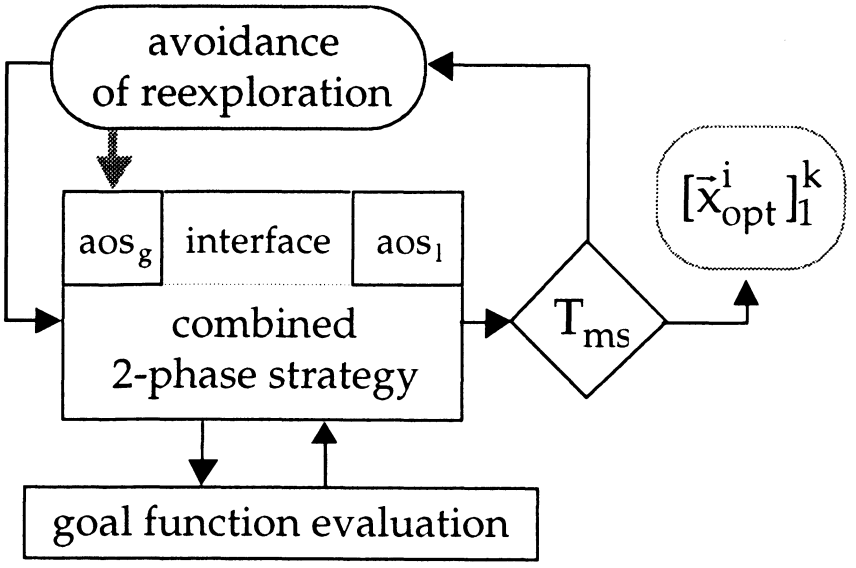


FIG. 3. *Basic structure of a multiple-stage optimization strategy* os_{ms}.

tion-based goal functions, etc.) have to be optimized, supplementary methods for reduction of optimization effort are of great importance. A very simple and obvious way to save goal function evaluations is to avoid reevaluations of search points which are generated several times during the optimization process. This can be done very easily by search of the optimization trajectory, which comprises all generated search points together with their corresponding goal function values.

Within a combined 2-phase strategy the pre-optimization phase offers an additional possibility to save goal function evaluations. This is due to the primary goal of pre-optimization, which is not to achieve high optimization quality but only a rough approximation of a promising region in the search space. This property as well as the robustness of probabilistic global optimization strategies against inaccurately evaluated goal function values makes it possible to also employ approximate goal functions values. Such approximations are performed in frequently visited regions of the search space. That way a lot of possibly very expensive goal function evaluations can be saved without a substantial loss of optimization success. Especially multiple-stage optimization makes the application of a goal function approximator very advantageous. Here, with each additional optimization stage more information about the goal function is gathered, which in turn can be exploited in subsequent optimization stages for approximation.

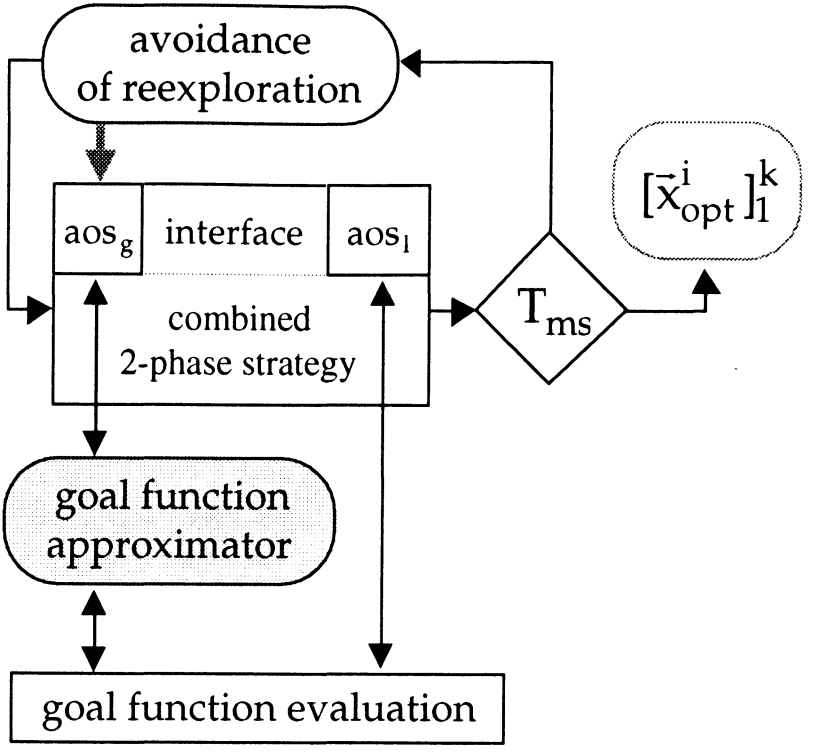


FIG. 4. *Acceleration of pre-optimization through goal function approximation.*

Fig. 4 shows the multiple-stage optimization strategy of Fig. 3 extended by a goal function approximator which is embedded between the pre-optimization process and the process of goal function evaluation.

For approximation we use a simple grid-based technique as well as a special kind of neural networks called Rectangular Basis Function Networks [4]. A more detailed description of our approach to accelerate pre-optimization by goal function approximation can be found in [12].

7. Realization. The direct optimization algorithms presented above already have been implemented and integrated into the software tool REMO (REsearch Model Optimization Package). REMO was primarily developed to automate the process of model optimization (especially optimization of simulation models). For that purpose REMO provides an interface to connect it with conventional modelling tools. Together with a modelling tool REMO enables the user

- to implement and analyze the model which has to be optimized,
- to specify the optimization problem,
- to automatically perform the model optimization process.

The architecture of the model optimization tool REMO is presented in Fig. 5.

REMO, currently running on SUN-SPARCstations 10, is composed of three basic building blocks:

- a graphical user-interface

  When working with REMO, the user has to handle a multitude of input parameters and to cope with a high quantity of output data, generated during the optimization process. In order to get a well structured user-interface which is easy to survey, it was subdivided into an input-, control-, and output part.

- a library of powerful direct optimization algorithms

  As elementary components REMO offers to the user the direct optimization strategies shown in Table 1. Outgoing from these atomar global and local optimization methods there exist the following possibilities to realize a combined 2-phase strategy:
  - Genetic Algorithm + Pattern Search,
  - Simulated Annealing + Pattern Search.

  All optimization methods can be used either in single-stage or in multiple-stage mode. For special adaptation of the implemented optimization algorithms to intricate model optimization problems REMO offers supplementary methods for

  - acceleration of the pre-optimization process by goal function approximation,
  - enhancement of the robustness to goal functions which are heavily distorted by stochastic inaccuracies.

- an interface for coupling REMO with conventional modelling tools

  In order to automate the process of model optimization a coupling of optimization and model analysis has to be realized. Therefore, a specific interface was developed, comprising a data and a synchronization part. Input parameters and goal function values are exchanged using the data part. Synchronization signals which manage the alternated switching between optimization and model analysis are exchanged using the synchronization part.

A more detailed description of REMO can be found in [9].

The direct optimization strategies which are implemented in REMO enable the user not only to deal with simulation models but also to optimize analytical models or just mathematical functions. For validation and performance analysis of our developed methods we have chosen a set of mathematical test functions. In the following Section some of these func-

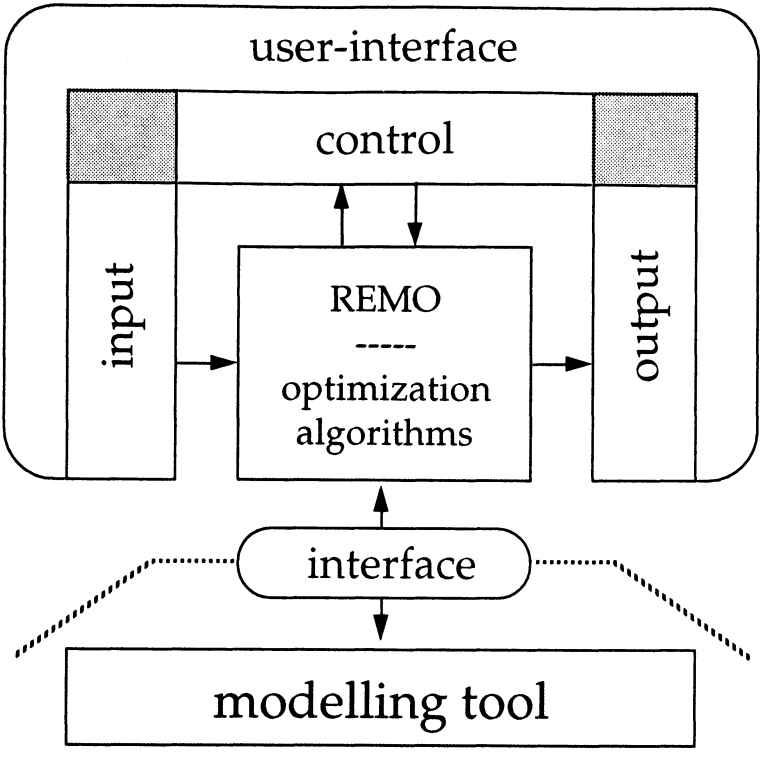


FIG. 5. *Architecture of REMO.*

tions as well as some interesting optimization results are presented and discussed.

8. Results. The optimization results presented in this Section were achieved with a combined 2-phase optimization strategy. For pre-optimization we used a Genetic Algorithm. Fine-optimization was performed by the Pattern Search algorithm of Hooke and Jeeves. The combined 2-phase optimization algorithm os_{2P} was used for single-stage optimization within a multiple-stage optimization algorithm os_{ms}. To avoid that previously found optimum points are located again in subsequent optimization stages AR (avoidance of reexploration) was applied. The method for acceleration of pre-optimization by goal function approximation, presented in Section 6, was not used.

In the following, the outcome of three extensive optimization experiments with os_{ms} is presented. We start with a brief description of the optimized mathematical test problems, which are used for maximization. Subsequently, the control parameter values for os_{ms} are considered. Finally, some optimization results are presented and discussed.

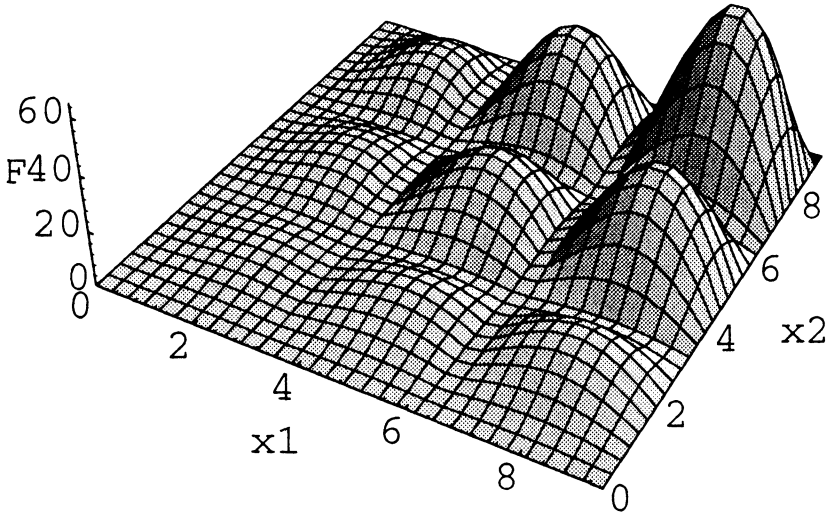


FIG. 6. *Graphical 3D-representation of the 2-dimensional goal function* F_1^2.

We demonstrate the capabilities of our developed optimization methods by means of three mathematical test problems with multimodal goal functions and a varying level of difficulty. In Table 2 the first test problem (S_1^n, F_1^n) is defined. Here, a trigonometrical goal function has to be optimized, which comprises exactly one global maximum point $\vec{x}^* = (7.98, 7.98, ..., 7.98)$ for all problem dimensions $n \in N$. Attention should be paid to the fact, that the number of local maximum points raises exponentially with an increase of the problem dimension. Fig. 6 shows a graphical 3D-representation of the 2-dimensional goal function F_1^2.

TABLE 2
Mathematical test problem (S_1^n, F_1^n).

search space	$S_1^n = \{\vec{x} \in R^n \mid 0 \leq x_i \leq 3 \cdot \pi; i \in \{1, ..., n\}\}$
goal function	$F_1^n(\vec{x}) = \left\lvert \prod_{i=1}^{n} x_i \cdot \sin(x_i) \right\rvert$

TABLE 3
Mathematical test problem (S_2^n, F_2^n).

search space	$S_2^n = \{\vec{x} \in R^n \mid 0 \le x_i \le 10; i \in \{1, ..., n\}\}$
goal function	$F_2^n(\vec{x}) = \sum_{j=1}^{10} \frac{1}{c_j + \sum_{i=1}^{n} (x_i - a_{ij})^2}$; with a_{ij} from Table 4 and c_j from Table 5

The goal function of the second mathematical test problem (S_2^n, F_2^n), defined in Table 3, is an often used benchmark function for direct optimization algorithms. Functions of this type are called Shekel functions [8]. For all problem dimensions (S_2^n, F_2^n) comprises exactly one global maximum point $\vec{x}^* = (4, 4, ..., 4)$ and nine local ones.

The coordinates $a_{ij}; i, j \in \{1, ..., 10\}$ of the maximum points of (S_2^n, F_2^n) are summarized in Table 4.

TABLE 4
Coordinates of the maximum points of test problem (S_2^n, F_2^n).

a_{ij}	$i1$	$i2$	$i3$	$i4$	$i5$	$i6$	$i7$	$i8$	$i9$	$i10$
$1j$	4	1	8	5	6	3	6	7	2	8
$2j$	4	1	8	5	6	7	2	3.6	9	1
$3j$	4	1	8	5	6	3	6	7	2	8
$4j$	4	1	8	5	6	7	2	3.6	9	1
$5j$	4	1	8	5	6	3	6	7	2	8
$6j$	4	1	8	5	6	7	2	3.6	9	1
$7j$	4	1	8	5	6	3	6	7	2	8
$8j$	4	1	8	5	6	7	2	3.6	9	1
$9j$	4	1	8	5	6	3	6	7	2	8
$10j$	4	1	8	5	6	7	2	3.6	9	1

The constants $c_j; j \in \{1, ..., 10\}$ shown in Table 5 determine the goal function values of the maximum points of (S_2^n, F_2^n).

TABLE 5
Constants which determine the maximum values of test problem (S_2^n, F_2^n).

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.5	0.6	0.7

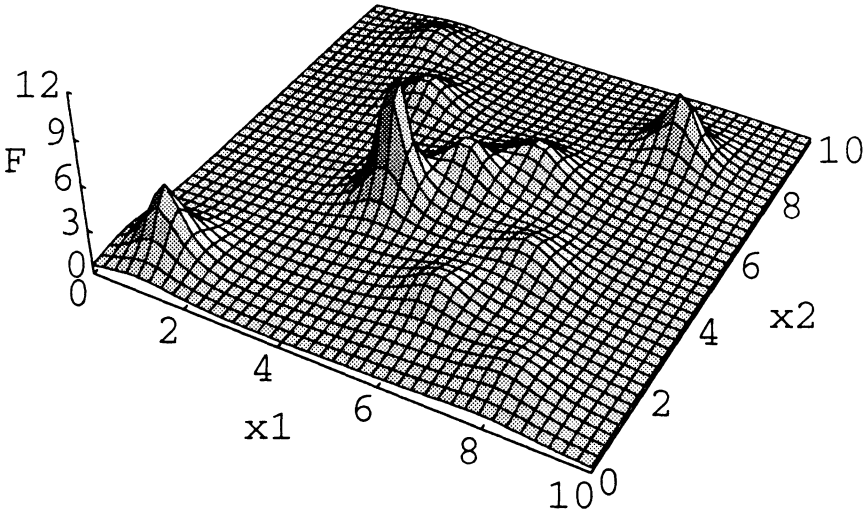


FIG. 7. *Graphical 3D-representation of* F_2^2.

Fig. 7 shows a graphical $3D$-representation of the 2-dimensional Shekel function F_2^2.

The third test problem (S_3^n, F_3^n), defined in Table 6, was developed with the intention to create a quasi-unsolvable problem.

TABLE 6
Mathematical test problem (S_3^n, F_3^n).

search space	$S_3^n = \{\vec{x} \in R^n \mid -3 \cdot \pi \leq x_i \leq 3 \cdot \pi, i \in \{1, ..., n\}\}$
goal function	$F_3^n(\vec{x}) = 1 + \prod_{i=1}^{n} \sin(x_i) + \dfrac{1}{0.05 + \sum_{i=1}^{n} (x_i - 4)^2}$

The goal function F_3^n represents a combination of a Sinus function with a one peak Shekel function. For all problem dimensions (S_3^n, F_3^n) comprises exactly one global maximum point $\vec{x}^* = (4, 4, ..., 4)$. In contrary to (S_2^n, F_2^n) however, the number of local maximum points raises exponentially with an increase of the problem dimension.

Fig. 8 shows the graphical 3D-representation of the 2-dimensional goal function F_3^2. It is easy to recognize, that with every increase of the problem dimension (S_3^n, F_3^n) more and more corresponds to the search for a pin in a haystack.

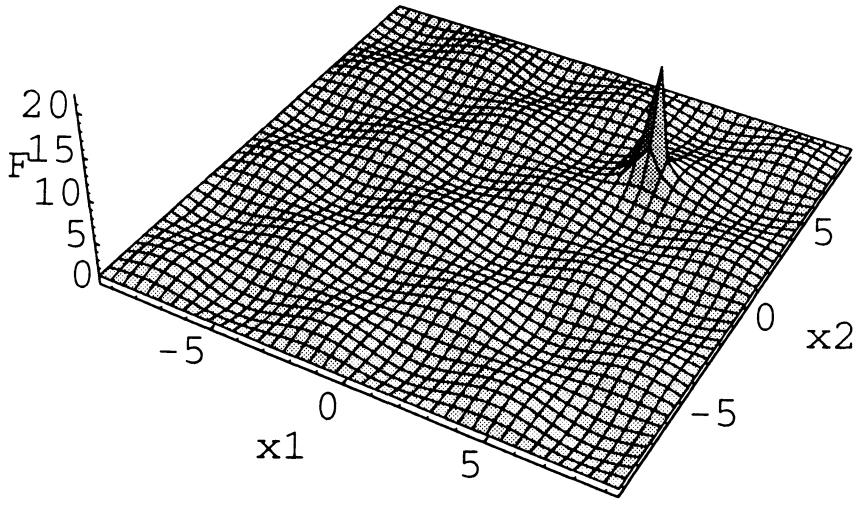


FIG. 8. *Graphical 3D-representation of* F_3^2.

Before starting our multiple-stage optimization experiments we have to determine the control parameter values for the combined 2-phase strategy os_{2P}. For the GA we chose control parameter settings recommended in literature [6]: For all considered problem dimensions $n \in \{2, ..., 10\}$ the crossover rate was set to 0.75, and the mutation rate was set to 0.075. The length of a parameter segment in the fixed-length binary strings (individuals of a population) representing parameter vectors was set to 10 bit. The parameter mc for monte-carlo initialization, the population size ps (number of individuals in a population) and the control parameter t for the termination criterion T_{po} were set dimension dependent (see Table 7). Switch-over to fine-optimization occurred, if the GA could not improve the goal function by 5% within t generations.

After pre-optimization with the GA the Pattern Search algorithm was applied in order to achieve high approximation quality. The demanded accuracy ε of the PS algorithm was set to 0.01 for each coordinate axis.

In the following, the basic outcome of three large-scale optimization experiments E_i with the multiple-stage optimization strategy os_{ms} applied to optimization problem $(S_i^n, F_i^n); i \in \{1, 2, 3\}$ is described. First of all, in Diagram 1 some fundamental results of experiment E_1 are summarized. In this experiment os_{ms} was used to maximize (S_1^n, F_1^n). With each of the nine optimization problems (S_1^n, F_1^n); $n \in \{2, ..., 10\}$ a sub-experiment was performed comprising 200 independent optimization runs with os_{ms}.

TABLE 7
Dimension dependent control parameter settings of the Genetic Algorithm.

Optimization Experiment E_1									
n	2	3	4	5	6	7	8	9	10
mc	40	40	40	60	60	60	80	80	80
ps	20	20	20	30	30	30	40	40	40
t	2	3	4	4	5	6	6	7	8

Experiments E_2, E_3					
n	2	4	6	8	10
mc	60	80	80	80	80
ps	30	40	40	40	40
t	4	4	5	7	9

In each multiple-stage run a sequence of 20 maximum points was generated. Hence, in one sub-experiment altogether 4000 maximum points were localized.

The x-axis of Diagram 1 shows the problem dimension n. The columns represent the optimization effort oe (average number of goal function evaluations) required in one optimization stage. The white (grey) part of a column shows the optimization effort of the GA (PS). The lines show the optimization success su(s) achieved after $s \in \{1, 2, 3\}$ optimization stages, which is defined as follows:

$$su(s) = \sum_{i=1}^{s} p^i_{\vec{x}^*,\varepsilon}.$$

s denotes the optimization stage. $p^i_{\vec{x}^*,\varepsilon}$ is the empirical probability of finding the global maximum point $\vec{x}^*$ for the first time in optimization stage i with accuracy ε:

$$p^i_{\vec{x}^*,\varepsilon} = \frac{\text{number of first } (\vec{x}^*, \varepsilon) - \text{hits in stage } i}{\text{total number of stage } i \text{ optimization runs}}.$$

Curve su(1) [GA+PS] of Diagram 1 shows, that os_{ms} already after the first optimization stage (single execution of $os_{2P} = GA + PS$) achieves an almost optimal success rate, which only slowly declines with an increase of the problem dimension. Obviously, (S_1^n, F_1^n) is well-suited for pre-optimization with a GA. This is due to the exponentially growing maxima of goal function F_1^n (see Fig. 6), which almost always lead the GA into the region of the global maximum point $\vec{x}^*$. Optimization problems which are much more difficult to pre-optimize with a GA are considered in E_2 and E_3.

The black line su(1) [PS] shows the optimization success after the first optimization stage, if multiple-stage optimization is not performed with os_{2P} but only with PS, started at a randomly chosen start point

$\vec{x}_{\text{start}}$. Obviously, the omission of pre-optimization entails a drastic decline of optimization success.

The two curves su(2) [GA+PS] and su(3) [GA+PS] show, that each further optimization stage causes a monotonical increase of the optimization success. After the third stage the optimization success has almost reached the maximum value of 1 for all considered problem dimensions.

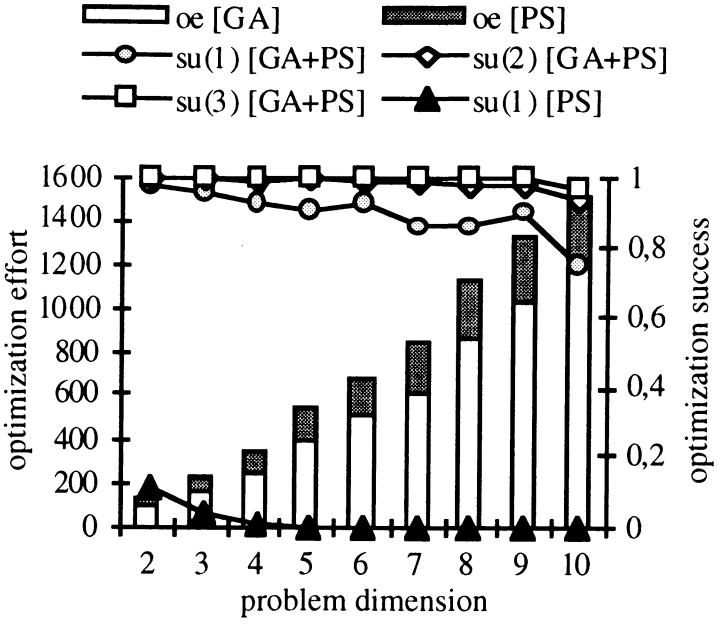


DIAGRAM 1. *Optimization effort and optimization success achieved in experiment* E_1.

oe [PS] indicates, that the computational effort required for fine-optimization raises linearly with an increase of the problem dimension. Because exploration of the whole search space is a much more difficult task than local optimization in a particular search space region, this does not apply to the pre-optimization effort oe [GA]. Here, attention must be paid to the fact, that with every increase of the problem dimension the size of the search space raises exponentially.

The approximation quality of the computed optimization results is not considered in a special diagram, because the Pattern Search algorithm ensures, that the user specified quality requirement of $\varepsilon = 0.01$ is always met.

Maximization of optimization success is not the one and only goal of multiple-stage optimization. Another very important objective is to generate heterogeneous sequences of optimum points. This is the task of AR (avoidance of reexploration) which has to ensure, that already found

optimum points are not localized again in subsequent optimization stages. Diagram 2 shows the effect of AR on the optimization result of os_{ms}. For each optimization stage $s \in \{1, ..., 20\}$ the number of first localizations of the 9 maximum points of the 2-dimensional test problem (S_1^2, F_1^2) is presented.

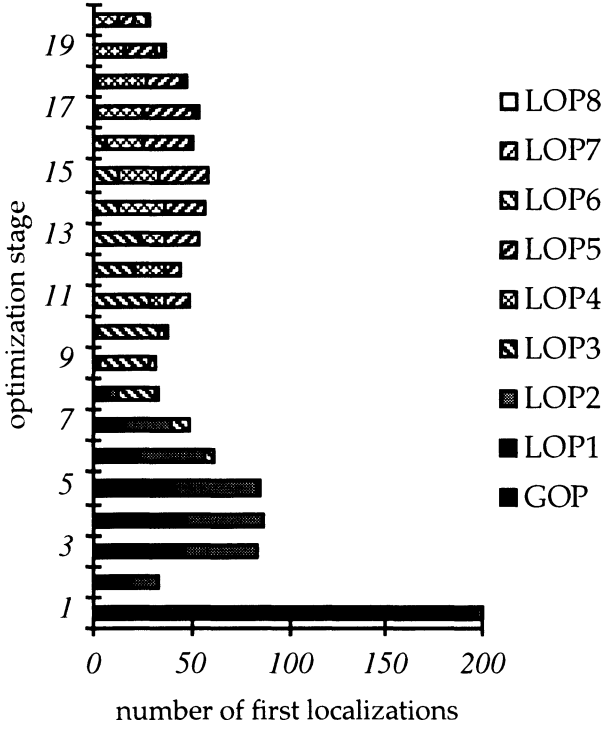


DIAGRAM 2. *Number of first localizations of the nine maximum points of test problem* (S_1^2, F_1^2) *in optimization stage* $s \in \{1, ..., 20\}$.

First of all, Diagram 2 very impressively shows that the most prominent maximum points of (S_1^2, F_1^2) are localized in early optimization stages. In accordance with the results of Diagram 1, the global maximum point $\vec{x}^*$ (GOP) is localized 196 of 200 times in the first optimization stage. In the following 7 optimization stages the two local maximum points LOP1 and LOP2 with the second best goal function value are found for the first time. Subsequently follows LOP3 with the third best goal function value and so on. Mind, that in case of deactivating AR, the localization of the global maximum point $\vec{x}^*$ would dominate not only the first but all optimization stages.

Diagram 3 summarizes the basic results of experiment E_2. For each of the five optimization problems (S_2^n, F_2^n); $n \in \{2, 4, 6, 8, 10\}$ a subexperiment was performed comprising 200 independent optimization runs

with os_{ms}. In each multiple-stage run a sequence of 20 maximum points was generated.

Curve su(1) of Diagram 3 clearly shows, that (S_2^n, F_2^n) is much more difficult to pre-optimize than test problem (S_1^n, F_1^n). This is due to the vicious properties of the Shekel function. As shown in Fig. 7, F_2^n consists of 10 peaks which have the property to contract more and more with every increase of the problem dimension. From dimension $n = 4$ onwards the Shekel-peaks deform to extreme thin pins which only differ in their heights. This feature makes it very difficult for the pre-optimization algorithm to distinguish the region of the global maximum point from sub-optimal regions.

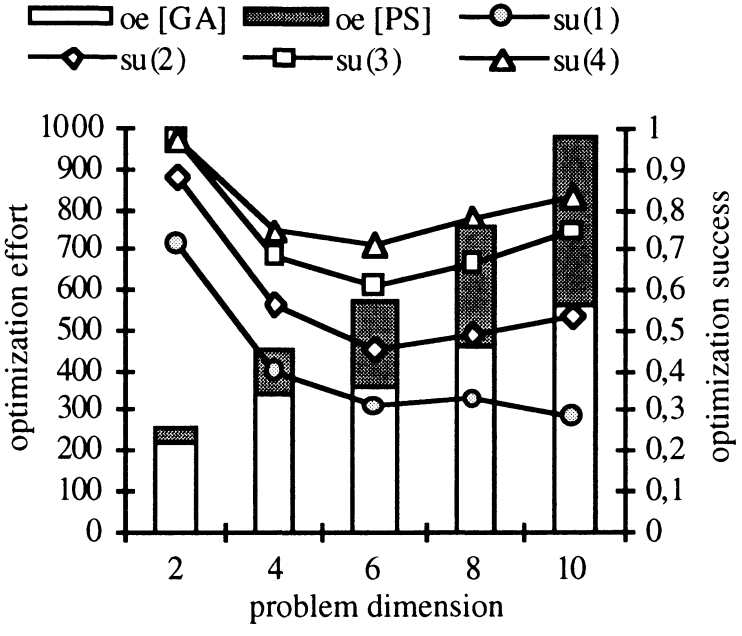


DIAGRAM 3. *Optimization effort and optimization success achieved in experiment E_2.*

The decline of su(1) clearly shows the limits of os_{2P}, especially the limits of pre-optimization with GA (and SA which behaves similarly). At $n \in \{6, 8, 10\}$ su(1) has almost dropped to the probability of finding $\vec{x}^*$ with PS started at a randomly chosen start point. In case of intricate optimization problems like (S_2^n, F_2^n) also a considerable increase of pre-optimization effort (increase of the control parameter t of T_{po}) would not cause a substantial improvement of su(1). In that case it turned out to be much more effective to perform additional optimization stages instead of only one expensive execution of os_{2P}. The curves su(i), $i \in \{2, 3, 4\}$ demonstrate that every further optimization stage substantially improves the optimization

success. After the fourth optimization stage the empirical probability of finding $\vec{x}^*$ already raised up to more than 0.7 for all considered problem dimensions $n \in \{2, 4, 6, 8, 10\}$. This advantageous property of multiple-stage optimization is mainly caused by avoidance of reexploration which effects, that with each optimization stage another region of the search space is explored. However, as we will see in E_3, this only works successfully, if the optimization problem comprises a very limited number of optimum points.

The low pre-optimization effort oe [GA] shown in Diagram 3 also indicates that (S_2^n, F_2^n) must be a non-trivial problem. For the higher problem dimensions $n \in \{4, ..., 10\}$ oe [GA] fairly decreased compared with the pre-optimization effort of E_1. This behaviour is due to the extreme thin peaks of (S_2^n, F_2^n), which cause that the termination criterion T_{po} of the GA very often is fulfilled prematurely. In this case, the GA cannot achieve an improvement of the goal function by 5% already after the first t computed generations.

The average number of goal function evaluations required for fine-optimization oe [PS] as well as the approximation quality of the fine-optimization results are very similar to E_1. Although (S_2^n, F_2^n) turned out to be very difficult for pre-optimization the PS algorithm works on (S_2^n, F_2^n) without any difficulties. AR also works very successfully on the Shekel function and ensures heterogeneous sequences of maximum points. After 10 optimization stages os_{ms} had already localized an average of 7 of the 10 maximum points of (S_2^n, F_2^n). Just as in E_1, the most prominent maximum points were localized in early optimization stages.

Diagram 4 summarizes the basic results of experiment E_3. Just as in E_2, for each of the five optimization problems (S_3^n, F_3^n); $n \in \{2, 4, 6, 8, 10\}$ a sub-experiment was performed comprising 200 independent optimization runs with os_{ms}. In each multiple-stage run a sequence of 20 maximum points was generated.

As shown in Fig. 8, F_3^n comprises exactly one global maximum point which is homogeneously surrounded by many local maximum points having all the same small goal function value. With an increase of the problem dimension the peak of the global maximum is getting thinner while the number of local maximum points raises exponentially. Exactly this vicious property makes an efficient solution of (S_3^n, F_3^n) impossible for os_{ms} in higher problem dimensions.

As can be observed in Diagram 4, os_{ms} still works very well at problem dimension $n = 2$. From dimension $n = 4$ onwards however, the optimization success su(1) drastically goes down to 0. Because of the multitude of local maximum points additional optimization stages also fail. The optimization effort oe and the approximation quality of the computed optimization results behave as in E_2.

Summing-up, the optimization results presented above show, that our multiple-stage optimization algorithm os_{ms} is able to efficiently and effectively perform a systematic analysis of the extreme points of multimodal

goal functions. This methodology also enables to successfully deal with intricate optimization problems like the Shekel function. Multiple-stage optimization is mainly based on the power and efficiency of combined 2-phase optimization, which tries to employ its atomar components only in that phases of the optimization process where they have full scope to unfold their abilities.

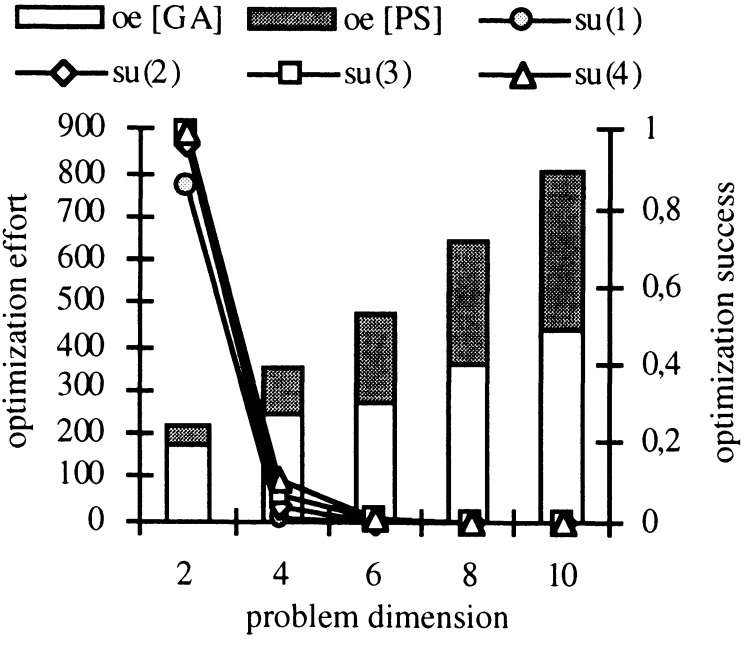


DIAGRAM 4. *Optimization effort and optimization success achieved in experiment* E_3.

However, the example of (S_3^n, F_3^n) clearly shows, that there also exist optimization problems where os_{ms} fails. This is not particularly surprising because global parameter optimization has turned out to belong to the class of np-hard problems. Nevertheless there is good reason to suppose that artificial problems like (S_3^n, F_3^n) do not very often occur in practice.

9. Conclusions. In this paper the basic structure of a combined 2-phase optimization strategy was presented, which is composed of a direct global and a direct local optimization algorithm. The primary goal of this hybrid method is to combine the advantages of global and local search, i.e.

- to achieve a high success rate (high probability of finding the global optimum),
- to generate optimization results which satisfy user-specified quality demands,
- to ensure high efficiency.

The excellent heuristic properties of a combined 2-phase strategy can be viewed as the basic requirement for multiple-stage optimization. Here the most important goals are

- to monotonically increase the optimization success with every optimization stage,
- to generate heterogeneous sequences of optimum points,
- to localize the most prominent optimum points in early optimization stages.

Altogether, combined 2-phase and multiple-stage optimization can be viewed as a substantial improvement compared to existing direct optimization methods like GA, SA, and PS. For that reason, our approach is not only applicable to mathematical test problems but also qualified for parameter optimization of expensive to evaluate simulation-based goal functions. The optimization results presented in Section 8 give a good impression of the potentialities but also the limits of our developed methods.

The topics of our future work concern adaptation of combined 2-phase optimization to combinatorial optimization problems, dynamic adjustment of control parameters to the surface of a given goal function, further exploitation of the multitude of data stored in the optimization trajectory, and parallelization of our developed algorithms.

10. Acknowledgements. We want to thank Prof. D. Schmid for his encouragement and support of our work. We also thank our students, especially H. Renfranz, T. Sommer, D. Haag, J. Gramlich and A. Kehl for their engagement and contributions.

REFERENCES

[1] E. AARTS, J. KORST, *Simulated Annealing and Boltzmann Machines*, Wiley, 1990.
[2] D.E. GOLDBERG , *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.
[3] R.A. HOOKE, T.A. JEEVES, *Direct Search Solution for Numerical and Statistical Problems*, Journal ACM 8, pp. 212–221, 1961.
[4] K.-P. HUBER, M.R. BERTHOLD, *Building Precise Classifiers with Automatic Rule Extraction*, Proceedings of the IEEE International Conference on Neural Networks ICNN'95, Perth, Western Australia, Vol. 3, pp. 1263–1268, 1995.
[5] Z. MICHALEWICZ, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, 1992.
[6] J.D. SCHAFFER, R.A. CARUANA, L.J. ESHELMAN, R. DAS, *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization*, Proceedings of the third International Conference on Genetic Algorithms, June 4–7, George Mason University, pp. 51–60, 1989.
[7] H.-P. SCHWEFEL, *Numerical Optimization of Computer Models*, Wiley, 1981.
[8] J. SHEKEL, *Test Functions for Multimodal Search Techniques*, in Fifth Annual Princeton Conference on Information Science and Systems, 1971.
[9] M. SYRJAKOW, H. SZCZERBICKA, *Optimization of Simulation Models with REMO*, Proceedings of the European Simulation Multiconference ESM'94, Barcelona, Spain, June 1–3, pp. 274–281, 1994.

[10] M. SYRJAKOW, H. SZCZERBICKA, *Simulation and Optimization of Complex Technical Systems*, Proceedings of the 1995 Summer Computer Simulation Conference SCSC'95, Ottawa, Ontario, Canada, July 24–26, pp. 86–95, 1995.

[11] M. SYRJAKOW, H. SZCZERBICKA, *Combination of Direct Global and Local Optimization Methods*, Proceedings of the International Conference on Evolutionary Computing ICEC'95, Perth, Western Australia, November 29 – December 1, pp. 326–333, 1995.

[12] M. SYRJAKOW, H. SZCZERBICKA, M.R. BERTHOLD, K.-P. HUBER, *Acceleration of Direct Model Optimization Methods by Function Approximation*, Proceedings of the 8th European Simulation Symposium ESS'96, Genoa, Italy, October 24–26, Volume II, pp. 181–186, 1996.

WHAT ARE GENETIC ALGORITHMS? A MATHEMATICAL PRESPECTIVE*

MICHAEL D. VOSE†

Abstract. This talk presents a "big picture" view of genetic search in a general, abstract setting. It limits consideration to simple generational versions of time invariant Markovian GAs (the next generation depends only upon the current population) with the aim of uncovering, in general terms, their inherent emergent behavior. Along the way, a few issues related to classical "GA theory" are touched upon so as to sketch the context out of which the material presented in the remainder of the talk grew, and to indicate a few of the problems it partially addresses.

1. Introduction. This talk concerns simple generational versions of time invariant Markovian GAs (the next generation depends only upon the current population). The introduction briefly touches upon a very few of the many parts of the mosaic that, historically, has been referred to as "GA threory", so as to provide context and contrast for the remainder of the talk. There results are presented which were motivated by, and partially respond to, shortcomings of the classical theory. It is assumed throughout that the audience is acquainted with the jargon commonly used in the field.

2. Intrinsic parallelism and no free lunch. Beginning with John Holland (if not before) in his book "adaptation in natural and artificial systems" (1975), the concept of schema was introduced as a mechanism whereby genetic search could be analyzed and understood. Although sets of elements, as opposed to individuals, were emphasized, Holland consistently stressed the importance of creating *new* samples (elements not in the current or past populations) throughout his development. To summarize the central ideas:

- Collections of elements (schemata) in the population gain representation in rough correspondence to their average fitness.
- An above average schema increases its share of trials exponentially.
- Genetic search rapidly explores sets of schemata of above average performance which can be produced from each other by relatively few crossovers, while not significantly slowing the overall search for better optima.

Presented in harmony with this schema-based view was the conjecture that by virtue of an element simultaneously belonging to a plethora of schemata (which is what *intrinsic parallelism* refers to), a tremendous flow of information is available to the genetic paradigm by which it gains tremendous power.

*This resarch was supported by the National Science Foundation IRI-9224917.

†C.S. Dept., 107 Ayres Hall, The University of Tennessee, Knoxville, TN 37996-1301. Email: vose@cs.utk.edu

This latter point – power flowing from intrinsic parallelism – has been laid to rest by the "no free lunch" theorem (Wolpert, D. and W. Macready, 1994). The NFL (no free lunch) theorem, roughly speaking, asserts that all black box search (BBS) strategies have similar performance when considered over all test functions[1]. The basic principle is straightforward. Suppose it is not a priori known what value will be returned by the objective function; this may be regarded as uncertainty as to what function g is being optimized. By defining the value to be returned – in an arbitrary way – when evaluating at a new point in the domain, one can arrange for any behavior to be manifest while incrementally defining the objective function as the search progresses. When all points in the finite domain have been explored, the objective function will have thereby been defined, and no uncertainty remains as to which function g was to be optimized.

Since the values returned as described above may be arbitrarily chosen, a GA can be made – through suitable choices – to perform either better or worse than any (other) BBS strategy applied to a given (typical) function f.[2] Of course, such performance might be manifest only when the GA is applied to an objective function g which differs from f; the incremental definition as described above need not result in the determination of f as the function which the GA was optimizing. In its full generality, the NFL theorem has the consequence that, on average, the effectiveness of genetic search is no better than that achieved by enumeration: all BBS strategies have equivalent performance; they merely exhibit it on different functions.[3]

3. Building blocks and the schema theorem. The belief in the power of intrinsic parallelism spawned attendant conjectures, like, for instance, the notion that mutation is an inferior search operator, and the idea that high cardinality alphabets yield inferior results. For a period of time, perspectives assuming the central importance of the *number* of schemata which survived, or that were processed, or which simply existed (given some particular representation), became, for better or for worse, dominant.

Popularized by David Goldberg in his book "Genetic Algorithms, in Search, Optimization, & Machine Learning" (1989), the *building block hypotheses* is the assertion that highly fit, short, low order schemata are recombined ("processed" by crossover) to lead to better performance (i.e., to create new, more highly fit elements). By the light the NFL theorem sheds, a GA will work well on some problems, poorly on others, and on average no better than enumeration. Under that light, the building block hypotheses might best be interpreted as a conjecture as to *what kinds* of functions a GA would perform well on: those for which new and better

[1] from a finite domain $\mathcal{X}$ to a finite codomain $\mathcal{Y}$.

[2] Provided performance is measured with respect to the sequence of values encountered during search, ignoring repeats.

[3] There are, naturally, conditions and technicalities; they do not, however, detract from the negative conclusions about the power of intrinsic parallelism.

elements will be produced, by crossover, from highly fit, short, low order schemata.

This leads naturally to the question: What new elements *are* produced by a genetic algorithm? The *schema theorem* is an inequality giving a lower bound on the expected number of instances of schemata in the next generation. The lower bound is zero, however, for all schemata not already present in the current population. A little thought will reveal that *which* elements outside of the current population become members of the next generation is crucial to the course of genetic search. To worry about *how many* within the current population survive, are processed, or exist, is to miss an important point.

A devastating consequence follows from the fact that the schema theorem does not nontrivially address what proportions, of which strings, schemata are expected to be comprised. Without such knowledge, schemata utilities become unknown in the next generation. Consequently, the schema theorem's ability to predict – even about strings within the current population – evaporates, in general, after a single generation.

Apparently neglected for some period of time, a paper by Bridges and Goldberg ("An analysis of reproduction and crossover in a binary-coded genetic algorithm") remedied that situation in 1987. A formula was presented which gave the expected representation of *every* schema in the next generation. Although limited to proportional selection, one point crossover, and zero mutation, their result was progress, in that an inequality had been sharpened to an equality, and the focus on "how many schemata?" had been broadened – in terms of proven analytical results – to include the question "*which new schemata?*".

4. Sampling error and facetwise analysis. Facetwise analysis (ignoring or eliminating problem aspects; treating the details of their interaction as inconsequential) emerged as a popular method to reason (albeit heuristicly) in the face of complexity and uncertainty. This practice was fostered by the reality that genetic algorithms are stochastic and nonlinear.

In circumstances where the building block hypothesis was thought appropriate, the idea arose that the GA's inherent mechanism could be an internal source of *sampling error*, potentially interfering with anticipated outcomes. Selection, for example, might choose unrepresentative parents, mutation or crossover could produce unrepresentative offspring. From the schemata perspective, the population size might be too small, setting the stage for an unrepresentative estimate of schemata utilities. Considerations like these spawned alternative operators less prone to sampling error, and also influenced theories of population sizing. Taking the limit, as population size goes to infinity, ameliorates these concerns, however. The resulting dynamics are deterministic and equivalent to that induced by the function which produces the expected next generation.

In the field of genetic algorithms, this methodology was initiated by

Holland and carried forward by his students. For example, the central ideas summarized in section 2, as well as the schema theorem, are statements about expectations or are justified by an implicit appeal to a trajectory of expectations (as, for example, the exponential increase in above average schemata). Not to suggest anything amiss with facetwise analysis, it is nevertheless fair to say that for a period of time the description of what, precisely, it established, had too frequently been left to the imagination or treated in a cavalier manner. The following subsections illustrate *fallacious* conclusions based on interpretations of "results" of "GA theory" which too commonly are made.

5. Static schema analysis. Consider the objective function $\{(00, 2.7), (01, 1.0), (10, 1.0), (11, 1.1)\}$ and assume the crossover rate χ is 0.6. The nontrivial schemata, with their associated fitness, are

$$0* \longleftrightarrow 1.85$$
$$*0 \longleftrightarrow 1.85$$
$$1* \longleftrightarrow 1.05$$
$$*1 \longleftrightarrow 1.05$$

The optimal element is over 145% more fit than any other, and the schemata containing it ($0*$ and $*0$) are over 76% more fit than their competitors; there is absolutely no deception (the schemata containing 00 win every "competition"). Schemata gain representation in correspondence to their fitness, and above average schemata increase exponentially; thus $0*$ and $*0$ – if initially present in the population – will quickly come to dominate. These schemata cannot be disrupted by crossover (they are too short), so in the absence of mutation, the exploratory operators cannot interfere with the exploitation of these building blocks. Sampling error is a potential problem, but that can be tamed by choosing population size sufficiently large.

It can safely be concluded in this case that given a zero mutation rate and an initial population containing fixed nonzero proportions of every element, a GA will with high probability converge to the optimal, provided the population size is not too small.

The conclusion of the previous paragraph sounds reasonable. It is false however. In fact, there exists a function h of the crossover rate χ which increases to infinity as χ increases to 1.0, such that the previous paragraph is false with respect to the function $\{(00, h(\chi)), (01, 1.0), (10, 1.0), (11, 1.1)\}$.

It is interesting to note, for this example, what the problem is *not*. The computation of schema utilities with respect to a uniform population (equal representation of strings), instead of with respect to the initial population from which convergence to a suboptimal is likely, is *not* to blame. Initial populations exist for which the initial conditions hold with respect to nonuniform schema utilities[4], yet convergence to a suboptimal remains

[4]The optimal element remains over 145% more fit than any other, and the schemata

the likely event (even though fixed nonzero proportions of every element are initially present and the population size may be arbitrarily large). The principal behind the example is the fact that the schema theorem's ability to predict – even about strings within the current population – evaporates, in general, after a single generation.

6. Neglecting finite population effects. Because taking the limit, as population size goes to infinity, yields a deterministic system whose dynamics coincide with that induced by the function which produces the expected next generation, the infinite population model can be used to determine the expected behavior of a genetic algorithm.

The conclusion of the previous paragraph is slippery, because the statement is so vague. Following are specific interpretations, each of which is a *false* assertion.

- The path followed by the infinite population model is the expected path followed by a GA.
- If the infinite population model converges to a population q, then q is representative of the GA's steady state distribution (as a Markov chain).
- If the infinite population model indicates that certain elements are likely to emerge in early generations, then it is probable for these elements to likewise emerge during genetic search.

This list of errors is not exhaustive, but these are not uncommon misconceptions.

Equally illusory is the ignis fatuus that because such notions are false, the infinite population model is not of fundamental importance to the theory of genetic search.

7. Convergence proofs. Genetic algorithms have been touted as robust optimization methods, similar to simulated annealing in some sense. There are differences of course, one of which being that whereas simulated annealing has a global convergence theory, it is arguably the case that an unimpeachable convergence theory for the simple genetic algorithm does not exist.

Even though there are a variety of "convergence results", those which apply to a simple GA without requiring its alteration amount to little more than the following when distilled to their essence:

> Visit each state, remembering the best state encountered. Therefore, as search progresses and every state is eventually encountered, the best state so far visited will converge to the best state which could be visited – the global optimum.

---

containing it (0* and *0) remain over 76% more fit than their competitors when nonuniform utilities are computed with respect to the initial population.

Unless severe restrictions are placed on the objective function, this virtual tautology, suitably dressed in the language of Markov chains, is the current state of the art – at least in terms of proven analytical results. To be fair, the convergence theory for simulated annealing is, from a practical applied perspective, hardly better.

In light of the no free lunch theorem, this may very well be as it should be. But the NFL theorem only addresses the sequence of values encountered during search, ignoring repeats. It does not prohibit a theory which could relate observed behavior to mathematical objects determined by the search strategy and the objective function. A nontrivial convergence theory is beginning to appear in this direction, though it speaks to inherent emergent behavior and not to function optimization.

8. Framework. As intimated in the introduction, the majority of theoretical work on genetic search has been facetwise (which, again, is not to suggest anything amiss with the practice of facetwise analysis). It was not until recently that more holistic attempts to deal with the complexities of genetic search were initiated in an attempt to address the shortcomings of incomplete information. As will become apparent, schemata are left behind, playing no part in the development. This is not to suggest that they have no role to play, but their natural place in the general landscape has yet to be discovered.

The view presented is general and abstract, dealing with a class of black box search strategies referred to as Random Heuristic Search (RHS). The simple genetic algorithm is a *special case* of the algorithms discussed. The advantage of this approach is to focus on basic principles by which inherent emergent behavior can be initially approached before delving into specific details. Even so, the analysis proceeds from simple objects encoding *complete* information, rather than a starting point based on preconceived (traditional) designs or assumptions about what may safely be ignored.

9. Representation. Random heuristic search can be thought of as an initial collection of elements P_0 chosen from some search space Ω of finite cardinality n together with some *transition rule* τ which from P_i will produce another collection P_{i+1}. In general, τ will be iterated to produce a sequence of collections

$$P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \ldots$$

The beginning collection P_0 is referred to as the *initial population*, the first population (or *generation*) is P_1, the second population (or generation) is P_2, and so on. Populations are multisets.

Not all transition rules are allowed. Obtaining a good representation for populations is a first step towards characterizing admissible τ. Define the *simplex* to be the set

$$\Lambda \;=\; \{< x_0, \ldots, x_{n-1} > : \; \mathbf{1}^T x = 1, \; x_j \geq 0\}$$

where angle brackets $< \cdots >$ denote a tuple which is to be regarded as a column vector and **1** denotes the column vector of all 1 s. An element p of Λ corresponds to a population according to the following rule for defining its components

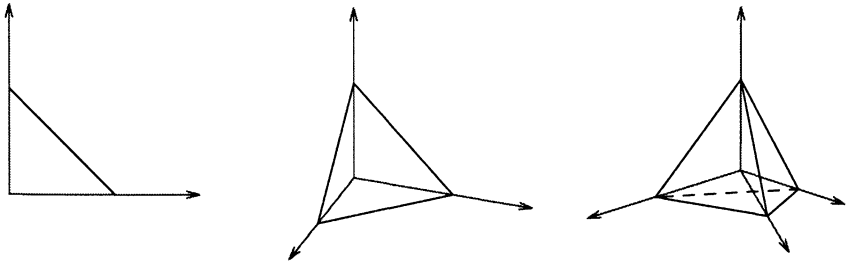
p_j = the proportion in the population of the j th element of Ω

For example, if $\Omega = \{0, 1, 2, 3, 4, 5\}$ then $n = 6$. The population $\{1, 0, 3, 1, 1, 3, 2, 2, 4, 0\}$ is represented by $p = < .2, .3, .2, .2, .1, .0 >$, given that

coordinate	corresponding element of Ω	percentage of P_0
p_0	0	2/10
p_1	1	3/10
p_2	2	2/10
p_3	3	2/10
p_4	4	1/10
p_5	5	0/10

The cardinality of each generation $P_0, P_1, \ldots$ is a parameter r called the *population size.* Hence the proportional representation given by p unambiguously determines a population once r is known. The vector p is referred to as a *population vector.* The distinction between population and population vector will often be blurred, because the population size is usually fixed. In particular, τ may be thought of as mapping the current population vector to the next.

To get a feel for the geometry of the representation space, Λ is shown in the following sequence of diagrams for n equals 2, 3, and 4. The figures represent Λ (a line segment, triangle, and solid tetrahedron). The arrows show the coordinate axes of the ambient space (the projection of the coordinate axes are being viewed in the second figure, which is three dimensional, and in the last figure where the ambient space is four dimensional).



In general, Λ is a tetrahedron of dimension $n - 1$ contained in an ambient space of dimension n. Note that each vertex of Λ corresponds to a unit basis vector of the ambient space; Λ is their convex hull. For example,

the vertices of the solid tetrahedron (right most figure) are at the basis vectors $<1,0,0,0>$, $<0,1,0,0>$, $<0,0,1,0>$, and $<0,0,0,1>$. Assuming that Ω is the ordered set $\{0,1,2,3\}$, they correspond (respectively) to the following populations: r copies of 0, r copies of 1, r copies of 2, and r copies of 3. The center diagram will later be used as a schematic for general Λ, representing it for arbitrary n.

It should be realized that not every point of Λ corresponds to a finite population. In fact, only those rational points expressible with common denominator r correspond to populations of size r. They are the intersection of a rectangular lattice of spacing $\frac{1}{r}$ with Λ:

$$\frac{1}{r} X_n^r \;=\; \frac{1}{r}\{<x_0,\ldots,x_{n-1}>:\, x_j \in \mathcal{Z},\, x_j \geq 0,\, \mathbf{1}^T x = r\}$$

where $\mathcal{Z}$ denotes the set of integers. As $r \to \infty$, these rational points become dense in Λ. The next theorem makes this precise. Since, without a priori knowledge of r, a rational point may represent arbitrarily large populations, a point p of Λ carries little information concerning population size. A natural view is therefore that Λ corresponds to populations of indeterminate size. This is but one of several useful interpretations. Another is that Λ corresponds to sampling distributions over Ω: since the components of p are non negative and sum to 1, p may be viewed as indicating that i is sampled with probability p_i.

THEOREM 9.1. *Let $p \in \Lambda$ denote an arbitrary population vector for population size r, and let ξ denote an arbitrary element of Λ. Then*

$$\sup_{\xi} \inf_{p} \|\xi - p\| \;=\; O(1/\sqrt{r})$$

where the constant (in the "big oh") is independent of the dimension of Λ.

In summary, random heuristic search appears to be a *discrete dynamical system* on Λ through the identification of populations with population vectors. That is, there is some transition rule

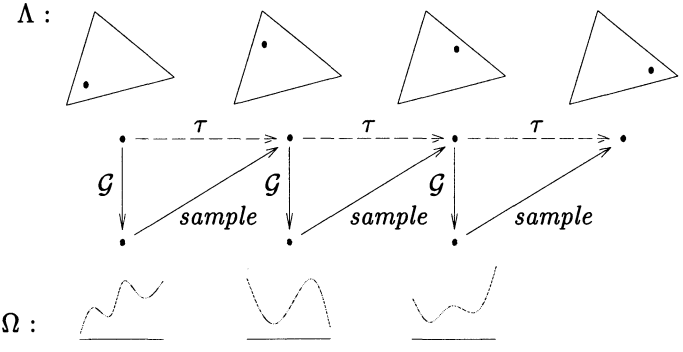
$$\tau : \Lambda \longrightarrow \Lambda$$

and what is of interest is the sequence of iterates beginning from some initial population vector p

$$p, \quad \tau(p), \quad \tau^2(p), \quad \ldots$$

This view is incomplete however, because the transitions are in general nondeterministic and not all transition rules are allowed. In the next section the stochastic nature of τ will be explained and admissible τ will be characterized.

10. Nondeterminism. Given the current population vector p, the next population vector $\tau(p)$ cannot be predicted with certainty because τ is stochastic. It is most conveniently thought of as resulting from r independent, identically distributed random choices. Let $\mathcal{G} : \Lambda \to \Lambda$ be a *heuristic function* (heuristic for short) which given the current population p produces a vector whose ith component is the probability that the ith element of Ω is chosen (with replacement). That is, $\mathcal{G}(p)$ is that probability vector which specifies the sampling distribution by which the aggregate of r choices forms the next generation. A transition rule τ is admissible if it corresponds to a heuristic function $\mathcal{G}$ in this way. The following diagram depicts the relationship between p, Λ, Ω, $\mathcal{G}$, and τ through a sequence of generations (the illustration does not correspond literally to any particular case, it depicts how transitions between generations take place in general):



The triangles along the top row represent Λ, one for each of four generations. Each Λ contains a dot representing a population. These same populations are also represented in the second row with dots; τ maps from one to the next. The transition arrow for τ is dashed to indicate that it is an induced map, computed by following the solid arrows. The lower row of dots represent images of populations under $\mathcal{G}$. Below each is a curve, suggesting the sampling distribution over Ω which it represents. The line segments in the bottom row represent Ω.

The transition from one generation (upper dot) to the next proceeds as follows. First $\mathcal{G}$ is applied to produce a vector (lower dot) which represents a sampling distribution (curve) over Ω (line segment). Next, r independent samples with replacement (represented in the diagram by *"sample"*) are made from Ω according to this distribution to produce the next generation.

For example, let $\Omega = \{0, 1, 2, 3\}$ and suppose the heuristic is

$$\mathcal{G}(p) \quad = \quad <0, p_1, 2p_2, 3p_3> / \sum ip_i$$

Let the initial population be $p = <.25, .25, .25, .25>$. Because $\mathcal{G}(p) = <0, 1/6, 1/3, 1/2>$, the probability of sampling 0 is 0, of sampling 1 is $1/6$, of sampling 2 is $1/3$, and of sampling 3 is $1/2$. With population

size $r = 100$, the transition rule corresponds to making 100 independent samples, with replacement, according to these probabilities.

A plausible next generation is therefore $\tau(p) = <0, .17, .33, .50>$. Note that the sampling distribution $\mathcal{G}(p)$ used in forming the next generation $\tau(p)$ depends on the current population p. Going one generation further, the new population is $\tau(p)$ and the sampling distribution for producing the next generation is given by $\mathcal{G}(\tau(p)) \approx < 0, .07, .28, .64 >$. It is therefore plausible that the second generation could be $\tau^2(p) = <0, .07, .28, .65>$.

Note the conceptually dual interpretation of Λ. It serves as both the space of populations and as the space of probability distributions over Ω. The first natural question is: What is the expected next generation?

THEOREM 10.1. *Let p be the current population vector. The expected next population vector is $\mathcal{G}(p)$.*

A more specific question is: Given current population p, what is the probability that the next generation is q? Let $\theta \in [0, 1]$ be defined by the following form of Sterling's theorem (for $x \in \mathcal{Z}^+$)

$$x! = \left(\frac{x}{e}\right)^x \sqrt{2\pi x} \exp\left\{\frac{1}{12x + \theta}\right\}$$

The function $\theta(x)$ appears in the next theorem.

THEOREM 10.2. *Let p be the current population vector. The probability that $q \in \frac{1}{r} X_n^r$ is the next population vector is*

$$r! \prod \frac{(\mathcal{G}(p)_j)^{rq_j}}{(rq_j)!} =$$

$$\exp\left\{-r \sum q_j \ln \frac{q_j}{\mathcal{G}(p)_j} - \sum \left(\ln \sqrt{2\pi r q_j} + \frac{1}{12rq_j + \theta(rq_j)}\right) + O(\ln r)\right\}$$

where summation is restricted to indices for which $q_j > 0$ and where r is the population size.

Theorem 10.2 provides qualitative information concerning probable next generations. The expression

$$\sum q_j \ln \frac{q_j}{\mathcal{G}(p)_j}$$

is the *discrepancy* of q with respect to $\mathcal{G}(p)$ and is a measure of how far q is from the expected next population $\mathcal{G}(p)$. Discrepancy is nonnegative and is zero only when q is the expected next population. Hence the factor

$$\exp\left\{-r \sum q_j \ln \frac{q_j}{\mathcal{G}(p)_j}\right\}$$

occurring on the right hand side of theorem 10.2 indicates the probability that q is the next generation decays exponentially (with constant $-r$) as the discrepancy between q and the expected next population increases.

12. Summary. As should be now apparent, the infinite population model – i.e., the expected next generation operator $\mathcal{G}$ – is of fundamental importance to the theory of random heuristic search in general, and to simple genetic search in particular (since it is a special case of RHS).

First, it is identical to the heuristic function which specifies the sampling distribution governing the formation of the next generation. In other words, it repairs the deficiencies inherent in the schema theorem by giving *complete* information about the probabilities with which elements occur in the next generation.

Second, it emerges as the defining component of the transition matrix by which RHS is expressed as a Markov chain. Anything that could ever be proved about simple genetic search therefore corresponds to some property of $\mathcal{G}$, which argues for the investigation of $\mathcal{G}$ as a mathematical object. Abstractly, the heuristic corresponding to the simple GA may be determined by considering the following procedure. This procedure serves as the *definition* of what the terms "simple genetic search" and "simple genetic algorithm" (SGA) refer to:

1. Generate a random population x containing r fixed length binary strings.
2. Choose, with replacement, parents u and v from x (by any fixed selection scheme).
   - Cross u and v (by any fixed crossover rate and type) to produce children u' and v'.
   - Mutate u' and v' (by any fixed mutation rate and type) to produce u'' and v''.
   - Keep, with uniform probability, one of u'' and v'' for the next generation.
3. If the next generation is incomplete, repeat step 2.
4. Replace x by the new generation just formed and go to step 2.

The corresponding heuristic is abstractly defined by

$$\mathcal{G}_i(x) = \Pr\{i \text{ is the result of step 2} \mid x \text{ is the current population}\}$$

Concretely, $\mathcal{G}$ is explicitly known for a fairly wide range of operators (proportional, ranking, or tournament selection used with arbitrary mutation masks and arbitrary crossover masks). Moreover, a wealth of information is beginning to emerge as the role played by the Walsh transform in the theory of the SGA's heuristic is becoming clear. In this theory, the Walsh transform does not, however, appear to have anything to do with schemata, deception, or representing the objective function (the interested reader is referred to chapter 2, contributed by Vose and Wright, in the book "Genetic Algorithms for Pattern Recognition", 1996).

The remainder of this talk is aimed at uncovering, in general terms, the inherent emergent behavior of random heuristic search. Towards that end, presentation of the specific details of $\mathcal{G}$ (which would specialize it to the simple genetic algorithm) will be postponed in favor of pursuing

The expression

$$\sum \left(\ln \sqrt{2\pi r q_j} + \frac{1}{12 r q_j + \theta(r q_j)} \right)$$

measures the *dispersion* of the population vector q. A minimally disperse population is homogeneous (r identical population members) and corresponds to $q = e_i$ for some i (where e_i is the ith column of the identity matrix). The corresponding dispersion is $O(\ln r)$. If $n \geq r$, a maximally disperse population has no duplication (q has r nonzero components which are all $1/r$) and dispersion r. The factor

$$\exp\left\{ - \sum \left(\ln \sqrt{2\pi r q_j} + \frac{1}{12 r q_j + \theta(r q_j)} \right) \right\}$$

occurring on the right hand side of theorem 10.2 indicates the probability that q is the next generation decays exponentially with increasing dispersion.

The combined effect of the two influences of discrepancy and dispersion is that random heuristic search is biased towards a less dispurse population near the expected next generation.

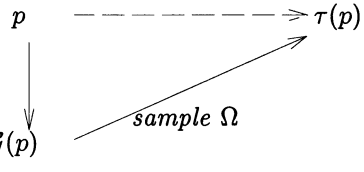
11. Markov Chain. The characterization of random heuristic search completed in the previous section was in terms of the sequence

$$p, \quad \tau(p), \quad \tau^2(p), \quad \ldots$$

produced by sampling Ω. This can be thought of as a sequence of random vectors (p may be regarded as the random vector which with probability one assumes only the single value p). Moreover, each random vector in the sequence depends only on the value of the preceding one. Such a sequence is called a *Markov chain.* The matrix defined by

$$Q_{p,q} \quad = \quad r! \prod \frac{(\mathcal{G}(p)_j)^{r q_j}}{(r q_j)!}$$

for $p, q \in \frac{1}{r} X_n^r$, is the *transition matrix* of the Markov chain. By theorem 10.2, $Q_{p,q}$ is the probability that $\tau(p) = q$. The definition of τ in the previous section rests ultimately on Ω as the induced map



This conceptualization can now be replaced by an abstraction which makes no reference to Ω at all: from current configuration p, produce $q = \tau(p)$ with probability $Q_{p,q}$.

consequences suggested by the structural properties of the framework just presented.

13. Large populations. This section is mainly about the relationship between random heuristic search and its heuristic, as population size goes to infinity. A consequence of the results obtained is a view of a simple genetic algorithm's transient and asymptotic behavior in the large population case, given a well behaved heuristic.

Before proceeding, RHS algorithms are first classified according to the behavior of $\mathcal{G}$. An instance of random heuristic search is *focused* if $\mathcal{G}$ is continuously differentiable and for every $p \in \Lambda$ the sequence

$$p, \quad \mathcal{G}(p), \quad \mathcal{G}^2(p), \quad \ldots$$

converges. In this case $\mathcal{G}$ is also called focused. In terms of search, the latter condition means that the path determined by following at each generation what τ is expected to produce will lead to some state x. By the continuity of $\mathcal{G}$, such points x satisfy $\mathcal{G}(x) = x$ and are therefore *fixed points* of $\mathcal{G}$. In the case of the simple genetic algorithm, there are no known counter examples to $\mathcal{G}$ being focused when the mutation rate is less than $1/2$. Moreover, Vose and Wright have proved that $\mathcal{G}$ is focused if proportional selection is used, the mutation rate is small, and the objective function has low epistasis.

An instance of random heuristic search is *hyperbolic* if $\mathcal{G}$ is continuously differentiable and its differential $d\mathcal{G}_x$ at x has no eigenvalues of unit modulus when x is a fixed point. In this case $\mathcal{G}$ is also called hyperbolic. In the case of the simple genetic algorithm, it has been proved by Eberlein and Vose that if proportional selection is used, then the set of fitness functions for which $\mathcal{G}$ is hyperbolic is dense and open (this was Mary Eberlein's Ph.D. dissertation). Generic hyperbolicity is believed to be the general case.

An instance of random heuristic search is *ergodic* if the Markov chain which represents it is ergodic for all $r > 0$. In this case $\mathcal{G}$ is also called ergodic. In the case of the simple genetic algorithm, ergodicity is insured by positive mutation.

The remainder of the talk deals with focused, hyperbolic, ergodic random heuristic search. It will turn out that fixed points are particularly relevant to both the transient (short term) and asymptotic (long term) behavior.

14. Transient and steady state behavior. The next theorem shows as $r \to \infty$ that, with probability converging to 1, the transient behavior of a population trajectory converges to the path determined by iterates of $\mathcal{G}$, and the initial transient occupies a time span diverging to infinity.

THEOREM 14.1. *Given $k > 0$, $\varepsilon > 0$ and $\gamma < 1$, there exists N such that with probability at least γ and for all $0 \leq t \leq k$*

$$r > N \implies \|\tau^t(x) - \mathcal{G}^t(x)\| < \varepsilon$$

Theorem 14.1 suggests (for large r) that some aspects of steady state behavior may be manifestations of transient behavior when $\mathcal{G}$ is focused. Let π be the probability measure corresponding to the steady state distribution of random heuristic search,

$$\pi(A) = \sum_{v \in \frac{1}{r} X_n^k} x_v \, [v \in A]$$

where x is the steady state distribution (probability vector) satisfying $x^T = x^T Q$ and [*expression*] denotes 1, if *expression* is true, and 0 otherwise. Thus $\pi(A)$ represents the proportion of time that populations spend in A, averaged over infinitely many generations. Since π is, for each population size r, a probability measure over the compact set Λ, a theorem of Prokhorov implies that every infinite sequence of π (corresponding to a sequence of r) has an infinite subsequence which converges weakly to some probability measure π'. Passing to the subsequence, this means that for every continuous function $h : \Lambda \to [0, 1]$,

$$\int h \, d\pi \longrightarrow \int h \, d\pi'$$

Let $\mathfrak{S}$ be the set of fixed points of $\mathcal{G}$. The next theorem provides a partial answer to how transient behavior influences steady state behavior.

THEOREM 14.2. *Suppose $\mathcal{G}$ is focused and ergodic. For every open set U containing $\mathfrak{S}$,*

$$\lim_{r \to \infty} \pi(U) = 1$$

In the large population case, theorem 14.2 indicates where population trajectories predominately spend time; near fixed points of $\mathcal{G}$. Moreover, theorem 14.1 indicates that a trajectory from x moves towards a fixed point of $\mathcal{G}$ by approximately following the path $x, \mathcal{G}(x), \mathcal{G}^2(x), \ldots$ The the next section investigates how quickly this path approaches a fixed point.

15. Logarithmic convergence. The definition of logarithmic time to convergence faces several obstacles. Perhaps the most obvious is the existence of a sequence of initial populations along which the time to convergence diverges to infinity. To circumvent this difficulty, let a probability density ρ be given over Λ and define the probability that the initial population is contained in A as

$$\int_A \rho \, d\lambda$$

where λ is Lebesgue measure. The task is then to show that for every ρ and every $\varepsilon > 0$, there exists a set A of probability at least $1 - \varepsilon$ such that if the initial population is in A, then the time to convergence is logarithmic.

The next difficulty is that, typically, a orbit under $\mathcal{G}$ will never reach the limit it is approaching. It is natural, therefore, to let $0 < \delta < 1$ denote how close trajectories are required to get to the limit, and then to require that they do so, within $O(-\ln \delta)$ generations.

In summary, *logarithmic convergence* is defined as follows: for every probability density ρ and every $\varepsilon > 0$, there exists a set A of probability at least $1 - \varepsilon$ such that for all $x \in A$ and $0 < \delta < 1$, the number k of generations required for $\|\mathcal{G}^k(x) - \omega(x)\| < \delta$ is $O(-\ln \delta)$, where $\omega(x)$ denotes the limit of $\mathcal{G}^t(x)$ as $t \to \infty$.

The next theorem makes use of the following technical condition. $\mathcal{G}$ is said to be *regular* if whenever C has measure zero, then so does the set $\mathcal{G}^{-1}(C)$. In the case of the simple genetic algorithm, Vose and Wright have proved that $\mathcal{G}$ is regular if the crossover rate is less than 1 and the mutation rate is strictly between 0 and 1/2 (provided that every string has positive selection probability).

THEOREM 15.1. *If $\mathcal{G}$ is focused, hyperbolic, and regular, then $\mathcal{G}$ is logarithmically convergent.*

16. Punctuated equilibria. Assuming $\mathcal{G}$ is focused, hyperbolic, ergodic, and regular, the view of RHS behavior that emerges for the large population case is the following.

As $r \to \infty$, and then with probability converging to 1, the initial transient of a population trajectory converges to following the path determined by iterates of $\mathcal{G}$, and that transient occupies a time span diverging to infinity. Consequently, populations will predominately appear near some fixed point ω of $\mathcal{G}$ since the path $x, \mathcal{G}(x), \mathcal{G}^2(x), \ldots$ approaches a fixed point relatively quickly.

This appears in contrast to the fact that the RHS is an ergodic Markov chain; every state must be visited infinitely often. This is reconciled in *punctuated equilibria*: Random events may eventually move the system to a population x' contained within the basin of attraction (with respect to the underlying dynamical system corresponding to $\mathcal{G}$) of a different fixed point ω'. Since the behavior of random heuristic search is time independent, the anticipated behavior follows the trajectory $x', \mathcal{G}(x'), \mathcal{G}^2(x'), \ldots$ – as if x' were the initial population – to reach a new temporary stasis in the vicinity of ω'.

This cycle of a period of relative stability followed by a sudden change to a new dynamic equilibrium is the picture provided by the results of this section. It is an open question as to how large r must be before these qualitative aspects of RHS are typically exhibited in the general case.

It is of interest that this phenomenon (punctuated equilibria) has been observed in practice when using GAs on optimization tasks. It is not uncommon for a GA to be described as undergoing a period of relative stability, after which it "discovers" a better solution which transforms the population. Neither is it uncommon for several such cycles to be manifest

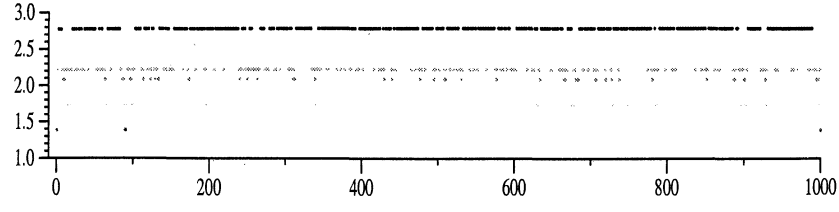
during long optimization runs. These observations are compatible with the conjecture that simple genetic search, as commonly used in practice, is influenced by the fixed points of the underlying dynamical system (corresponding to $\mathcal{G}$ on Λ).

17. Empirical evidence. This section considers a few computational examples for the simple genetic algorithm. Unlike the previous section whose results were based on arbitrarily large populations, the phenomena documented here were observed subject to a constraint on r. These examples took $r \approx \sqrt{n}$. Further empirical study is required (or better theorems need to be proved) to sort out the required linkage between search space size and population size in order for the behavior presented below to typically emerge. The conjecture that a logarithmic coupling (or some power of a log) might be appropriate for a wide class of objective functions is not incompatible with the empirical evidence.

When considering emergent behavior, perhaps the most fundamental question is: Where in Λ is the simple genetic algorithm at time t? Since the state space Λ has dimensionality too large for direct visualization (except in the case $\ell = 2$), alternate means of monitoring the progression from one generation to the next are required. A primitive means of reducing dimensionality is by measuring distance from populations to a reference point, say to the center $\mathbf{1}/n$ of Λ. The following graph shows what this looks like for string length 4, a random fitness function, (one-point) crossover rate 0.8, and mutation rate 0.01. Motivated by theoretical observations following theorem 10.2, the vertical axis measures distance as discrepancy,

$$distance(x, y) = \sum x_j \ln \frac{x_j}{y_j}$$

and the horizontal axis spans 1,000 generations. This example has population size 4 and corresponding state space of 3,876 populations in a simplex of dimension 15. The initial population p is random, and subsequent generations are produced by τ.

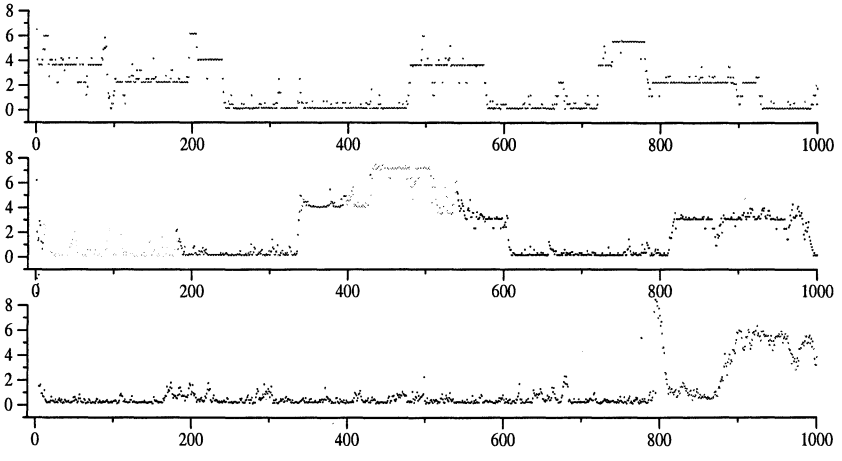


Certainly populations equidistant from $\mathbf{1}/n$ (i.e., with equal entropy) need not be near each other, but nevertheless there may be relatively few regions where the SGA is spending most of its time. One natural way to explore this is to locate regions where it seems reasonable that the SGA could be spending time, and then plot distance from the current population to such a place. The result should be essentially flat since at a typical

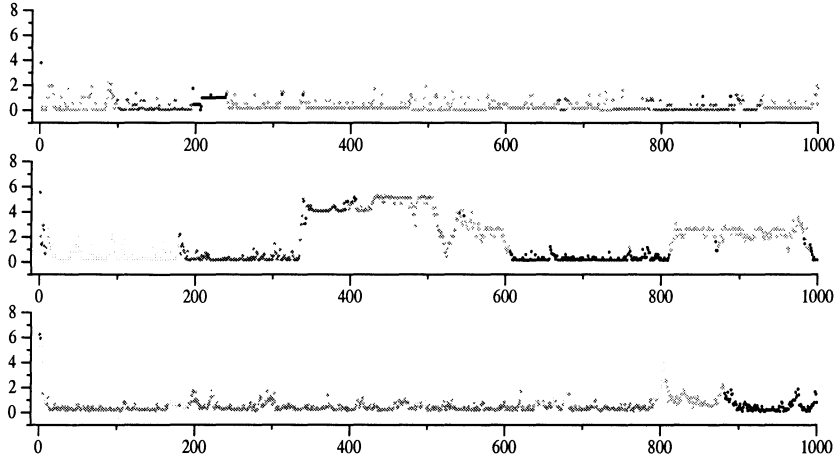
generation (abscissa), the distance to one of these regions (ordinate) should be small.

Candidate regions are suggested by the discussion following theorem 10.2. Likely next generations are strongly related to the expected next generation. Near a fixed point, expected behavior is for the next generation to be near the current one, so fixed points of $\mathcal{G}$ may indicate areas where there is little pressure for change. It is plausible that the SGA could spend more time near such regions of Λ.

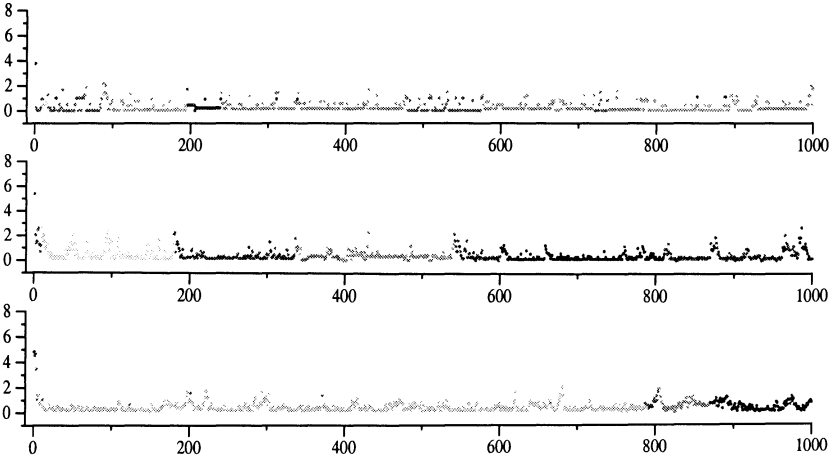
One method of locating attracting fixed points is by iterating $\mathcal{G}$. The following series of graphs shows generations (horizontal) versus distance to the stable fixed point to which iterates of $\mathcal{G}$ converge (vertical). The initial population p is random, and subsequent generations are produced by τ. The fitness function is random, a (one-point) crossover rate of 0.8 and mutation rate of 0.01 is used, and 1,000 generations are spanned. The string length ℓ begins at 4, increasing by 2 with each graph. The population size is approximately $\sqrt{n}$.



Given the complexity of these graphs, one might suspect fixed points of $\mathcal{G}$ do not help explain emergent behavior after all. However, the possibility remains that the SGA is particularly adept at seeking out fixed points of $\mathcal{G}$, more so than is the method of iterating $\mathcal{G}$ to locate them. The plateaus in these graphs suggest populations which could be concentrated in some localized region of Λ, perhaps a region near a fixed point not found by iterating $\mathcal{G}$. Extending the domain of $\mathcal{G}$ to the real affine space containing the simplex, using a minimization method to locate fixed points – whether stable or not – and then measuring distance to the nearest fixed point results in the following.



The majority of fixed points are unstable, outside the simplex, and near a vertex. The instability of population p within Λ near such a fixed point may be counterbalanced by the preference of RHS for populations having low dispersion. Another counterbalancing influence is the coarseness of the lattice $\frac{1}{r}X_n^r$ of points available to populations for occupation. Since $\mathcal{G}(p)$ is nearly p, the influence of discrepancy favors the next generation being identical to p, and this influence grows with increasing coarseness of the lattice. Even though overall the graphs are much lower, indicating a typical population's proximity to some fixed point, a few regions remain far from any fixed point. The next series of graphs is as the previous, except the fixed point equations were considered over *complex* space.



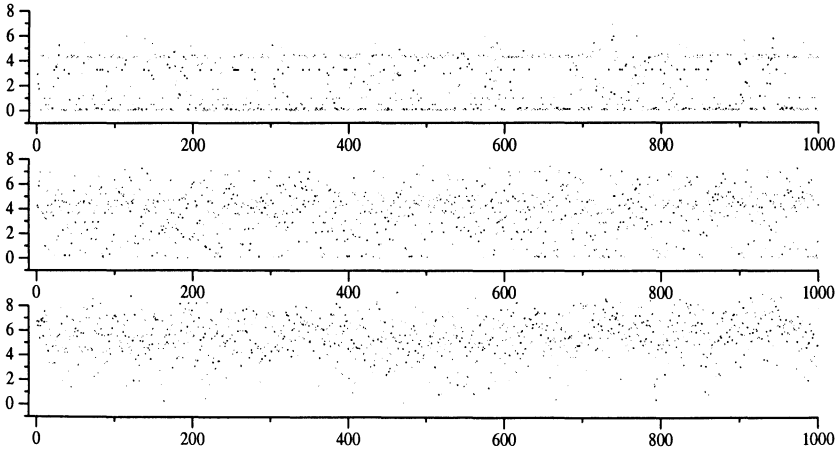
All graphs exhibit small height now that complex fixed points are in-

cluded. Even though these examples are anecdotal, they are representative of several thousands of random cases.

It should not be supposed that irregularities in the graphs indicate an inability of fixed points to partially explain behavior. Finding fixed points is a difficult task exacerbated by high dimensionality which increases exponentially with ℓ. It is unlikely that all relevant fixed points have been found.

On the other hand, it is equally uncertain whether fixed points will continue to exert such a pronounced influence over emergent behavior as string length increases. Uncertainty revolves around the linkage between population size and string length. Whereas the large population case is fairly well understood (there the answer is yes), the small population case is not, and it is precisely that case which is the more interesting one from the point of view of search/optimization.

A natural question is whether the graphs presented in this section would retain their general appearance if, while leaving the fitness functions unchanged, alternate initial populations or different seeds for the random number generator were used. The answer is typically yes; sufficiently many fixed points corresponding to each fitness function were found so that populations are generally nearby. This begs the question of whether, with many fixed points available, a random point in Λ would be close to one of them. If that were the case, then these graphs have little meaning. The following sequence of graphs address this issue. Distance is measured to the nearest fixed point from random points having the same entropy distribution as populations in the previous graphs.



18. Summary. The evidence presented seems to indicate that simple genetic search is particularly adept at locating regions in the vicinity of fixed points of $\mathcal{G}$. This picture is conjectural, however, since no theorems were proved. Although smaller population sizes, $r = O(\ln n)$ for example,

may seem more realistic, there is insignificant difference between $3\log_2 n$ and $\sqrt{n}$ for the string lengths considered ($3 < \ell < 9$).

While different in important respects, the results presented here are reminiscent of the large population case. The increased influence of the dispersion term (in theorem 10.2) on transition probabilities, given smaller r, may contribute to increased importance of unstable and complex fixed points located near vertices of Λ. Because $\mathcal{G}$ is continuous, such fixed points locate regions where there is decreased pressure for change. The natural preference of RHS for low dispersion may counterbalance the instability of the underlying dynamical system (corresponding to $\mathcal{G}$) for populations in such regions. Moreover, the coarseness of the lattice of points available for occupation also serves to counterbalance the instability.

19. Small populations: Signal and noise. The previous considerations may seem, from a practitioner's viewpoint, academic. There are several reasons for this. One is that it is the initial transient (or first few) of the small population case that is of primary interest. Another is that s/he has a problem to solve, and wants an answer: Should a GA be used? If genetic search *is* being used, then *how* can it be made effective?

In general, I believe it can be shown that categorical answers to questions like these are of comparable hardness to finding the optimum by enumeration (again, there is no "free lunch"). The best one can hope for is results concerning a specific class of functions, and even then, the problem of deciding whether a general function belongs to the class is a hard problem. Nevertheless, it is quite likely that simple qualitative results providing insight into the general mechanism of RHS for all population sizes are near at hand.

Consider, for example, the following asymptotic result (as $r \to \infty$). Let $q = \mathcal{G}(p)$ and let C be an n by $n-1$ matrix having orthonormal columns perpendicular to $h = <\sqrt{q_0}, \ldots, \sqrt{q_{n-1}}>$.

THEOREM 19.1. *If $\mathcal{G}$ maps into the interior of Λ, then for any open subset U of $\mathbf{1}^{\perp}$,*

$$Pr\{\tau(p) \in \mathcal{G}(p) + U/\sqrt{r}\} = (2\pi)^{-(n-1)/2} \int_{C^T \operatorname{diag}(h)^{-1} U} e^{-y^T y/2}\, dy + o(1)$$

If an error term was provided for this result (i.e., an explicit form for "$o(1)$" in terms of r and n), a fairly simple decomposition of τ into a deterministic *signal* component, given by $\mathcal{G}$, and a stochastic *noise* component, given by the multinormal distribution, would result.

If one does not mind the multinomial distribution, theorem 10.2 shows, for any r, that $\tau(p)$ is given by a single sample from a multinomial distribution having mean $\mathcal{G}(p)$. Again, $\mathcal{G}(p)$ emerges as the deterministic *signal*. The stochastic *noise* component is given by the multinomial distribution.

Note how, in this decomposition into signal and noise, the signal is *invariant* in the sense that it is *independent* of population size. The noise

is partially characterized by the following theorem.

THEOREM 19.1. *Let $\mathcal{E}$ denote expectation.*

$$\mathcal{E}(\| \tau(p) - \mathcal{G}(p) \|^2) = (1 - \| \mathcal{G}(p) \|^2)/r.$$

What complicates the small population case is threefold. First, as r decreases, the noise component increases. Second, the relative influence of dispersion grows. Third, the lattice of allowable values for population vectors, $\frac{1}{r} X_n^r$, becomes increasingly coarse, as fewer points, located in lower dimensional faces of Λ, become available for occupation. Genetic search is conducted in a low dimensional "skeleton" of Λ which constrains the system's ability to follow the signal.

It is of interest to understand which directions, or pathways through this skeleton, are more probable than others. In particular, one would like to know what strings are typically encountered while traversing the pathways. At this point, there are no proven simple answers (there is, of course, the Markov chain, but that quickly becomes unwieldy as r increases).

20. Metalevel chain. Assuming simple genetic algorithms are adept at locating regions in the vicinity of fixed points of $\mathcal{G}$, the transition probabilities from one such region to another are significant. In that case, simple genetic search could be modeled by a Markov chain over the fixed points. If the transition probabilities from temporary stasis in the vicinity of one fixed point to temporary stasis near another can be determined, then some aspects of the punctuated equilibria could in principle be analyzed.

The goal of constructing a meta level Markov chain, as described in the previous paragraph, has been achieved for general random heuristic search (subject to the technical conditions that $\mathcal{G}$ is hyperbolic, ergodic, and has a complete Lyapunov function), but only in the large population case.

Let $\rho = x_0, \ldots, x_k$ be a sequence of points from Λ, referred to as a *path* of length k from x_0 to x_k. The *cost* of ρ is

$$| \rho | = \alpha_{x_0,x_1} + \cdots + \alpha_{x_{k-1},x_k}$$

where

$$\alpha_{u,v} = \sum v_j \ln \frac{v_j}{\mathcal{G}(u)_j}$$

and it is assumed that $\mathcal{G}$ maps Λ into its interior so as to avoid division by zero. Let the stable fixed points of $\mathcal{G}$ in Λ be $\{\omega_0, \ldots, \omega_w\}$ and define

$$\rho_{\omega_i,\omega_j} = \inf \{ | \rho | : \rho \text{ is a path from } \omega_i \text{ to } \omega_j \}$$

Let $\mathcal{C}_r$ be a Markov chain defined over $\{1, \ldots, w\}$ with $i \to j$ transition probability (for $i \neq j$)

$$\exp\{-r\, \rho_{\omega_i,\omega_j} + o(r)\}$$

Up to the uncertainly in the $o(r)$ terms, the desired Markov chain is C_r.

As section 4 demonstrates, C_r cannot possibly be appropriate for small r, because unstable, complex, and stable fixed points outside Λ make no contribution to C_r. Nevertheless, the form of the transition probabilities above is instructive. The likelihood of a transition from i to j is determined by the minimal cost path from ω_i to ω_j where a path incurs cost to the extent that it is made up of steps which end at a place differing from where $\mathcal{G}$ maps their beginning. By theorem 10.2, this roughly corresponds to the probability of the most likely sample path leading from ω_i to ω_j in the Markov chain with transition matrix Q. In other words, when r is large, and aside from the dispersion being unimportant, behavior is determined by a *single* sample path, rather than a *sum* over sample paths.

Is not unthinkable that in order to make significant progress in the small population case, the *particular* nature of $\mathcal{G}$ will have to be brought into play; perhaps general properties (hyperbolicity, ergodicity, etc.) will not suffice. Either way, this points to the pivotal importance of $\mathcal{G}$ – either in terms of its general properties, or else in terms of the details of its specific nature – and argues for its study as an abstract mathematical object.

21. The SGA's heuristic. This section presents a special case of the simple genetic algorithm's heuristic. It is not feasible to summarize a nontrivial part of all that is currently known, so simple *definition* of a particular case will have to do for this talk. For simplicity, the binary case will be described (Koehler, Bhattacharyya, and Vose have extended results to the general cardinality case) and only proportional selection will be considered.

For positive integer ℓ, the set of length ℓ binary strings is the Cartesian product

$$\Omega = \underbrace{Z_2 \times \cdots \times Z_2}_{\ell \text{ times}}$$

Since the ℓ-digit binary representations of integers in the interval $[0, 2^\ell)$ coincide with the elements of Ω, they are regarded as being the same. Elements of Z_2 form a finite field under the operations of addition and multiplication modulo 2

$\oplus$	0	1
0	0	1
1	1	0

$\otimes$	0	1
0	0	0
1	0	1

These operations are extended to Ω by applying them coordinate-wise. By convention, $\otimes$ takes precedence over $\oplus$, and both bind more tightly than operations which are not modulo 2.

For $x \in \Omega$, let $\overline{x}$ abbreviate $\mathbf{1} \oplus x$. In standard computer science nomenclature, $\oplus$ is *exclusive-or* on integers, $\otimes$ is *and*, and $x \mapsto \overline{x}$ is *not*. Note that $\otimes$ distributes over $\oplus$.

Let σ_k be the permutation matrix defined by

$$(\sigma_k)_{i,j} \quad = \quad [i \oplus j = k]$$

The permutation σ_k corresponds to applying the map $i \mapsto i \oplus k$ to subscripts. That is,

$$\sigma_k < x_0, \ldots, x_{n-1} > \quad = \quad < x_{0 \oplus k}, \ldots, x_{(n-1) \oplus k} >$$

22. Selection. The symbol s will be used for three equivalent (though different) things. This overloading of s does not take long to get used to because context makes meaning clear. The benefits are clean and elegant presentation and the ability to use a common symbol for ideas whose differences are often conveniently blurred.

First, $s \in \Lambda$ can be regarded as a *selection distribution* describing the probability s_i with which i is selected (with replacement) from the current population for participation in forming the next generation. A selected element is an intermediate step towards producing the next population, not typically a member of it. In total, $2r$ such selections will be made, the aggregate of which is sometimes referred to as the *gene pool.*

Second, $s : \Lambda \to \Omega$ can be regarded as a *selection function* which is nondeterministic. The result $s(p)$ of applying s to p is i with probability given by the ith component s_i of the selection distribution. Of course, for there to be a nontrivial dependence on p, the selection distribution must be some function $\mathcal{F}$ of p. The function $\mathcal{F}$ is referred to as the *selection scheme.*

Third, $s \in \Lambda$ can be regarded as a population vector.

In analogy with survival of the fittest, an integral part of $\mathcal{F}$ is a *fitness function* $f : \Omega \to \Re$ which can be used (in a variety of ways) to determine a selection scheme. The fitness function is assumed to be injective. The value $f(i)$ is called the *fitness* of i. Through the identification $f_i = f(i)$, the fitness function may be regarded as a vector.

Proportional selection refers to the selection function corresponding to the selection scheme

$$\mathcal{F}(x) \quad = \quad f \cdot x / f^T x$$

(where $f \cdot x$ denotes $\mathrm{diag}(f)x$). When proportional selection is being used, it is assumed that the fitness function is positive. Since proportional selection is homogeneous, without loss of generality $f \in \Lambda$.

By letting the heuristic $\mathcal{G}$ be the selection scheme, results from previous sections apply to selection. For example, with population size $2r$, $\tau(p)$ becomes the gene pool. Invoking theorem 10.1, the expected gene pool is described by the population vector $s = \mathcal{F}(p)$. By definition, the selection distribution is $s = \mathcal{F}(p)$. Hence, as elements of Λ, the selection distribution is identical to the expected gene pool.

23. Mutation. The symbol μ will also be used for three different (though related) things.

First, $\mu \in \Lambda$ can be regarded as a distribution describing the probability μ_i with which i is selected to be a *mutation mask* (additional details will follow).

Second, $\mu : \Omega \to \Omega$ can be regarded as a *mutation function* which is nondeterministic. The result $\mu(x)$ of applying μ to x is $x \oplus i$ with probability given by the ith component μ_i of the distribution μ. The i occurring in $x \oplus i$ is referred to as a mutation mask. The application of μ to x is referred to as *mutating* x.

Third, $\mu \in [0, 0.5)$ can be regarded as a *mutation rate* which implicitly specifies the distribution μ according to the rule

$$\mu_i \quad = \quad (\mu)^{\mathbf{1}^T i} \, (1 - \mu)^{\ell - \mathbf{1}^T i}$$

The distribution μ need not correspond to any mutation rate, although that is certainly the classical situation. Any element $\mu \in \Lambda$ whatsoever is allowed.

The effect of mutating x using mutation mask i is to alter the bits of x in those positions where the mutation mask i is 1. When mutation is affected by a rate, the probability of selecting mask i depends only on the number of 1s that i contains.

If the mutation rate is nonzero (the typical case), then every element of Ω has a positive probability of being the result of $\mu(x)$. Mutation is said to be *zero* if $\mu_i = [i = 0]$. For arbitrary $\mu \in \Lambda$, mutation is called *positive* if $\mu_i > 0$ for all i.

24. Crossover. It is convenient to use the concept of *partial probability* . Let $\zeta : A \to B$ and suppose that $\phi : A \to [0, 1]$. To say "$\xi = \zeta(a)$ with partial probability $\phi(a)$" means that $\xi = b$ with probability $\sum_a [\zeta(a) = b] \, \phi(a)$.

The description of crossover parallels the description of mutation given in the previous section; the symbol χ will be used for three different (though related) things.

First, $\chi \in \Lambda$ can be regarded as a distribution describing the probability χ_i with which i is selected to be a *crossover mask* (additional details will follow).

Second, $\chi : \Omega \times \Omega \to \Omega$ can be regarded as a *crossover function* which is nondeterministic. The result $\chi(x, y)$ is $x \otimes i \oplus \bar{i} \otimes y$ with partial probability $\chi_i/2$ and is $y \otimes i \oplus \bar{i} \otimes x$ with partial probability $\chi_i/2$. The i occurring in the definition of $\chi(x, y)$ is referred to as a crossover mask. The application of $\chi(x, y)$ to x, y is referred to as *recombining* x and y.

The arguments x and y of the crossover function are called *parents*, the pair $x \otimes i \oplus \bar{i} \otimes y$ and $y \otimes i \oplus \bar{i} \otimes x$ are referred to as their *children*. Note that crossover produces children by exchanging the bits of parents in

those positions where the crossover mask i is 1. The result $\chi(x, y)$ is called their *child.*

Third, $\chi \in [0, 1]$ can be regarded as a *crossover rate* which specifies the distribution χ according to the rule

$$\chi_i \quad = \quad \begin{cases} \chi c_i & \text{if } i > 0 \\ 1 - \chi + \chi c_0 & \text{if } i = 0 \end{cases}$$

where the distribution $c \in \Lambda$ is referred to as the *crossover type.* Classical crossover types include *1-point crossover*, for which

$$c_i \quad = \quad \begin{cases} 1/(\ell - 1) & \text{if } \exists k \in (0, \ell) \,.\, i = 2^k - 1 \\ 0 & \text{otherwise} \end{cases}$$

and *uniform crossover*, for which $c_i = 2^{-\ell}$. However, any element $c \in \Lambda$ whatsoever is allowed.

Obtaining child z from parents x and y via the process of mutation and crossover is called *mixing* and has probability denoted by $m_{x,y}(z)$.

THEOREM 24.1. *If mutation is performed before crossover, then*

$$m_{x,y}(z) \quad = \quad \sum_{i,j,k} \mu_i \, \mu_j \, \frac{\chi_k + \chi_{\overline{k}}}{2} \, [(x \oplus i) \otimes k \oplus \overline{k} \otimes (y \oplus j) = z]$$

If mutation is performed after crossover, then

$$m_{x,y}(z) \quad = \quad \sum_{j,k} \mu_j \, \frac{\chi_k + \chi_{\overline{k}}}{2} \, [x \otimes k \oplus \overline{k} \otimes y = z \oplus j]$$

The *mixing scheme* $\mathcal{M} : \Lambda \to \Lambda$ is defined by the component equations

$$\mathcal{M}(x)_i \quad = \quad x^T \sigma_i M \sigma_i x$$

THEOREM 24.2. *The heuristic $\mathcal{M}$ corresponds to the instance of RHS which produces elements for the next generation by mixing the results of uniform choice (with replacement) from the population.*

25. SGA's Heuristic. The simple genetic algorithm's heuristic is the composition of mixing and selection

$$\mathcal{G} = \mathcal{M} \circ \mathcal{F}$$

which depends on the three control parameters, $f, \mu, \chi \in \Lambda$. This completes the definition of the simple genetic algorithm as an instance of random heuristic search.

26. Closing remarks. What is known about the theory of the SGA's heuristic has only been hinted at in this talk. The forthcoming book "The Simple Genetic Algorithm: Foundations and Theory" (Mit Press, in press) will be a good place for the interested reader to begin.

I would like to indicate where the example of section 1.3.1 (Static Schema Analysis) came from. The key is that, in the zero mutation case, the spectrum of the differential $d\mathcal{G}_x$ is known explicitly at every fixed point x which corresponds to an absorbing state of the Markov chain. For further details, see "Stability of Vertex Fixed Points and Applications" (Vose and Wright) in the book "Foundations of Genetic Algorithms III" (Whitley & Vose, Editors).

I owe a lot to friends, colleagues, and the National Science Foundation, for supporting this line of inquiry which so radically departs from the classical schemata-based "GA theory." Many have provided inspiration and companionship along the way, but Alden Wright and Gary Koehler deserve special mention.

SURVEY OF PROJECTS INVOLVING EVOLUTIONARY ALGORITHMS SPONSORED BY THE ELECTRIC POWER RESEARCH INSTITUTE

A. MARTIN WILDBERGER, PH.D.*

Abstract. Over the last five years, the Electric Power Research Institute (EPRI) has been engaged in a number of research and development projects using evolutionary algorithms for applications in the electric power industry. (EPRI 1994b) Most of these projects also involve theory, since advances in theoretical understanding is often necessary in order to address critical barriers faced in practical applications. Projects reviewed briefly in this paper include:

- successful use of global optimization by genetic algorithms combined with parallel local solutions by coupled gradient neural networks to solve the unit commitment problem (a highly constrained, mixed-integer programming problem usually attacked by Lagrangian relaxation methods).
- projects co-sponsored with the National Science Foundation that are using evolutionary methods in combination with other techniques in the context of "intelligent control systems."
- successful use of a genetic algorithm to test an expert system for improving the performance of fossil-fueled power plants.
- successful use of genetic algorithms to optimize a neural network for heat-rate improvement of nuclear plant operation; now being extended to construct a generic tool for the automated design and optimization of neural networks in any context.
- exploration of ways to use autonomous evolving agents as a modeling technique for the electric power market and ultimately for the industry itself as it moves toward increased deregulation and competition.
- plans for the use of multiple adaptive agents as a modeling method to allow real-time, distributed control of the electric power grid.

1. Unit commitment and contingency analysis. The problem of planning which power generating units to use to supply consumer demands and to allow some reserve for contingencies is known as the "Unit Commitment Problem." This is a highly constrained mixed-integer programming problem of the class known as "portfolio" or "knapsack." Most software used by most electric utilities employs Lagrangian multipliers solved numerically. Dynamic programming is also used by a few power system control centers. Both methods are slow, not particularly adapted to having their speed increased by parallel computing, and very sensitive to initial conditions.

M. H. Hassoun of Wayne State University used global optimization by genetic algorithms (GA), along with parallel local solutions by coupled gradient neural networks, to solve the unit commitment problem. This EPRI project provided the proof of concept for a faster, global optimization technique that may eventually save considerable money for electric utilities.

*Mgr., Mathematics & Information Science, Strategic Research & Development, Electric Power Research Institute, 3412 Hillview Ave., Palo Alto, CA 94304-1395; email: mwildber@epri.com phone: (415) 855-1043, FAX: (415) 855-2287.

(EPRI 1994a) Used in this context, genetic algorithms have an additional advantage. They provide multiple suboptimal solutions not much worse than the true optimum. Between the time the planning software is run and the time that the "dispatcher" (the operator of the power system) must actually perform an "economic dispatch" of power, a significant change in the power system topology may have occurred due to equipment failure, weather, of even an unexpected increase/decrease in customer demand. One or more of the originally suboptimal solutions may fit within the new constraints, and be able to be used for guidance while the planning software in being run again. Furthermore, the genetic algorithm can be expected to converge very rapidly when it is restarted with the whole set of previous trial solutions and the new constraints.

2. Multi-agent software framework for risk-based planning & control of dynamic contingencies. Planning for the avoidance of system failures, and recovery from those that cannot be avoided, is called "contingency analysis" by the electric utilities. It is typically handled by starting with the actual power flows in the network, removing one component at a time (with replacement) and adding generation as necessary to guarantee uninterrupted operation should that component fail. In the competitive business environment resulting from deregulation, utilities can gain price advantage by accepting some risk of failure and balancing this against the cost of recovery, including performance penalties in their negotiated contracts. In an EPRI-sponsored project started in 1996, V. C. Ramesh at the Illinois Institute of Technology is investigating a multiple adaptive agent approach to risk-based contingency analysis in a dynamically changing environment. He has retained the conventional model of contingency analysis as a constrained non-linear optimization problem, but he is building (approximately ten) individual intelligent agent solvers, each of which will attack the solution of the model in a different way. Some of the agents will employ evolutionary algorithms while others will use more conventional methods such as Lagrangian relaxation and dynamic programming. Neural networks and other non-linear statistical techniques may also be included. Other agents, called "destroyers," will weed out infeasible solutions in order to keep the population size manageable. Cooperation and competition among all these agents is expected to achieve a faster resolution to a true optimum than could any one alone. Agents are also expected to evolve improved methods based on experience with the specific network structure for each utility as well as the availability of mutual support. It is planned to build this simulation using JAVA and the SWARM software package from the Santa Fe Institute (SFI). (1996)

3. Evolutionary methods for supervisory adaptive control. From 1992 through 1995 EPRI co-sponsored with the National Science Foundation a group of twenty-one projects which addressed various techniques for intelligent and adaptive control. A few of the researchers in-

volved in these projects are now exploring the use of evolutionary methods for supervisory adaptive control.

In a project at the University of Texas at Arlington, Frank L. Lewis, Kai Liu, and others completed several controls applications in power system load frequency control (LFC) and automatic generation control (AGC), including a fuzzy logic AGC controller. They are now investigating genetic algorithms for design of fuzzy logic and neural network power system AGC controllers, to set up the initial rule-bases and neuron structures, as well as to tune them adaptively on line.

The use of Flexible AC Transmission Systems (FACTS), and other high-power, active control devices, throughout power delivery systems promises significant increases in the performance, efficiency, and capacity of existing networks. These solid-state electronic devices, analogous to integrated circuits but operational at multi-megawatt power levels, can convert electricity to a wide range of voltages, numbers of phases, and frequencies with minimal electrical loss and component wear. They enable control and tuning of all power circuits for maximum performance and cost-effectiveness, promising enhanced energy efficiency for the U.S. economy and flexible value-added service offerings from electricity providers. (EPRI 1996c) However, to realize these benefits, new control techniques are required. In a project at the University of Wisconsin, Madison, William Sethares developed new techniques, based on a Lyapunov energy function approach, to coordinate the activities of the numerous dispersed controllers that would be involved in such applications, and to optimize the exchange of information between them. He is now exploring the use of the genetic algorithm to help set up high-level "goals" (such as load management) for the FACTS devices to accomplish.

4. Using a genetic algorithm to "stress-test" an adaptive system . There is, at present, neither a firm theory nor a set of "engineering handbook" procedures to guide the design of any adaptive system, whether it is trained (like a neural network) or evolved (like a genetic algorithm), and whether or not it is permitted to continue adapting on-line while it is in actual service. Most of these systems continue to be built almost entirely by trial and error, with some guidance from the designer's prior experience. This approach to design, as well as the nature of adaptability itself, makes it extremely difficult either to guarantee the performance of an untried design or even to validate it by testing after it is implemented.

The most promising approach to automating the validation of adaptive control systems after their implementation may be a purely pragmatic one that, itself, uses adaptive techniques. Originally suggested by John Grefenstette (1988) of the Naval Research Laboratory (NRL), this approach to validating an adaptive system employs a genetic algorithm (GA) that serves as a "devil's advocate" by attempting to make the adaptive control system fail. This approach requires a validated simulation of the "plant" (the

underlying system to be controlled) as well as the adaptive controller to be tested. The need for a valid simulation of the plant may seem merely to move the problem of validation down one level. However, as long as the plant without its controller is not self-adaptive, representing it by a validated model and simulation is possible with well known technology. The individual population members of the "devil's advocate" GA represent possible operating conditions for the plant and its environment as well as possible plant failure modes (if the adaptive control system is expected to overcome these also). The fitness function for the GA encourages the evolution of population members that cause unsatisfactory performance by the control system. The successful evolution of one or more such individuals demonstrates that the adaptive control system is inadequate in its handling of the particular situation represented by each such individual. Either the control system must be improved or that particular combination of circumstances must be prevented from occurring by a separate safety device or other defensive mechanism.

Clearly, this approach has the potential to uncover all the modes of failure in the adaptive control system, but does not provide absolute assurance if, after many generations of evolution, no more instances of unsatisfactory control occur. As is well known, testing only reveals "bugs," not the absence of them. Some subsequent generation of the algorithm might yet overpower the controller. However, subject to compactness of the parameter space, after a sufficient number of generations have past, the most successful individuals in each successive generation represent bounds within which the adaptive controller performs satisfactorily. If guaranteed performance is required to be unbounded in one or more dimensions, this pragmatic approach fails and proofs using formal logic are needed to ensure the adaptive system's unbounded correctness.

A recent EPRI project has tested the use of genetic algorithms to validate a rule-based expert system. (Roache et al. 1995)(EPRI 1996a) In any practical application, validation is critical to expert system success. Most fielded expert systems are validated by testing whether the expert system provides appropriate conclusions for specific cases. Since exhaustive testing of all rule combinations is not computationally feasible in any real-world expert system, cases where the system provides inappropriate conclusions are easily missed. This project was conducted jointly by EPRI, NRL, New York Electric & Gas (NYSEG), and DHR Technologies. The Fossil Thermal Performance Advisor (FTPA), is an expert system developed by personnel at DHR for NYSEG. It provides the operator of a coal-fired steam power plant with recommendations for improving the performance of the plant. The FTPA is currently in operation in several plants, and some early installations have been running continually for over five years. NRL provided the GA software. NYSEG furnished the FTPA along with data from one of their plants, and, under the leadership of Edward Roache, personnel from DHR Technologies programmed the required modifications to the FTPA

as well as additional bridging software. DHR also provided a neural network model of an actual power plant, in lieu of a more conventional plant simulation based on a causal model.

The individual population members of the GA were designed to represent possible operating conditions for the plant and its environment. Its fitness function encouraged the evolution of population members that produced unsatisfactory performance by the plant but were either inadequately diagnosed by the expert system or caused the expert system to recommend actions which actually decreased plant performance. Besides errors purposely inserted for test purposes, the GA successfully exposed an error in the expert system which had not previously been detected by designers or operators over three years of actual use.

This successful proof-of-concept test suggests that genetic algorithms may be useful not only for the initial validation of a symbolic artificial intelligence "expert system," but also for adapting that system to changes in the configuration of the plant or for customizing it for other plants that are only basically similar to the one for which it was originally designed. Customization typically requires almost as many engineering man-hours as designing the original expert system, and it is a critical barrier to the greater use of symbolic artificial intelligence in industry. Even if an accurate simulation is not available for the new plant, the original expert system could be installed there and a genetic algorithm used to "tune" it to the new plant over some period of actual operation.

5. Genetic synthesis and optimization of neural networks. A key shortcoming of the current state of neural network technology is the lack of any effective design methodology. Neural network technology is becoming widely accepted in various industries, including the electricity enterprise. Numerous operating applications have demonstrated significant performance improvements over prior methods. But current approaches to developing neural network applications are a critical barrier to realizing the full potential of this technology. Optimizing neural network applications is a formidable design task requiring myriad choices in the number of neurons and layers, their interconnections, and the training algorithm to be employed as well as all its parameters.

EPRI has been sponsoring research by Tariq Samad and Steven A. Harp at Honeywell Technology Center in the use of genetic algorithms to synthesize and optimize the design of neural networks. Initial investigation was directed at validating the approach by optimizing an already trained neural network developed manually in a realistic application of heat rate modeling performed for EPRI by Robert Uhrig at the University of Tennessee. The same data sets, from TVA's Sequoyah nuclear power plant, were used for network training and testing. Appropriate criteria for optimization were determined to include accuracy (e.g. low error prediction of plant gross heat rate), learning speed, and network simplicity (e.g. low

number and density of connections). Neural network design experiments and evaluations began with a review of various network designs appropriate for the thermodynamic modeling application. Experiments were conducted that showed the genetically optimized networks had a significant performance improvement over manually developed networks. This research also demonstrated that the choice of input variables is critical in neural network applications. The genetic algorithm's ability to simultaneously optimize input selections, along with network structures and learning parameters, was vital to its accurate modeling. Another result was that genetic optimization need not be limited to evolving individual trained networks. Neural network architectures can be designed for classes of applications and then later trained on specific data for different applications within a given class. (EPRI 1994c) (Harp & Samad 1995)

Experiments on two additional applications have been completed: engine NO_x emissions and ozone levels in New York City. Both resulted in the automatic production of models competitive with the best conventional non-linear statistical models developed for these problems by the exercise of considerable statistical expertise and manual adjustment.

Demonstration software has been developed that runs under Microsoft Windows 3.x on a PC. The software features an easy-to-use graphical interface and incorporates tutorial material on neural networks, genetic algorithms, and the neurogenetic design technique. Two problems are included-heat rate modeling and engine NO_x emission prediction-along with optimized network models for them. The software also contains a neural network specification and training facility, allowing users to test and compare their own hand-crafted designs with the genetically optimized ones, and to have the GA optimize their design within the constraints they establish for it.

Research is continuing with the objective of demonstrating conclusively the power of genetic synthesis of neural networks for modeling and analysis applications throughout the electric power industry. An in-depth assessment of this design technique requires applying it to a variety of additional problems relevant to EPRI members. The first of these is the real-time pricing (RTP) of electric power in a competitive market. A GA is being used to design and optimize a neural network model using data from the Marriott Marquis Hotel in New York City. The data includes two years of hourly loads, rates, weather information, etc. obtained in a joint EPRI and Consolidated Edison RTP experiment. Accurately forecasting the loads and the rates would be of value to any electricity customer who had energy storage facilities or co-generation capability.

The investigators are also exploring ways to extend the state of the art for this general approach by exploring extensions to both the genetic and neural components of the system.

6. Simulation of the electric power market by evolving adaptive agents. Experiments with real people engaging in artificial economic systems have shown considerable deviations of actual behavior from theory. (Anderson *et al.* 1988) To gather more information quickly, researchers have been developing computer simulations by applying various forms of evolutionary algorithms to the modeling of economic systems, especially those embodying financial or commodity markets. (Friedman & Rust 1993) Most of these models treat the participants in the economic system as relatively autonomous agents representing individuals or institutions whose strategies can change and evolve as they engage both in cooperative and in competitive behavior so as to survive and prevail in a basically competitive environment. Evolutionary algorithms provide a convenient way to model the bounded rationality of real human beings as contrasted with the perfect rationality assumed by neo-classical economic theory. Studying economic systems using discrete event simulations with multiple adaptive agents makes it easy to include effects of imperfect information as well as the costs and benefits of information. The market, or other economic system, can be started in any prescribed state and the researcher can observe whether equilibrium (or any other hypothesized state) will actually evolve from the actions of the agents. Various strategies and regulations can be prescribed or be allowed to evolve, and the effects of new strategies or regulations can be immediately ascertained.

Commodity markets are a form of Dutch auction in which each prospective seller sets a high price, and reduces it gradually until some buyer accepts the sale at that price. Meanwhile, each prospective buyer sets a low price and gradually raises it until some prospective seller agrees to sell at that price. In simulated auction markets, many intelligent and adaptive agents buy and sell, each with whatever physical and economic constraints the modeler wishes to impose. Models of each type of auction provide general frameworks but allow behavior to emerge by actual or simulated human interaction with the simulations as with an artificial world. The market and the strategies of the agents evolve in a series of computer experiments which explore the various possible configurations that the market can take subject to different degrees and kinds of cooperation, competition and regulation. Results may include development of conditions for Nash equilibria, strategies that "clear the market", mutually beneficial strategies, the implications of differential information, and the likelihood of "chaos" developing. John Miller, of Carnegie-Mellon University and the Santa Fe Institute (SFI), along with others associated with SFI, has been exploring adaptive models of auction markets. They have run over 66 million auctions on machines. (Friedman & Rust 1993)

Bulk electric power has, in the past, been traded between separate electric utilities through bilateral bargaining agreements. With the advent of deregulation and competition in the power industry, "power pool" arrangements that resemble Dutch auction commodity markets are now

being set up both by institutions and by private parties. One example is the New York State Power Pool. However, there are significant differences that must be addressed in modeling the electric power market as compared with other commodity markets. Electricity has the shortest shelf-life of any manufactured product. Its perishability is partly compensated by the ability to transport it at almost the speed of light. But the infrastructure required for that transportation is made up of local parts that have limited capacities, and the viability of the whole system depends in an extremely complex way on the performance of each of those parts. While electricity also "flows" (from high voltage to low voltage locations), its transmission is inherently different from that of gas or water:

- Power flows through the grid in inverse relation to the impedance on each line.
- Electric power systems use phase shifters rather than valves.
- Providing the required flow on one line often results in "loop flows" on several other lines.
- Despite batteries and capacitors, and in contrast to "line packing" of gas or the use of reservoirs for water, there is no practical way to store large amounts of electricity for any significant length of time.
- Reliable electric service is critically dependent on the whole grid's ability to respond to changed conditions instantaneously.

In brokering electric power, the location of the buyer and seller (load and generation) have a significant effect on the price because the cost of delivery depends on all the other deliveries that are being made at that instant.

In order to gain practical insights into the anticipated future problem of brokering the use of the national transmission grid, EPRI has been sponsoring research by Gerald Sheblé of Iowa State University into this central deregulation issue whose financial basis is inherently nonlinear due both to nonlinear costs and to nonlinear network operation characteristics. As envisioned by those proposing to deregulate the electric power industry, there are two possible approaches to brokering. The first would be to auction incremental blocks of power and energy over a short time horizon. The second would include the longer-term problem of continually rescheduling power units as the network loading changes. The second approach involves not only power system non-linearity but also nonlinear cost and the nonlinear operational behavior of the power units. The EPRI project addressed only the first approach.

The toy stock market and computerized auction (used in the earlier research at the Santa Fe Institute) provided the starting point for this development which was based on an adaptive agent auction mechanism combined with economic dispatch models for generation and distribution companies. The evolutionary algorithm used is an extension of the brokerage programs for commodity markets described above. Agents repre-

sent the producers (generators) and the buyers (distributors). This project began EPRI's exploration of the practicality of using the bids of evolving agents to solve the power system network optimization by economic market mechanisms in place of economic dispatch of power by a central authority based on the solution of a mixed integer portfolio problem. (See earlier example.)

Previous studies of proposed brokerage schemes have not included transmission losses or equipment constraints because is presently unknown just what form deregulation may take. If deregulation takes the same form as in the oil/gas pipeline industry, the transmission companies will have to auction the buying and the selling of electricity at various points throughout the network. This research assumed that brokering will be expanded into a regional (i.e., multiple State) auctioning system. However, a reduced or an expanded problem can be handled in the same manner. The basic problem is to match the generation resources of each production company to the native (i.e., commercial and residential) demand while enabling economic interchange between transmission companies, interchange sales between generation companies and industrial loads, interchange sales between generation companies and transmission companies, and interchange sales between transmission companies and industrial loads. This formulation assumes that each transmission company will retain some generation in order to serve the native demand and to meet regulatory and reliability constraints, but that assumption is not required nor does the demonstration software depend on it.

A major problem in many brokerage situations is that the parties may have no idea of the true value of the commodity they are allocating through the process of buying and selling. For instance, they may not know the actual cost of production or delivery. Often, in standard brokerage systems, different decisions could be made using the same resources (capacity and money) in such a way that all parties would be better off. If it is impossible for any party to be better off without some party being worse off, the allocation is Pareto optimal. Decisions that are not Pareto optimal may occur frequently in buying and selling electrical power. When buying power, an electrical utility may not know the cost of generating and transmitting an equivalent amount of power. Similarly, utilities selling power may not know the actual cost they will incur to provide that power. The problem is very complicated due to the nonlinear nature of power generation itself as well as the nonlinear losses incurred in transmission. Therefore, it is critical that, when faced with buying and selling opportunities, utilities make decisions based on a price which that includes total system costs and capabilities.

In this project, a fixed schedule for power generation was assumed to exist over a short time horizon. The utility's primary concern in optimization under these conditions is the loading of equipment and the allocation of contracts. The non-linearity of loading is due to the complex simultaneous power flow equations that have to be solved to predict equipment loading

and equipment losses. The demonstration software uses linear approximations to power system operation, but it addresses the non-linearity of the interchange auctioning by using a genetic algorithm to achieve optimal brokerage.

The demonstration software, running under Windows 3.x on a Pentium PC, meets the goal of evaluating the interchange bids fast enough to simulate real-time auctioning. The auctioning system closely approximates the mechanics of the kind of commodities exchange envisioned for deregulated operation, although there is presently no consensus among EPRI members as to how, or even whether, a "free market" in electric power will develop in the next few years. The simulations resulting from this project are expected to help members' understanding of the potential problems and opportunities that may arise from any of the various scenarios that are being explored within the industry.

7. Multiple agent modeling for distributed control of the electric power grid. EPRI is pursuing a conceptual design for distributed control of the power system of the next century by intelligent agents operating locally with minimal supervisory control. The design integrates modeling, computation, sensing and control to meet the goals of efficiency and security in a geographically distributed system, subject to unavoidable natural disasters, and operated by partly competing and partly cooperating business organizations.

The computational model underlying this distributed control system will employ multiple intelligent agents to represent individual components of the grid – generators, transformers, buses – as if they were intelligent robots ("softbots"), cooperating and sometimes competing to keep the whole system operating in the most efficient manner despite whatever contingencies might befall it.

Each isolatable component of the electric power grid will be represented by an agent of limited intelligence which seeks to ensure its own survival while optimizing its performance in the context of the other agents. For instance, a single bus will strive to stay within its voltage and power flow limits while still operating in the context of the voltages and flows imposed on it by the combination of other agents representing generators, loads, transformers, etc. All lines, and most other components, have safety and capacity restrictions which are relatively "soft." High and low voltage limits, for instance, may not be exceeded by specified percentages for more than specified time periods. Maximum thermal limits, expressed in megavolt-amperes (MVA), are also set in percentage-time, but ultimately, most components would melt and overhead lines would sag until they caused a short circuit to the earth.

Any specific class of component may have additional survival constraints which may not be exceeded and, if exceeded, may require replacement or off-line maintenance. For instance, the chemical state of the oil

bath in a large transformer must stay within specified limits for safe operation. It's state may be tested periodically by taking samples, or instrumentation may be installed for continuous monitoring. The cost-benefit of this instrumentation for each transformer is an example of the management information the distributed model and simulation will provide.

More complex components, such as a generating plant or a substation, will be analyzed using object-oriented methods to model them as class and object hierarchies of simpler components. In addition to being a natural way to implement multi-agent simulations, object oriented programming is also a convenient technology for building libraries of reusable software and, in this case, will facilitate the exchange of agent (component) descriptions and potential improvements among EPRI's members and all interested researchers in the electric power community. These agents and subagents, represented as semi-autonomous "active objects" can be made to evolve (Wildberger 1995) by using combination of genetic algorithms (Goldberg 1988, Wildberger 1994a) and genetic programming. (Koza 1992, 1994) In this context, classes are treated as an analogy of biological genotypes and the objects instantiated from them as an analogy of their phenotypes.

Class attributes, which may stand for characteristics, capabilities, limitations or strategies possessed by potential agents, can be selected and recombined when instantiating objects to form different individuals by the operations typical of genetic algorithms, such as crossover and mutation. The physics of each component will determine its allowable strategies and behaviors. Existing instrumentation and control capabilities will be augmented in computer experiments with hypothetical capabilities which could be made optional in order to evaluate their benefit and the actual way in which their use might evolve. The operational parts of each class, its services or methods, may also be evolved through genetic programming using similar techniques. Success or "fitness" in this context combines the agents own survival with meeting the global security and efficiency goals.

A recent project has produced an Integrated Knowledge Framework (IKF) that describes the management and operational functions at a generic, coal-fired, steam power plant. (EPRI 1996b) This framework identifies the data, information and knowledge required for the important functions typically performed at a fossil power plant, as well as the flow of that data, information and knowledge between the function that generates it and those that use it. The project was conducted by a team of utility, EPRI, and equipment vendor personnel lead by the author. The resultant, object-oriented model contains 422 classes, interconnected through three different types of relations. While not solely developed as a design for distributed agent control and containing many classes which would not be appropriate for instantiation as active objects, this model defines, at an abstract level, all the structure needed to model the generation aspects of electric power with intelligent agents.

7.1. Issues: Time, credit, communication. A major issue in the design of agents is the interplay of short and long term evolution to provide what may be thought of as real-time adaptation to events that can only be anticipated in a very general way, viz.: lighting strikes. In the longer time frame, the softbots must learn new strategies and/or modify old ones in such a way that they can become better able to handle real-time control. Their long term fitness should be based mainly on the evolution of strategies for cooperation, methods for raising the sum of the game for all players. (Axelrod 1984) (Brams & Mattli 1993) However, their strategies for short term operation must evolve in real-time and may create temporary conflicts in real-time control, which must also be resolved in that same time frame. To reduce conflicts in the short term requires "look ahead" for real-time resolution. The strategies for where and how to "look" must have evolved over the long term. This is complicated in the electric power system case by the occurrence of non-ergodic effects. The state of the network after a failure, for instance, does not always allow a determination of the root cause of the problem. An analysis of the sequence of events may be necessary. While recovery from the degraded state may not hinge on knowing the root cause, efficient recovery and future prevention would.

Credit assignment is not particularly difficult when the situation provides immediate reward and/or precise information about the effects of actions taken. (Holland 1993) However, in the present case, the large number of parallel decisions and actions make it difficult to determine in real-time the exact benefits to the overall system of each individual's behavior. The present plan is to address this by taking advantage of the large number of individual components of each class that are found in the network. For purposes of long term evolution, they are treated as members of the same species so that combinations of strategies that were beneficial for real-time operation tend to be preserved and spread over the components of that same type.

Since individual agents are expected to collaborate and, indeed, are "in it together" as far as successful operation of the whole grid is concerned, it would appear necessary that they all have complete information about each other's state, actions and even decisions to take actions. However, this degree of detailed communication is not practical because of the combination of elapsed time and bandwidth that would be required. Therefore, it is assumed that each individual agent, in its own context, will contain a very rudimentary model (representation) of the entire system, which will also be subject to the long term evolution process. Only information about exceptions will be transmitted intentionally and explicitly over communication channels set up for that purpose. Even the frequency of these exception messages can be reduced by employing "tags" (Holland 1993) that might include unused capacity, reaction (ramp-up) time, etc. communicated as standing messages that remain valid until revised. However, the electric power grid itself provides some virtually instantaneous information about

its own state that is available locally to each component through instrumentation that is required in any case. Local voltage, current and frequency information may not provide all the necessary information but they can produce a first order approximation on which exception reporting can then be based.

8. Modeling the electric enterprise with multiple, adaptive agents. Over the next five years, the North American power network will undergo dramatic transformation as its operators–the individual electric utilities–respond to deregulation, competition, tightening environmental/land-use restrictions, and other industry trends. Requirements for wholesale and retail wheeling are beginning to decrease security margins, increase susceptibility to disturbances, and further complicate control of today's stressed, highly interconnected power systems. Meanwhile, competitive pressures are placing a premium on maximum use of transmission assets and more efficient dispatch of generation facilities. Deregulation and the introduction of competition is being achieved through the unbundling of electrical services: converting the historic, vertical integration of generation, transmission and distribution into separate companies, or at least separate services, each optimizing its performance based on different criteria and all operating at arms length. Common wisdom, based on the experience of other industries and other nations, expects that in five years there will be only about fifty companies. The generation companies will be completely deregulated, except for some lingering environmental constraints. The distribution companies will still be regulated, along the lines of today's local telephone companies, but major industrial/commercial customers, and cooperatives of individual residential customers, may generate their own power or buy it from the lowest bidder. The transmission companies will be partly regulated in an attempt to ensure open access and non-discriminatory pricing for "wheeling" power between any generator and any user or distributor, while maintaining some level of system security despite their lack of control of either generation or load. However, the above description represents just one hypothetical future scenario.

Current approaches to predicting the new business structure of the electric power industry are all driven by assumed scenarios like the one outlined above. The topology of the alternative scenarios/business structures dictate features of the future power system infrastructure which, in turn, create a need for new tools and make obsolete some of the existing tools for analysis, modeling, simulation and control. This is essentially a "top-down" approach to the setting of technological requirements. Its predictive accuracy depends entirely on the actual occurrence of the scenario or family of scenarios postulated. Recently, a joint PSERC-EPRI Working Session, at the University of Arizona's computer meeting support facility, used the scenario approach to investigate the underlying technical issues in electricity deregulation. From a set of nine alternatives prepared in advance

of the meeting the participants selected four specific scenarios for in-depth development during the workshop: Base Case, Dispersed, Centralized, and Minimum-ISO (Independent System Operator). The results of the groups' development of these four alternative cases identified the engineering and economic tools needed in the various structures and the degree to which these tools are departures from current practice or represent totally new tools.

8.1. Economic applications of a distributed agents-based model. Regardless of what conceivable future scenario actually takes place, one mutually beneficial objective for all interested parties–utilities, regulators, non-utility generators (NUGs) and consumers–will involve the enhanced coordination of system components so as to reduce outages, boost efficiency, enhance power quality, and increase use of existing assets. Automated management of future power networks may be necessary for utilities to achieve–in real time–the most satisfactory compromise between adequate security and optimally efficient operation. (Wildberger 1994b) Hence the design for distributed real-time control described above will be needed, or at least extremely useful, under any assumed scenario.

However, the multiple intelligent agent model and the computer simulation derived from it, although they are designed ultimately for distributed control, will also serve as a "scenario-free" method of studying the largely unknown effects from deregulation, unbundling and competition, without requiring any *a priori* assumptions about global scenarios that involve assumed political agendas. Its major constraints will be the laws of physics and the cost or availability of possible technological and economic solutions. Political accommodations and corporate restructuring will appear as global emergent behavior from these locally fixed agents cooperating and/or competing among themselves. This view, of course, has considerable similarity to the mathematical theory of games of strategy, but, unlike the generalized games solved by von Neuman or Nash, these are repeated games with non-zero sum payoffs. Information theoretic considerations are pertinent and these may, in turn, be represented by entropy in the state or phase space in which the system operates.

This model of the power system, and of the industry, can be used repeatedly for "what if" studies and computer experiments intended to provide insight into the evolution of the entire electric power industry under various forms of exogenous constraints. Individual companies may use it to examine the potential of entering into new partnerships or attempting to exploit new market segments. In addition, it will serve as a practical way to estimate the benefits of implementing any proposed new technology or making hypothetical changes to existing equipment and operating practices.

After the initial efforts toward economic market modeling which were described earlier, the next step in the development of a comprehensive,

high-fidelity, scenario-free modeling and optimization tool began in January, 1997, at Honeywell Technology Center. This project is intended to produce a meaningful first prototype implementation in one year. The first version of this tool will treat several aspects of the operation of the electric power industry in a simplified manner, but it will nevertheless be of immediate benefit to EPRI members for gaining strategic insights into the electricity marketplace. Two specific applications will be implemented in the tool. Candidates include: real-time pricing, co-generation, retail wheeling, and effects of new technological developments. The base functionality of the tool and the selected applications will be implemented in a PC Windows platform and will feature a graphical user interface designed to be familiar to utility personnel. To the extent practicable, file input/output formats will be adopted that are in common use today, such as the PSS/E data format for transmission networks, so that EPRI members will be able to tailor the applications for their own organizations. The tool will use object-oriented technology, along with a suitably abstract agent model definition, in order to facilitate future enhancements.

This prototype version of the model will include base-classes of the following entities:

G : Generation unit agents
T : Transmission system agents
L : Load agents
C : Corporate agents

The design and implementation of these agents will be sufficiently generic as not to limit how users may extend the system by specializing these classes or by defining new ones to allow for different kinds of generation, transmission, load, and corporate agents.

As these artificial agents evolve in a series of experiments, this simulation should expose the various possible configurations that the market and the industry could take, subject to different degrees and kinds of cooperation, competition and regulation. Possible results will be the development of conditions for equilibria, strategies or regulations that destabilize the market, mutually beneficial strategies, the implications of differential information, and the conditions under which chaotic behavior might develop. It is intended to gradually extend this model and simulation to include all four of the applications mentioned above as well as the effects of trading in futures, options and various derivatives. Further enhancements will emphasize greater fidelity in modeling the implication for each transaction of the resulting power flow (stability, security, etc.) on the existing network.

9. Summary. The Electric Power Research Institute is exploiting and enlarging the practical uses of evolutionary algorithms through internal and sponsored research and development. Examples described in this paper include:

- Real-time computation in the solution of non-linear, global optimization problems involving economic allocation of generation facilities and risk-based decision making in contingency planning.
- Automated supervision, design and validation of other adaptive systems using methods such as neural networks or symbolic artificial intelligence in various control and diagnostic applications.
- Distributed control of the electric power grid with components represented as autonomous evolving agents.
- Modeling the evolving market in electric power as it is deregulated and re-organizes itself for competition.
- Exploring the evolution of the electric enterprise itself in order to gain insights into how efficiency and security are affected by various combinations of economic control (by free markets) and command control (by a central agency).

Most of these applications require the invention of new algorithms and all of them would profit from the creation of new theory to guarantee their performance.

REFERENCES

[1] Anderson, P.W., Arrow, K.J. and D. Pines. eds. *The Economy as an Evolving Complex System*, Reading, MA: Addison Wesley, 1988.

[2] Axelrod, R. *The Evolution of Cooperation*, New York: Basic Books, 1994.

[3] Brams, S.T. and Mattli, W. Theory of Moves: Overview and Examples. *Conflict Management and Peace Science* 12(2):1–29, 1993.

[4] Electric Power Research Institute, *Optimization of the Unit Commitment Problem by a Coupled Gradient Network and by a Genetic Algorithm*, EPRI Technical Report TR-103697, Pleasant Hill, CA: EPRI Dist. Ctr. (May), 1994a.

[5] Electric Power Research Institute, *Evolutionary Computing*, EPRI Technical Brief, TB-104097. EPRI Dist. Ctr. Pleasant Hill, CA (Jun.), 1994b.

[6] Electric Power Research Institute, *Genetic Optimization of Neural Network Architecture* TR-104074, EPRI Dist. Ctr. Pleasant Hill, CA, 1994c.

[7] Electric Power Research Institute, *Genetic Algorithm Testbed for Expert System Testing* EPRI Report, TR-106004. EPRI Dist. Ctr. Pleasant Hill, CA (Jan.) 1996a.

[8] Electric Power Research Institute, *Integrated Knowledge Framework (IKF) for Coal-Fired Power Plants*, EPRI Technical Report TR-106211-V1/2/3, Pleasant Hill, CA: EPRI Dist. Ctr. (Mar), 1996b.

[9] Electric Power Research Institute, *High-Power Electronics Advanced Technology Program*, EPRI Brochure BR-1006800, Pleasant Hill, CA: EPRI Dist. Ctr, 1996c.

[10] Friedman, D. and J. Rust. ed., *The Double Auction Market*, Reading, MA: Addison Wesley, 1993.

[11] Goldberg, D., *Genetic Algorithms in Search Optimization and Machine Learning*, Reading, MA: Addison Wesley, 1988.

[12] Grefenstette, J.J. ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications* (Pittsburgh, July 1985), Lawrence Erlbaum, Hillsdale, NJ, 1988.

[13] Harp, S.A. and T. Samad, *Genetic algorithms and neural networks for optimized modeling and control* In *Mission Earth: Modeling and Simulation for a Sustainable Future, Proceedings of the 1995 Western Multiconference* (Las Vegas, Nevada, Jan. 15-18) A.M. Wildberger, ed., 14–20, San Diego, CA: Society for Computer Simulation, 1988.

[14] Holland, J.H., *The Effect of Labels (Tags) on Social Interactions*, Technical Report, 93-10-064, Santa Fe Institute, 1993.

[15] Koza, J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge. MA: MIT Press, 1992.

[16] Koza, J.R., *Genetic Programming II: Scalable Automatic Programming by Means of Automatically Defined Functions*, Cambridge. MA: MIT Press, 1994.

[17] Roache, E.A., K.A. Hickok, K.F. Loje, M.W. Hunt and J.J. Grefenstette, *Genetic Algorithms for Expert System Validation* In *Mission Earth: Modeling and Simulation for a Sustainable Future, Proceedings of the 1995 Western Multiconference* (Las Vegas, Nevada, Jan. 15–18) A.M. Wildberger, ed., 45–51, San Diego, CA: Society for Computer Simulation, 1995.

[18] Santa Fe Institute (sfi@santafe.edu), ftp://ftp.santafe.edu/pub/swarm/, or from WEB site: http://www.santafe.edu/projects/swarm/, 1996.

[19] Wildberger A.M., *Genetic Algorithms as a Modeling Tool* In *Proceedings of the 1994 European Simulation Symposium*, 24–28, Ghent, Belgium: Society for Computer Simulation International, 1994a.

[20] Wildberger, A.M., *Automated Management for Future Power Networks: A Long-Term Vision*, Public Utilities Fortnightly, 132, 20 (Nov.): 38–41, 1994b.

[21] Wildberger, A.M., *Some Implications of Soft Computing for Simulation*, In *Proceedings of the 1995 Summer Computer Simulation Conference*, 237–241, San Diego, CA: Society for Computer Simulation, 1995.

IMA SUMMER PROGRAMS

1987 Robotics
1988 Signal Processing
1989 Robust Statistics and Diagnostics
1990 Radar and Sonar (June 18 - June 29)
New Directions in Time Series Analysis (July 2 - July 27)
1991 Semiconductors
1992 Environmental Studies: Mathematical, Computational, and Statistical Analysis
1993 Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations
1994 Molecular Biology
1995 Large Scale Optimizations with Applications to Inverse Problems, Optimal Control and Design, and Molecular and Structural Optimization
1996 Emerging Applications of Number Theory (July 15 – July 26)
Theory of Random Sets (August 22 – August 24)
1997 Statistics in the Health Sciences
1998 Coding and Cryptography (July 6 – July 18)
Mathematical Modeling in Industry (July 22 – July 31)
1999 Codes, Systems and Graphical Models

SPRINGER LECTURE NOTES FROM THE IMA:

The Mathematics and Physics of Disordered Media
Editors: Barry Hughes and Barry Ninham
(Lecture Notes in Math., Volume 1035, 1983)

Orienting Polymers
Editor: J.L. Ericksen
(Lecture Notes in Math., Volume 1063, 1984)

New Perspectives in Thermodynamics
Editor: James Serrin
(Springer-Verlag, 1986)

Models of Economic Dynamics
Editor: Hugo Sonnenschein
(Lecture Notes in Econ., Volume 264, 1986)

Current Volumes:

1 **Homogenization and Effective Moduli of Materials and Media** J. Ericksen, D. Kinderlehrer, R. Kohn, and J.-L. Lions (eds.)

2 **Oscillation Theory, Computation, and Methods of Compensated Compactness** C. Dafermos, J. Ericksen, D. Kinderlehrer, and M. Slemrod (eds.)

3 **Metastability and Incompletely Posed Problems** S. Antman, J. Ericksen, D. Kinderlehrer, and I. Muller (eds.)

4 **Dynamical Problems in Continuum Physics** J. Bona, C. Dafermos, J. Ericksen, and D. Kinderlehrer (eds.)

5 **Theory and Applications of Liquid Crystals** J. Ericksen and D. Kinderlehrer (eds.)

6 **Amorphous Polymers and Non-Newtonian Fluids** C. Dafermos, J. Ericksen, and D. Kinderlehrer (eds.)

7 **Random Media** G. Papanicolaou (ed.)

8 **Percolation Theory and Ergodic Theory of Infinite Particle Systems** H. Kesten (ed.)

9 **Hydrodynamic Behavior and Interacting Particle Systems** G. Papanicolaou (ed.)

10 **Stochastic Differential Systems, Stochastic Control Theory, and Applications** W. Fleming and P.-L. Lions (eds.)

11 **Numerical Simulation in Oil Recovery** M.F. Wheeler (ed.)

12 **Computational Fluid Dynamics and Reacting Gas Flows** B. Engquist, M. Luskin, and A. Majda (eds.)

13 **Numerical Algorithms for Parallel Computer Architectures** M.H. Schultz (ed.)

14 **Mathematical Aspects of Scientific Software** J.R. Rice (ed.)

15 **Mathematical Frontiers in Computational Chemical Physics** D. Truhlar (ed.)

16 **Mathematics in Industrial Problems** A. Friedman

17 **Applications of Combinatorics and Graph Theory to the Biological and Social Sciences** F. Roberts (ed.)

18 ***q*-Series and Partitions** D. Stanton (ed.)

19 **Invariant Theory and Tableaux** D. Stanton (ed.)

20 **Coding Theory and Design Theory Part I: Coding Theory** D. Ray-Chaudhuri (ed.)

21 **Coding Theory and Design Theory Part II: Design Theory** D. Ray-Chaudhuri (ed.)

22 **Signal Processing Part I: Signal Processing Theory** L. Auslander, F.A. Grünbaum, J.W. Helton, T. Kailath, P. Khargonekar, and S. Mitter (eds.)

23 **Signal Processing Part II: Control Theory and Applications of Signal Processing** L. Auslander, F.A. Grünbaum, J.W. Helton, T. Kailath, P. Khargonekar, and S. Mitter (eds.)

24 **Mathematics in Industrial Problems, Part 2** A. Friedman

25 **Solitons in Physics, Mathematics, and Nonlinear Optics** P.J. Olver and D.H. Sattinger (eds.)

26 **Two Phase Flows and Waves** D.D. Joseph and D.G. Schaeffer (eds.)

27 **Nonlinear Evolution Equations that Change Type** B.L. Keyfitz and M. Shearer (eds.)

28 **Computer Aided Proofs in Analysis** K. Meyer and D. Schmidt (eds.)

29 **Multidimensional Hyperbolic Problems and Computations** A. Majda and J. Glimm (eds.)

30 **Microlocal Analysis and Nonlinear Waves** M. Beals, R. Melrose, and J. Rauch (eds.)

31 **Mathematics in Industrial Problems, Part 3** A. Friedman

32 **Radar and Sonar, Part I** R. Blahut, W. Miller, Jr., and C. Wilcox

33 **Directions in Robust Statistics and Diagnostics: Part I** W.A. Stahel and S. Weisberg (eds.)

34 **Directions in Robust Statistics and Diagnostics: Part II** W.A. Stahel and S. Weisberg (eds.)

35 **Dynamical Issues in Combustion Theory** P. Fife, A. Liñán, and F.A. Williams (eds.)

36 **Computing and Graphics in Statistics** A. Buja and P. Tukey (eds.)

37 **Patterns and Dynamics in Reactive Media** H. Swinney, G. Aris, and D. Aronson (eds.)

38 **Mathematics in Industrial Problems, Part 4** A. Friedman

39 **Radar and Sonar, Part II** F.A. Grünbaum, M. Bernfeld, and R.E. Blahut (eds.)

40 **Nonlinear Phenomena in Atmospheric and Oceanic Sciences** G.F. Carnevale and R.T. Pierrehumbert (eds.)

41 **Chaotic Processes in the Geological Sciences** D.A. Yuen (ed.)

42 **Partial Differential Equations with Minimal Smoothness and Applications** B. Dahlberg, E. Fabes, R. Fefferman, D. Jerison, C. Kenig, and J. Pipher (eds.)

43 **On the Evolution of Phase Boundaries** M.E. Gurtin and G.B. McFadden

44 **Twist Mappings and Their Applications** R. McGehee and K.R. Meyer (eds.)

45 **New Directions in Time Series Analysis, Part I** D. Brillinger, P. Caines, J. Geweke, E. Parzen, M. Rosenblatt, and M.S. Taqqu (eds.)

46 **New Directions in Time Series Analysis, Part II**
D. Brillinger, P. Caines, J. Geweke, E. Parzen, M. Rosenblatt, and M.S. Taqqu (eds.)

47 **Degenerate Diffusions**
W.-M. Ni, L.A. Peletier, and J.-L. Vazquez (eds.)

48 **Linear Algebra, Markov Chains, and Queueing Models**
C.D. Meyer and R.J. Plemmons (eds.)

49 **Mathematics in Industrial Problems, Part 5** A. Friedman

50 **Combinatorial and Graph-Theoretic Problems in Linear Algebra**
R.A. Brualdi, S. Friedland, and V. Klee (eds.)

51 **Statistical Thermodynamics and Differential Geometry of Microstructured Materials**
H.T. Davis and J.C.C. Nitsche (eds.)

52 **Shock Induced Transitions and Phase Structures in General Media** J.E. Dunn, R. Fosdick, and M. Slemrod (eds.)

53 **Variational and Free Boundary Problems**
A. Friedman and J. Spruck (eds.)

54 **Microstructure and Phase Transitions**
D. Kinderlehrer, R. James, M. Luskin, and J.L. Ericksen (eds.)

55 **Turbulence in Fluid Flows: A Dynamical Systems Approach**
G.R. Sell, C. Foias, and R. Temam (eds.)

56 **Graph Theory and Sparse Matrix Computation**
A. George, J.R. Gilbert, and J.W.H. Liu (eds.)

57 **Mathematics in Industrial Problems, Part 6** A. Friedman

58 **Semiconductors, Part I**
W.M. Coughran, Jr., J. Cole, P. Lloyd, and J. White (eds.)

59 **Semiconductors, Part II**
W.M. Coughran, Jr., J. Cole, P. Lloyd, and J. White (eds.)

60 **Recent Advances in Iterative Methods**
G. Golub, A. Greenbaum, and M. Luskin (eds.)

61 **Free Boundaries in Viscous Flows**
R.A. Brown and S.H. Davis (eds.)

62 **Linear Algebra for Control Theory**
P. Van Dooren and B. Wyman (eds.)

63 **Hamiltonian Dynamical Systems: History, Theory, and Applications**
H.S. Dumas, K.R. Meyer, and D.S. Schmidt (eds.)

64 **Systems and Control Theory for Power Systems**
J.H. Chow, P.V. Kokotovic, R.J. Thomas (eds.)

65 **Mathematical Finance**
M.H.A. Davis, D. Duffie, W.H. Fleming, and S.E. Shreve (eds.)

66 **Robust Control Theory** B.A. Francis and P.P. Khargonekar (eds.)

67 **Mathematics in Industrial Problems, Part 7** A. Friedman

68 **Flow Control** M.D. Gunzburger (ed.)

69 **Linear Algebra for Signal Processing**
A. Bojanczyk and G. Cybenko (eds.)

70 **Control and Optimal Design of Distributed Parameter Systems**
J.E. Lagnese, D.L. Russell, and L.W. White (eds.)

71 **Stochastic Networks** F.P. Kelly and R.J. Williams (eds.)

72 **Discrete Probability and Algorithms**
D. Aldous, P. Diaconis, J. Spencer, and J.M. Steele (eds.)

73 **Discrete Event Systems, Manufacturing Systems, and Communication Networks**
P.R. Kumar and P.P. Varaiya (eds.)

74 **Adaptive Control, Filtering, and Signal Processing**
K.J. Åström, G.C. Goodwin, and P.R. Kumar (eds.)

75 **Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations** I. Babuska, J.E. Flaherty, W.D. Henshaw, J.E. Hopcroft, J.E. Oliger, and T. Tezduyar (eds.)

76 **Random Discrete Structures** D. Aldous and R. Pemantle (eds.)

77 **Nonlinear Stochastic PDEs: Hydrodynamic Limit and Burgers' Turbulence** T. Funaki and W.A. Woyczynski (eds.)

78 **Nonsmooth Analysis and Geometric Methods in Deterministic Optimal Control** B.S. Mordukhovich and H.J. Sussmann (eds.)

79 **Environmental Studies: Mathematical, Computational, and Statistical Analysis** M.F. Wheeler (ed.)

80 **Image Models (and their Speech Model Cousins)**
S.E. Levinson and L. Shepp (eds.)

81 **Genetic Mapping and DNA Sequencing**
T. Speed and M.S. Waterman (eds.)

82 **Mathematical Approaches to Biomolecular Structure and Dynamics**
J.P. Mesirov, K. Schulten, and D. Sumners (eds.)

83 **Mathematics in Industrial Problems, Part 8** A. Friedman

84 **Classical and Modern Branching Processes**
K.B. Athreya and P. Jagers (eds.)

85 **Stochastic Models in Geosystems**
S.A. Molchanov and W.A. Woyczynski (eds.)

86 **Computational Wave Propagation**
B. Engquist and G.A. Kriegsmann (eds.)

87 **Progress in Population Genetics and Human Evolution**
P. Donnelly and S. Tavaré (eds.)

88 **Mathematics in Industrial Problems, Part 9** A. Friedman

89 **Multiparticle Quantum Scattering With Applications to Nuclear, Atomic and Molecular Physics** D.G. Truhlar and B. Simon (eds.)

90 **Inverse Problems in Wave Propagation** G. Chavent, G. Papanicolau, P. Sacks, and W.W. Symes (eds.)

91 **Singularities and Oscillations** J. Rauch and M. Taylor (eds.)

92 **Large-Scale Optimization with Applications, Part I: Optimization in Inverse Problems and Design**
L.T. Biegler, T.F. Coleman, A.R. Conn, and F. Santosa (eds.)

93 **Large-Scale Optimization with Applications, Part II: Optimal Design and Control**
L.T. Biegler, T.F. Coleman, A.R. Conn, and F. Santosa (eds.)

94 **Large-Scale Optimization with Applications, Part III: Molecular Structure and Optimization**
L.T. Biegler, T.F. Coleman, A.R. Conn, and F. Santosa (eds.)

95 **Quasiclassical Methods**
J. Rauch and B. Simon (eds.)

96 **Wave Propagation in Complex Media**
G. Papanicolaou (ed.)

97 **Random Sets: Theory and Applications**
J. Goutsias, R.P.S. Mahler, and H.T. Nguyen (eds.)

98 **Particulate Flows: Processing and Rheology**
D.A. Drew, D.D. Joseph, and S.L. Passman (eds.)

99 **Mathematics of Multiscale Materials** K.M. Golden, G.R. Grimmett, R.D. James, G.W. Milton, and P.N. Sen (eds.)

100 **Mathematics in Industrial Problems, Part 10** A. Friedman

101 **Nonlinear Optical Materials** J.V. Moloney (ed.)

102 **Numerical Methods for Polymeric Systems** S.G. Whittington (ed.)

103 **Topology and Geometry in Polymer Science** S.G. Whittington, D. Sumners, and T. Lodge (eds.)

104 **Essays on Mathematical Robotics** J. Baillieul, S.S. Sastry, and H.J. Sussmann (eds.)

105 **Algorithms For Parallel Processing** M.T. Heath, A. Ranade, and R.S. Schreiber (eds.)

106 **Parallel Processing of Discrete Problems** P.M. Pardalos (ed.)

107 **The Mathematics of Information Coding, Extraction, and Distribution** G. Cybenko, D.P. O'Leary, and J. Rissanen (eds.)

108 **Rational Drug Design** D.G. Truhlar, W. Howe, A.J. Hopfinger, J. Blaney, and R.A. Dammkoehler (eds.)

109 **Emerging Applications of Number Theory** D.A. Hejhal, J. Friedman, M.C. Gutzwiller, and A.M. Odlyzko (eds.)

110 **Computational Radiology and Imaging: Therapy and Diagnostics** C. Börgers, F. Natterer (eds.)

111 **Evolutionary Algorithms** L.D. Davis, K.D. Jong, M.D. Vose, and L.D. Whitley (eds.)

112 **Statistics in Genetics** S. Geisser and M.E. Halloran (eds.)

FORTHCOMING VOLUMES

1992–1992: *Control Theory*
Robotics

1996 Summer Program: *Emerging Applications of Number Theory*

1996–1997: *Mathematics in High Performance Computing*
Algorithms for Parallel Processing
Evolutionary Algorithms
The Mathematics of Information Coding, Extraction and Distribution
Structured Adaptive Mesh Refinement Grid Methods
Computational Radiology and Imaging: Therapy and Diagnostics
Mathematical and Computational Issues in Drug Design
Rational Drug Design
Grid Generation and Adaptive Algorithms
Parallel Solution of Partial Differential Equations

1997 Summer Program: *Statistics in the Health Sciences*
Week 1: Genetics
Week 2: Imaging
Week 3: Diagnosis and Prediction
Weeks 4 and 5: Design and Analysis of Clinical Trials
Week 6: Statistics and Epidemiology: Environment and Health

1997–1998: *Emerging Applications for Dynamical Systems*
Numerical Methods for Bifurcation Problems
Multiple-time-scale Dynamical Systems
Dynamics of Algorithms