GLOBAL OPTIMIZATION FOR CONSTRAINED NONLINEAR PROGRAMMING

BY

TAO WANG

B.E., Zhejiang University, 1989
M.E., Zhejiang University, 1991

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2001

Urbana, Illinois

Abstract

In this thesis, we develop *constrained simulated annealing* (CSA), a global optimization algorithm that asymptotically converges to constrained global minima (CGM_{dn}) with probability one, for solving discrete constrained nonlinear programming problems (NLPs). The algorithm is based on the necessary and sufficient condition for constrained local minima (CLM_{dn}) in the theory of discrete constrained optimization using Lagrange multipliers developed in our group. The theory proves the equivalence between the set of discrete saddle points and the set of CLM_{dn}, leading to the first-order necessary and sufficient condition for CLM_{dn}.

To find a CGM_{dn}, CSA searches for a discrete saddle point with the minimum objective value by carrying out both probabilistic descents in the original-variable space of a discrete augmented Lagrangian function and probabilistic ascents in the Lagrange-multiplier space. We prove that CSA converges asymptotically to a $CGM_{dn}$with probability one. We also extend CSA to solve continuous and mixed-integer constrained NLPs. By achieving asymptotic convergence, CSA represents one of the major developments in nonlinear *constrained* global optimization today, which complements *simulated annealing* (SA) in *unconstrained* global optimization.

Based on CSA, we have studied various strategies of CSA and their trade-offs for solving continuous, discrete, and mixed-integer NLPs. The strategies evaluated include adaptive neighborhoods, distributions to control sampling, acceptance probabilities, and cooling schedules. An optimization software package based on CSA and its various strategies has been implemented.

Finally, we apply CSA to solve a collection of engineering application benchmarks and design filters for subband image coding. Much better results have been reported in comparison with other existing methods.

To my wife Xiao, and my parents

Acknowledgments

First of all, I would like to express my sincere gratitude to my advisor, Professor Benjamin W. Wah, for his invaluable guidance, advice and encouragement during the course of my graduate study. His constant confidence and persistence in face of research uncertainties inspire me much and teach me to be a real scientist.

I would like to thank Professor Michael T. Heath for granting me a one-year fellowship in the computational science & engineering (CSE) program, and Professor Pierre Moulin, my co-advisor in the CSE program, for his helpful discussions and suggestions toward my research in subband image coding. I wish to thank Professors Narendra Ahuja, Michael T. Heath, Pierre Moulin, and Sylvian R. Ray for serving on my Ph.D. committee and for providing many useful comments and suggestions.

I would also like to thank all the members in our research group for providing crucial comments on the work and for providing a congenial environment for me to work in.

I wish to thank my parents and my wife, Xiao, for their everlasting love and support.

Finally, I would like to acknowledge the support of National Science Foundation Grant MIP 96-32316, National Aeronautics and Space Administration Grant NAG 1-613, and the CSE program in the University of Illinois at Urbana-Champaign, without which this work would not have been possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A Variety of applications in engineering, decision science, and operations research have been formulated as constrained *nonlinear programming problems* (NLPs). Such applications include neural-network learning, digital signal and image processing, structural optimization, engineering design, computer-aided-design (CAD) for VLSI, database design and processing, nuclear power plant design and operation, mechanical design, and chemical process control [68, 145, 180]. Due to the availability of a lot of unconstrained optimization algorithms, many real applications that are inherently nonlinear and constrained have been solved in various unconstrained forms. Optimal or good solutions to these applications have significant impacts on system performance, such as low-cost implementation and maintenance, fast execution, and robust operation.

Every constrained NLP has three basic components: a set of unknowns or variables to be determined, an objective function to be minimized or maximized, and a set of constraints to be satisfied. Solving such a problem amounts to finding values of variables that optimize (minimize or maximize) the objective function while satisfying all the constraints.

In this chapter, we first define constrained NLPs, some related basic concepts, and our research goals. We then describe the contributions of this research and outline the organization of this thesis.

1.1 Problem Formulation

A general constrained *nonlinear programming problem* (NLP) that we study in this thesis takes the following form

$$\begin{array}{lll} \textit{minimize} & f(x) & \\ \textit{subject to} & h(x) = 0 & x = (x_1, \ldots, x_n) \\ & g(x) \le 0 & \end{array} \tag{1.1}$$

where $f(x)$ is an objective function that we want to minimize. $h(x) = [h_1(x), \cdots, h_m(x)]^T$ is a set of m equality constraints, and $g(x) = [g_1(x), \cdots, g_k(x)]^T$ is a set of k inequality constraints. All $f(x)$, $h(x)$, and $g(x)$ are either linear or nonlinear, convex or nonconvex, continuous or discontinuous, analytic (*i.e.*, in closed-form) or procedural (*i.e.*, evaluated by some procedure or simulation). Variable space X is composed of all possible combinations of variables x_i, $i = 1, 2, \cdots, n$. In contrast to many existing NLP theory and methods, our formulation has no requirements on convexity, differentiability, and continuity of the objective and constraint functions.

Without loss of generality, we discuss our results with respect to minimization problem (1.1), knowing that maximization problems can always be transformed into minimization problems by negating their objective functions. Therefore, we use optimization and minimization interchangeably in this thesis. Two special cases are involved: a) an *unconstrained NLP* if there is no constraint, and b) a *constraint-satisfaction problem* if there is no objective function.

With respect to minimization problem (1.1), we make the following assumptions.

- Objective function $f(x)$ is lower bounded, but constraints $h(x)$ and $g(x)$ can be either bounded or unbounded.
- All variables x_i ($i = 1, 2, \cdots, n$) are bounded.
- All functions $f(x)$, $h(x)$, and $g(x)$ can be either linear or nonlinear, convex or nonconvex, continuous or discontinuous, differentiable or non-differentiable.

In some applications, variables are restricted to take prespecified values. According to the values that variable x takes, we have three classes of constrained NLPs [141]:

- **Discrete problems:** Variable x is a vector of discrete variables, where component x_i takes discrete and finite values, such as integers. Although variable space X at this time is finite (because variable x is bounded), it is usually very huge, making it impossible to enumerate every combination of x.
- **Continuous problems:** Variable x is a vector of continuous variables, $x_i \in R$, and $x \in R^n$. Variable space X is infinite.
- **Mixed-integer problems (MINLP):** Some variables take discrete values while others take continuous values. Let I_d be the set of indices of discrete variables, and I_c be those of continuous variables. Then $I_d \bigcup I_c = \{1, 2, \cdots, n\}$, and $I_d \bigcap I_c = \emptyset$. Variable space X is infinite because of continuous variables.

1.2 Basic Concepts

To characterize the solutions sought, we introduce some basic concepts [31, 125, 145, 69] on neighborhoods, feasible solutions, and constrained local and global minima here.

Definition 1.1 $\mathcal{N}(x)$, the *neighborhood* of point x in variable space X, is a set of points $x' \in X$ such that $x' \in \mathcal{N}(x) \Longleftrightarrow x \in \mathcal{N}(x')$.

Neighborhood $\mathcal{N}(x)$ has different meanings for the three types of constrained NLPs defined in (1.1). For a discrete problem on a discrete space, neighborhood $\mathcal{N}_{dn}(x)$ is in general user-defined and application-dependent [1, 221]. The choice of neighborhood, however, does not affect the validity of a search as long as one definition is used consistently throughout. Normally, one may choose $\mathcal{N}_{dn}(x)$ to include nearby discrete points to x so that neighborhood carries its original meaning. However, one may also choose the neighborhood to contain "far away" points.

For a continuous problem, neighborhood $\mathcal{N}_{cn}(x)$ is well-defined and application independent [31, 125, 106]. It includes those points that are sufficiently close to x, *i.e.*, $\mathcal{N}_{cn}(x)$ is a set of points x' such that $||x' - x|| < \varepsilon$ for some small $\varepsilon > 0$. For a mixed-integer problem, neighborhood $\mathcal{N}_{mn}(x)$ can be defined as a joint neighborhood of its discrete subspace x_i ($i \in I_d$) and its continuous subspace x_j ($j \in I_c$), where the neighborhood of its discrete subspace is user-defined and application-dependent.

Definition 1.2 Point $x \in X$ is called a *feasible point*, if x satisfies all the constraints; that is, $h(x) = 0$ and $g(x) \leq 0$.

Definition 1.3 Point $x^* \in X$ is called a *constrained local minimum* (CLM), iff a) x^* is a feasible point, and b) for every feasible point $x \in \mathcal{N}(x^*)$, $f(x^*) \leq f(x)$.

Note that when all neighboring points of x^* are infeasible, that is, x^* is the only feasible point surrounded by infeasible points, x^* is still a CLM. To avoid confusion of CLM in discrete, continuous, and mixed-integer spaces, we denote, respectively, CLM_{dn} to be a CLM in discrete space, CLM_{cn} to be a CLM in continuous space, and CLM_{mn} to be a

CLM in mixed-integer space. Further, in a discrete or mixed-integer space, point x may be a CLM_{dn} or CLM_{mn} for one definition of $\mathcal{N}_{dn}(x)$ or $\mathcal{N}_{mn}(x)$, but may not be for another definition of $\mathcal{N}'_{dn}(x)$ or $\mathcal{N}'_{mn}(x)$.

Definition 1.4 Point $x^* \in X$ is called a *constrained global minimum* (CGM), iff a) x^* is a feasible point, and b) for every feasible point $x \in X$, $f(x^*) \leq f(x)$. The set of all the CGM is denoted by X_{opt}.

1.3 Research Goals

The general goal of this research is to develop a global optimization method for solving constrained NLP (1.1). Global optimization methods are methods that are able to find CGM of constrained NLPs by either hitting a CGM during the search or converging to one when the search stops.

As (1.1) does not have closed-form solutions and cannot be solved analytically except in some trivial cases, it is generally solved by some iterative procedure ψ [125, 141]. Formally, let Ω be a search space of ψ. Search space Ω is determined by procedure ψ, and may or may not be equal to variable space X. Given starting point $\omega(k=0) \in \Omega$, search procedure ψ generates a sequence of iterative points, $\omega(1), \omega(2), \cdots, \omega(k), \cdots$ in search space Ω, until some stopping conditions hold.

ψ is called a *deterministic* procedure [105, 225] if $\omega(k)$ is generated deterministically. Otherwise, ψ is called a *probabilistic* or *stochastic* procedure.

Finding CGM of (1.1) is challenging as well as difficult. First, $f(x)$, $h(x)$, and $g(x)$ may be non-convex and highly nonlinear, making it difficult to even find a feasible point or a feasible region. Moreover, it is not useful to keep a search within a feasible region, as feasible regions may be disjoint and the search may need to visit a number of feasible regions before finding a CGM. Second, $f(x)$, $h(x)$, and $g(x)$ may be discontinuous or may not have derivatives, rendering it impossible to apply existing theories and methods developed for solving continuous problems. Last, there may be a large number of CLM, trapping trajectories that only utilize local information.

Constrained global optimization is NP-hard [105, 191], because it takes exponential time to verify whether a feasible solution is optimal or not for a general constrained NLP. This is true even for quadratic programming problems with linear or box constraints. As stated by Griewank [88], global optimization is mathematically ill-posed in the sense that a lower bound for $f(x)$ cannot be given after any finite number of evaluations, unless $f(x)$ satisfies certain subsidiary conditions, such as Lipschitz conditions and the condition that the search area is bounded.

Global optimization algorithms can be further divided into deterministic or stochastic approaches. Deterministic approaches consist of analytic and decomposition methods that are applicable in special cases. Analytic methods can only solve some trivial cases of constrained NLPs, while decomposition methods are enumerative methods that decompose a large problem into smaller subproblems that are easier to solve. These methods, such as branch-and-bound search, do not work when constraints are highly nonlinear and cannot be linearized. Other methods, such as interval methods, require the continuity and differentiability of functions and, thus, do not work for discrete and mixed-integer NLPs. In addition, decomposition methods are computationally expensive, because they have to enumerate all possibilities of points or subproblems.

Stochastic methods sample search space Ω based on some probability distributions that generate $\omega(k)$. Depending on the coverage of search space Ω and the way that the best solution is kept, stochastic methods can at best converge to a CGM with probability one when time approaches infinity. When terminated in finite time, stochastic methods may be better than enumerative methods because they may find a CGM with high probability.

Stochastic methods have two interpretations [10, 225] with respect to global optimization. Let Ω_s be a subset of Ω that has a one-to-one correspondence with X_{opt}, the set of all CGM.

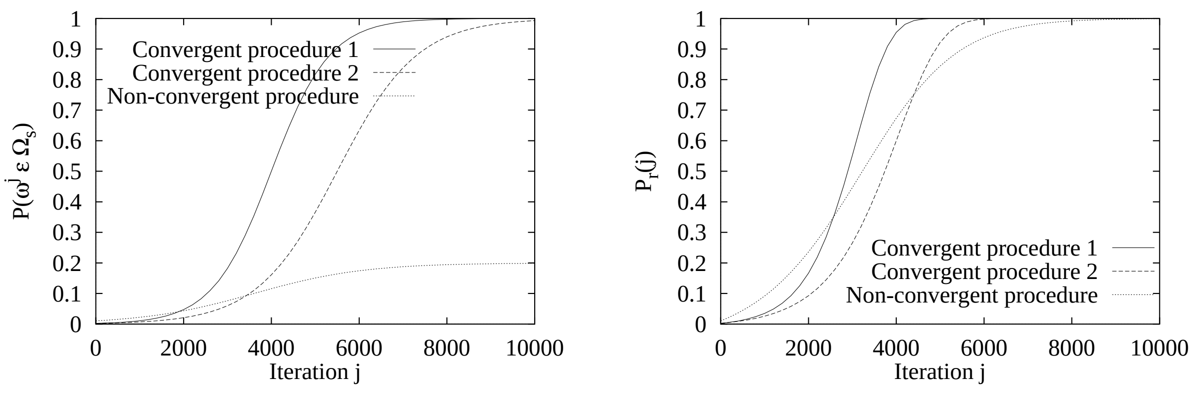
Definition 1.5 Procedure ψ is said to have *asymptotic convergence to global minimum*, or simply *asymptotic convergence* [10, 225], if ψ converges with probability one to an element of Ω_s, that is, $\lim_{k\to\infty} P(\omega(k) \in \Omega_s) = 1$, independent of starting point $\omega(k = 0)$.

Otherwise, ψ is called *non-asymptotically convergent* if it converges with a probability less than one to an element in Ω_s; that is, $\lim_{k\to\infty} P(\omega(k) \in \Omega_s) < 1$. Such a procedure is also said to have *reachability of global minimum* [10, 225] or *convergence in the best solution to a CGM*, because by keeping the best solution found in the course of ψ, ψ will eventually visit one of the CGM as time approaches infinity

Definition 1.6 Procedure ψ is said to have *reachability of global minimum* [10, 225], if probability $\lim_{k\to\infty} P(\omega(l) \in \Omega_s,\ \exists l,\ 0 \le l \le k) = 1$.

Reachability is much weaker than asymptotic convergence as it only requires $\omega(k)$ to hit a CGM sometime during a search. A pure random search is one example of global optimization with reachability. In contrast, asymptotic convergence requires ψ to converge to a CGM in Ω_s with probability one. Consequently, the probability of hitting a global solution increases as a search progresses, making it more likely to find a CGM than an algorithm with reachability alone if the search were stopped before it converges.

Here we assume that each iterative point $\omega(k)$ generated by procedure ψ is independent of the best solution visited so far. Otherwise, we can always transform an algorithm with reachability into the one with asymptotic convergence: For example, one simply remembers



a) Probabilities of hitting a CGM b) Corresponding reachability probabilities

Figure 1.1: An illustration showing the difference between asymptotic convergence and reachability. Algorithms with asymptotic convergence always have better reachability probabilities than algorithms without when the number of iterations exceeds some finite threshold.

the best solution (call it the incumbent) visited up to now, and for the next iteration, selects with probability p the incumbent and with probability $1 - p$ a new point according to the original reachability algorithm. Let p tend to approach one as the iterations progress. Then one obtains an algorithm with asymptotic convergence. This procedure, however, destroys the independence assumption.

Figure 1.1a illustrates the difference between asymptotically convergent and non-asymptotically convergent stochastic procedures. Note that the non-asymptotically convergent procedure in Figure 1.1a may have higher probabilities of hitting a CGM when the number of iterations is small.

In practice, asymptotic convergence can never be achieved because any algorithm must terminate within a finite amount of time. When an algorithm completes in t iterations, one is interested in its *reachability probability* $P_r(t)$ of hitting a CGM in any of its past t iterations, assuming all the iterations are independent,

$$P_r(t) = 1 - \prod_{j=1}^{t}(1 - P(\omega(j) \in \Omega_s)). \tag{1.2}$$

Figure 1.1b illustrates the reachability-probability curves corresponding to the curves in Figure 1.1a. It shows that algorithms with asymptotic convergence always have a better chance of finding a CGM when the number of iterations exceeds some finite threshold. For this reason, it is important to design algorithms with asymptotic convergence, especially in solving complex constrained NLPs.

In short, the first goal of this thesis is to develop a stochastic global optimization algorithm, called constrained simulated annealing (CSA), which achieves asymptotic convergence for solving discrete constrained NLP (1.1) and to extend it to solve continuous and mixed-integer NLPs. These constrained NLPs do not have the requirements on convexity, differentiability, and continuity of the objective and constraint functions. This goal is of great theoretical importance in nonlinear constrained global optimization.

There are three performance measures to evaluate a stochastic algorithm: the quality of solution f^* found, the maximum time (or iterations) T_r taken by the algorithm, and the probability P_r of finding such a solution f^*. Hence, the *average completion time* of finding a solution with quality f^* using multiple runs of the algorithm from random starting points is [198]:

$$\sum_{i=1}^{\infty} i \cdot T_r \cdot (1 - P_r)^{i-1} \cdot P_r = \frac{T_r}{P_r} \tag{1.3}$$

Here, P_r is a monotonically non-decreasing function of T_r. In general, if T_r is very small, it is very unlikely to find a solution of desired quality, leading to very small P_r and large T_r/P_r. As T_r increases, P_r will increase until it approaches one. After that, further increases in T_r will cause T_r/P_r to increase monotonically. Hence, there exists an optimal running time T_r such that T_r/P_r is minimized.

Because the T_r/P_r measure is only meaningful when dealing with solutions of the same quality f^*, the second goal of the thesis is to develop efficient strategies to implement our stochastic algorithm, measure its performance by depicting T_r/P_r with respect to various levels of quality of solution f^*, and compare them with some existing methods.

1.4 Contributions of This Thesis

The main contributions of this thesis are as follows:

- *Global optimization algorithm for discrete constrained NLPs (Chapter 3) [202, 200].* We have developed constrained simulated annealing (CSA), a global optimization algorithm that converges to a CGM_{dn} with probability one, for solving discrete constrained NLPs. CSA carries out both probabilistic descents of the discrete augmented Lagrangian function in the original-variable space and probabilistic ascents in the Lagrange-multiplier space. By achieving asymptotic convergence, CSA is one of the major developments in nonlinear constrained global optimization today. One of the papers published [200] was given the best paper award at the 11th IEEE Int'l Conf. on Tools with Artificial Intelligence, Nov., 1999.

- *Efficient strategies of CSA (Chapter 4) [203].* Our CSA procedure involves choosing neighborhoods, sampling distributions, acceptance probabilities, as well as cooling schedules. We have studied different strategies for each component, and have explored their tradeoffs for solving discrete, continuous, and mixed-integer constrained NLPs. Finally, an optimization software package based on CSA and its various strategies has been implemented.

- *Global optimization for solving engineering applications (Chapter 5 and 6) [202, 200, 203, 209].* We have applied CSA to solve a set of engineering benchmarks and design wavelet filters in subband image coding. We have obtained much better results on these benchmarks than others and have improved coding quality of real images.

- *Inequality constraint-handling and dynamic-weighting strategies (Appendix A) [201, 208, 205, 204, 210].* We have proposed adaptive and robust strategies to handle inequality constraints by *MaxQ* and dynamic weighting in order to speed up convergence in first-order methods for solving continuous constrained NLPs. Although this enhanced first-order method is not comparable to sequential quadratic programming (SQP) in terms of convergence speed, it represents new results on first-order methods. To highlight our major results on CSA in this thesis, we have described this method in Appendix A.

1.5 Outline of the Thesis

This thesis is organized as follows. In Chapter 2, we survey existing work on solving continuous, discrete, and mixed-integer constrained NLPs. These methods are classified based on the problem formulations they use, and for each formulation, further classified into local search, global search, and global optimization methods. We comment these methods based on four criteria: (c_1) what kind of optimality they achieve, local or global; (c_2) if global optimality, which convergence they achieve, asymptotic convergence or reachability, if they are stochastic methods; (c_3) whether they are able to cope with nonlinear objective and constraints; and (c_4) whether they require continuity and differentiability of the objective and constraint functions.

In Chapter 3, we develop *constrained simulated annealing* (CSA), a global optimization algorithm that converges to CGM_{dn} with probability one, for solving discrete constrained NLPs. Based on the necessary and sufficient conditions for CLM_{dn} (Theorem 2.2) in the theory of discrete constrained optimization using Lagrange multipliers (see Section 2.2.4), CSA aims to find a saddle point with the minimum objective value, *i.e.*, a CGM_{dn}. CSA carries out in an annealing fashion both probabilistic descents in the original variable subspace and probabilistic ascents in the Lagrange-multiplier subspace. By modeling CSA as a finite inhomogeneous Markov chain, we prove the asymptotic convergence of CSA to a CGM_{dn} with probability one. By achieving asymptotic convergence, CSA is one of the major developments in nonlinear *constrained* global optimization today and complements simulated annealing (SA) in nonlinear *unconstrained* global optimization.

In Chapter 4, we evaluate various strategies used in CSA that may affect its performance in solving discrete, continuous, and mixed-integer constrained NLPs. The strategies studied consist of adaptive neighborhoods, distributions to control sampling, acceptance probabilities, as well as cooling schedules.

In Chapter 5, we choose the best strategy among all the combinations of strategies tested in Chapter 4, and apply CSA with this strategy to solve three sets of continuous constrained benchmarks, and their derived discrete and mixed-integer NLPs. We report some improvements over the best-known solutions in comparison with two sequential-quadratic-programming (SQP) packages DONLP2 and LANCELOT, evolutionary algorithms (EAs) with specific constraint handling techniques, and interval methods.

In Chapter 6, we address filter-design issues in subband image coding by studying different problem formulations for filter design, including two constrained NLPs and one unconstrained NLP. We then solve them using such optimization algorithms as FSQP [226] and CSA. We demonstrate the performance of CSA by improving the coding quality of the filters in terms of their peak signal-to-noise ratios (PSNRs), and compare their performance with that of some well-known subband or wavelet filters.

Finally, in Chapter 7, we briefly summarize the major work we have presented in this thesis and point out future directions to extend and enhance this research.

In Appendix A, we develop two strategies to speed up the convergence of first-order methods for solving continuous constrained NLPs. To avoid oscillations or even divergence of the slack-variable method for handling inequality constraints, we propose the *MaxQ* method and study some techniques to speed up its convergence. We also propose a dynamic weighting strategy to balance the relative weights between the objective and the constraints. It monitors search progress and adaptively adjusts the weights once imbalance is detected. Using such a strategy, we are able to eliminate divergence, reduce oscillations, and greatly speed up convergence.

Chapter 2

Previous Work

Active research in the past four decades has produced a variety of methods for solving general constrained NLPs [185, 106, 69, 91, 127]. They fall into one of two general formulations, direct solution or transformation-based. The former aims to directly solve constrained NLP (1.1) by searching its feasible regions, while the latter first transforms (1.1) into another form before solving it. Transformation-based formulations can be further divided into penalty-based and Lagrangian-based. For each formulation, strategies that can be applied are classified as local search, global search, and global optimization.

Local search. Local search methods use local information, such as gradients and Hessian matrices, to generate iterative points and attempt to locate CLM quickly. Local search methods may not guarantee to find CLM, and their solution quality is heavily dependent on starting points. These CLM are CGM [141] only if (1.1) is *convex*, namely, the objective function $f(x)$ is convex, every inequality constraint $g_i(x)$ is convex, and every equality constraint $h_i(x)$ is linear.

Global search. Global search methods employ local search methods to find CLM and, as they get stuck at local minima, utilize some mechanisms, such as multistart, to escape from these local minima. Hence, one can seek as many local minima as possible and pick the best one as the result. These mechanisms can be either deterministic or probabilistic and do not guarantee to find CGM.

Global optimization. Global optimization methods are methods that are able to find CGM of constrained NLPs. They can either hit a CGM during their search or converge to a CGM when they stop (see Section 1.3 for more details).

In this chapter, we survey existing methods for solving each class of discrete, continuous, and mixed-integer constrained NLPs. We comment these methods based on four criteria: (c_1) what kind of optimality they can have, local or global; (c_2) if global optimality, what kind of convergence they can achieve, and whether asymptotic convergence or reachability is achieved, if they are stochastic methods; (c_3) whether they are able to cope with nonlinear objective and constraints; and (c_4) whether they require continuity and differentiability of the objective and constraint functions.

2.1 Search Methods for Continuous NLPs

In this section, we survey previous work on solving continuous constrained NLPs. We classify them based on the problem formulations they use, and for each formulation, classify them as local search, global search and global optimization. Figure 2.1 classifies existing search methods for solving continuous NLPs.

2.1.1 Direct Solutions for Continuous NLPs

Direct solution aims to solve (1.1) directly without transforming it into another form. Based on this formulation, solution methods can be classified into local search, global search, or global optimization.

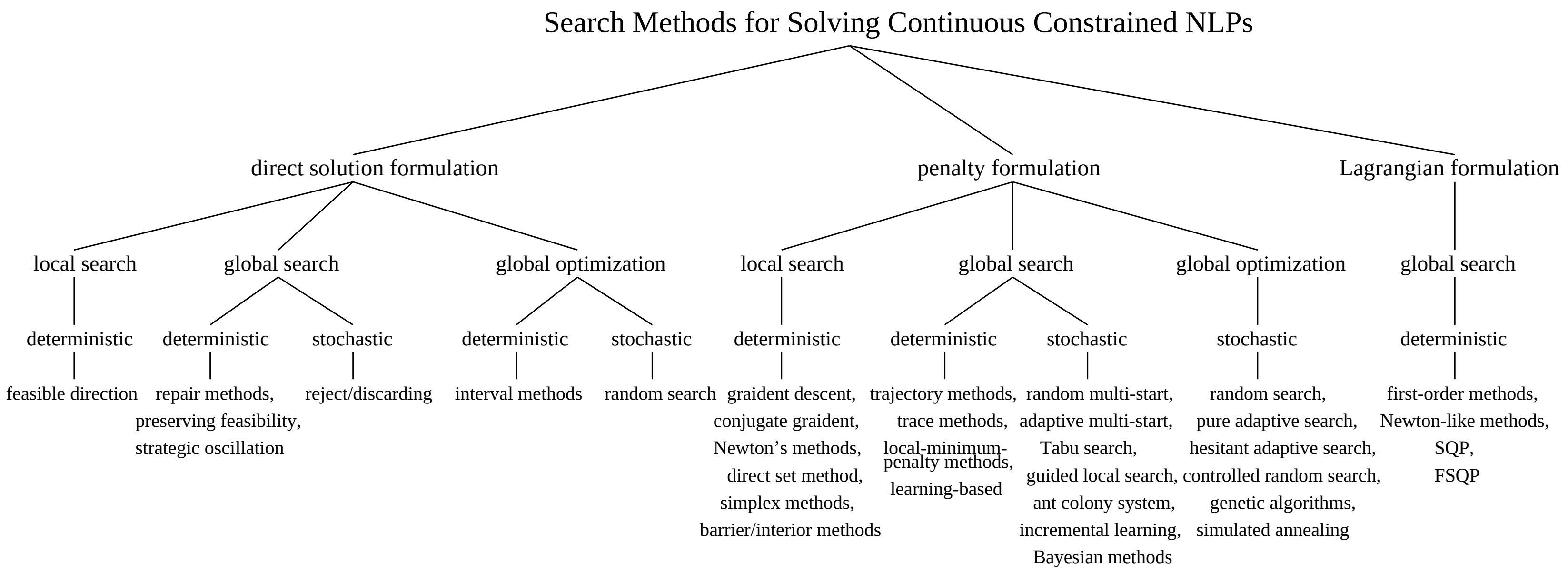

Search Methods for Solving Continuous Constrained NLPs
direct solution formulation
penalty formulation
Lagrangian formulation
local search
global search
global optimization
local search
global search
global optimization
global search
deterministic
deterministic
stochastic
deterministic
stochastic
deterministic
deterministic
stochastic
stochastic
deterministic
feasible direction
repair methods, preserving feasibility, strategic oscillation
reject/discarding
interval methods
random search
graident descent, conjugate graident, Newton's methods, direct set method, simplex methods, barrier/interior methods
trajectory methods, trace methods, local-minimum-penalty methods, learning-based
random multi-start, adaptive multi-start, Tabu search, guided local search, ant colony system, incremental learning, Bayesian methods
random search, pure adaptive search, hesitant adaptive search, controlled random search, genetic algorithms, simulated annealing
first-order methods, Newton-like methods, SQP, FSQP

Figure 2.1: Classification of search methods for solving continuous constrained NLPs.

Local Search. Feasible-direction methods [31, 122, 125] start from feasible points and iteratively maintain feasible directions by searching along gradients of the objective and projecting them into feasible regions. They try to reduce the value of objective function $f(x)$ during their search while staying within feasible regions. Such deterministic methods perform well with linear constraints. However, when there are nonlinear constraints, feasible regions may be disconnected, and keeping a search within a feasible region may lead to poor solutions. In addition, it is very difficult or expensive to project a trajectory into feasible regions for nonlinear constraints.

Global Search. Rejecting/discarding methods [110, 14, 160, 154] are stochastic procedures. They iteratively generate random points and only accept feasible points, while dropping infeasible points during their search. Although they are simple and easy to implement, they are very inefficient when constraints are nonlinear and feasible regions are difficult to find, because they spend a lot of time in generating and rejecting infeasible points. Adaptive simulated annealing (ASA) [13] implements one form of rejecting/discarding methods that requires the user to provide feasible points during its search.

Repair methods [113, 142] attempt to maintain feasibility by repairing infeasible points and transforming them into feasible points by some repair operators. Such deterministic methods, however, are usually very problem-specific and limited in their applications. Besides, restoring feasibility in case of nonlinear constraints may be as difficult as solving the original problem and is sometimes computationally expensive.

Preserving feasibility [131] and strategic oscillation [81, 172] are examples of deterministic repair methods. They involve specialized operators to maintain feasibility or keep a search on the boundaries of feasible regions. Both methods are problem dependent and cannot deal with general nonlinear constraints. Sometimes, keeping a search within a feasible region or on the boundary of a feasible region may be as difficult as solving the original constrained NLP. In addition, both methods are easy to get stuck at local minima when feasible regions are disconnected.

Global Optimization. Covering methods [106, 65, 91, 137] are deterministic algorithms. They detect subregions that do not contain CGM_{cn} and exclude them from further consideration. They can guarantee the quality of solutions and approximate CGM_{cn} by iteratively tightening bounds.

Interval methods are examples of covering methods that use interval arithmetic to recursively divide regions into subregions. These methods keep subregions that may have good solutions while dropping unpromising subregions. In order to verify the existence of feasible points for a given subregion, they usually utilize interval Newton methods [116, 115]. GlobSol [80] and ILOG solver [108] are popular software packages that implement these methods.

Interval methods, however, are unable to cope with general nonlinear constraints and may be misled by inaccurate evaluations of lower bounds. Obtaining solutions of guaranteed quality implies exhaustive search over a search space. Hence, interval methods have huge computational costs, with computational time increasing dramatically as problem size increases [23, 24], and can only be used to solve small problems. Besides, interval methods require the continuity and differentiability of functions and cannot be used for solving discrete and mixed-integer NLPs.

Pure random search samples uniformly a search space, while adaptive random search guides sampling based on information derived from its previous samples [104]. Both methods are stochastic approaches that guarantee the reachability of CGM_{cn} because any point in a search space has a chance to be found. The sequence of sample points, however, may not converge, or even diverge, and fail to provide asymptotic convergence. In addition, both methods may waste most of their time sampling infeasible points and are very inefficient for solving large problems or problems with nonlinear constraints.

2.1.2 Penalty Formulations for Continuous NLPs

Static-penalty formulations [31, 125, 145] transform (1.1) into a single unconstrained NLP by using a weighted sum of the objective and constraints,

$$min_x \;\; L_\rho(x, \gamma) = f(x) + \sum_{i=1}^{m} \gamma_i |h_i(x)|^\rho + \sum_{j=1}^{k} \gamma_{m+j} max^\rho(0, g_j(x)) \tag{2.1}$$

where constant $\rho > 0$, and penalty $\gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_{m+k}\}$ is *fixed* and chosen to be a large positive value. Note that other forms of penalty formulations are also available in the literature [31]. Because this formulation is simple to apply and can be solved by many existing well-defined unconstrained optimization methods, it has been widely used [125, 123, 102, 213, 150].

Selecting a suitable value of γ, however, proves to be very difficult. If it is chosen to be too large, the search terrain of $L_\rho(x, \gamma)$ may become too rugged to be searched by gradient-based methods. If it is too small, the solution to unconstrained problem (2.1) may not be a CLM_{cn} to the original problem or even may not be a feasible solution.

Dynamic-penalty formulations address these difficulties by increasing penalties gradually. They transform the original problem (1.1) into a sequence of unconstrained NLPs:

$$min_x \;\; L_\rho(x, \lambda(\kappa)) = f(x) + \sum_{i=1}^{m} \lambda_i(\kappa) |h_i(x)|^\rho + \sum_{j=1}^{k} \lambda_{m+j}(\kappa) max^\rho(0, g_j(x)) \tag{2.2}$$

using an increasing sequence $\lambda(\kappa), \kappa = 1, 2, \cdots, \mathcal{K}$, where $0 \leq \lambda(\kappa) < \lambda(\kappa+1)$, and $\lambda(\mathcal{K}) = \gamma$. Here $\lambda \leq \lambda'$ iff $\lambda_i \leq \lambda'_i$ for every $i = 1, 2, \cdots, m + k$, and $\lambda < \lambda'$ iff $\lambda \leq \lambda'$ and there exists at least one i such that $\lambda_i < \lambda'_i$.

Dynamic-penalty formulations avoid guessing the value of γ. Based on dynamic-penalty formulation (2.2), the solution obtained at one stage of $\lambda(k)$ is employed as a starting point for the next stage of $\lambda(k + 1)$. Therefore, the different stages are correlated, and the final solution to (1.1) depends on every stage of (2.2) for a given $\lambda(k)$.

Local Search. After transforming (1.1) into (2.1), one can solve it by many existing unconstrained local-search algorithms [125, 145, 161, 128], such as gradient descent, conjugate gradient, and Newton's methods and their variants. Direction-set (or Powell's) method [155] can be viewed as an approximation to conjugate gradient-descent methods. Because a large penalty γ is required, the search terrain of $L_\rho(x, \gamma)$ may be very rugged, and its unconstrained local minima may be very deep. Such a rugged search terrain causes the derivatives of $L_\rho(x, \gamma)$ to change very rapidly from one point to another, making it difficult for gradient-based methods to converge. It is also difficult for these methods to get out of deep local minima after getting stuck in them.

Simplex methods [138, 155] utilize a set of sample points to form a simplex, and iterate such a simplex by the reflection, expansion, and contraction operators until its volume is sufficiently small. Everytime, the worst point in a simplex is replaced by a new and better point. The rugged search terrain of $L_\rho(x, \gamma)$ and its associated deep local minima make simplex methods shrink quickly into a simplex of a small volume and get trapped at local minima before exploring a large search space.

Using dynamic-penalty formulation (2.2), if one increases λ slowly, the search is hopefully close to the solution before λ reaches a large value. Therefore, if one solves (2.2) at each stage of $\lambda(k)$ using gradient-based local-search methods [125, 145, 161, 128] or simplex methods [138, 155], the rugged terrain at large λ may have little effect. However, this may be computationally expensive [31], as more unconstrained problems need to be solved when λ increases slowly.

As an alternative to dynamic-penalty formulation (2.2), barrier methods [69, 145] also solve (1.1) by solving a sequence of unconstrained problems, each of which is controlled by a barrier parameter to prevent a search from going out of feasible regions. Such methods, however, require initial feasible points that may be as difficult as solving the original problem when there are nonlinear constraints. They only work for inequality constraints but not equality constraints. Interior point methods [214, 215, 141] extend the barrier methods by keeping non-negative variables strictly positive while allowing the search outside feasible regions. This gives them freedoms to start and proceed in infeasible regions. Since solutions of both methods depend highly on starting points, they are usually used for local search.

Global Search. To solve a single unconstrained problem (2.1) or a sequence of unconstrained problems (2.2), many global-search methods employ some heuristics to escape from local minima, identify good starting points, and use local-search methods to find local minima. Among all the CLM_{cn} found, the best one is reported as the search result. Hence, global-search methods are generally built on top of local-search methods. There is no guarantee that such global-search approaches are able to find CGM_{cn}. This is the major difference between global search and global optimization as discussed in Section 1.3.

According to strategies utilized to escape from local minima, global-search approaches can be either deterministic or stochastic. Deterministic global-search approaches consist of generalized descent methods (including trajectory or tunneling methods [194, 57, 170, 182], trace methods [197, 174] and local-minimum penalty methods [44, 75]) and learning-based methods [38, 39].

Generalized descent methods improve over local-search methods by continuing their search process whenever a local minimum is found. There are three approaches to achieve this. First, trajectory or tunneling methods [194, 57, 170, 182] modify the differential equations that control the local-descent trajectory in order to escape from local minima. These methods require the differentiability of the objective and constraint functions, and their main disadvantage is the large number of function evaluations that are spent at unpromising regions. Second, a trace-based method [197, 174] is a trajectory method that uses an external force to pull a search out of local minima and picks up good starting points from a couple of stages of global-search trajectories. This method is only applicable to continuous problems with differentiability and may also be limited to a small search space. Third, local-minimum penalty methods [44, 75] prevent multiple visits to the same local minima by adding to the objective a penalty term on each local minimum found. Their major problem is that, as more local minima are found, the modified objective function becomes more difficult to minimize and may not lead to good solutions.

A learning-based approach [38, 39] learns some relationships expressed as evaluation functions between starting points and solution quality of a local search method in order to predict good starting points using these evaluation functions. The evaluation functions derived must be simple (such as quadratic functions), although their simplicity limits their ability to capture information about complicated search terrains and leads to poor starting points.

Stochastic global-search approaches consist of random multistarts [171, 169, 96, 177], adaptive multi-starts [34], tabu search [82, 25], guided local search (GLS) [195], ant colony system [60, 59], population-based incremental learning [20] and its enhancements [36, 21, 22], as well as Bayesian methods [134, 196, 135].

Multi-start [171, 169, 96, 177] restarts a search by randomly generating new starting points after it gets trapped at local minima. This method may not find good solutions, because good starting points may not be easy to generate when the problem size is large. As the number of starting points is gradually increased, multi-start will approach the average quality of all the solutions, due to a consequence of the Central Limit Theorem [34]. Adaptive multi-start [34] explores a so-called globally convex structure, meaning that the best local minima appear to be at the center of other local minima. This method, however, is very problem specific and cannot be used for solving general optimization problems. Tabu search [82, 25] employs a tabu list to prevent a local search from revisiting recently found solutions and to avoid getting stuck at local minima. Because this heuristic is independent of solution quality, it may waste a lot of time in unpromising regions.

Explicit exploitations of search-related information have recently been used for global search. Guided local search (GLS) [195] extracts such information called solution features from previously found local minima of (2.1) and uses it to guide the next local search by adaptively updating the objective function of (2.1). Ant colony system [60, 59] uses positive feedbacks from the behavior of a group of ants or search agents for the discovery of good solutions. The major problems of GLS and ant colony system are two folds. First, they only use information about previously found local minima to guide the next local search but do not fully use information during the search process. Second, both methods are specific to some problems, such as the traveling salesman problem (TSP), and may not work well for general optimization problems. When there are nonlinear constraints, feasible regions may be disconnected, and information about one local minimum may not be useful for finding other local minima.

Population-based incremental learning (PBIL) [20] uses a population of solutions and captures the dependencies between individual solution parameters and solution quality using a simple model of unconditional probabilities. This model is entirely rederived after each generation. Mutual information maximization for input clustering (MIMIC) [36] extends PBIL by utilizing pairwise dependencies between solution parameters through a model of conditional probabilities. This pairwise probability model is further improved by more accurate dependency trees [21], whose high computational efforts have been reduced by combining optimizers with mutual information tree (COMIT) [22] and using fast search techniques like hillclimbing. However, these methods are still computationally expensive and assume some relationships between solution quality and solution parameters. Like GLS and ant colony system, such an assumption may not be correct, because the feasible regions modeled by nonlinear constraints may be disconnected, leading to little relevance among local minima in different feasible regions.

Bayesian methods [134, 196, 135] model original problem variables by random variables, construct stochastic functions, and minimize the expected deviation of the estimate from actual global minima. These methods do not work well because most of the samples that they collect randomly from the error surface are close to the average error value and are inadequate to model the behavior at minimum points. They are also too expensive to be applied to problems with more than twenty variables [185].

Global Optimization. To find CGM_{cn} using static-penalty formulation (2.1), one requires penalty γ to be large enough such that

$$L_\rho(x^*, \gamma) < L_\rho(x, \gamma) \quad \forall x \in X - X_{opt} \text{ and } x^* \in X_{opt}. \tag{2.3}$$

In this case, an unconstrained global minimum of (2.1) over x is a CGM_{cn} for (1.1). Hence, it is sufficient to minimize (2.1), whose search space is the same as variable space

X. The requirement of such a large penalty γ, however, may cause the search terrain of $L_\rho(x, \gamma)$ to be very rugged and its local minima to be very deep, making it very difficult to find optimal solutions to (2.1) in practice.

Instead of directly minimizing (2.1) using a large penalty γ, dynamic-penalty methods solve a sequence of unconstrained problems formulated as (2.2). They achieve asymptotic convergence if, for every $\lambda(\kappa)$, (2.2) is solved optimally [31, 125]. The requirement of obtaining an unconstrained global minimum of (2.2) in every stage is, however, difficult to achieve in practice, given only finite amount of time in each stage. If the result in one stage is not a global minimum, then the process cannot be guaranteed to find a CGM_{cn}. Some approximations that sacrifice global optimality of solutions have been developed, such as two-phase evolutionary programming [117] and two-phase neural networks [126].

Many stochastic global optimization algorithms have been developed to solve (2.1). Pure random search (PRS) randomly samples an entire search space X, whereas pure adaptive search (PAS) [148] tries to sample uniformly regions that may have better solutions. Hesitant adaptive search (HAS) [40] extends PAS by allowing the procedure to 'hesitate' before improvement continues, instead of requiring every iteration to produce an improved solution. Controlled random search (CRS) [156, 4, 7] maintains a population of points and is a generalization of the simplex method with randomization. Various versions of CRS differ in their ways of generating random points and replacement policies in the population.

Although PRS, PAS, HAS, and CRS are able to achieve reachability of optimal solutions, their major problems are two folds. First, it is impossible or very difficult for them to identify promising regions when constraints are nonlinear, not to mention that it is difficult to verify whether nonlinear constraints can be satisfied or not in a given region. In such a situation, these methods spend a lot of time in sampling infeasible regions. Second, because these methods focus too much on improving current solutions, they have small chances to overcome rugged search terrains and deep local minima and get stuck at local minima easily.

Genetic algorithms (GAs) [85, 71, 127, 167, 140, 147, 94] maintain a population of points in every generation and uses some genetic operators, such as cross-overs and mutations, to generate new points. These old and new points compete for survival in the next generation based on a fitness function modeled by (2.1). Variations of GAs include evolution strategies (ES) [14, 15], evolutionary programming (EP) [70, 71] as well as differential evolution (DE) [181]. By carefully choosing control parameters of cross-overs and mutations, GAs and its variants are capable of reaching every point from every other point in a search space. Therefore, these methods assure the reachability of optimal solutions, albeit asymptotic convergence.

Some variants of penalty formulations have also been used in GAs to handle constraints. The method of multi-level static-penalties [101] divides constraint violations into levels, each of which has its own penalty values. This method is very problem-dependent and cannot be generalized to other optimization problems. Generation-based dynamic-penalties [112], annealing penalties [130] and adaptive penalties [27, 89, 151] can be viewed as approximate implementations of dynamic-penalty formulation (2.2). Although they differ in their ways of modifying the penalties, all of them adjust penalties at the end of each generation, instead of when unconstrained problem (2.2) at previous penalty levels has been minimized. Accordingly, these mehods cannot achieve asymptotic convergence.

Behavioral memory with a linear order of constraints [173] copes with constraints in some predefined linear order. The major problem is that it is difficult to satisfy all the nonlinear constraints simultaneously, in the sense that enforcing the satisfaction of one constraint may cause other constraints to be violated. Furthermore, its results depend heavily on the order of constraints. A co-evolutionary algorithm [146, 132] is based on a coevolutionary model that utilizes two populations: one containing a set of constraints difficult to be satisfied and the other composed of potential solutions to the problem. These two populations are competitively evolved in such a way that fitter constraints are violated by less solutions while fitter solutions satisfy more constraints. This method is good for solving constraint satisfaction problems but may not work well for constrained NLPs, because it does not consider the objective seriously. Besides, it calculates fitness based on historical records, making it easy to get stuck in local minima.

All these methods have at least one of the following problems [129, 133, 130]: (a) difficulty in finding feasible regions or in maintaining feasibility for nonlinear constraints, (b) requiring specific domain knowledge or problem-dependent genetic operators, and (c) tendency to get stuck at local minima. A series of software packages, GENOCOP I, II, and III [76], utilize some of the above constraint handling techniques.

In contrast to GAs that generally accept points with reduced objective value of (2.1), simulated annealing (SA) [**?**, 43, 190, 152, 90, 124, 1, 63] accepts with some probabilities points that increase the objective. These probabilities are managed by a control parameter called temperature that is set initially to be a large value and then reduced slowly.

Besides achieving reachability like other stochastic global optimization methods, SA can achieve asymptotic convergence in solving unconstrained problem (2.1). This important property is promising in practice and has led to the development of many variants of SA with improved efficiency. These include varying the neighborhood during a search [221], combining with simplex method [155, 120], sampling by an improved Hit-and-Run (IHR) method [162, 223], incorporating SA with some local-search steps [56], aspiration-based SA [5] with a self-regulatory mechanism, and direct search SA (DSA) [6].

However, SA cannot be applied to solve *constrained* optimization problems for the following reasons. In order to solve a constrained problem, one has to first transform it into an unconstrained problem by a penalty formulation before applying SA. Large penalties required by (2.1) lead to rugged search terrains and deep local minima. In such a situation, the search has to overcome some infeasible regions by large increases in the values of its penalty functions, making it difficult for SA to move from one feasible region to another or escape from local minima, especially at low temperatures.

2.1.3 Lagrangian Formulations for Continuous NLPs

Lagrangian methods [31, 125, 145, 69, 104] were traditionally developed to solve *continuous* NLPs with differentiability. They utilize Lagrange multipliers to combine constraints with the objective function to form a Lagrangian function and then gradually resolve the constraints through iterative updates in Lagrangian search space (the joint space of variable x and Lagrange multipliers). Therefore, a local search in the Lagrangian space can also be viewed as a global search in variable space x, because updating Lagrange multipliers may bring the search out of local minima in x space.

Theory of Continuous Constrained Optimization Using Lagrange Multipliers. Lagrangian methods generally work with equality constraints and transform inequality constraints into equality constraints using slack variables [31, 125] before solving it. For example, inequality constraint $g_i(x) \leq 0$ can be transformed into either $g_i(x) + z_i = 0$ with $z_i \geq 0$ or $g_i(x) + z_i^2 = 0$. The former is rarely used because it increases the number of variables and has another form of inequality, whereas the latter does not have such drawbacks by eliminating slack variable z_i [31, 125]. Please also see Appendix A.1.1 for detail.

A general continuous equality-constrained NLP is defined as:

$$\begin{array}{lll} \text{minimize} & f(x) & x = (x_1, x_2, \ldots, x_n) \text{ is a vector} \\ \text{subject to} & h(x) = 0 & \text{of continuous variables} \end{array} \tag{2.4}$$

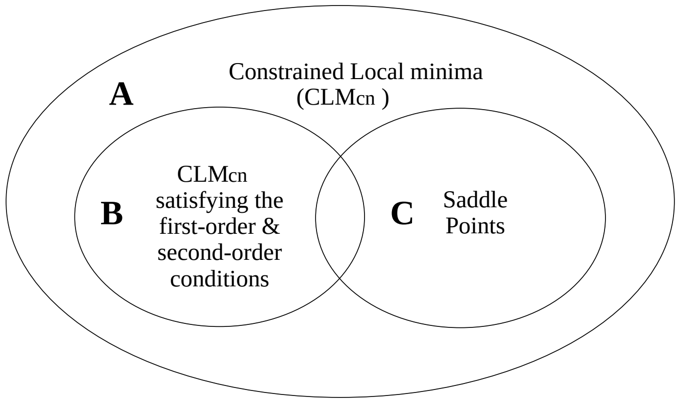


Figure 2.2: Relationship among solution sets of nonlinear continuous constrained NLPs.

where $h(x) = [h_1(x), \ldots, h_m(x)]^T$ is a set of m equality constraints. Both $f(x)$ and $h(x)$ are assumed to be differentiable.

Definition 2.1 The *augmented Lagrangian function* of (2.4) is defined as

$$L(x, \lambda) = f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2 \tag{2.5}$$

where λ is a vector of Lagrange multipliers, and $||h(x)||^2 = \sum_{i=1}^{m} h_i^2(x)$.

$L(x, \lambda)$ can be viewed a sum of objective $f(x)$ and constraint $h(x)$ weighted by Lagrange multiplier λ. In contrast to penalty and other heuristic formulations whose search space is X, a Lagrangian search works in the joint search space of $S = X \times \Lambda$ with variables x and λ. Therefore, any point in search space S is a vector (x, λ).

Definition 2.2 Point (x^*, λ^*) is called a *saddle point* of $L(x, \lambda)$, iff it satisfies $L(x^*, \lambda) \leq L(x^*, \lambda^*) \leq L(x, \lambda^*)$ for all (x^*, λ) and all (x, λ^*) sufficiently close to (x^*, λ^*).

Definition 2.3 Point x is said to be a *regular point* [125] if gradient vectors [125]

$$\nabla h_1(x),\ \nabla h_2(x),\ \cdots,\ \nabla h_m(x)$$

at point x are linearly independent.

Theorem 2.1 *First-order necessary conditions for continuous problems* [125]. Let x be a local extremum point of $f(x)$ subject to $h(x) = 0$, and assume that $x = (x_1, x_2, \cdots, x_n)$ is a regular point. Then there exists $\lambda \in R^m$ such that

$$\nabla_x L(x, \lambda) = 0; \qquad \nabla_\lambda L(x, \lambda) = 0. \tag{2.6}$$

Additional second-order sufficient conditions are needed to ensure that the solution to (2.6) is a CLM_{cn} of the continuous equality-constrained NLP (2.4).

Figure 2.2 depicts the relationships among the three solution sets [216, 206]: the set of CLM_{cn} (**A**), the set of solutions satisfying the first-order necessary and second-order sufficient conditions (**B**), and the set of saddle points (**C**). **A**-**B** is the set of CLM_{cn} that are either not differentiable or not regular points. If both the objective and constraint functions are differentiable, then $\mathbf{B} \subseteq \mathbf{C}$; that is, the set of points satisfying the first-order necessary and second-order sufficient conditions is a subset of the set of saddle points.

Global Search. Various Lagrangian methods aim to locate CLM_{cn} based on solving the first-order necessary conditions (2.6). First-order methods [125, 145, 69, 224, 46, 189] do both gradient descents in the original-variable subspace and gradient ascents in the Lagrange-multiplier subspace and can be written as a dynamic system that consists of a set of ordinary differential equations (ODEs),

$$\begin{aligned} \frac{dx}{dt} &= -\nabla_x L(x, \lambda) = -\nabla_x f(x) - \lambda^T \nabla_x h(x) \\ \frac{d\lambda}{dt} &= \nabla_\lambda L(x, \lambda) = h(x) \end{aligned} \tag{2.7}$$

where t is an autonomous time variable. From a starting point $(x(t = 0), \lambda(t = 0))$, the dynamic system in (2.7) evolves over time t by performing gradient descents in the original-variable subspace of x and gradient ascents in the Lagrange-multiplier subspace of λ. When the system reaches an equilibrium point where

$$\frac{dX}{dt} = 0 \quad \text{and} \quad \frac{d\lambda}{dt} = 0, \tag{2.8}$$

the first-order necessary conditions in (2.6) are satisfied. Because first-order methods only use gradients instead of Hessian matrices, their rate of convergence is linear [31, 141].

One can also solve these first-order necessary conditions in (2.6) by using Newton's methods or quasi-Newton methods [31, 125] in order to achieve quadratic or superlinear convergence and by using starting points that were sufficiently close to some local minima. To have global convergence from remote starting points, a variety of line-search techniques have been developed [31, 141].

Sequential quadratic programming (SQP) [41, 79, 35, 178, 107] approximates solutions to the first-order necessary conditions by solving a series of quadratic programming (QP) subproblems, each involving the minimization of a quadratic approximation of the objective function, subject to a linear approximation of the constraints. Such a QP problem is then solved efficiently using Newton-like methods in order to have at least superlinear convergence when near some local minima. To have global convergence from remote starting points, trust-region techniques have been used [141]. Feasible SQP (FSQP) is a variation of SQP in which all iterations are within feasible regions of relaxed constraints [143, 144]. It is very useful if the objective function is undefined or difficult to compute outside feasible regions. DONLP2 [58], FSQP [73], and LANCELOT [121] are examples of software packages that implement the above search methods for solving continuous constrained NLPs.

The major problems of first-order methods, Newton-like methods and SQP methods are two folds. First, they require the derivatives of both the objective and constraint functions, restricting them to continuous NLPs with differentiability but not discrete and mixed-integer

NLPs. Second, they are not guaranteed to find a CGM_{cn} even if they can find all the solutions satisfying the first-order necessary and second-order sufficient conditions in Theorem 2.1, because a CGM_{cn} may be in set **A**-**B** in Figure 2.2.

Table 2.1 summarizes existing methods for solving continuous constrained NLPs based on the four criteria for evaluating search methods.

Table 2.1: Summary of search methods for solving continuous constrained NLPs based on the four criteria of optimality achieved, convergence property for stochastic methods with global optimality, ability to handle nonlinear functions, and requirement on function continuity and differentiability.

Problem Formulations	Search Strategies	Search Methods	Achieved Optimality(c_1)	Convergence Property(c_2)	Ability to Handle Nonlinear F's(c_3)	Require Cont'y & Diff'y(c_4)
Direct Solutions	Local Search	feasible direction	local	none	no	yes
	Global Search	repair methods	local	none	no	no
		preserving feasibility	local	none	no	no
		startegy oscillation	local	none	no	no
		reject/discarding	local	none	yes	no
	Global Optimization	interval methods	global	none	no	yes
		random search	global	reachability	yes	no
Penlaty Formulations	Local Search	gradient descent	local	none	yes	yes
		conjugate gradient	local	none	yes	yes
		Newton's methods	local	none	yes	yes
		direct set methods	local	none	yes	yes
		simplex methods	local	none	yes	no
		barrier/interior	local	none	yes	yes
	Global Search	trajectory methods	local	none	yes	yes
		trace methods	local	none	yes	yes
		local-minimum penalty	local	none	yes	yes
		learning-based methods	local	none	yes	no
		random multi-start	local	none	yes	no
		adaptive multi-start	local	none	yes	no
		Tabu search	local	none	yes	no
		guided local search	local	none	yes	no
		ant colony system	local	none	yes	no
		incremental learning	local	none	yes	no
		Bayesian methods	local	none	yes	no
	Global Optimization	random search	global	reachability	yes	no
		pure adaptive search	global	reachability	yes	no
		hesitant adaptive search	global	reachability	yes	no
		controlled random search	global	reachability	yes	no
		genetic algorithms	global	reachability	yes	no
		simulated annealing	global	asymptotic	yes	no
Lagrangian Formulations	Global Search	first-order methods	local	none	yes	yes
		Newton-like methods	local	none	yes	yes
		SQP	local	none	yes	yes
		FSQP	local	none	yes	yes

2.2 Search Methods for Discrete NLPs

In this section, we survey existing work for solving discrete constrained NLPs whose variables take discrete values. In this case, derivatives are not available, causing many gradient- or Hessian-based approaches to fail, even when the objective and constraint functions are differentiable. Figure 2.3 classifies existing search methods for solving discrete constrained NLPs.

2.2.1 Direct Solutions for Discrete NLPs

Global Search. Direct-solution methods try to directly solve (1.1) based either on rejecting/discarding [110, 14, 160, 154] infeasible points or on repairing [113, 142] infeasible points into feasible ones. Both methods are not efficient in handling nonlinear constraints, because the former wastes a lot of time in generating and rejecting infeasible points whereas the latter is very problem-specific and has high computational cost to maintain feasibility.

Global Optimization. Direct solution can be either random search [104] or enumeration. Enumerative methods [211] utilize branch-and-bound algorithms to find lower bounds using linearized constraints. Lower bounds found this way are inaccurate when constraints are highly nonlinear. Hence, branch-and-bound methods do not work well on general discrete NLPs. Besides, such methods are computationally expensive and can only be used to solve small problems.

2.2.2 Penalty Formulations for Discrete NLPs

Local Search. Penalty-based methods first transform constrained problem (1.1) into a single unconstrained problem (2.1) and then solve it by greedy search or hillclimbing [157].

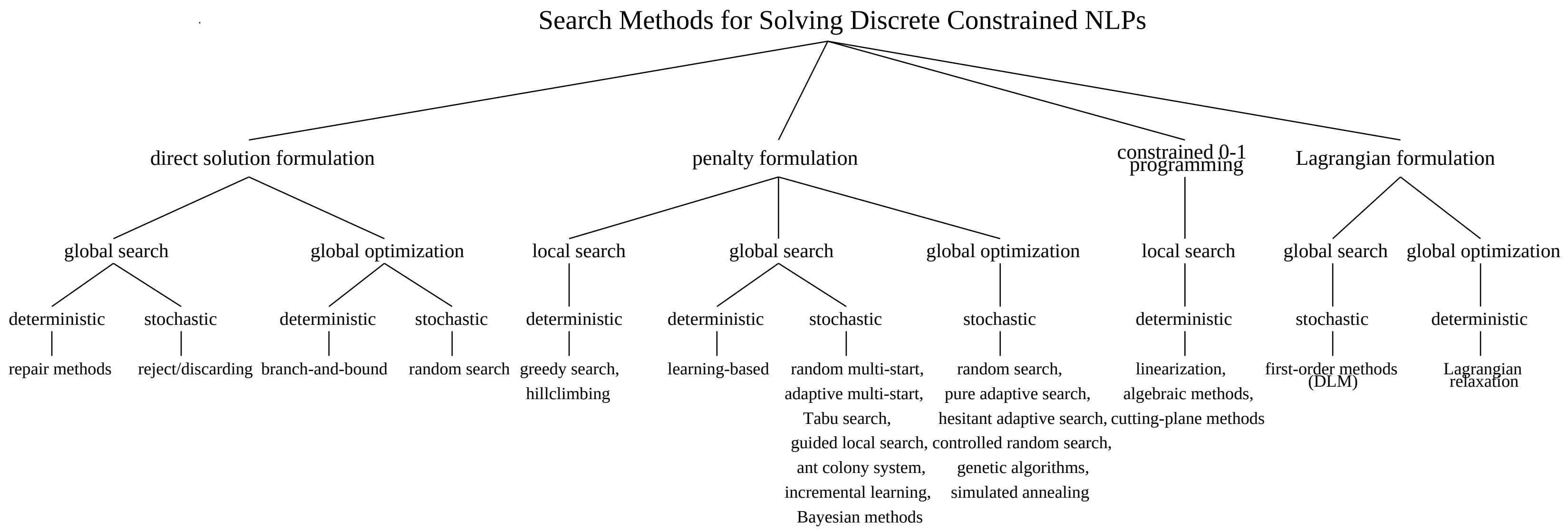

Search Methods for Solving Discrete Constrained NLPs
direct solution formulation
penalty formulation
constrained 0-1 programming
Lagrangian formulation
global search
global optimization
local search
global search
global optimization
local search
global search
global optimization
deterministic
stochastic
deterministic
stochastic
deterministic
deterministic
stochastic
stochastic
deterministic
stochastic
deterministic
repair methods
reject/discarding
branch-and-bound
random search
greedy search, hillclimbing
learning-based
random multi-start, adaptive multi-start, Tabu search, guided local search, ant colony system, incremental learning, Bayesian methods
random search, pure adaptive search, hesitant adaptive search, controlled random search, genetic algorithms, simulated annealing
linearization, algebraic methods, cutting-plane methods
first-order methods (DLM)
Lagrangian relaxation

Figure 2.3: Classification of search methods for solving discrete constrained NLPs.

These methods, although popular, generally lead to CLM_{dn} and do not guarantee constraint satisfaction if penalties were not chosen properly.

Global Search. Penalty-based methods first transform problem (1.1) into either a single unconstrained problem (2.1) or a sequence of unconstrained problems (2.2) and then solve them using unconstrained global-search strategies. Stochastic global-search strategies include random multi-start [171, 169, 96, 177], adaptive multi-start [34], learning-based approach [38, 39], tabu search [82, 25], guided local search (GLS) [195], ant colony system [60, 59], incremental learning [20, 36, 21, 22], and Bayesian methods [134, 196, 135]. Deterministic global-search methods consist of learning-based approaches [38, 39]. Section 2.1.2 gives more details on each of the above methods.

Global Optimization. Penalty-based global-optimization approaches solve unconstrained problems (2.1) or (2.2) using pure random search (PRS), pure adaptive search (PAS) [148], hesitant adaptive search (HAS) [40], controlled random search (CRS) [156, 4, 7], genetic algorithm (GA) and its variants [85, 71, 127, 167, 140, 147, 94, 14, 15, 70, 181], or simulated annealing (SA) [**?**, 43, 190, 152, 90, 124, 1, 63] and its improved implementations [221, 155, 120, 162, 223, 56, 5, 6]. See Section 2.1.2 for more details.

2.2.3 Nonlinear Constrained 0-1 Programming

This approach rewrites a discrete constrained NLP as a constrained 0-1 NLP before solving it. There are three local-search methods based on nonlinear 0-1 integer programming [92]. First, a nonlinear problem can be linearized by replacing each distinct product of variables by a new 0-1 variable and by adding some new constraints [83, 84]. This method only works for simple nonlinear problems and cannot be applied to solve general nonlinear problems. Second, algebraic methods [165] express the objective function as a polynomial function of the variables and their complements. These methods can only solve problems whose constraints can all be removed. Last, cutting-plane methods [86, 87] further reduce a constrained nonlinear 0-1 problem into a generalized covering problem. This approach is very limited because many nonlinear 0-1 problems cannot be transformed this way. The major problem with these methods is that they can only be applied to linear or simple nonlinear problems and, therefore, are very limited in solving general discrete constrained NLPs.

2.2.4 Lagrangian Formulations for Discrete Constrained NLPs

Lagrangian Relaxation. Lagrangian relaxation [74, 26] is a class of algorithms for solving *linear* programming (LP) problem based on Lagrangian Duality theory [184]. It reformulates a *linear* integer optimization problem

$$\begin{aligned} z = \text{minimize}_x \quad & Cx \\ \text{subject to} \quad & Gx \le b \quad \text{where } x \text{ is an integer vector of variables} \\ & x \ge 0 \qquad \text{and } C \text{ and } G \text{ are constant matrices} \end{aligned} \tag{2.9}$$

into a relaxed problem

$$\begin{aligned} L(\lambda) = \text{minimize}_x \quad & (Cx + \lambda(b - Gx)) \\ \text{subject to} \quad & x \ge 0. \end{aligned} \tag{2.10}$$

that can be solved efficiently for any given vector λ. Lagrangian relaxation aims to find an optimal primal solution given an optimal dual solution, or vice versa. Although some work [16] addresses nonlinear NLPs, the approach does not work for general nonlinear functions.

Theory of Discrete Constrained Optimization Using Lagrange Multipliers. Here we briefly overview a new theory of discrete constrained optimization using Lagrange multipliers [206, 216] developed in our research group. In contrast to Lagrangian methods that work only for continuous constrained NLPs [31, 125], this new theory was derived for discrete constrained NLPs and can be extended to solve both continuous and mixed-integer NLPs. More importantly, its first-order necessary and sufficient conditions for CLM_{dn} provide a strong theoretic foundation for developing global optimization methods for solving constrained NLPs.

We first consider a special case of (1.1) with only equality constraints and then discuss ways to handle inequality constraints. A general discrete equality-constrained NLP is:

$$\begin{array}{lll} \text{minimize} & f(x) & x = (x_1, x_2, \ldots, x_n) \text{ is a vector} \\ \text{subject to} & h(x) = 0 & \text{of discrete variables.} \end{array} \tag{2.11}$$

Similar to that in the continuous case [125], the *generalized discrete augmented Lagrangian function* [206] of (2.11) is defined as

$$L_d(x, \lambda) = f(x) + \lambda^T H(h(x)) + \frac{1}{2}||h(x)||^2, \tag{2.12}$$

where $\lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_m\}$ is a set of Lagrange multipliers, H is a continuous transformation function that satisfies $H(x) = 0 \Leftrightarrow x = 0$, and $||h(x)||^2 = \sum_{i=1}^{m} h_i^2(x)$.

A Lagrangian search treats original variable x and Lagrange multiplier λ equally in $L_d(x, \lambda)$, and defines its search space to be $S = X \times \Lambda$ with both x and λ being variables. Hence, any point in S can be represented by a vector (x, λ). This is quite different from other penalty formulations whose search space is X.

We cannot use L_d to derive first-order necessary conditions similar to those in continuous space [125] because there are no gradients or differentiation in discrete space. Without these concepts, none of the calculus in continuous space is applicable in discrete space.

An understanding of gradients in continuous space shows that they define directions in a small neighborhood in which function values decrease. To this end, we define in discrete space a *direction of maximum potential drop* (*DMPD*) for $L_d(x, \lambda)$ at point x for fixed λ as a vector [1] that points from x to a neighbor of $x \in \mathcal{N}_{dn}(x)$ with the minimum L_d:

$$\Delta_x L_d(x, \lambda) = \vec{\nu}_x = y \ominus x = (y_1 - x_1, \ldots, y_n - x_n) \tag{2.13}$$

$$\text{where} \qquad y \in \mathcal{N}_{dn}(x) \cup \{x\} \text{ and } L_d(y, \lambda) = \min_{\substack{x' \in \mathcal{N}_{dn}(x) \\ \cup \{x\}}} L_d(x', \lambda).$$

Here, $\ominus$ is the vector-subtraction operator for changing x in discrete space to one of its "user-defined" neighborhood points $\mathcal{N}_{dn}(x)$. Intuitively, $\vec{\nu}_x$ is a vector pointing from x to y, the point with the minimum L_d value among all neighboring points of x, including x itself. If x itself has the minimum L_d, then $\vec{\nu}_x = \vec{0}$. It is important to emphasize that, with this definition of discrete descent directions, DMPDs cannot be added and subtracted in discrete space [216]. Consequently all the results derived in continuous space are not applicable here. Based on DMPD, we define the concept of discrete saddle points [175, 216] in discrete space similar to those in continuous space [31, 125].

Definition 2.4 Point (x^*, λ^*) is a *discrete saddle point* iff

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*), \tag{2.14}$$

for all $x \in \mathcal{N}_{dn}(x^*)$ and all possible λ.

Note that the first inequality in (2.14) holds only when all the constraints are satisfied, which implies that it must be true for all λ. Further, (x^*, λ') is also a saddle point if $\lambda' \geq \lambda^*$, where $\lambda' \geq \lambda^*$ means that every component of λ' is no less than the corresponding component of λ^*. Both saddle points have the same x^* but different λ's.

The concept of saddle points is of great importance to discrete NLPs because, starting from saddle points, we can derive first-order necessary and sufficient conditions for discrete

[1] To simplify our symbols, we represent points in x space without the explicit vector notation.

constrained NLPs. These conditions are stronger than their continuous counterparts since they are necessary and sufficient (rather than necessary alone). Although these conditions look similar to those for continuous problems, they were derived from the concept of saddle points rather than from that of regular points [31, 125].

Theorem 2.2 *First-order necessary and sufficient conditions for discrete CLM_{dn}* [206, 216]. In discrete space, if $H(x)$ in (2.12) is a continuous function satisfying $H(x) = 0 \Leftrightarrow x = 0$ and is non-negative (or non-positive), then point x^* is a CLM_{dn} if and only if there exists λ^* such that

- (x^*, λ^*) is a saddle point, or
- (x^*, λ^*) satisfies the following first-order conditions:

$$\Delta_x L_d(x, \lambda) = 0 \quad \text{and} \quad h(x) \ = \ 0, \tag{2.15}$$

  where Δ_x is the DMPD operator defined in (2.13) for discrete space.

Let **A** be the set of all CLM_{dn}, **B** be the set of all solutions that satisfy the discrete-space first-order necessary and sufficient conditions (2.15), and **C** be the set of all discrete saddle points satisfying (2.14). The meaning of Theorem 2.2 is shown in Figure 2.4.

Theorem 2.2 is very important in the following aspects. First, it is sufficient to search for saddle points in order to find CLM_{dn}. Second, global optimization, aiming to find a CGM_{dn}, amounts to finding saddle points with the minimum objective value. In contrast, continuous-space global optimization methods based on continuous first-order necessary and second-order sufficient conditions are not guaranteed to find any CGM_{cn}, because these CGM_{cn} may be outside the set of points that satisfy the first-order and second-order conditions. This fact has been illustrated in Figure 2.2.

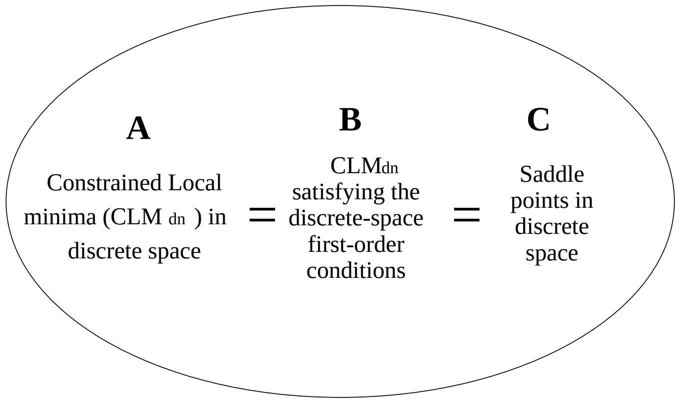

A
Constrained Local minima (CLM dn) in discrete space
=
B
CLMdn satisfying the discrete-space first-order conditions
=
C
Saddle points in discrete space

Figure 2.4: Relationship among solution sets of nonlinear discrete constrained NLPs.

Handling Inequality Constraints. The results discussed so far apply only to discrete NLPs with equality constraints (2.11). To handle (1.1) with inequality constraints, we need to transform them into equivalent equality constraints. Here we choose to transform inequality constraint $g_j(x) \leq 0$ into equality constraint by using a maximum function, $\max(0, g_j(x)) = 0$. Obviously, the new constraint is satisfied iff $g_j(x) \leq 0$. Hence, the discrete augmented Lagrangian function for (1.1) is

$$\begin{aligned} L_d(x, \lambda, \mu) = \quad & f(x) + \lambda^T H(h(x)) + \frac{1}{2}||h(x)||^2 \\ & + \sum_{i=1}^{k} \mu_i H(\max(0, g_i(x))) + \frac{1}{2}\sum_{i=1}^{k} \max{}^2(0, g_i(x)) \end{aligned} \tag{2.16}$$

where $\lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_m\}$ and $\mu = \{\mu_1, \mu_2, \cdots, \mu_k\}$ are, respectively, Lagrange multipliers for the equality and inequality constraints.

Discrete Lagrangian Methods (DLM) have been developed in [206, 175, 216] based on the first-order necessary and sufficient conditions (2.15). Given a starting point (x^0, λ^0), DLM iterates the following steps:

$$x^{k+1} = x^k \oplus \Delta_x L_d(x^k, \lambda^k) \tag{2.17}$$

$$\lambda^{k+1} = \lambda^k + c_1 \cdot H\left(h(x^k)\right), \tag{2.18}$$

where $\oplus$ is the vector-addition operator, $x \oplus y = (x_1 + y_1, x_2 + y_2, \ldots x_n + y_n)$, and c_1 is a constant controlling the speed of changing the Lagrange multipliers.

DLM is a local search method in discrete Lagrangian space, and may get stuck at CLM_{dn} or infeasible points. To escape from these traps, many global search strategies have been proposed [219, 218, 216], such as global perturbation, multi-start, updating Lagrange multipliers, and adding distance-related penalties. None of these global search strategies, however, can guarantee the reachability of global solutions, let alone asymptotic convergence.

In addition, both GAs and SA cannot be used to minimize Lagrangian function $L_d(x, \lambda)$, because $L_d(x, \lambda)$ is not minimum at CGM_{dn}, and because minimization of the Lagrangian function cannot guarantee satisfaction of all the constraints.

Extensions to continuous and mixed-integer NLPs. Theorem 2.2 has been extended to apply to continuous and mixed-integer NLPs [217] if all continuous variables are represented in floating-point types (single or double precision in digital computers, for example). Such a representation can be viewed as a kind of discretization to the continuous variables. Here we summarize the main results in [217].

Let c^* be the CGM_{cn} to $\mathcal{P}_{org}$, the original continuous constrained NLP, and d^* be the CGM_{dn} to $\mathcal{P}_{sub}$, the corresponding NLP with its continuous variables represented by floating-point numbers, whose interval size is $s_{grid,j}$. Let ℓ_f, ℓ_{h_i} be the *Lipschitz constants* for objective function $f(x)$ and constraint function $h_i(x)$, such that for any two points $x, y \in X$, $|f(x) - f(y)| \leq \ell_f \cdot \|x - y\|$, $|h_i(x) - h_i(y)| \leq \ell_{h_i} \cdot \|x - y\|$, $i = 1, \ldots, m$. We also define *minimum Lipschitz constants* ℓ_f^{min}, $\ell_{h_i}^{min}$, $i = 1, 2, \ldots, m$ as the minima of all possible Lipschitz constants ℓ_f and ℓ_{h_i}, $i = 1, 2, \ldots, m$, respectively. The following theorem shows the worst-case bound between $f(c^*)$ and $f(d^*)$.

Theorem 2.3 *Worst-case error on objective function due to floating-point representation of continuous variables* [217]. Assume the following for $\mathcal{P}_{org}$ and $\mathcal{P}_{sub}$.

1. ℓ_f^{min}, and $\ell_{h_i}^{min}, i = 1, \ldots, m$, are finite.

2. Constraint $h_i(x), i = 1, 2, \ldots, m$, is considered to be satisfied if $|h_i(x)| \leq \Phi$, where Φ is a pre-specified maximum violation tolerance.

3. The interval sizes for all dimensions satisfy

$$G = \frac{1}{2}\sqrt{\sum_{j=1}^{n} s_{grid,j}^2} \leq \frac{\Phi}{max_{i=1}^{m} \ell_{h_i}^{min}}. \tag{2.19}$$

Then the error in objective function f due to a floating-point representation of d^* is:

$$f(d^*) - f(c^*) \leq \ell_f^{min} \cdot G. \tag{2.20}$$

Theorem 2.3 reveals that solving continuous or mixed-integer constrained NLPs $\mathcal{P}_{org}$ by digital computers is equivalent to solving discrete constrained NLP $\mathcal{P}_{sub}$ mapped on a finite, discrete space represented by floating-point numbers. Therefore, continuous and mixed-integer NLPs can be solved in a similar way as discrete NLPs if their continuous variables are first discretized to the precision of computers.

Table 2.2 summarizes existing methods for solving discrete constrained NLPs based on the four criteria that we use to evaluate search methods.

Table 2.2: Summary of search methods for solving discrete constrained NLPs based on the four criteria of achieved optimality, convergence property for stochastic methods with global optimality, ability to handle nonlinear functions, and requirement on function continuity and differentiability.

Problem Formulations	Search Strategies	Search Methods	Achieved Optimality(c_1)	Convergence Property(c_2)	Ability to Handle Nonlinear F's(c_3)	Require Cont'y & Diff'y(c_4)
Direct Solutions	Global Search	repair methods	local	none	no	no
		reject/discarding	local	none	yes	no
	Global Optimization	branch-and-bound	global	none	no	no
		random search	global	reachability	yes	no
Penlaty Formulations	Local Search	greedy/hillclimbing	local	none	yes	no
	Global Search	learning-based methods	local	none	yes	no
		random multi-start	local	none	yes	no
		adaptive multi-start	local	none	yes	no
		Tabu search	local	none	yes	no
		guided local search	local	none	yes	no
		ant colony system	local	none	yes	no
		incremental learning	local	none	yes	no
		Bayesian methods	local	none	yes	no
	Global Optimization	random search	global	reachability	yes	no
		pure adaptive search	global	reachability	yes	no
		hesitant adaptive search	global	reachability	yes	no
		controlled random search	global	reachability	yes	no
		genetic algorithms	global	reachability	yes	no
		simulated annealing	global	asymptotic	yes	no
Constrained 0-1 Programming	Local Search	linearization	local	none	no	no
		algebraic methods	local	none	no	no
		cutting-plane methods	local	none	no	no
Lagrangian Formulations	Global Search	DLM	local	none	yes	no
	Global Optimization	Lagrangian relaxation	global	none	no	no

2.3 Search Methods for Constrained MINLPs

In this section, we survey previous work for solving constrained MINLPs, where some variables take discrete values while others take continuous values. In such a situation, derivatives are not available, rendering many gradient or Hessian-matrix based methods not applicable, even when the objective and constraint functions are differentiable. Figure 2.5 shows a classification of search methods for solving constrained MINLPs.

2.3.1 Direct Solutions for Constrained MINLPs

Global Search. Many global search methods surveyed for solving continuous constrained NLPs can be extended to solve constrained MINLPs. Reject/discarding [110, 14, 160, 154] or repair methods [113, 142] iteratively sample a search space and either discard infeasible points or transform them into feasible points. Different sampling strategies may be used for continuous and discrete variables. Their major problems are their inability to cope with nonlinear constraints and their high computational costs.

2.3.2 Penalty Formulations for Constarined MINLPs

Local Search. The most popular way to solve MINLPs is to first transform them into unconstrained problems and then solve them by greedy search or hillclimbing [157]. This approach requires different sampling strategies to be used for continuous and discrete variables. Convergence to CLM_{mn} may not be guaranteed if one does not select or adapt penalties correctly.

Global Search. Penalty-based methods, surveyed in Section 2.1.2, first transform (1.1) into either a single unconstrained problem (2.1) or a sequence of unconstrained problems (2.2), and then solve them using unconstrained global-search strategies, such as random

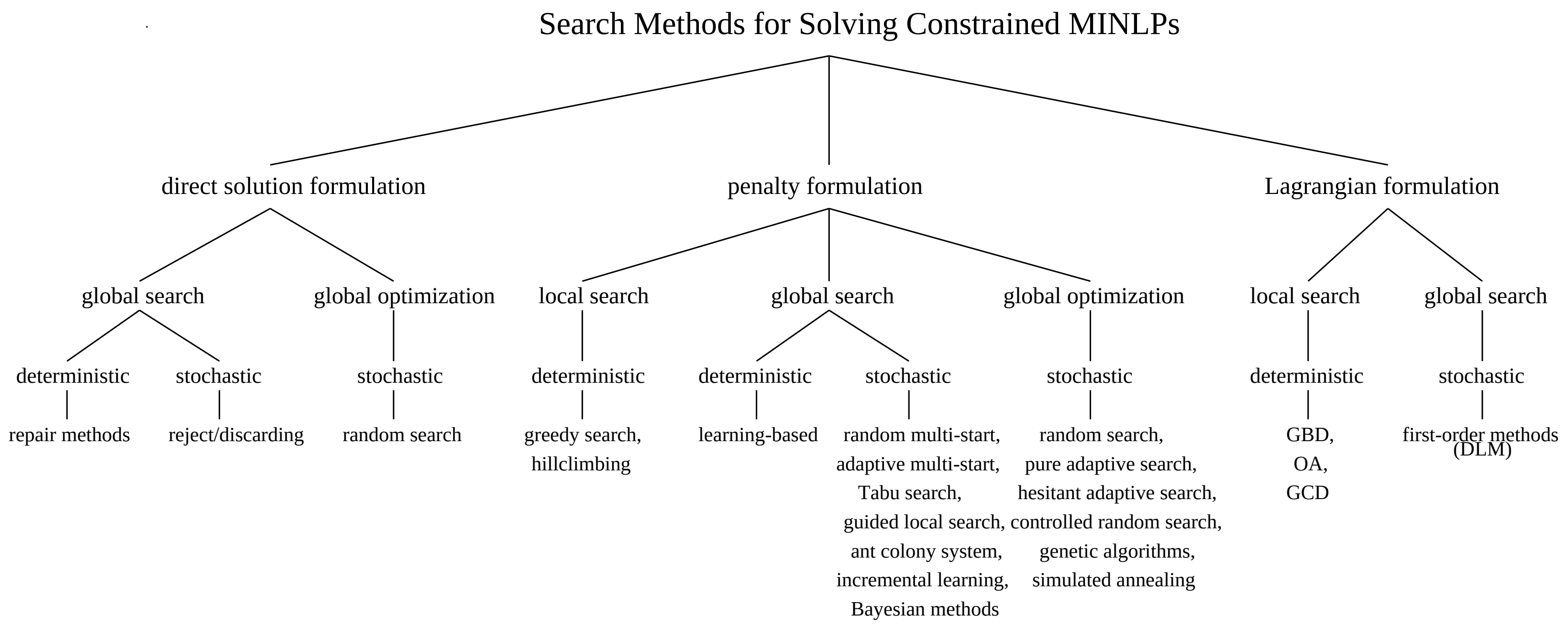

Search Methods for Solving Constrained MINLPs
direct solution formulation
penalty formulation
Lagrangian formulation
global search
global optimization
local search
global search
global optimization
local search
global search
deterministic
stochastic
stochastic
deterministic
deterministic
stochastic
stochastic
deterministic
stochastic
repair methods
reject/discarding
random search
greedy search, hillclimbing
learning-based
random multi-start, adaptive multi-start, Tabu search, guided local search, ant colony system, incremental learning, Bayesian methods
random search, pure adaptive search, hesitant adaptive search, controlled random search, genetic algorithms, simulated annealing
GBD, OA, GCD
first-order methods (DLM)

Figure 2.5: Classification of search methods for solving constrained MINLPs.

multi-start [171, 169, 96, 177], adaptive multi-start [34], learning-based approach [38, 39], tabu search [82, 25], guided local search (GLS) [195], ant colony system [60, 59], incremental learning [20, 36, 21, 22], Bayesian methods [134, 196, 135], as well as learning-based approaches [38, 39].

Global Optimization. Existing global-optimization approaches for solving constrained MINLPs are based on penalty formulations, which transform them into unconstrained problems using static-penalty (2.1) or dynamic-penalty (2.2) formulations. Then the unconstrained problems can be solved by pure random search (PRS), pure adaptive search (PAS) [148], hesitant adaptive search (HAS) [40], controlled random search (CRS) [156, 4, 7], genetic algorithms with non-uniform mutations [127], or simulated annealing (SA) [1, 63] and its variants [221, 155, 162, 223, 56, 5, 6]. See Section 2.1.2 for more details.

All these methods except SA can only guarantee the reachability of global minima by choosing proper sample distributions and neighborhoods. As mentioned before, combining penalty formulations and SA is not efficient. First, choosing suitable penalties prove to be difficult. Second, rugged search terrains and deep local minima, caused by large penalties, prevent SA from traversing large search spaces and trap it in local minima, especially at low temperature. Such traps seriously degrade the performance of SA.

2.3.3 Lagrangian Formulations for Constrained MINLPs

Local Search Using Function Convexity. This approach generally formulates a constrained MINLP in a Lagrangian formulation and then decomposes it into subproblems in such a way that, after fixing a subset of the variables, the resulting subproblem is convex and can be solved easily. There are three methods to implement this approach.

Generalized Benders decomposition (GBD) [67, 77, 29] computes at each iteration an upper bound on the solution sought by solving a primal problem and a lower bound on a master problem. The primal problem corresponds to the original problem with fixed discrete variables, while the master problem is derived through nonlinear duality theory. The sequence of upper (resp. lower) bounds have been shown to be non-increasing (resp. nondecreasing) and converge in a finite number of iterations. Its major disadvantage is that it is only applicable to a class of MINLPs with restrictions on their variable space, such as a nonempty and convex continuous subspace with convex objective and constraint functions.

Outer approximation (OA) [62, 61] solves constrained MINLPs by a sequence of approximations where each approximated subproblem contains the original feasible region. OA is similar to GBD except that the master problem is formulated based on primal information and outer linearization. This method requires the continuous subspace to be a nonempty, compact, and convex set, and the objective and constraint functions to be convex.

Generalized cross decomposition (GCD) [67, 99, 100, 166] iteratively alternates between two phases: phase 1 solving the primal and dual subproblems and phase 2 solving the master problem. Similar to OA and GBD, GCD also requires the objective and constraint functions to be proper convex functions.

The major problems of these convexity-based methods are that they are only applicable to specific classes of constrained MINLPs whose convex subproblems can be constructed and solved. Accordingly, their application to solve general constrained MINLPs is very restricted. Even in cases that convex subproblems can be constructed, they are prohibitively expensive because the number of convex subproblems may be too large to be enumerated.

Global Search. Discrete Lagrangian methods (DLM) [206, 175, 216], developed for solving discrete constrained NLPs based on the theory of discrete constrained optimization using

Table 2.3: Summary of search methods for solving constrained MINLPs based on the four criteria of achieved optimality, convergence property for stochastic methods with global optimality, ability to handle nonlinear functions, and requirement on function continuity and differentiability.

Problem Formulations	Search Strategies	Search Methods	Achieved Optimality(c_1)	Convergence Property(c_2)	Ability to Handle Nonlinear F's(c_3)	Require Cont'y & Diff'y(c_4)
Direct Solutions	Global Search	repair methods	local	none	no	no
		reject/discarding	local	none	yes	no
	Global Optimization	random search	global	reachability	yes	no
Penlaty Formulations	Local Search	greedy/hillclimbing	local	none	yes	no
	Global Search	learning-based methods	local	none	yes	no
		random multi-start	local	none	yes	no
		adaptive multi-start	local	none	yes	no
		Tabu search	local	none	yes	no
		guided local search	local	none	yes	no
		ant colony system	local	none	yes	no
		incremental learning	local	none	yes	no
		Bayesian methods	local	none	yes	no
	Global Optimization	random search	global	reachability	yes	no
		pure adaptive search	global	reachability	yes	no
		hesitant adaptive search	global	reachability	yes	no
		controlled random search	global	reachability	yes	no
		genetic algorithms	global	reachability	yes	no
		simulated annealing	global	asymptotic	yes	no
Lagrangian Formulations	Local Search	GBD	local	none	no	no
		OA	local	none	no	no
		GCD	local	none	no	no
	Global Search	DLM	local	none	yes	no

Lagrange multipliers, has been extended in [217] to solve constrained MINLPs. To escape from local minima, many global search strategies have been proposed [219, 218, 216], such as global perturbation, multi-start, ways to update Lagrange multipliers, and additional distance-related penalties. Such strategies, however, do not guarantee that a CGM_{mn} will be found.

Table 2.3 summarizes existing methods for solving constrained MINLPs based on the four criteria described earlier to evaluate search methods.

2.4 Summary

In this chapter, we have surveyed existing work for solving continuous, discrete, and mixed-integer constrained NLPs. We have organized these methods based on the hierarchy of problem formulations they use, search strategies (local search, global search, and global optimization), and their optimality and requirements.

To achieve the goal of this thesis and to have asymptotic convergence in solving *constrained* NLPs, we propose to transform (1.1) into an unconstrained NLP using penalty formulations and solve it using simulated annealing (SA), an unconstrained global optimization algorithm that achieves asymptotic convergence in solving *unconstrained* NLPs.

For static-penalty formulation (2.1), it proves to be difficult to choose suitable penalty γ. If the penalty is too large, SA tends to find feasible solutions rather than optimal solutions. If the penalty is too small, unconstrained local or global minima may not be a CLM or CGM to (1.1), and may even not be a feasible point. For dynamic-penalty formulation (2.2), unconstrained problem (2.2) at every stage of $\lambda(k)$ has to be solved optimally [31, 125] in order to have asymptotic convergence. Hence, a sequence of unconstrained global optimization is required. This is difficult to achieve in practice, given only a finite amount of time in each stage. If the result in one stage is not a global minimum, then the process cannot be guaranteed to find a CGM. Note that SA cannot be applied to search in Lagrangian space, because minimizing (2.12) does not guarantee constraint satisfaction.

In the next chapter, we develop a stochastic global-optimization algorithm, called constrained simulated annealing (CSA), that achieves asymptotic convergence for solving *constrained* NLPs. In contrast to using SA in penalty formulations, CSA searches in a Lagrangian space, carries out a single global optimization process, and does not depend on initial values of its penalty.

Chapter 3

Discrete Constrained Optimization: Constrained SA

In this chapter, we develop *constrained simulated annealing* (CSA), a global optimization algorithm that converges to CGM_{dn} with probability one, for solving discrete constrained NLPs. The algorithm is based on the necessary and sufficient conditions for CLM_{dn} (Theorem 2.2) in the theory of discrete constrained optimization using Lagrange multipliers (see Section 2.2.4). To find CGM_{dn}, *i.e.*, discrete saddle points with the minimum objective value, we model CSA by a finite inhomogeneous Markov chain that carries out in an annealing fashion both probabilistic descents of the discrete augmented Lagrangian function in the variable space of x and probabilistic ascents in the Lagrange-multiplier space of λ. Then we prove the asymptotic convergence of the algorithm to a CGM_{dn} with probability one. By achieving asymptotic convergence, CSA is one of the major developments in nonlinear *constrained* global optimization today and complements simulated annealing (SA) in nonlinear *unconstrained* global optimization.

It is following, we first outline the differences between traditional SA [1, 118] and CSA developed in this thesis.

- *Targeted problems:* SA was developed for solving *unconstrained* NLPs, whereas CSA is for solving *constrained* NLPs. In addition to minimizing objective function $f(x)$, CSA has to find solutions that satisfy a set of nonlinear constraints $h(x) = 0$ and $g(x) \leq 0$. Therefore, SA can be viewed as a special case of CSA in the absence of constraints. Here we assume that $f(x)$ is lower bounded, that $g(x)$ and $h(x)$ need not be bounded, and that variable space X is bounded.

- *Search space:* SA searches in variable space X, whereas CSA searches in a joint space of variable x and Lagrange multiplier λ. SA looks for solution points with the minimum objective value, whereas CSA looks for saddle points in its search space.

- *Search procedure:* SA does probabilistic descents in variable x space with acceptance probabilities governed by a temperature, while CSA does both probabilistic ascents in the λ subspace and probabilistic descents in the x subspace. Therefore, SA is explicit in minimizing objective function $f(x)$, whereas CSA is implicit in minimizing a virtual energy according to the GSA framework [186, 187] instead of $L(x, \lambda)$, since minimizing $L(x, \lambda)$ cannot satisfy all the constraints.

- *Converged solutions:* SA has asymptotic convergence to an *unconstrained* global minimum [1], while CSA has asymptotic convergence to a CGM_{dn} [202].

3.1 Theory of Simulated Annealing

In this section, we briefly overview simulated annealing (SA) and its theory [1, 90, **?**] for solving discrete *unconstrained* NLPs or *combinatorial optimization problems.* A general unconstrained NLP is defined as

$$minimize_i \quad f(i) \quad \text{for} \quad i \in S \tag{3.1}$$

```
1.  procedure SA
2.      set starting point $i = i_0$;
3.      set starting temperature $T = T^0$ and cooling rate $0 < \alpha < 1$;
4.      set $N_T$ (number of trials per temperature);
5.      while stopping condition is not satisfied do
6.          for $k \leftarrow 1$ to $N_T$ do
7.              generate trial point $i'$ from $S_i$ using $q(i, i')$;
8.              accept $i'$ with probability $A_T(i, i')$
9.          end_for
10.         reduce temperature by $T \longleftarrow \alpha \times T$;
11.     end_while
12. end_procedure
```

Figure 3.1: Simulated annealing (SA) algorithm.

where $f(i)$ is an objective function to be minimized, and S is the *solution space* denoting the *finite* set of all possible solutions.

A solution i_{opt} is called a *global minimum* if it satisfies $f(i_{opt}) \leq f(i)$, for all $i \in S$. Let S_{opt} be the set of all the global minima and $f_{opt} = f(i_{opt})$ be their objective value. *Neighborhood* S_i of solution i is the set of discrete points j satisfying $j \in S_i \Leftrightarrow i \in S_j$.

Figure 3.1 shows the procedure of SA for solving unconstrained problem (3.1). $q(i, i')$, the *generation probability*, is defined as $q(i, i') = 1/|S_i|$ for all $i' \in S_i$, and $A_T(i, i')$, the *acceptance probability* of accepting solution point i', is defined by:

$$A_T(i, i') = exp\left(-\frac{(f(i') - f(i))^+}{T}\right), \tag{3.2}$$

where $a^+ = a$ if $a > 0$, and $a^+ = 0$ otherwise.

Accordingly, SA works as follows. Given current solution i, SA first generates trial point i'. If $f(i') < f(i)$, i' is accepted as a starting point for the next iteration; otherwise, solution i' is accepted with probability $exp\left(-\frac{f(i')-f(i)}{T}\right)$. The worse the i' is, the smaller is the probability that i' is accepted for the next iteration. The above procedure is repeated N_T times until temperature T is reduced. Theoretically, if T is reduced sufficiently slowly in logarithmic scale, then SA will converge asymptotically to an optimal solution $i_{opt} \in S_{opt}$ [1, 90, **?**] (see below). In practice, a geometric cooling schedule, $T \leftarrow \alpha T$, is generally utilized to have SA settle down at some solution i^* in a finite amount of time.

SA can be modeled by an inhomogeneous Markov chain that consists of a sequence of homogeneous Markov chains of finite length, each at a specific temperature in a given temperature schedule. According to generation probability $q(i, i')$ and acceptance probability $A_T(i, i')$, the *one-step transition probability* of the Markov chain is:

$$P_T(i, i') = \begin{cases} q(i, i')A_T(i, i') & \text{if } i' \in S_i \\ 1 - \sum_{j \in S_i, j \neq i} P_T(i, j) & \text{if } i' = i \\ 0 & \text{otherwise,} \end{cases} \tag{3.3}$$

and the corresponding *transition matrix* is $P_T = [P_T(i, i')]$.

It is assumed that, by choosing neighborhood S_i properly, the Markov chain is *irreducible* [1, 90], meaning that for each pair of solutions i and j, there is a positive probability of reaching j from i in a finite number of steps.

Consider the sequence of temperatures $\{T_k, k = 0, 1, 2, \cdots\}$, where $T_k > T_{k+1}$ and $\lim_{k\to\infty} T_k = 0$, and choose N_T to be the maximum of the minimum number of steps required to reach an i_{opt} from every $j \in S$. Since the Markov is irreducible and search space S is finite, such N_T always exists. The asymptotic convergence theorem of SA is stated as follows [1, 90].

Theorem 3.1 The Markov chain modeling SA converges asymptotically to a global minimum of S_{opt} [1, 90] if the sequence of temperatures satisfies:

$$T_k \geq \frac{N_T \triangle}{log_e(k+1)}, \tag{3.4}$$

where $\triangle = \max_{i,j \in S}\{f(j) - f(i) | j \in S_i\}$.

The proof of this theorem is based on so-called *local balance equation* [1], meaning that:

$$\pi_T(i) P_T(i, i') = \pi_T(i') P_T(i', i), \tag{3.5}$$

where $\pi_T(i)$ is the stationary probability of state i at temperature T.

Although SA works well for solving unconstrained NLPs, it cannot be used directly to solve *constrained* NLPs that have a set of constraints to be satisfied, in addition to minimizing the objective. The widely used strategy is to transform constrained NLP (1.1) into an unconstrained NLP using penalty formulations as described in Section 2.1.2. For static-penalty formulation (2.1), it is very difficult to choose suitable penalty γ: if the penalty is too large, SA tends to find feasible solutions rather than optimal solutions. For dynamic-penalty formulation (2.2), unconstrained problem (2.2) at every stage of $\lambda(k)$ has to be solved optimally [31, 125] in order to have asymptotic convergence. However, this requirement is difficult to achieve in practice, given only a finite amount of time in each stage. If the result in one stage is not a global minimum, then the process cannot be guaranteed to find a CGM_{dn}. Therefore, applying SA to a dynamic-penalty formulation does not always lead to asymptotic convergence. Besides, SA cannot be used to search in a Lagrangian space, because minimizing Lagrangian function (2.12) does not guarantee constraint satisfaction.

In the next section, we develop a novel stochastic global optimization algorithm, called constrained simulated annealing (CSA), that achieves asymptotic convergence for solving discrete constrained NLPs. The method searchs in a discrete Lagrangian space and aims to find a discrete saddle point with the minimum objective value. CSA carries out a single global optimization process and does not need to guess a penalty for each constraint.

3.2 Constrained Simulated Annealing Algorithm

The goal of constrained simulated annealing (CSA) [202] is to find a saddle point with the minimum objective value, *i.e.*, a CGM_{dn} based on Theorem 2.2, assuming H is the absolute function. Therefore, CSA works in a discrete Lagrangian space, the joint space $S = X \times \Lambda$ of both x and λ variables, according to theory of discrete Lagrange multiplier in Section 2.2.4. Without loss of generality, we discuss CSA for solving discrete equality-constrained problem (2.11) with the following augmented Lagrangian function:

$$L(x, \lambda) = f(x) + \lambda^T |h(x)| + \frac{1}{2}||h(x)||^2, \tag{3.6}$$

since inequality constraints can always be transformed into equality constraints and incorporated into Lagrangian function (2.16).

Figure 3.2 describes the procedure of CSA. CSA begins from starting point $\mathbf{x} = (x, \lambda)$ (Line 2), where x can be either user-provided or randomly generated and $\lambda = 0$. Initial temperature T^0 is set (Line 3) to be so large that allows almost all trial points $\mathbf{x}'$ to be accepted. Line 4 sets N_T, the number of iterations at each temperature. CSA stops (Line 5) when the current point $\mathbf{x}$ is not changed, *i.e.*, no other new point $\mathbf{x}'$ is accepted, for a couple of successive temperatures, or when the current temperature T is small enough (*e.g.* $T < 10^{-6}$).

Line 7 generates a random trial point $\mathbf{x}'$ in neighborhood $\mathcal{N}(\mathbf{x})$ of current point $\mathbf{x} = (x, \lambda)$ in search space $S = X \times \Lambda$ using generation probability $q(\mathbf{x}, \mathbf{x}')$, where $\mathcal{N}(\mathbf{x})$, $\mathcal{N}^1_{dn}(x)$,

```
1.  procedure CSA
2.      set starting point $\mathbf{x} = (x, \lambda)$;
3.      set starting temperature $T = T^0$ and cooling rate $0 < \alpha < 1$;
4.      set $N_T$ (number of trials per temperature);
5.      while stopping condition is not satisfied do
6.          for $k \leftarrow 1$ to $N_T$ do
7.              generate trial point $\mathbf{x}'$ from $\mathcal{N}(\mathbf{x})$ using $q(\mathbf{x}, \mathbf{x}')$;
8.              accept $\mathbf{x}'$ with probability $A_T(\mathbf{x}, \mathbf{x}')$
9.          end_for
10.         reduce temperature by $T \longleftarrow \alpha \times T$;
11.     end_while
12. end_procedure
```

Figure 3.2: CSA: the constrained simulated annealing algorithm.

neighborhood of x at $\mathbf{x}$, and $\mathcal{N}^2_{cn}(\lambda)$, neighborhood of λ at $\mathbf{x}$, are defined as follows:

$$\mathcal{N}(\mathbf{x}) = \{(x', \lambda) \in S \text{ where } x' \in \mathcal{N}^1_{dn}(x)\} \bigcup \{(x, \lambda') \in S \text{ where } \lambda' \in \mathcal{N}^2_{cn}(\lambda)\} \quad (3.7)$$

$$\begin{aligned} \mathcal{N}^1_{dn}(x) &= \{x' | \text{ neighboring points of x such that } x' \in \mathcal{N}^1_{dn}(x) \Longleftrightarrow x \in \mathcal{N}^1_{dn}(x')\} \\ \mathcal{N}^2_{cn}(\lambda) &= \{\mu \in \Lambda \mid \mu < \lambda \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\} \\ &\quad \bigcup \{\mu \in \Lambda \mid \mu > \lambda \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\}, \end{aligned} \quad (3.8)$$

where relation "<" on two vectors has been defined earlier. Neighborhood $\mathcal{N}^2_{cn}(\lambda)$ prevents λ_i from being changed when the corresponding constraint is satisfied, *i.e.*, $h_i(x) = 0$.

An example of $\mathcal{N}^2_{cn}(\lambda)$ is that μ differs from λ in one variable (*e.g.* $\mu_i \neq \lambda_i$, and $\mu_j = \lambda_j$ for $j \neq i$), and $\{\mu_i | \, i \neq j\}$ is a set of values, some of which are larger than λ_i and others are smaller. Hence, point $\mathbf{x} = (x, \lambda)$ has two kinds of neighbors: (x', λ) or (x, λ'). Trial point (x', λ) is a neighboring point to (x, λ) if x' is a neighboring point to x in variable space X, and

(x, λ') is a neighboring point to (x, λ) if λ' is a neighboring point to λ in Lagrange-multiplier space Λ and $h(x) \neq 0$.

Note that CSA does not try to satisfy the first-order necessary and sufficient conditions (2.15) of CLM_{dn}, because CSA is unable to compute *DMPD* defined by (2.13) when the number of neighboring points is large. Hence, if CSA were stopped in finite time, it would only stop at a feasible point that may not be a saddle point.

$q(\mathbf{x}, \mathbf{x}')$, the *generation probability* from $\mathbf{x}$ to $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$, satisfies:

$$q(\mathbf{x}, \mathbf{x}') > 0 \quad \text{and} \quad \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} q(\mathbf{x}, \mathbf{x}') = 1. \tag{3.9}$$

The choice of $q(\mathbf{x}, \mathbf{x}')$ is arbitrary as long as it satisfies (3.9). In our illustrative example 3.1 below, we use a uniform probability over $\mathcal{N}(\mathbf{x})$, independent of T,

$$q(\mathbf{x}, \mathbf{x}') = 1/|\mathcal{N}(\mathbf{x})|. \tag{3.10}$$

Experimentally, we have found better performance using a nonuniform probability over $\mathcal{N}(\mathbf{x})$ in which trial point (x', λ) is generated more frequently than trial point (x, λ'). For this reason, we use a nonuniform distribution in our experiments.

We accept $\mathbf{x}'$ (Line 8) according to acceptance probability $A_T(\mathbf{x}, \mathbf{x}')$, which consists of two components, depending on whether x or λ is changed in $\mathbf{x}'$:

$$A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} exp\left(- \frac{(L(\mathbf{x}') - L(\mathbf{x}))^+}{T}\right) & \text{if } \ \mathbf{x}' = (x', \lambda) \\ exp\left(- \frac{(L(\mathbf{x}) - L(\mathbf{x}'))^+}{T}\right) & \text{if } \ \mathbf{x}' = (x, \lambda'), \end{cases} \tag{3.11}$$

where $(a)^+ = a$ if $a > 0$, and $(a)^+ = 0$ otherwise for all $a \in R$.

$A_T(\mathbf{x}, \mathbf{x}')$ differs from (3.3) used in SA [1, 118] that only has the first part of (3.11). The goal of SA is to look for global minima in the x subspace. Hence, it only needs probabilistic descents in that space and does not need the λ subspace.

In contrast, our goal here is to look for saddle points in the joint space $X \times \Lambda$ of x and λ, which exist at local minima in the x subspace and at local maxima in the λ subspace. To this end, the first part of (3.11) carries out *probabilistic descents* of $L(x, \lambda)$ with respect to x for fixed λ. That is, when we generate a new point x' given fixed λ, we accept it with probability one when $\delta_x = L(x', \lambda) - L(x, \lambda)$ is negative; otherwise, we accept it with probability $e^{-\delta_x/T}$. This is performing exactly descents while allowing occasional ascents in the x subspace, as done in conventional SA [1, 118].

However, descents in the x subspace alone only lead to local/global minima of the Lagrangian function without satisfying the constraints. To this end, the second part of (3.11) carries out *probabilistic ascents* of $L(x, \lambda)$ with respect to λ for fixed x in order to increase the penalties of violated constraints and to force them into satisfaction. Hence, when we generate a new point λ' given fixed x, we accept it with probability one when $\delta_\lambda = L(x, \lambda') - L(x, \lambda)$ is positive; otherwise, we accept it with probability $e^{+\delta_\lambda/T}$. This is performing exactly ascents in the λ subspace while allowing occasional descents (and reducing the ruggedness of the terrains). Note that when a constraint is satisfied, the corresponding Lagrange multiplier will not be changed according to (3.8).

Although our algorithm only changes either the x or λ variable one at a time, CSA can be implemented with simultaneous changes of x and λ. In this case, we can decompose the aggregate change from (x, λ) to (x', λ') into two steps: (a) $(x, \lambda) \longrightarrow (x', \lambda)$ and (b) $(x', \lambda) \longrightarrow (x', \lambda')$, each of which fits into the CSA procedure.

Finally, Line 10 reduces T using the following geometric *cooling schedule* after looping N_T times at a given T:

$$T \longleftarrow \alpha \times T, \tag{3.12}$$

where α is a constant smaller than 1 (typically between 0.5 and 0.99). Theoretically, if T is reduced slow enough, then CSA will converge to a CGM_{dn} of (1.1) with probability one as T approaches 0 (see the next section).

Acceptance probability (3.11) enables any trial point to be accepted with high probabilities at high T, allowing the search to traverse a large space and overcome infeasible regions. As T is gradually reduced, the acceptance probability decreases, and at very low temperatures the algorithm behaves like a local search.

3.3 Asymptotic Convergence of CSA

In this section, we prove the asymptotic convergence of CSA to a CGM_{dn} by modeling the process as an inhomogeneous Markov chain, showing that the Markov chain is strongly ergodic, proving that the Markov chain minimizes an implicit virtual energy based on the framework of generalized SA (GSA) [186, 187], and showing that the virtual energy is at its minimum at any CGM_{dn}. We also illustrate the results by a simple example.

As discussed in Section 2.2.4, if (x^*, λ^*) is a discrete saddle point, then (x^*, λ') is also a saddle point for $\lambda' \geq \lambda^*$. This means that the exact value of λ' is not important as long as it is large enough to make (x^*, λ') a saddle point. Therefore, without loss of generality, we assume that Lagrange multiplier λ takes discrete values with the maximum value γ larger than or equal to λ^* for all CLM_{dn} x^*. Because variable subspace X is finite for discrete problems, γ always exists and is finite. Lagrange-multiplier subspace Λ is then the finite set of all possible combinations of discrete λ, and the search space of CSA $S = X \times \Lambda$ is finite.

3.3.1 Inhomogeneous Markov Chain

CSA can be modeled by an inhomogeneous Markov chain consisting of a sequence of homogeneous Markov chains of finite length, each at a specific temperature in a given temperature schedule. According to generation probability $q(\mathbf{x},\mathbf{x}')$ and acceptance probability $A_T(\mathbf{x},\mathbf{x}')$, the *one-step transition probability* of the Markov chain is similar to (3.3):

$$P_T(\mathbf{x},\mathbf{x}') = \begin{cases} q(\mathbf{x},\mathbf{x}')A_T(\mathbf{x},\mathbf{x}') & \text{if } \mathbf{x}' \in \mathcal{N}(\mathbf{x}) \\ 1 - \sum_{\mathbf{y}\in\mathcal{N}(\mathbf{x})} P_T(\mathbf{x},\mathbf{y}) & \text{if } \mathbf{x}' = \mathbf{x} \\ 0 & \text{otherwise,} \end{cases} \tag{3.13}$$

and the corresponding *transition matrix* is $P_T = [P_T(\mathbf{x},\mathbf{x}')]$.

Example 3.1 The following simple example illustrates the Markov chain for a problem that minimizes a quadratic objective with one quadratic constraint:

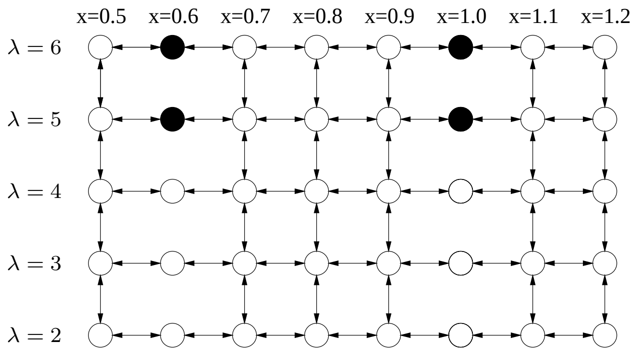
$$\begin{aligned} &\textit{minimize} && f(x) = -x^2 \\ &\textit{subject to} && h(x) = |(x-0.6)(x-1.0)| = 0, \end{aligned} \tag{3.14}$$

where $x \in X = \{0.5, 0.6, \cdots, 1.2\}$ and $\lambda \in \Lambda = \{2,3,4,5,6\}$ are both discrete, and γ, the maximum Lagrange multiplier, is 6. The state space is, therefore, $S = \{(x,\lambda)|\ x \in X, \lambda \in \Lambda\}$, and the total number of states is $|S| = 8 \times 5 = 40$.

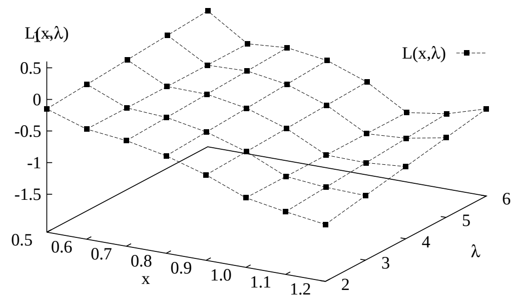
In the Markov chain, we define the neighborhoods of x and λ as follows:

$$\begin{aligned} \mathcal{N}^1_{dn}(x) &= \{x-1, x+1|\ 0.6 \le x \le 1.1\} \cup \{x+1|\ x = 0.5\} \cup \{x-1|\ x = 1.2\} \\ \mathcal{N}^2_{cn}(\lambda) &= \{\lambda-1, \lambda+1|\ 3 \le \lambda \le 5, x \ne 0.6, \text{ and } x \ne 1.0\} \cup \{\lambda-1|\ \lambda = 6, \\ &\quad x \ne 0.6, \text{ and } x \ne 1.0\} \cup \{\lambda+1|\ \lambda = 2, x \ne 0.6, \text{ and } x \ne 1.0\}. \end{aligned} \tag{3.15}$$

Given $\mathcal{N}^1_{dn}(x)$ and $\mathcal{N}^2_{cn}(\lambda)$, $\mathcal{N}(\mathbf{x})$ is defined as in (3.7).



a) State-space with non-zero transition probabilities



b) Lagrangian function value at each state

Figure 3.3: The Markov chain modeling the Lagrangian space of (3.14) and the corresponding Lagrangian-function values. The four saddle points are shaded in a).

Figure 3.3 shows the Markov chain constructed based on $\mathcal{N}(\mathbf{x})$. In the chain, node $\mathbf{x} = (x, \lambda)$ represents a point in S, and an arrow from $\mathbf{x}$ to $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$ (where $\mathbf{x}' = (x', \lambda)$ or (x, λ')) means that there is a one-step transition from $\mathbf{x}$ to $\mathbf{x}'$ with $P_T(\mathbf{x}, \mathbf{x}') > 0$. For $x = 0.6$ and $x = 1.0$, there is no transition among the λ's because the constraints are satisfied at these points according to (3.8).

Two saddle points at $(0.6, 5)$ and $(0.6, 6)$ in this Markov chain corresponds to CLM_{dn} $x = 0.6$, while the other two saddle points at $(1.0, 5)$ and $(1.0, 6)$ correspond to CLM_{dn} $x = 1.0$. Since $h(x)$ is non-negative, each saddle point has an associated CLM_{dn} according to Theorem 2.2. Hence, the solution space is the set of four saddle points or the set of two CLM_{dn}. CSA is designed to locate the saddle point corresponding to CGM_{dn} $x^* = 1.0$ and $\lambda = \gamma$ at $(1.0, 6)$. ■

Let $\mathbf{x}_{opt} = \{(x^*, \gamma) | \; x^* \in X_{opt}\}$, and N_L be the maximum of the minimum number of transitions required to reach $\mathbf{x}_{opt}$ from all $\mathbf{x} \in S$. By properly constructing $\mathcal{N}(\mathbf{x})$, we assume that P_T is irreducible, and that N_L can always be found. This property is illustrated in Figure 3.3 in which any two nodes can reach each other. To verify whether the Markov chain P_T is irreducible or not, one can calculate the $|S|$-step transition probability $P' = [P_T(\mathbf{x}, \mathbf{x}')]^{|S|}$ for any temperature T, where $|S|$ is the number of states in the Markov chain. If, for every $\mathbf{x}$ and $\mathbf{x}'$, $P'(\mathbf{x}, \mathbf{x}') > 0$, then the Markov chain P_T is irreducible. In fact, if $[P_T(\mathbf{x}, \mathbf{x}')]^p(\mathbf{x}, \mathbf{x}') > 0$ and $[P_T(\mathbf{x}, \mathbf{x}')]^{p-1}(\mathbf{x}, \mathbf{x}') = 0$, then p is the minimum number of steps to reach $\mathbf{x}'$ from $\mathbf{x}$.

Consider the sequence of temperatures $\{T_k, k = 0, 1, 2, \cdots\}$, where $T_k > T_{k+1}$ and $\lim_{k\to\infty} T_k = 0$, and set N_T, the number of trials per temperature, to be N_L. The following theorem proves the strong ergodicity of the Markov chain.

Theorem 3.2 The inhomogeneous Markov chain is *strongly ergodic* if the sequence of temperatures $\{T_k\}$ satisfy:

$$T_k \geq \frac{N_L \triangle_L}{log_e(k+1)}, \tag{3.16}$$

where $\triangle_L = \max_{\mathbf{x}}\{|L(\mathbf{x}') - L(\mathbf{x})|, \mathbf{x}' \in \mathcal{N}(\mathbf{x})\}$.

Proof. The proof of strong ergodicity follows the steps used to show the weak ergodicity of SA [1] and use the strong ergodicity conclusions [9, 10].

a) Let $\triangle_G = \min_{\mathbf{x}\in S, \mathbf{x}'\in\mathcal{N}(\mathbf{x})} q(\mathbf{x}, \mathbf{x}')$. For all $\mathbf{x} \in S$ and $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$, we have

$$P_{T_k}(\mathbf{x}, \mathbf{x}') = q(\mathbf{x}, \mathbf{x}') A_{T_k}(\mathbf{x}, \mathbf{x}') \geq \triangle_G \; e^{-\triangle_L/T_k}, \tag{3.17}$$

because $(L(\mathbf{x}') - L(\mathbf{x}))^+ \leq \triangle_L$ for $\mathbf{x}' = (x', \lambda)$ and $(L(\mathbf{x}) - L(\mathbf{x}'))^+ \leq \triangle_L$ for $\mathbf{x}' = (x, \lambda')$, according to the definition of $\triangle_L$.

b) Let $\hat{S}$ be the set of local maxima of $L(x, \lambda)$ over x given any λ. Then for every $\mathbf{x} = (x, \lambda) \in S - \hat{S}$, there always exists some $\mathbf{x}'' = (x'', \lambda) \in \mathcal{N}(\mathbf{x})$ such that $L(\mathbf{x}'') > L(\mathbf{x})$. Let $\delta = \min_{\mathbf{x} \in S - \hat{S}, \mathbf{x}'' \in \mathcal{N}(\mathbf{x})} \{L(\mathbf{x}'') - L(\mathbf{x})\} > 0$. We have

$$\begin{aligned} P_{T_k}(\mathbf{x}, \mathbf{x}) &= 1 - \sum_{\mathbf{y} \in \mathcal{N}(\mathbf{x})} q(\mathbf{x}, \mathbf{y}) A_{T_k}(\mathbf{x}, \mathbf{y}) \\ &\geq 1 - \left[\sum_{\mathbf{y} \in \mathcal{N}(\mathbf{x}), \mathbf{y} \neq \mathbf{x}''} q(\mathbf{x}, \mathbf{y}) \right] - q(\mathbf{x}, \mathbf{x}'') e^{-\delta/T_k} \\ &= q(\mathbf{x}, \mathbf{x}'')(1 - e^{-\delta/T_k}) \geq \triangle_G (1 - e^{-\delta/T_k}) \end{aligned}$$

Because T_k is a decreasing sequence, it is always possible to find $k_0 > 0$ such that for all $k \geq k_0$, $1 - e^{-\delta/T_k} \geq e^{-\triangle_G/T_k}$. Thus, for $\mathbf{x} \in S - \hat{S}$, we get:

$$P_{T_k}(\mathbf{x}, \mathbf{x}) \geq \triangle_G e^{-\triangle_L/T_k}. \tag{3.18}$$

c) Based on (3.17) and (3.18), for all $\mathbf{x}, \mathbf{y} \in S$ and $k \geq k_0$, the N_T-step transition probability from $\mathbf{x} = \mathbf{x}_0$ to $\mathbf{y} = \mathbf{x}_{N_T}$ satisfies the following:

$$P_{T_k}^{N_T}(\mathbf{x}, \mathbf{y}) \geq P_{T_k}(\mathbf{x}_0, \mathbf{x}_1) P_{T_k}(\mathbf{x}_1, \mathbf{x}_2) \cdots P_{T_k}(\mathbf{x}_{N_T - 1}, \mathbf{x}_{N_T}) \geq \left[\triangle_G e^{-\triangle_L/T_k} \right]^{N_T}.$$

Let $\tau_1(P)$ be the coefficient of ergodicity of transition matrix P. Then the lower bound of $1 - \tau_1(P_{T_k}^{N_T})$ is:

$$\begin{aligned} 1 - \tau_1(P_{T_k}^{N_T}) &= \min_{\mathbf{x}, \mathbf{x}' \in S} \sum_{\mathbf{y} \in S} \min\{P_{T_k}^{N_T}(\mathbf{x}, \mathbf{y}), P_{T_k}^{N_T}(\mathbf{x}', \mathbf{y})\} \\ &\geq \min_{\mathbf{x}, \mathbf{x}' \in S} \min_{\mathbf{y}' \in S} \{P_{T_k}^{N_T}(\mathbf{x}, \mathbf{y}'), P_{T_k}^{N_T}(\mathbf{x}', \mathbf{y}')\} \\ &\geq \left[\triangle_G e^{-\triangle_L/T_k} \right]^{N_T} = \triangle_G^{N_T} e^{-\triangle_L N_T/T_k}. \end{aligned}$$

Then using any temperature schedule that satisfies (3.16), the following holds:

$$\sum_{k=0}^{\infty}[1-\tau_1(P_{T_k}^{N_T})] \geq \sum_{k=k_0}^{\infty} \triangle_G^{N_T} e^{-\triangle_L N_T/T_k} \geq \triangle_G^{N_T} \sum_{k=k_0}^{\infty} \frac{1}{k+1} = \infty. \qquad (3.19)$$

Therefore, the Markov chain is weakly ergodic.

d) In addition, because transition probability $P_{T_k}(\mathbf{x}, \mathbf{x}')$ for all $\mathbf{x}, \mathbf{x}' \in S$ belongs to the exponential rationals in a closed class of asymptotically monotone functions (CAM) [9, 10], the Markov chain is strongly ergodic. ■

Strongly ergodicity implies that the Markov chain has a unique stationary distribution π_T, where $\pi_T(\mathbf{x})$ is the probability of hitting point $\mathbf{x}$ during the search of CSA. Hence, the Markov chain in Figure 3.3 is strongly ergodic if we set $N_L = 5 + 8 = 13$ and the cooling schedule as $T_k \geq \frac{N_L \triangle_L}{log_e(k+1)}$, where $\triangle_L = \max_{\mathbf{x}}\{|L(\mathbf{x}') - L(\mathbf{x})|, \mathbf{x}' \in \mathcal{N}(\mathbf{x})\} = 0.411$ in Example 3.1.

3.3.2 Asymptotic Convergence to Constrained Global Minima

In this section, we first discuss the framework of generalized simulated annealing (GSA) [186, 187] and show how our Markov chain fits into this framework in which the Markov chain minimizes a virtual energy. Then we focus on proving the main theorem that CSA has asymptotic convergence to a CGM_{dn}.

Generalized Simulated Annealing (GSA). As we mentioned earlier, the proof of asymptotic convergence of simulated annealing (SA) in *unconstrained* NLPs [1] utilizes a *sufficient* condition of local balance equation (3.5). This sufficient condition, however, may not be satisfied by many general applications. GSA [186, 187] aims to establish a new framework for those applications, where the family of transition probabilities is given as follows:

Definition 3.1 *Communication Cost* "(Definition 2.1 in [186, 187]). Let $(Q_T)_{T>0}$ be a family of Markov kernels on E. We say that $(Q_T)_{T>0}$ is admissible for q and k if there exists a family of positive real-valued numbers $(V(i,j))_{(i,j)\in E\times E}$ (some of them may take the value $+\infty$) such that:

- $V(i,j) < +\infty$, iff $q(i,j) > 0$,
- for all $T > 0$, all $i, j \in E$,

$$\frac{1}{\kappa} q(i,j) e^{-V(i,j)/T} \leq Q_T(i,j) \leq \kappa q(i,j) e^{-V(i,j)/T}, \tag{3.20}$$

where function $V \ : E \times E \to [0, +\infty]$ is called the *communication cost function*."

Any one-step transition probability $Q_T(i,j)$ that satisfies the two conditions in Definition 3.1 is called generalized simulated annealing (GSA). There is no need of satisfying the local balance equation for $Q_T(i,j)$, because $q(i,j)Q_T(i,j)$ may not equal to $q(j,i)Q_T(j,i)$. For the special case where $V(i,j) = (f(j) - f(i))^+$ used in SA, we have the local balance equation $q(i,j)Q_T(i,j) = q(j,i)Q_T(j,i)$. In the following, we quote the notion of *A-graph* and *virual energy* as defined in [72, 186, 187]

Definition 3.2 *A-Graph* "(Definition 2.4 in [72, 186, 187]). Let $A \subset E$. A set g of arrows $i \to j$ in $A^c \times E$ is an *A-graph*, where $j \in \mathcal{N}(i)$, iff a) for each $i \in A^c$, there exists a unique $j \in E$ such that $i \to j \in g$; b) for each $i \in A^c$, there is a path in g ending on a point $j \in A$."

Here E is the set of all states (or nodes) in the Markov chain defined in Definition 3.1, A is a subset of E, and A^c is the complement of A in E. Accordingly, A-graph g is actually a spanning tree rooted at set A, where the tree is defined over the digraph constructed by neighborhood $\mathcal{N}(i)$. Let $G(A)$ be the set of A-graphs. For each $g \in G(A)$, the cost of g is defined as $V(g) = \sum_{i \to j \in g} V(i,j)$.

Definition 3.3 *Virtual Energy* "(Definition 2.5 in [186, 187]). For each state $i \in E$, its *virtual energy* $W(i)$ is defined as:

$$W(i) = \min_{g \in G(\{i\})} V(g), \tag{3.21}$$

which is the cost of the minimum spanning tree rooted at point i."

The following theorem shows the asymptotic convergence of GSA in minimizing virtual energy $W(i)$.

Proposition 3.1 "(Proposition 2.6 in [72, 186, 187]). For every $T > 0$, the unique stationary distribution π_T of the Markov chain satisfies:

$$\pi_T(i) \longrightarrow exp\left(-\frac{W(i) - W(E)}{T}\right) \quad \text{as} \quad T \longrightarrow 0, \tag{3.22}$$

where $W(i)$ is the virtual energy of i, and $W(E) = \min_{i \in S} W(i)$."

Asymptotic Convergence of CSA. Our Markov chain (3.13) fits into the framework of GSA (3.20), if we define an irreducible Markov kernel $Q_T(i,j) = P_T(\mathbf{x}, \mathbf{x}')$ and its associated communication cost $V(\mathbf{x}, \mathbf{x}')$:

$$V(\mathbf{x}, \mathbf{x}') = \begin{cases} (L(\mathbf{x}') - L(\mathbf{x}))^+ & \text{if } \mathbf{x}' = (x', \lambda) \\ \\ (L(\mathbf{x}) - L(\mathbf{x}'))^+ & \text{if } \mathbf{x}' = (x, \lambda'). \end{cases} \tag{3.23}$$

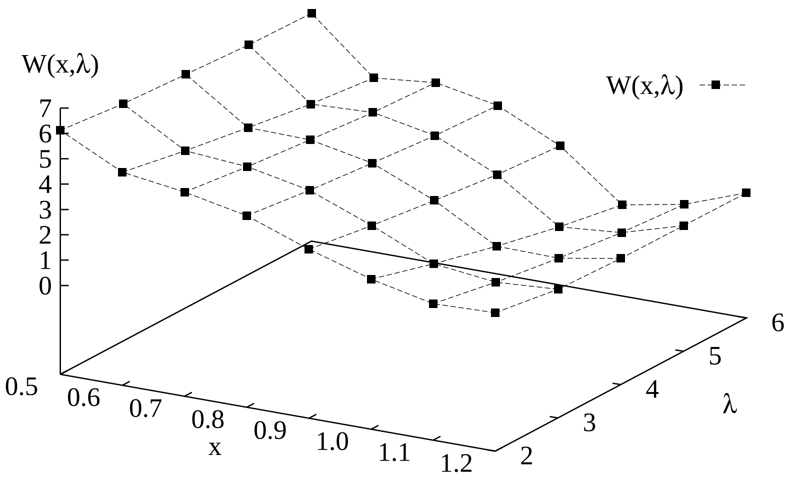


Figure 3.4: Virtual energy $W(\mathbf{x})$ of Example 3.1.

Obviously $V(\mathbf{x}, \mathbf{x}') \geq 0$ and function $V\colon\ S \times S \rightarrow [0, +\infty]$.

Hence, CSA minimizes an implicit virtual energy $W(\mathbf{x})$, according to GSA, and converges to the global minimum of $W(\mathbf{x})$ with probability one. Here, virtual energy $W(\mathbf{x})$ is the cost of the minimum spanning tree rooted at point $\mathbf{x}$ of the digraph governed by $\mathcal{N}(\mathbf{x})$. This is quite different from SA in solving unconstrained NLPs, whose explicit objective is to minimize a single objective $f(i)$ by performing probabilistic descents given by (3.3). Note that CSA does not minimize $L(x, \lambda)$ due to its probabilistic descents in the x subspace and probabilistic ascents in the λ subspace.

Example 3.1 (cont'd). Figure 3.4 shows the virtual energy $W(\mathbf{x})$ for Example 3.1. One can obtain $W(\mathbf{x})$ by enumerating all possible spanning trees rooted at $\mathbf{x}$ and finding the one with the minimum cost. This only works for a small problem like Example 3.1. Another more efficient way is to first compute stationary probability $\pi_T(\mathbf{x})$ based on (3.24) (discussed later) and then compute $W(\mathbf{x})$ using (3.22).

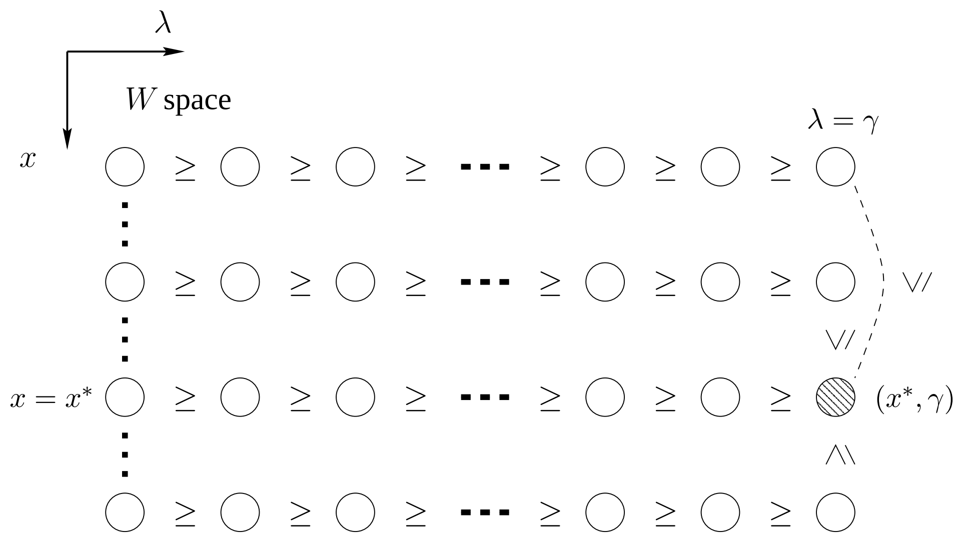


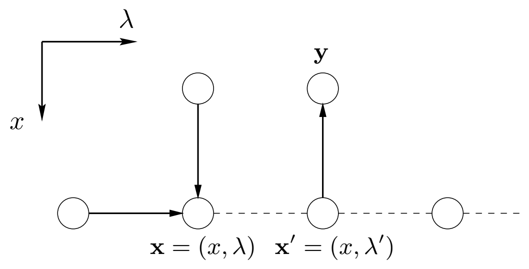
Figure 3.5: Proof strategy for Theorem 3.3.

Clearly, $L(x, \lambda) \neq W(x, \lambda)$. For a given x, as λ increases, $L(x, \lambda)$ is non-decreasing while $W(x, \lambda)$ is non-increasing. The minimum value of $W(x, \lambda)$ is at $(x = 1.0, \lambda = 6)$ (see Figure 3.4), which is a saddle point with the minimum objective value. However, the minimum value of $L(x, \lambda)$ is at $(x = 1.2, \lambda = 2)$ (see Figure 3.3b), which is not a feasible point. ■

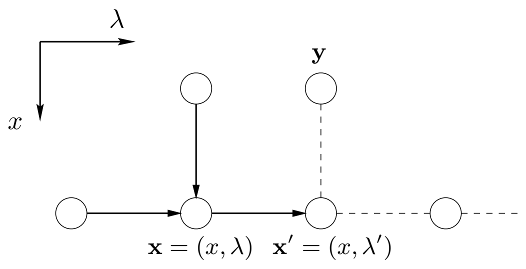
In order to prove that our Markov chain converges asymptotically to a CGM_{dn} of the original problem (1.1), we need to show that $W(\mathbf{x})$ is minimized at (x^*, γ), *i.e.*, $W((x^*, \gamma)) < W((x, \lambda))$ for $x^* \in X_{opt}$, all $x \in X - X_{opt}$ and $\lambda \in \Lambda$. This is stated in the following theorem.

Theorem 3.3 The Markov chain modeling CSA converges asymptotically to a CGM_{dn} $x^* \in X_{opt}$ with probability one.

Proof. The proof consists of two steps as shown in Figure 3.5. First, we show that for a given x, the virtual energy satisfies $W((x, \lambda')) \leq W((x, \lambda))$ for any $\lambda' > \lambda$ (the horizontal



a) $MT(x, \lambda)$



b) constructed tree $T(x, \lambda')$

Figure 3.6: Proof of part a1) in Theorem 3.3 (Solid arrow indicates edge in the spanning tree).

direction in Figure 3.5) [1]. Here both x and λ are vectors, and $\lambda' > \lambda$ if every $\lambda'_i \geq \lambda_i$ and there exists at least one j such that $\lambda'_j > \lambda_j$.

Second, we show that $W((x^*, \gamma)) < W((x, \gamma))$ at the maximum value γ of the Lagrange multipliers (the vertical direction along $\lambda = \gamma$ in Figure 3.5) [2], where $x^* \in X_{opt}$ and $x \in X - X_{opt}$. Hence, $W(\mathbf{x})$ is minimized at (x^*, γ) [3], and the Markov chain converges to CGM_{dn} $x^* \in X_{opt}$ with probability one.

a) Let us first compare virtual energy $W(\mathbf{x})$ with $W(\mathbf{x}')$ where $\mathbf{x} = (x, \lambda)$, $\mathbf{x}' = (x, \lambda')$, and $\lambda' > \lambda$. There exist two cases according to whether constraint $h(x)$ is satisfied or not.

a1) Consider the case in which $h(x) \neq 0$ (meaning that at least one $h_i(x) \neq 0$) and $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$, implying that there exists an edge $\mathbf{x} \rightarrow \mathbf{x}'$. Let $MT(\mathbf{x})$ be a minimum spanning tree rooted at $\mathbf{x}$ (see Figure 3.6a). We construct a spanning tree $T(\mathbf{x}')$ rooted at $\mathbf{x}'$ (see Figure 3.6b) as follows: i) add edge $\mathbf{x} \rightarrow \mathbf{x}'$ to $MT(\mathbf{x})$, and ii) delete edge $\mathbf{x}' \rightarrow \mathbf{y}$. Edge

[1]In Example 3.1, $W((0.6, 4)) = 4.03 \leq W((0.6, 3)) = 4.44$, and $W((0.8, 6)) = 3.14 \leq W((0.8, 2)) = 4.05$, shown in Figure 3.4.

[2]In Example 3.1, $W((1.0, 6)) = 0.097 < W((0.6, 6)) = 3.37$, and $W((1.0, 6)) = 0.097 < W((0.8, 6)) = 3.14$, shown in Figure 3.4.

[3]$(x^*, \gamma) = (1.0, 6)$ in Example 3.1

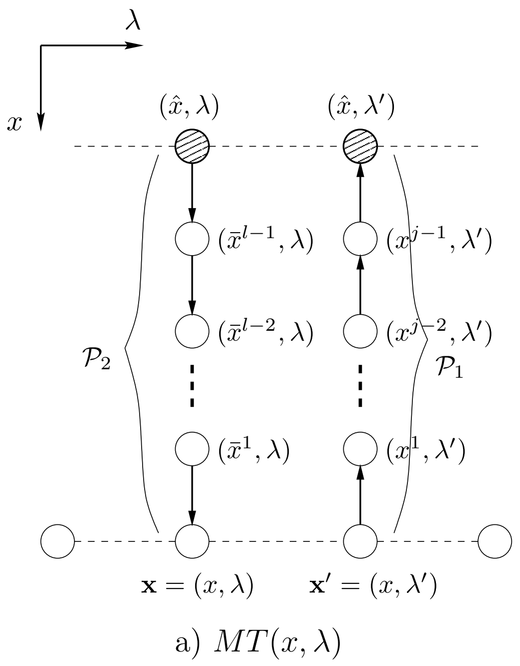


Figure 3.7: Proof of part a2) in Theorem 3.3 (Solid arrow indicates edge in the spanning tree).

$\mathbf{x}' \to \mathbf{y}$ always exists because there is a path from $\mathbf{x}'$ to $\mathbf{x}$ in $MT(\mathbf{x})$. Then cost $C(\mathbf{x}')$ of spanning tree $T(\mathbf{x}')$ satisfies:

$$C(\mathbf{x}') = W(\mathbf{x}) + V(\mathbf{x}, \mathbf{x}') - V(\mathbf{x}', \mathbf{y}) = W(\mathbf{x}) - V(\mathbf{x}', \mathbf{y}) \leq W(\mathbf{x}).$$

The equation is true because $V(\mathbf{x}, \mathbf{x}') = [L(\mathbf{x}) - L(\mathbf{x}')]^+ = [(\lambda - \lambda')^T h(x)]^+ = 0$ and $V(\mathbf{x}', \mathbf{y}) \geq 0$. In addition, $W(\mathbf{x}') \leq C(\mathbf{x}')$ due to the fact that $W(\mathbf{x}')$ is the cost of a minimum spanning tree. Therefore, we have $W(\mathbf{x}') \leq C(\mathbf{x}') \leq W(\mathbf{x})$.

a2) Consider the case in which $h(x) = 0$, implying that there is no edge from $\mathbf{x} = (x, \lambda)$ to $\mathbf{x}' = (x, \lambda')$ because constraint $h(x)$ is satisfied and λ is not allowed to change according to (3.8). The minimum spanning tree rooted at $\mathbf{x}$ must have has a directed path $\mathcal{P}_1 = (x, \lambda') \to (x^1, \lambda') \to \cdots, \to (x^{j-1}, \lambda') \to (\hat{x}, \lambda')$ from $\mathbf{x}' = (x, \lambda')$ to $(\hat{x}, \lambda')$ and a directed path $\mathcal{P}_2 = (\hat{x}, \lambda) \to (\bar{x}^{l-1}, \lambda) \to \cdots, \to (\bar{x}^1, \lambda) \to (x, \lambda)$ from $(\hat{x}, \lambda)$ to $\mathbf{x} = (x, \lambda)$ (see

Figure 3.7a), where $\hat{x}$ are points (shown as shaded nodes in Figure 3.7) such that $h(\hat{x}) \neq 0$ (meaning that at least one constraint is not satisfied at $\hat{x}$), $h(x^i) = 0$ $(i = 1, 2, \cdots, j-1)$, and $h(\bar{x}^i) = 0$ $(i = 1, 2, \cdots, l-1)$. Such j and l always exist due to the ergodicity of the Markov chain, and path $\mathcal{P}_1$ may differ from path $\mathcal{P}_2$. Note that there is no relationship between $f(x)$ and $f(\hat{x})$ and that the spanning tree at x^i and $\bar{x}^i$ can only move along the x subspace because they all satisfy the constraints.

In contrast, the minimum spanning tree rooted at $\mathbf{x}'$ must have a directed path $\mathcal{P}'_1 = (x, \lambda) \rightarrow (x^1, \lambda) \rightarrow \cdots, \rightarrow (x^{j-1}, \lambda) \rightarrow (\hat{x}, \lambda)$ from $\mathbf{x} = (x, \lambda)$ to $(\hat{x}, \lambda)$ and a directed path $\mathcal{P}'_2 = (\hat{x}, \lambda') \rightarrow (\bar{x}^{l-1}, \lambda') \rightarrow \cdots, \rightarrow (\bar{x}^1, \lambda') \rightarrow (x, \lambda')$ from $(\hat{x}, \lambda')$ to $\mathbf{x} = (x, \lambda')$ (see Figure 3.7b). Then the costs of $\mathcal{P}_1$ and $\mathcal{P}'_1$ satisfy:

$$\begin{aligned}
C(\mathcal{P}_1) &= V((x, \lambda'), (x^1, \lambda')) + \cdots + V((x^{j-2}, \lambda'), (x^{j-1}, \lambda')) + V((x^{j-1}, \lambda'), (\hat{x}, \lambda')) \\
&= V((x, \lambda), (x^1, \lambda)) + \cdots + V((x^{j-2}, \lambda), (x^{j-1}, \lambda)) + [L(\hat{x}, \lambda') - L(x^{j-1}, \lambda')]^+ \\
&\geq V((x, \lambda), (x^1, \lambda)) + \cdots + V((x^{j-2}, \lambda), (x^{j-1}, \lambda)) + [L(\hat{x}, \lambda) - L(x^{j-1}, \lambda)]^+ \\
&= C(\mathcal{P}'_1),
\end{aligned}$$

where $V((x^{i-1}, \lambda'), (x^i, \lambda')) = V((x^{i-1}, \lambda), (x^i, \lambda))$ for $i = 1, 2, \ldots, j-1$ (here $x^0 = x$) and $L(x^{j-1}, \lambda') = L(x^{j-1}, \lambda)$ are used since $h(x^i) = 0$, and $L(\hat{x}, \lambda') \geq L(\hat{x}, \lambda)$ is used because $h(\hat{x}) \neq 0$ and $\lambda' > \lambda$.

Similarly, the costs of $\mathcal{P}_2$ and $\mathcal{P}'_2$ satisfy:

$$\begin{aligned}
C(\mathcal{P}_2) &= V((\hat{x}, \lambda), (\bar{x}^{l-1}, \lambda)) + V((\bar{x}^{l-1}, \lambda), (\bar{x}^{l-2}, \lambda)) + \cdots + V((\bar{x}^1, \lambda), (x, \lambda)) \\
&= [L(\bar{x}^{l-1}, \lambda) - L(\hat{x}, \lambda)]^+ + V((\bar{x}^{l-1}, \lambda'), (\bar{x}^{l-2}, \lambda')) + \cdots + V((\bar{x}^1, \lambda'), (x, \lambda')) \\
&\geq [L(\bar{x}^{l-1}, \lambda') - L(\hat{x}, \lambda')]^+ + V((\bar{x}^{l-1}, \lambda'), (\bar{x}^{l-2}, \lambda')) + \cdots + V((\bar{x}^1, \lambda'), (x, \lambda')) \\
&= C(\mathcal{P}'_2),
\end{aligned}$$

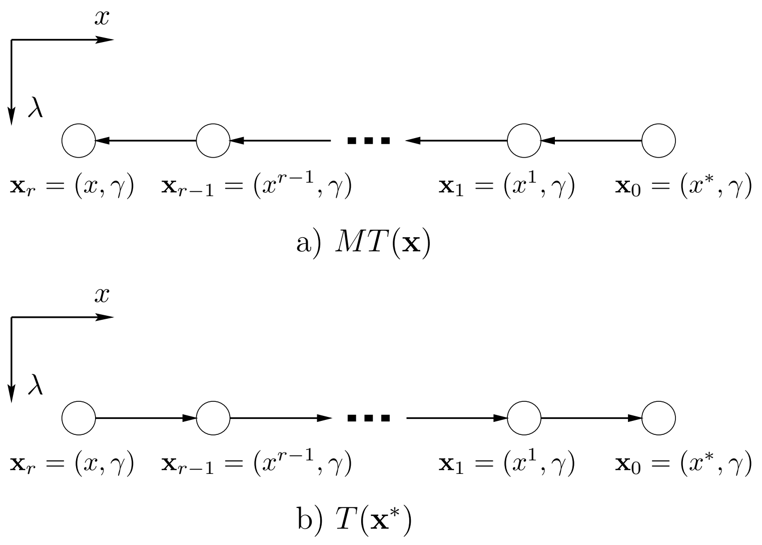

x
λ
$\mathbf{x}_r = (x,\gamma)$
$\mathbf{x}_{r-1} = (x^{r-1},\gamma)$
$\mathbf{x}_1 = (x^1,\gamma)$
$\mathbf{x}_0 = (x^*,\gamma)$
a) $MT(\mathbf{x})$
x
λ
$\mathbf{x}_r = (x,\gamma)$
$\mathbf{x}_{r-1} = (x^{r-1},\gamma)$
$\mathbf{x}_1 = (x^1,\gamma)$
$\mathbf{x}_0 = (x^*,\gamma)$
b) $T(\mathbf{x}^*)$

Figure 3.8: Proof of part b) in Theorem 3.3.

where $V((\bar{x}^i,\lambda'),(\bar{x}^{i-1},\lambda')) = V((\bar{x}^i,\lambda),(\bar{x}^{i-1},\lambda))$ for $i = 1,2,\ldots,l-1$ and $L(\bar{x}^{l-1},\lambda) = L(\bar{x}^{l-1},\lambda')$ are used since $h(x^i) = 0$, and $L(\hat{x},\lambda') \geq L(\hat{x},\lambda)$ is used because $h(\hat{x}) \neq 0$ and $\lambda' > \lambda$.

In addition, for any $\hat{x}$, $V((\hat{x},\lambda),(\hat{x},\lambda')) = [L(\hat{x},\lambda) - L(\hat{x},\lambda')]^+ = [(\lambda-\lambda')^T h(\hat{x})]^+ = 0$ in $MT(\mathbf{x}')$, and $V((\hat{x},\lambda'),(\hat{x},\lambda)) = [(\lambda'-\lambda)^T h(\hat{x})]^+ \geq 0$ in $MT(\mathbf{x})$. Therefore, we have $W(\mathbf{x}') \leq W(\mathbf{x})$.

b) For any $x \in X$ and $\lambda \in \Lambda$, there exists a path such that $\lambda < \lambda_1 < \lambda_2 < \cdots < \lambda_l < \gamma$. According to (a), we have $W((x,\gamma)) \leq W((x,\lambda_l)) \leq \cdots \leq W((x,\lambda_1)) \leq W((x,\lambda))$, where γ is the maximum value of λ. This means that the Markov chain try to push λ to its maximum γ, but this movement is probabilistic since occasional reductions of λ are also allowed. Accordingly, we only need to compare $W((x,\gamma))$ ($x \in X - X_{opt}$) with $W((x^*,\gamma))$ ($x^* \in X_{opt}$) at the maximum value γ of the Lagrange multipliers.

Let $MT(\mathbf{x})$ be the minimum spanning tree of $\mathbf{x} = (x,\gamma)$ and its associated virtual energy be $W(\mathbf{x})$. There must exist a path $\mathcal{P} = \mathbf{x}_0(=\mathbf{x}^*) \rightarrow \mathbf{x}_1 \rightarrow \cdots \rightarrow \mathbf{x}_{r-1} \rightarrow \mathbf{x}_r(=\mathbf{x})$ of length

r from $\mathbf{x}^* = (x^*, \gamma)$ to $\mathbf{x}$ in $MT(\mathbf{x})$ (see Figure 3.8a). Reversing this path, we obtain a path from $\mathbf{x}$ to $\mathbf{x}^*$ and also a spanning tree $T(\mathbf{x}^*)$ (see Figure 3.8b) of $\mathbf{x}^*$ with cost $C(\mathbf{x}^*)$, satisfying:

$$\begin{aligned} W(\mathbf{x}) - C(\mathbf{x}^*) &= \sum_{k=1}^{r} \{[L(\mathbf{x}_k) - L(\mathbf{x}_{k-1})]^+ - [L(\mathbf{x}_{k-1}) - L(\mathbf{x}_k)]^+\} \\ &= \sum_{k=1}^{r} [L(\mathbf{x}_k) - L(\mathbf{x}_{k-1})] = L(\mathbf{x}_r) - L(\mathbf{x}_0) \\ &= L(x, \gamma) - L(x^*, \gamma) > 0, \end{aligned}$$

based on the definition of γ. Because $W((x^*, \gamma)) \leq C((x^*, \gamma))$, we have $W((x^*, \gamma)) \leq C((x^*, \gamma)) < W((x, \gamma))$.

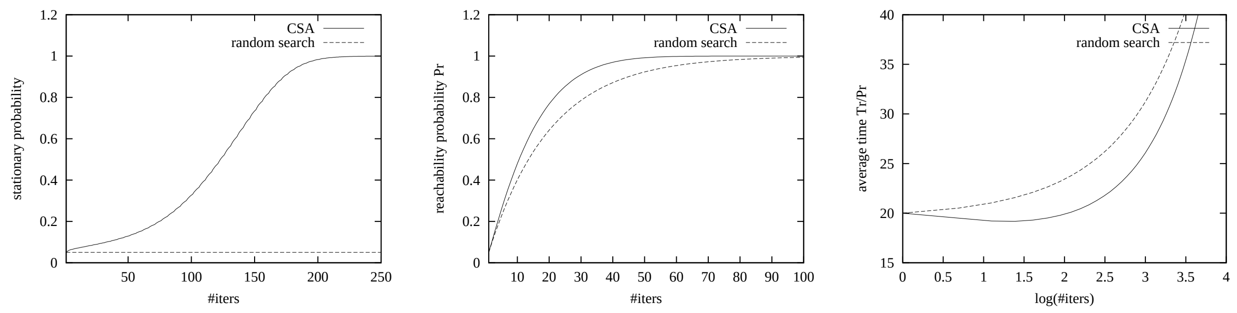
c) Summarizing a) and b), we conclude that virtual energy $W(\mathbf{x})$ of $\mathbf{x} = (x, \lambda)$ is minimized at CGM_{dn} (x^*, γ). Thus, the Markov chain converges to CGM_{dn} $x^* \in X_{opt}$ with probability one according to Proposition 3.1. ■

Example 3.1 (cont'd). Since multiple runs of CSA do not illustrate their asymptotic convergence to the global minimum, we evaluate the stationary probabilities π_T numerically at a given T by first computing acceptance probability $A_T(\mathbf{x}, \mathbf{x}')$ using (3.11) and one-step transition probability $P_T(\mathbf{x}, \mathbf{x}')$ using (3.13). The stationary distribution π_T of the Markov chain with transition matrix P_T evolves with iteration k as follows:

$$p(k+1) = p(k) P_T \qquad \text{for any given initial vector } p(k=0), \tag{3.24}$$

As $\pi_T = \lim_{k \to \infty} p(k)$, independent of starting vector $p(k=0)$, we set $p_i(k=0) = 1/|S|$ for all $i = 1, 2, \cdots, |S|$.

In this simple example, we set initial temperature $T^0 = 1.0$, $N_T = 5$, and cooling rate $\alpha = 0.9$. Figure 3.9a shows the stationary probabilities at $x^* = 1.0$ (the optimal solution) for both



a) Prob. of hitting CGM_{dn} b) reachability probability P_r c) T_r/P_r

Figure 3.9: Example showing the convergence probabilities, reachability probabilities P_r, and average time T_r/P_r (see section 1.3 and [198] for detail) to find the global solution for CSA and random search.

CSA and pure random search. The probability of hitting CGM_{dn}increases monotonically with iterations and approaches one as T is small in CSA, but remains constant for random search. Thus, CSA has a larger probability to find the CGM_{dn}than random search, and the Markov chain modeling CSA converges asymptotically to the CGM_{dn}.

Figure 3.9b and Figure 3.9c depict, respectively, the corresponding reachability probabilities at each iteration and average times T_r/P_r (see section 1.3 and [198] for detail) to find CGM_{dn}at each iteration in log scale for both CSA and random search. The curve T_r/P_r for random search is monotonously increased, whereas the curve T_r/P_r for CSA is convex. Such a convexity property for CSA has been proved and then utilized to derive an optimal any-time CSA in [198]. ■

3.4 Optimal Cooling Schedules

Theoretically, we have proved that CSA converges asymptotically to a $CGM_{dn}$with probability one if the cooling schedule satisfies the condition of (3.16). Such a cooling schedule, however, may be too slow to be applied to solve real applications when the goal is to find high-quality solutions efficiently.

Like conventional simulated annealing [**?**, 1], the difficulty in applying CSA is to determine an optimal cooling schedule that allows a CGM_{dn} to be found in the shortest average amount of time. Intuitively, such a schedule does not generally correspond to the case in which the success probability in one run of CSA is the highest, because the highest success probability in one run requires a cooling schedule that approaches infinity. Rather, the optimal schedule happens when CSA is allowed to be run multiple times, and the average total time of the multiple runs is the shortest. If each run is too short, it will have very little chance of finding a CGM_{dn}. This leads to a very large number of runs in order to find a CGM_{dn} and, thus, a long average total completion time. In contrast, if each run is too long, it will have a high probability of finding a CGM_{dn} but a very long completion time. Hence, the average total completion time of a few long runs of CSA will still be very long.

An *optimal cooling schedule* [198] is defined as one that leads to the shortest average total completion time of multiple runs of CSA in order to find a solution of prescribed quality. The optimal cooling schedule, however, is highly problem-dependent and is impossible to determine it in advance.

Let T_r be the maximum time used in CSA and P_r be the reachability probability of finding a feasible solution with prescribed quality f^* within T_r. The *average time* of finding a solution with quality f^* using multiple runs is T_r/P_r, as derived in (1.3).

By proving the convexity of T_r/P_r with respect to $log(T_r)$, Wah and Chen [198] have proposed CSA with iterative deepening, called CSA_{ID}, by applying iterative deepening to the cooling schedule. CSA_{ID} starts from a very fast cooling schedule and doubles it after a certain number of consecutive runs of CSA until a feasible solution with a prescribed solution quality is found. They have proved that the total time spent on CSA_{ID} is of the same order as the time of one CSA run with the optimal cooling schedule.

Because the curve of T_r/P_r is convex with respect to $log(T_r)$ and P_r is less than one, CSA with an optimal cooling schedule finds desired solutions with the shortest average time (corresponding to the valley of curve T_r/P_r) in multiple runs. On the other hand, the T_r/P_r curve in a random search is not convex (monotonously increasing in Figure 3.9c), and, thus, its performance is always worse than CSA with an optimal cooling schedule. For example, in Figure 3.9c, the T_r/P_r curve in a random search is always above that in CSA, leading to longer average time than CSA.

3.5 Summary

In this chapter, we have presented a new stochastic global optimization algorithm, called CSA, for solving discrete constrained NLPs. The algorithm is based on Theorem 2.2 of the first-order necessary and sufficient conditions for discrete CLM_{dn} in the theory of discrete constrained optimization using Lagrange multipliers. This theorem establishes a one-to-one correspondence between discrete-space saddle points and CLM_{dn}. Hence, global optimization can be achieved by searching in the space of saddle points.

Our proposed CSA performs both probabilistic descents in the variable subspace of x and probabilistic ascents in the Lagrange-multiplier subspace of λ. We have proved that CSA converges asymptotically to a saddle point with the minimum objective value, namely, a CGM_{dn}, with probability one. By achieving asymptotic convergence, CSA is one of the major developments in nonlinear *constrained* global optimization today and complements simulated annealing (SA) in nonlinear *unconstrained* global optimization. Based on Theorem 2.3 in the theory of discrete constrained optimization using Lagrange multipliers, CSA is a powerful method for solving discrete, continuous, and mixed-integer NLPs.

Chapter 4

Design of Efficient Strategies for CSA

In this chapter, we examine various strategies used in constrained simulated annealing (CSA) that may affect its performance in solving discrete, continuous, and mixed-integer constrained NLPs. The strategies consist of adaptive neighborhoods, distributions to control sampling, acceptance probabilities, as well as cooling schedules. For each strategy, we first briefly discuss related work in SA and then extend them to CSA or develop new ones for CSA. Because CSA performs probabilistic descent in the x subspace that is the same as that of SA, most strategies used in SA can be extended to CSA in its x subspace. But for the Lagrange-multiplier subspace, we need to develop new strategies.

4.1 Choice of Neighborhoods

CSA consists of two major steps: generating trial points and accepting them based on some acceptance probability. In theory, any neighborhoods $\mathcal{N}^1_{dn}(x)$ and $\mathcal{N}^2_{cn}(\lambda)$ that satisfy (3.8) and Definition 1.1 will guarantee asymptotic convergence. In practice, however, it is important to choose appropriate neighborhoods for generating proper trial points in x and λ in order to improve the probability of finding a CGM_{dn} when finite cooling schedules are used.

Neighborhoods describe how trial points differ from the current point, and their sizes determine the range of generating trial points [45]. A neighborhood commonly used in SA allows trial points to differ from the current point in one variable, because it has higher probability of accepting trial points than those neighborhoods with more than one variables changed [49]. Here we adopt the same strategy in both the variable and the Lagrange-multiplier subspaces.

In our implementation, we choose a simple neighborhood $\mathcal{N}_{dn}^1(x)$ as the set of points x' that differ from x in one variable x_i. Likewise, $\lambda' \in \mathcal{N}_{cn}^2(\lambda)$ differs from λ in one variable. In general, both x' and λ' can differ from x and λ in more than one variables, as long as the conditions in (3.8) and Definition 1.1 are satisfied.

We characterize $\mathcal{N}_{dn}^1(x)$ by vector σ, where σ_i controls the size of the neighborhood along x_i. Similarly, we characterize $\mathcal{N}_{cn}^2(\lambda)$ by vector ϕ, where ϕ_i denotes the maximum possible perturbation along λ_i.

4.2 Generation of Trial Points

Three general distributions used in SA to generate trial points include uniform [49], Gaussian [45, 222], and Cauchy [51]. Examples of other methods are logarithmic explorations [50] and tree annealing [33, 32] that organize neighborhoods in a tree. Such methods only work well for problems with specific objective or constraint forms and may not work for general problems. Here we test the three general distributions for generating trial points in x.

Generation of trial points (x', λ). In generating trial point $\mathbf{x}' = (x', \lambda)$ from $\mathbf{x} = (x, \lambda)$, we consider two cases. To generate a *continuous* trial point, we set:

$$x' = x + \theta_i \, \mathbf{e}_i, \tag{4.1}$$

where $\mathbf{e}_i$ is a vector with its i^{th} component being 1 and the other components being 0, and i is randomly generated from $\{1, 2, \cdots, n\}$.

There are three possible choices for θ_i: a) *uniform*, where θ_i is generated uniformly in $[-\sigma_i, \sigma_i]$ [49]; b) *Gaussian*, where θ_i is generated from a Gaussian distribution with zero mean and variance σ_i [222]; and c) *Cauchy*, where θ_i is generated from a Cauchy density [51, 222]:

$$f_d(x) = \frac{1}{\pi} \frac{\sigma_i}{\sigma_i^2 + x^2}.$$

The major advantage [51, 222] of using a Cauchy distribution lies in its long flat tail. In addition to generating samples close to the current point, there is also a high probability of sampling remote points, making it easy to escape from local minima, especially when temperatures are low and the basins of attraction to local minima are large.

To generate a *discrete* trial point x', we first generate a point by (4.1) and then round it to its closest discrete grid point. If it happens that $x' = x$, we set $x' = x + j/s$, where $1/s$ is the grid size and j has equal probability to take value $+1$ or -1.

Generation of trial points (x, λ')**.** In generating $\mathbf{x}' = (x, \lambda')$ from $\mathbf{x} = (x, \lambda)$, we apply the following rule:

$$\lambda' = \lambda + \eta_j \, \mathbf{e}_j, \tag{4.2}$$

where j is uniformly distributed in $\{1, 2, \cdots, m\}$.

We test three possible choices for η_j: a) *symmetric uniform* (S-uniform), where η_j is generated uniformly in $[-\phi_j, \phi_j]$; b) *non-symmetric uniform* (NS-uniform), where η_j is generated uniformly in $[-\frac{2}{3}\phi_j, \frac{4}{3}\phi_j]$; and c) *nonuniform*, where η_j is generated uniformly from $[-\phi_j, 0]$ and $[0, \phi_j]$ with probabilities 1/3 and 2/3, respectively.

We set the ratio of generating (x', λ) and (x, λ') from (x, λ) to be $10n$ to m; that is, every variable x_i is tried 10 times more often than each Lagrange multiplier λ_j. Hence, x is updated more frequently than λ.

4.3 Acceptance Probabilities

After generating $\mathbf{x}' = (x', \lambda)$ or $\mathbf{x}' = (x, \lambda')$, $\mathbf{x}'$ is accepted according to the Metropolis acceptance probability (3.11).

Besides the Metropolis rule, we also evaluate three other possible acceptance rules studied in SA: Logistic acceptance rule [1, 164], Hastings' rule [95, 78], and Tsallis' rule [8, 93]. All these acceptance rules lead to asymptotic convergence [163], although they differ in solution quality when applied under a finite cooling schedule.

Because CSA carries out probabilistic descents in the x subspace like SA, the acceptance probabilities proposed in SA can also be utilized in CSA when x is changed in generating a trial point. For probabilistic ascents of CSA in the λ subspace, we modify the three acceptance probabilities by favoring increases in Lagrangian values.

Let $\delta L_x = L(x', \lambda) - L(x, \lambda)$ and $\delta L_\lambda = L(x, \lambda') - L(x, \lambda)$. The three acceptance probabilities for CSA are defined as follows:

$$\text{Logistic rule:} \quad A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} \frac{1}{1+e^{+\delta L_x/T}} & \text{if } \mathbf{x}' = (x', \lambda) \\ \frac{1}{1+e^{-\delta L_\lambda/T}} & \text{if } \mathbf{x}' = (x, \lambda'), \end{cases} \tag{4.3}$$

$$\text{Hastings' rule:} \quad A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} \frac{1+2\left[\frac{1}{2}\min\{e^{\delta L_x/T},\, e^{-\delta L_x/T}\}\right]^\gamma}{1+e^{+\delta L_x/T}} & \text{if } \mathbf{x}' = (x', \lambda) \\ \frac{1+2\left[\frac{1}{2}\min\{e^{\delta L_\lambda/T},\, e^{-\delta L_\lambda/T}\}\right]^\gamma}{1+e^{-\delta L_\lambda/T}} & \text{if } \mathbf{x}' = (x, \lambda'), \end{cases} \tag{4.4}$$

$$\text{Tsallis' rule:} \quad A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} \min\left\{1, [1-(1-q)\delta L_x/T]^{1/(1-q)}\right\} & \text{if } \mathbf{x}' = (x', \lambda) \\ \min\left\{1, [1+(1-q)\delta L_\lambda/T]^{1/(1-q)}\right\} & \text{if } \mathbf{x}' = (x, \lambda'), \end{cases} \tag{4.5}$$

where $\gamma = 2$ in Hastings' rule, and q in Tsallis' rule starts from 2 and decreases exponentially to 1 as T is reduced.

4.4 Adaptive Neighborhoods

Adaptive neighborhoods for x. During the course of CSA, we adaptively adjust neighborhood $\mathcal{N}^1_{dn}(x)$ by updating scale vector σ for x using a modified $1:1$ rate rule developed in SA [49] in order to balance the ratio between accepted and rejected configurations:

$$\sigma_i = \begin{cases} \sigma_i \left[1 + \beta_0 (p_i - p_u)/(1 - p_u)\right] & \text{if } \ p_i > p_u \\ \sigma_i / \left[1 + \beta_1 (p_v - p_i)/p_v\right] & \text{if } \ p_i < p_v, \end{cases} \tag{4.6}$$

where p_i is the ratio of accepting x' in which x'_i differs from x_i. If p_i is low ($p_i < p_v$), then the trial points generated are rejected too often. In that case, we reduce σ_i in order to increase the chance of generating acceptable trial points. In contrast, if p_i is high ($p_i > p_u$), then the trial points generated are too close to (x, λ). In that case, we increase σ_i in order to generate more remote trial points.

Empirically, we chose the parameters as follows. When a trial point is generated by a uniform or a Gaussian distribution, we set $\beta_0 = 2$, $\beta_1 = 2$, $p_u = 0.6$, and $p_v = 0.4$. When a trial point is generated by a Cauchy distribution, we have two sets of parameters: a) Cauchy$_0$ with $\beta_0 = 2$, $\beta_1 = 2$, $p_u = 0.3$, and $p_v = 0.2$; and b) Cauchy$_1$ with $\beta_0 = 7$, $\beta_1 = 2$, $p_u = 0.3$, and $p_v = 0.2$. The only difference between them lies in β_0, the ratio of enlarging neighborhood size.

Adaptive neighborhoods for λ. We adjust ϕ according to the degree of constraint violations. Here we decompose ϕ as:

$$\phi = w \otimes h(x) = [w_1 h_1(x), w_2 h_2(x), \cdots, w_m h_m(x)] \tag{4.7}$$

where $\otimes$ represents vector product. When $h_i(x)$ is satisfied, $\phi_i = 0$ because there is no need to update the corresponding λ_i. On the other hand, when a constraint is not satisfied, we adjust ϕ_i by modifying w_i according to how fast $h_i(x)$ is changing:

$$w_i = \begin{cases} \eta_0 \, w_i & \text{if } \; h_i(x) > \tau_0 T \\ \eta_1 \, w_i & \text{if } \; h_i(x) < \tau_1 T, \end{cases} \tag{4.8}$$

where $\eta_0 = 1.25$, η_1=0.8, $\tau_0 = 1.0$, and $\tau_1 = 0.01$ are all chosen experimentally.

When $h_i(x)$ is reduced too quickly (*i.e.,* $h_i(x) < \tau_1 T$), $h_i(x)$ may be over-weighted, leading to possibly poor objective values or difficulty in satisfying under-weighted constraints. In this case, we reduce the neighborhood size of λ_i. In contrast, if $h_i(x)$ is reduced too slowly (*i.e.,* $h_i(x) > \tau_0 T$), we enlarge the neighborhood size of λ_i in order to improve its possibility of satisfaction. Note that w_i is adjusted using T as a reference because constraint violations are expected to decrease when T drops.

4.5 Cooling Schedules

A cooling schedule consists of initial temperature T^0, number of trials per temperature N_T, and the ratio α of reducing temperature T in Figure 3.2.

Initial temperature T^0 has to be set large enough to make initial acceptance probabilities close to 1. To avoid setting it too high, SA normally computes its initial temperature based on some initial probability of accepting degraded moves in x space [**?**, 1]. However, in addition to minimizing the objective, CSA has to satisfy a set of constraints. Accordingly, we need to consider constraint violations in determining the initial temperature.

We generate initial temperature T^0 by first randomly generating 100 points of x and their corresponding neighboring points x', where each component $|x'_i - x_i| \leq 0.001(u_i - l_i)$ and u_i and l_i are upper and lower bounds of variable x_i, respectively. We set T^0 =

$max_{x,x',i}\{|L(x',1) - L(x,1)|, |h_i(x)|\}$. Our rationale is based on the initial amount of violations observed in a problem.

In our implementation, we select $N_T = \zeta(10n + m)$ and $\zeta = 10(n + m)$, where n is the number of variables, and m is the number of constraints. This setting is based on the heuristic rule in SA [49] and uses $n + m$ instead of n because of the constraints.

There are many cooling schedules developed for SA. These include logarithmic annealing schedules [1, 90, 28], schedules inversely proportional to annealing steps [183], simulated quenching scheduling [109, 111], geometric cooling schedules [**?**, 153], constant annealing [30], arithmetic annealing [136, 159], polynomial-time cooling [2, 1], adaptive temperature scheduling based on acceptance ratio of bad moves [212], and non-equilibrium SA (NESA) [42, 158].

Geometric cooling schedules have been used widely in practice for its simplicity and easy implementation [1]. For this reason, we reduce T by a geometric cooling schedule (3.12) in CSA, where α is a constant smaller than 1. In our experiments, we have used four cooling rates: $\alpha = 0.1$, $\alpha = 0.5$, $\alpha = 0.8$, and $\alpha = 0.95$.

4.6 Selected Test Benchmarks

To evaluate various strategies used in CSA, we have chosen 12 difficult benchmark problems: G1, G2 and G5 from G1-G10 [133, 119], and 2.1, 2.7.5, 5.2, 5.4, 6.2, 6.4, 7.2, 7.3 and 7.4 from a collection of optimization benchmarks [68]. The former were originally developed for testing and tuning various constraint handling techniques in evolutionary algorithms (EAs), while the latter were derived from practical applications, mostly from chemical engineering.

The 12 selected test problems have objective functions of various types (linear, quadratic, and nonlinear) and linear/nonlinear constraints of equalities and inequalities. The number

Table 4.1: Statistics of the 12 benchmark problems used for tuning CSA.

Problem ID	number of variables	number of constraints	number of bounds	objective	equality constraints		inequality constraints	
					size	type	size	type
G1	13	9	26	quadratic	0	-	9	linear
G2	20	2	40	nonlinear	0	-	2	nonlinear
G5	4	5	8	nonlinear	3	nonlinear	2	linear
2.1	5	1	10	quadratic	0	-	1	linear
2.7.5	20	10	40	quadratic	0	-	10	linear
5.2	46	36	92	quadratic	36	quadratic	0	-
5.4	32	26	64	linear	26	quadratic	0	-
6.2	9	6	18	linear	4	quadratic	2	quadratic
6.4	9	6	18	linear	4	quadratic	2	quadratic
7.2	16	13	32	nonlinear	16	quadratic	0	-
7.3	27	19	54	nonlinear	19	quadratic	0	-
7.4	38	23	76	nonlinear	23	quadratic	0	-

of variables is up to about 50, and that of constraints, including simple bounds, is up to about 100. The ratio of feasible space with respect to the whole search space varies from 0% to almost 100%, and the topologies of feasible regions are quite different. Table 4.1 shows the statistics of these problems.

Due to a lack of discrete and mixed-integer benchmarks, we derive them from the two sets of continuous benchmarks [133, 119, 68] as follows. In generating a constrained MINLP, we assume that variables with odd indices are continuous and those with even indices are discrete. In discretizing continuous variable x_i in range $[l_i, u_i]$, where l_i and u_i are lower and upper bounds of x_i, respectively, we force x_i to take values from the set:

$$A_i = \begin{cases} \left\{a_i + \frac{b_i - a_i}{s} j, \ j = 0, 1, \cdots, s\right\} & \text{if } b_i - a_i < 1 \\ \left\{a_i + \frac{1}{s} j, \ j = 0, 1, \cdots, \lfloor (b_i - a_i) s \rfloor\right\} & \text{if } b_i - a_i \geq 1, \end{cases} \tag{4.9}$$

where $s = 10^7$. Hence, the search spaces produced for discrete and mixed-integer NLPs are very huge and are impossible to be enumerated. For example, for a problem with 10 discrete variables, the size of the search space is at least $(10^7)^{10} = 10^{70}$. Using such a finely discretized search space allows us to compare directly the quality of solutions between continuous problems and their discretized versions, since a CLM_{cn} in the continuous version should differ little from the corresponding solution in the discretized version.

4.7 Results on Evaluating CSA Strategies

The various strategies implemented in CSA can be represented by a triplet: (distribution for generating trial point (x', λ), distribution for generating trial point (x, λ'), acceptance probability for the trial point). The distribution for (x', λ) can be uniform, Gaussian, Cauchy$_0$ or Cauchy$_1$ (see Section 4.4); the distribution for (x, λ') can be symmetric uniform (S-uniform), non-symmetric uniform (NS-uniform), and nonuniform; and the acceptance probability can be Metropolis (M), Logistic (L), Hastings (H) or Tsallis (T).

Using a given strategy, we evaluated each problem by running CSA from randomly generated starting points until a feasible solution was found or until 100 runs of CSA had been made without finding a feasible solution. In the latter case, we declare that CSA fails to find a solution for the problem in this run. Here, we assume that an equality constraint is satisfied if $\Phi = 10^{-5}$ (see Theorem 2.3 in Section 2.2.4).

We then repeated each run 100 times to obtain at most 100 pairs of CPU time and solution quality. Let $t_x(i)$ and $f_x(i)$ be, respectively, the CPU time and objective value of the i^{th} run, assuming that a feasible solution has been found. Further, let $t_r(i)$ and $f_r(i)$ be, respectively, the CPU time and objective value of the baseline strategy (Cauchy$_1$, S-uniform, M) run using the same sequence of starting points. If the i^{th} run leads to feasible solutions, we normalize time and quality as follows:

$$r_t(i) = \begin{cases} t_x(i)/t_r(i) - 1 & \text{if } t_x(i) > t_r(i) \\ 1 - t_r(i)/t_x(i) & \text{if } t_x(i) < t_r(i), \end{cases} \tag{4.10}$$

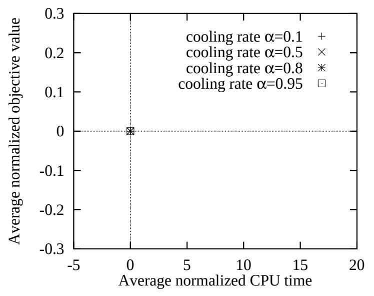
$$r_f(i) = (f_x(i) - f_r(i))/|f_{best}|, \tag{4.11}$$

where f_{best} is the best-known solution for the problem. We use (4.10) to measure the symmetric speedup [199] in order to give equal weights to speedups and slowdowns, but use (4.11) to measure improvements and degradations in the objective value because objective values can be negative. Finally, we evaluate the average normalized time r_t and average normalized quality r_f for the R (out of 100) runs that led to feasible solutions:

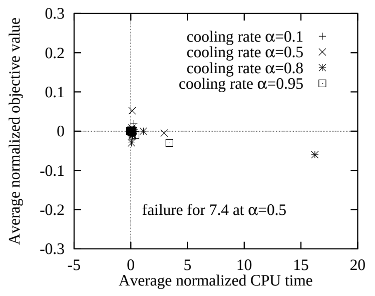
$$r_f = \frac{1}{R}\sum_{i=1}^{R} r_f(i), \qquad r_t = \frac{1}{R}\sum_{i=1}^{R} r_t(i). \tag{4.12}$$

A strategy is said to be better for a given problem if both r_f and r_t are smaller.

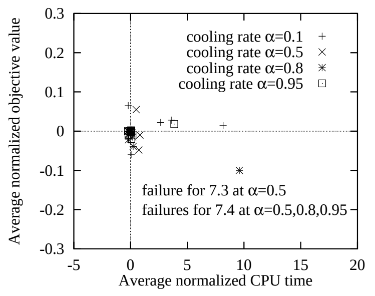
Figure 4.1 shows the results on evaluating the 12 mixed-integer benchmarks on a subset of the 48 combinations of strategies at cooling rates 0.1, 0.5, 0.8 and 0.95, respectively. Cauchy$_0$ has similar performance as Cauchy$_1$, but fails to find a solution for Problem 7.4 at cooling rate 0.5. CSA using Gaussian or uniform distribution tends to spend less times and obtain slightly better solutions than Cauchy$_1$ for some NLPs, but fails to solve some very difficult NLPs, such as 7.3 at cooling rates 0.5 and 7.4, because it generated more infeasible local points and got stuck there. Among the four acceptance probabilities, Logistic, Hastings' and Tsallis' rules are worse, using either more running times or reaching worse solutions



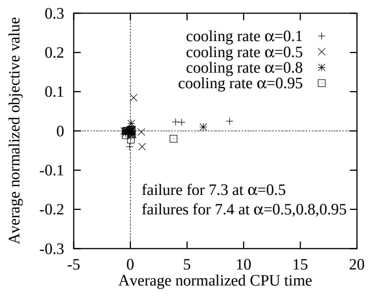
a) (Cauchy$_1$, S-uniform, M)



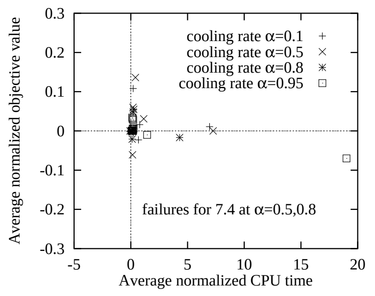
b) (Cauchy$_0$, S-uniform, M)



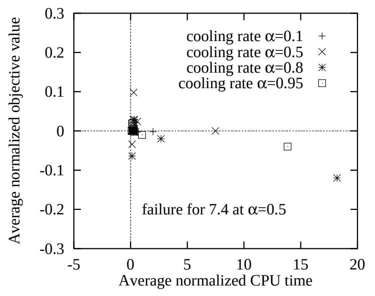
c) (Gauss, S-uniform, M)



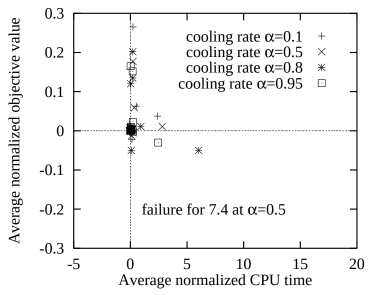
d) (Uniform, S-uniform, M)



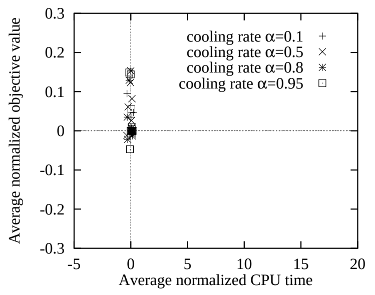
e) (Cauchy$_1$, S-uniform, L)



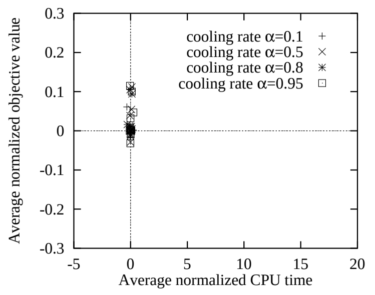
f) (Cauchy$_1$, S-uniform, H)



g) (Cauchy$_1$, S-uniform, T)

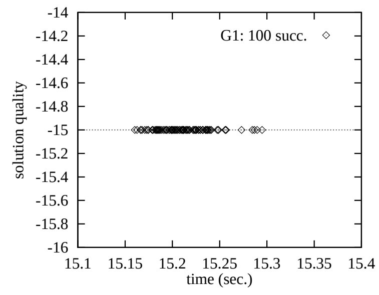


h) (Cauchy$_1$, NS-uniform, M)

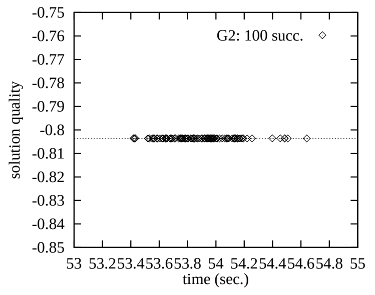


i) (Cauchy$_1$, Nonuniform,M)

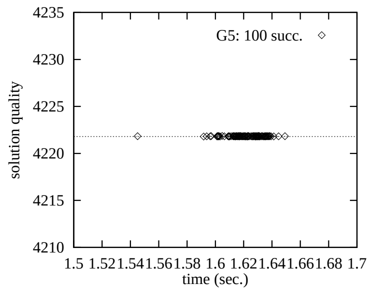
Figure 4.1: Comparisons of average relative CPU times and average relative solution qualities for different strategies normalized with respect to the baseline of (Cauchy$_1$, S-uniform, M) at various cooling rates for solving 12 difficult mixed-integer NLPs. All runs were made on a Pentium-III 500-MHz computer running Solaris 7.



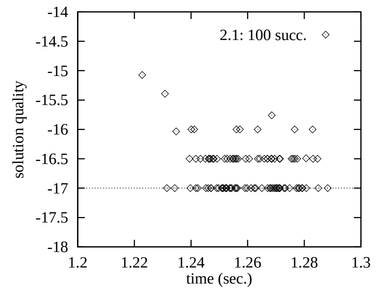
a) Problem G1



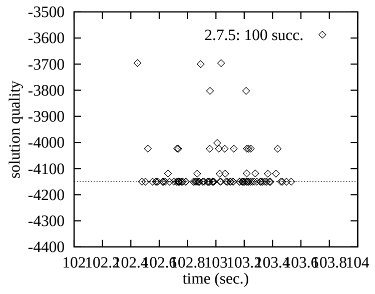
b) Problem G2



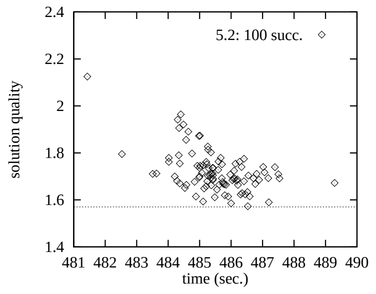
c) Problem G5



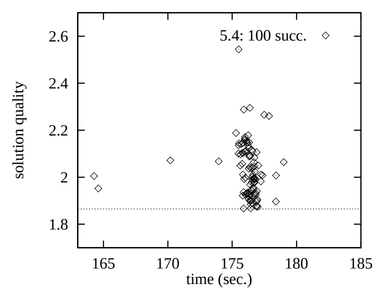
d) Problem 2.1



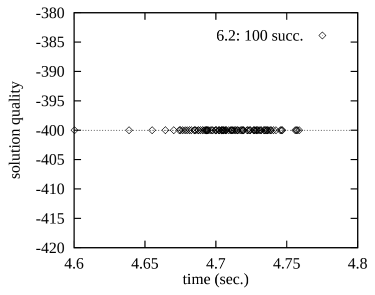
e) Problem 2.7.5



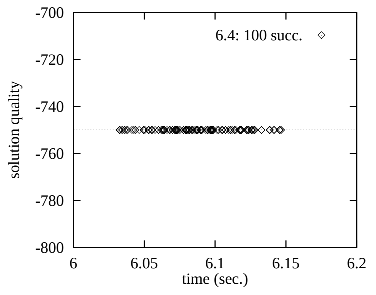
f) Problem 5.2



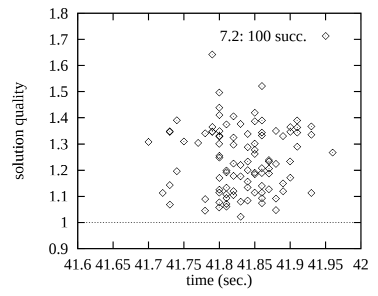
g) Problem 5.4



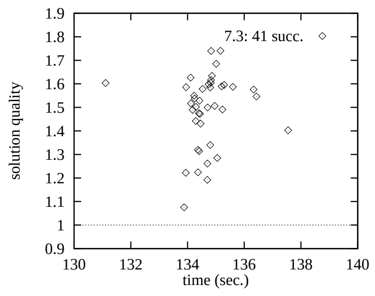
h) Problem 6.2



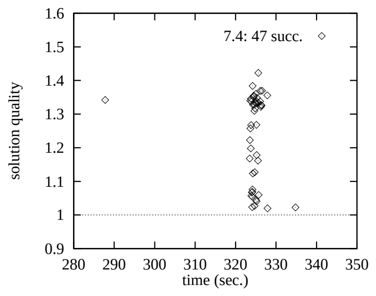
i) Problem 6.4



j) Problem 7.2

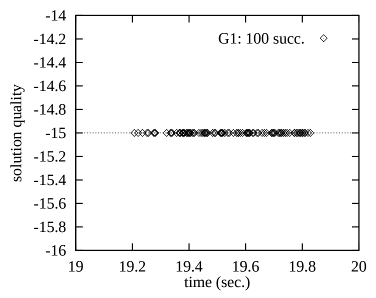


k) Problem 7.3

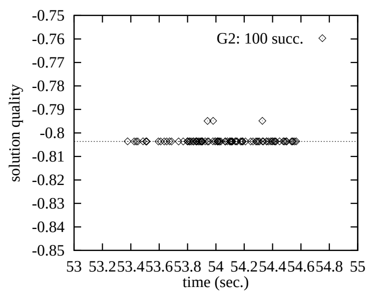


l) Problem 7.4

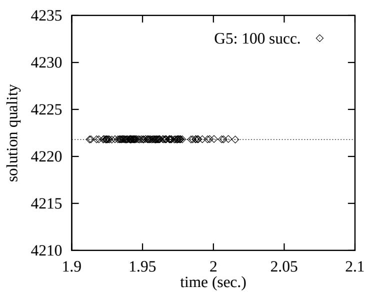
Figure 4.2: Performance of CSA based on (Cauchy$_1$, S-uniform, M) and cooling rate $\alpha = 0.8$ on 12 difficult *continuous* constrained NLPs. (The optimal solution in each problem is represented as a dotted line in the graph. All runs were made on a Pentium-III 500-MHz computer running Solaris 7. The optimal solutions in Problems 7.2-7.4 have been normalized to one.)



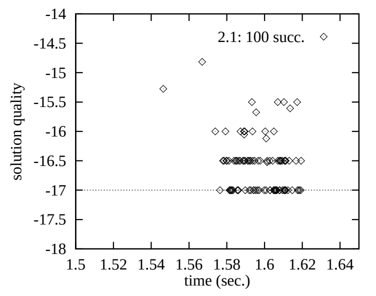
a) Problem G1



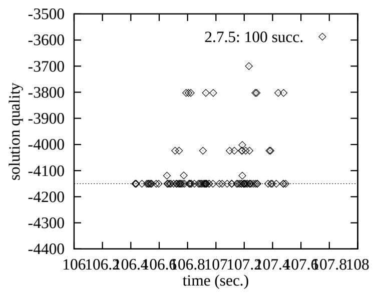
b) Problem G2



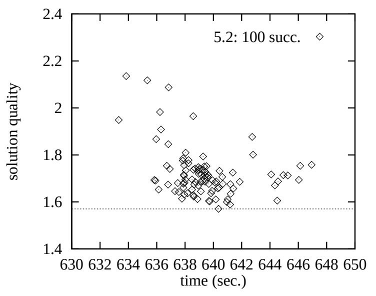
c) Problem G5



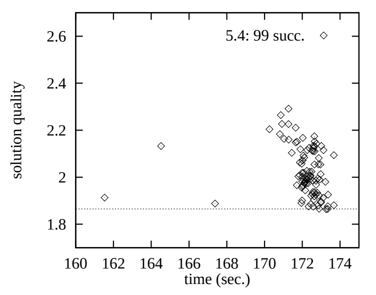
d) Problem 2.1



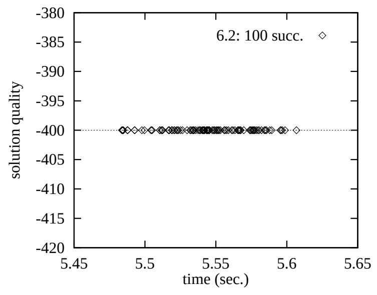
e) Problem 2.7.5



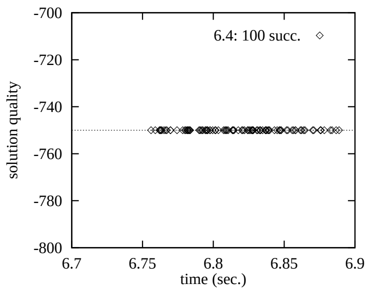
f) Problem 5.2



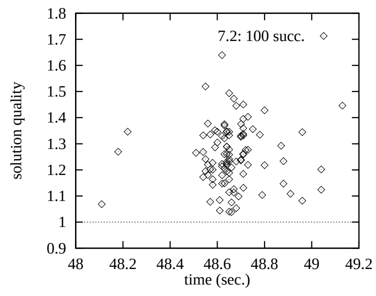
g) Problem 5.4



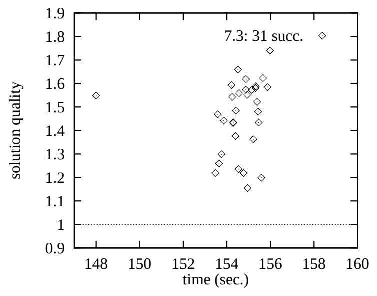
h) Problem 6.2



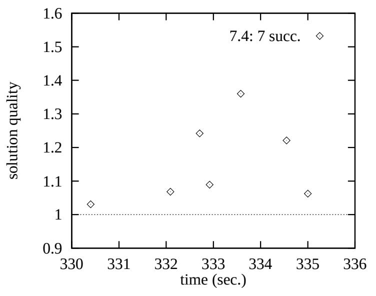
i) Problem 6.4



j) Problem 7.2

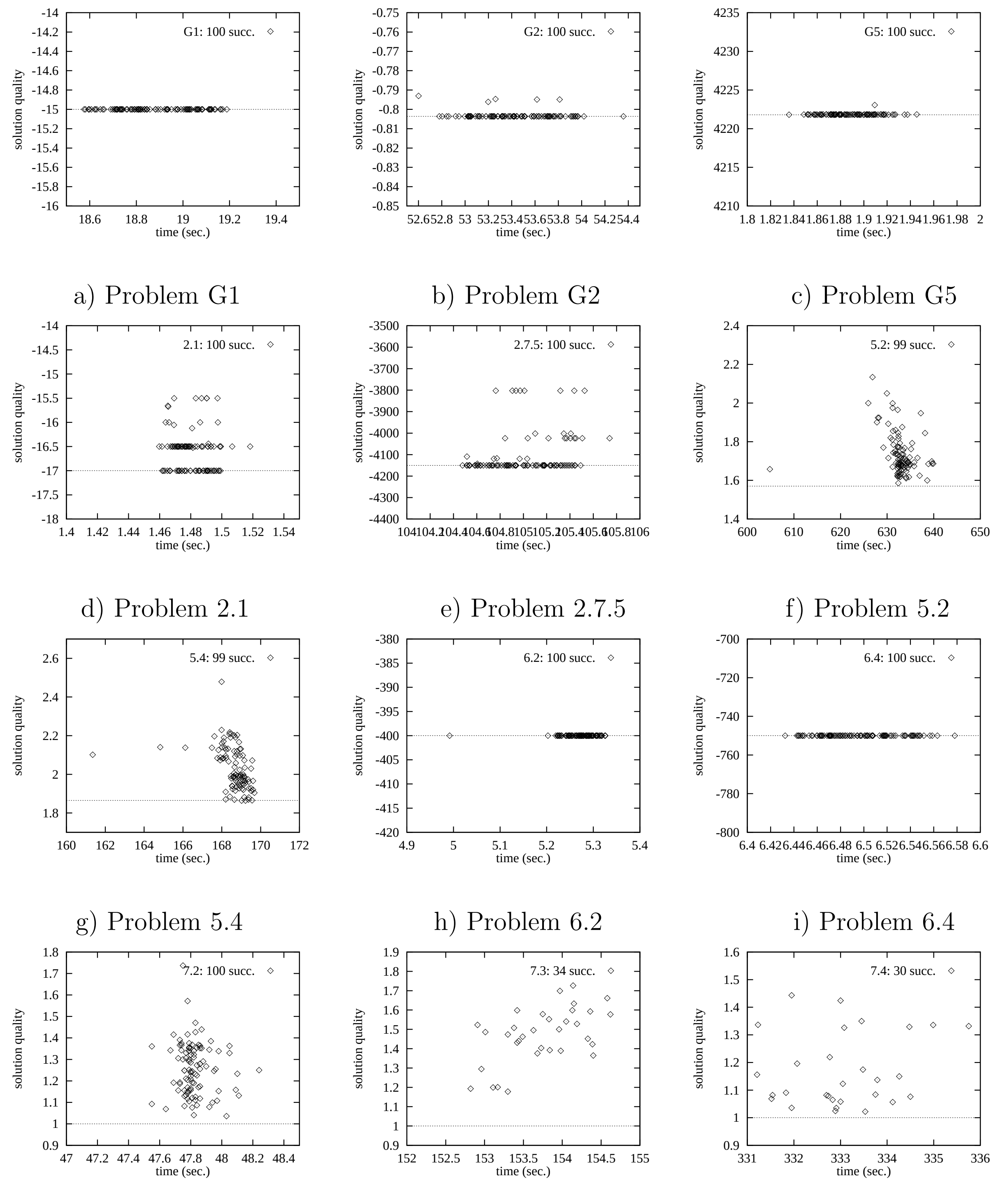


k) Problem 7.3



l) Problem 7.4

Figure 4.3: Performance of CSA based on (Cauchy$_1$, S-uniform, M) and cooling rate $\alpha = 0.8$ on 12 difficult derived *discrete* constrained NLPs. (The optimal solution in each problem is represented as a dotted line in the graph. All runs were made on a Pentium-III 500-MHz computer running Solaris 7. The optimal solutions in Problems 7.2-7.4 have been normalized to one.)



a) Problem G1 b) Problem G2 c) Problem G5

d) Problem 2.1 e) Problem 2.7.5 f) Problem 5.2

g) Problem 5.4 h) Problem 6.2 i) Problem 6.4

j) Problem 7.2 k) Problem 7.3 l) Problem 7.4

Figure 4.4: Performance of CSA based on (Cauchy$_1$, S-uniform, M) and cooling rate $\alpha = 0.8$ on 12 difficult derived *mixed-integer* constrained NLPs. (The optimal solution in each problem is represented as a dotted line in the graph. All runs were made on a Pentium-III 500-MHz computer running Solaris 7. The optimal solutions in Problems 7.2-7.4 have been normalized to one.)

on average. Unlike the Metropolis rule that always accepts better trial points, these three rules accept better trial points based on some probabilities. Among the three choices for generating trial points with λ changed, they are able to solve all the problems with similar running times, but S-uniform is the best in terms of average normalized solution quality.

In short, CSA with (Cauchy$_1$, S-uniform, M) performs the best among all the combinations of strategies tested. Accordingly, we test CSA using (Cauchy$_1$, S-uniform, M) at cooling rate 0.8 in the following experiments. Note that the performance results may differ if different cooling rates were used.

Figures 4.2 thru 4.4 show the performance of CSA using (Cauchy$_1$, S-uniform, M) at cooling rate 0.8 to solve the 12 difficult benchmark problems. In each case, we tried 100 random starting points and reported successes as the number of runs that found feasible solutions. It is clear that CSA performs consistently well in solving continuous, discrete, and mixed-integer constrained NLPs. The only exception is in solving discrete Problem 7.4 (Figure 4.3l) in which CSA has much lower success ratio than that of solving the original continuous version (Figure 4.2l). The main reason is that the size of feasible regions or the number of feasible points is greatly reduced after discretization, leading to lower success ratios in solving these problems.

4.8 An Illustration of a Search Trajectory for CSA

Figure 4.5 illustrates the run-time behavior of CSA in converging to a constrained global minimum at four different temperatures. The continuous constrained problem solved:

$$\text{minimize} \quad f(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i)) \tag{4.13}$$

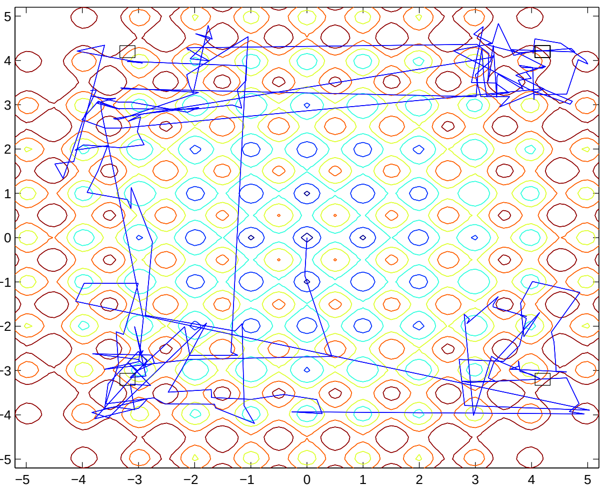
$$\text{subject to} \quad |(x_i - 4.2)(x_i + 3.2)| \leq 0.1 \qquad \text{where } n = 2,$$

has a Rastrigin function with 11^n local minima as its objective function $f(x)$, four local minima around four corners denoted by rectangles, and a global solution at $(-3.2, -3.2)$. The objective space is very rugged as shown in Figure 4.5.

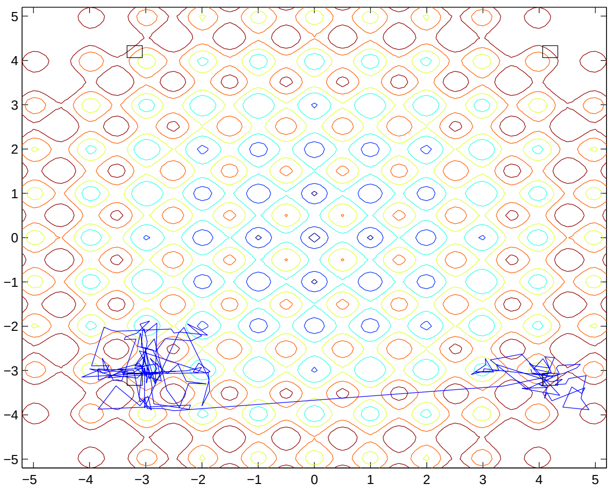
CSA starts from $x = [0, 0]$ with initial temperature $T^0 = 20$. At high temperatures (*e.g.* $T = 20$), the probability of accepting a trial point is high. Hence, the neighborhood size is large based on (4.6). Large jumps in Figure 4.5 are due to the use of Cauchy distribution for generating remote trial points, which increases the chance of getting out of infeasible local minima. As T is reduced, the acceptance probability of a trial point is reduced, leading to very small neighborhoods (*e.g.* at $T = 0.45$). Probabilistic ascents with respect to λ at different temperatures help increase penalties on constraint violation and push the search trajectory of CSA to feasible regions. Accordingly, CSA is focusing more heavily on feasible regions. Finally it converges to the constrained global minimum at $x^* = [-3.2, -3.2]$.

4.9 Summary

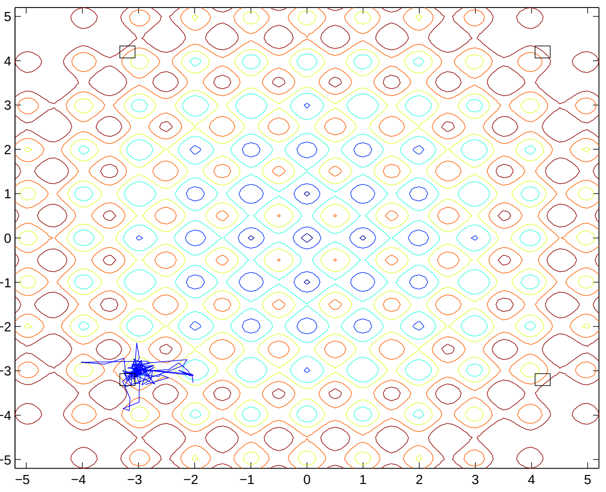
In this chapter, we have evaluated various strategies used in CSA that may affect its performance in solving discrete, continuous, and mixed-integer constrained NLPs. The strategies studied consist of adaptive neighborhoods, distributions to control sampling, acceptance probabilities, as well as cooling schedules. After extensive experiments, we found that the strategies of using Cauchy$_1$ distribution and adaptive neighborhoods in the variable subspace, symmetric-uniform distribution in the Lagrange-multiplier subspace, and Metropolis acceptance probability work the best in the CSA procedure.



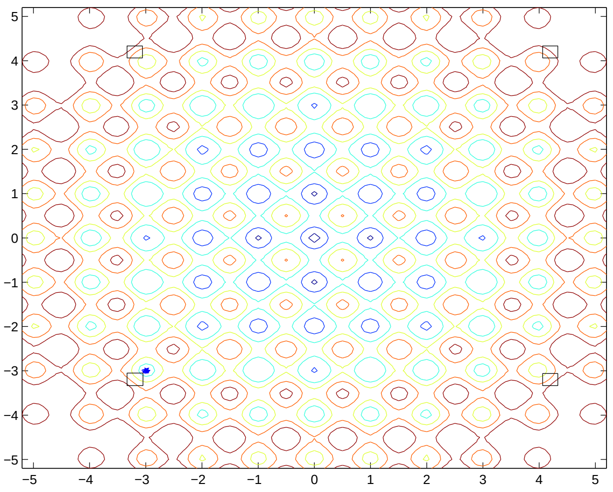
a) T = 20



b) T = 10.24



c) T = 8.192



d) T = 0.45

Figure 4.5: The run-time behavior of CSA based on (Cauchy$_1$, S-uniform,M) at different temperatures in solving (4.13).

Chapter 5

Experimental Results on Constrained NLP Benchmarks

In this chapter, we report experimental results of CSA based on (Cauchy$_1$, S-uniform, M) and cooling rate $\alpha = 0.8$ in solving three sets of continuous optimization benchmarks and their derived discrete, and mixed-integer versions. We also compare the results with those obtained by some existing search algorithms.

5.1 Continuous Benchmark Problems and Their Derived Discrete and Mixed-Integer Versions

We have chosen three sets of continuous constrained benchmarks: s$_1$) ten problems G1-G10 [133, 119], s$_2$) a collection of optimization benchmarks [68], and s$_3$) selected problems from CUTE [37], a constrained and unconstrained testing environment. These problems have objective functions of various types (linear, quadratic, cubic, polynomial, and nonlinear) and linear/nonlinear constraints of equalities and inequalities. The ratio of feasible space with respect to the whole search space varies from 0% to almost 100%, and the topologies of feasible regions are quite different.

Problems G1-G10 [133, 119] were originally developed for testing and tuning various constraint handling techniques in evolutionary algorithms (EAs). Examples of techniques developed in EAs include keeping the search within feasible regions by some specific genetic operators and dynamic and adaptive penalty methods. The number of variables in this test set is up to 20, and that of constraints, including simple bounds, is up to about 50.

The second set of benchmarks [68] were collected by Floudas and Pardalos and were derived from practical applications, mostly from chemical engineering. The number of variables in this test set is up to about 50, and that of constraints, including simple bounds, is up to about 100.

The last test problem set was selected from CUTE [37] based on the criterion that at least the objective or one of the constraints is nonlinear. Some of the CUTE problems were constructed specifically by researchers to test optimization algorithms, while others were from real applications, such as semiconductor analysis in physics, chemical reactions in chemistry, economic equilibrium in economy, and computer production planning in operations research. Both the number of variables and the number of constraints in CUTE can be as large as several thousand.

Derived discrete and mixed-integer problems. Due to a lack of large-scale discrete and mixed-integer benchmarks, we derive them from the above three sets of continuous benchmarks [133, 119, 68, 37]. In generating a constrained MINLP, we assume that variables with odd indices are continuous and those with even indices are discrete. In discretizing continuous variable x_i in range $[l_i, u_i]$, where l_i and u_i are lower and upper bounds of x_i, respectively, we force x_i to take values from the set:

$$A_i = \begin{cases} \left\{a_i + \frac{b_i - a_i}{s} j,\ j = 0, 1, \cdots, s\right\} & \text{if } b_i - a_i < 1 \\ \left\{a_i + \frac{1}{s} j,\ j = 0, 1, \cdots, \lfloor (b_i - a_i) s \rfloor\right\} & \text{if } b_i - a_i \geq 1, \end{cases} \tag{5.1}$$

where $s = 10000$. We also relax equality constraint $h_i(x) = 0$ into inequality constraint $|h_i(x)| \leq 0.001$ in order to have feasible solutions in the discretized MINLP.

5.2 Experimental Results on Discretized G1-G10 and Floudas and Pardalos' Benchmarks

In this section, we report experimental results of CSA based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$ on ten constrained NLPs G1-G10 [133, 119] and all of Floudas and Pardalos' benchmarks [68]. As a comparison, we also solved these problems using DONLP2 [179], a popular SQP package. SQP is an efficient local-search method widely used for solving continuous constrained NLPs. Its quality depends heavily on its starting points, since it is a local search.

For every problem, both DONLP2 and CSA were run 100 times with the same sequence of random starting points. For both discrete and mixed-integer problems, SQP first solves corresponding continuous problems with relaxed equality constraints (as described in previous section), and then discretizes its continuous solutions to the nearest grids defined by (5.1). We measure the performance of DONLP2 and CSA using T_r/P_r, where T_r is the average time to finish one run and P_r is the probability of finding a feasible solution with prescribed quality of solution. Therefore, T_r/P_r gives the average time to find a feasible solution with prescribed quality.

Table 5.1 compares the performance of CSA and DONLP2 on *discrete* problems G1-G10. The first two columns show the problem IDs and the best-known solutions. The next six columns list, respectively, T_r/P_r to find feasible solutions that differ within 0%, 1%, 5%, 10%, 20%, and 50% from the best-known solutions for DONLP2. The following six columns show the results for CSA. As expected, for both DONLP2 and CSA, average time T_r/P_r decreases

Table 5.1: Performance comparison of DONLP2 (SQP) and CSA in solving derived discrete constrained NLPs G1-G10. CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. '-' stands for no feasible solution found for the specified solution quality within 100 runs. Both SQP and CSA use the same sequence of starting points.

Problem	Best-known	DONLP2 for Discrete NLPs (100 runs)						CSA for Discrete NLPs (100 runs)					
ID	Solutions	0%	1%	5%	10%	20%	50%	0%	1%	5%	10%	20%	50%
G1 (min)	-15	1.13	1.13	1.03	0.364	0.163	0.124	16.8	7.62	5.98	5.14	4.43	0.69
G2 (max)	0.8036	-	-	-	-	-	-	56.4	32.0	27.6	26.4	26.2	22.6
G3 (max)	1.0	0.405	0.405	0.405	0.405	0.405	0.405	38.5	23.6	23.6	23.6	23.6	23.6
G4 (min)	-30665.5	1.17	1.17	1.17	1.17	1.17	1.17	1.83	0.517	0.25	0.17	0.16	0.16
G5 (min)	4211	-	0.105	0.105	0.105	0.105	0.0807	2.58	1.36	1.36	1.36	1.36	1.36
G6 (min)	-6961.81	-	-	-	-	-	-	0.68	0.25	0.105	0.051	0.019	0.0049
G7 (min)	24.3062	-	-	-	-	-	-	17.15	9.01	7.26	6.67	5.92	4.67
G8 (max)	0.095825	0.056	0.056	0.056	0.056	0.056	0.056	0.826	0.157	0.089	0.066	0.048	0.019
G9 (min)	680.63	0.0425	0.0425	0.0425	0.0425	0.0425	0.0425	4.65	1.03	0.39	0.13	0.026	0.0056
G10 (min)	7049.33	-	-	-	-	-	-	6.24	4.19	3.81	3.55	3.29	3.01

as the quality of solution decreases. Table 5.2 reports the results on *discrete* problems derived from Floudas and and Pardalos' continuous benchmarks [68]. Both DONLP2 and CSA fail to find a feasible solution to discrete problem 7.4. This may be due to the reason that discrete problem 7.4 does not have any feasible solutions.

Table 5.3 and Table 5.4 show the comparison results of DONLP2 and CSA on solving derived MINLPs.

Obviously, CSA performs much better than DONLP2 in solving both *discrete* and *mixed-integer* constrained NLPs. First, CSA is able to find best-known solutions to all the discrete and mixed-integer problems (except discrete 7.4), whereas SQP even fails to find feasible solutions to many problems, such as G2, G6, 5.2 and 5.4. This means that discretization of continuous solutions found by SQP sometimes may not lead to feasible discrete and mixed-integer solutions. Besides, SQP finds very poor feasible solutions for some problems such

Table 5.2: Performance comparison of DONLP2 (SQP) and CSA in solving derived discrete constrained NLPs from Floudas and Pardalos' continuous constrained benchmarks [68]. CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. '-' stands for no feasible solution found for the specified solution quality within 100 runs. Both SQP and CSA use the same sequence of starting points.

Problem	Best-known	DONLP2 for Discrete NLPs (100 runs)						CSA for Discrete NLPs (100 runs)					
ID	Solutions	0%	1%	5%	10%	20%	50%	0%	1%	5%	10%	20%	50%
2.1 (min)	-17	0.85	0.85	0.213	0.10	0.0708	0.0293	2.23	1.68	0.613	0.484	0.095	0.014
2.2 (min)	-213	0.022	0.022	0.022	0.022	0.022	0.022	1.57	0.368	0.046	0.0073	0.0029	0.0021
2.3 (min)	-15	1.13	1.13	1.03	0.364	0.163	0.124	16.8	7.62	5.98	5.14	4.43	0.69
2.4 (min)	-11	0.134	0.103	0.0848	0.0848	0.0629	0.0629	2.63	1.34	0.98	0.631	0.172	0.0076
2.5 (min)	-268	-	-	-	-	-	-	20.4	5.53	3.76	3.24	1.51	0.0927
2.6 (min)	-39	9.50	2.38	2.38	1.36	1.36	0.50	7.28	3.73	2.78	2.34	1.97	1.35
2.7.1 (min)	-394.75	-	-	-	14.2	14.2	14.2	80.5	39.1	30.7	27.0	21.6	18.2
2.7.2 (min)	-884.75	-	-	-	-	74.4	74.4	85.9	42.6	35.4	21.9	18.8	7.02
2.7.3 (min)	-8695.0	-	-	-	-	-	-	78.0	37.3	30.2	26.2	19.2	15.3
2.7.4 (min)	-754.75	-	-	-	-	71.6	71.6	76.5	34.0	26.7	21.6	19.2	6.03
2.7.5 (min)	-4150.4	-	-	-	-	-	-	125.6	79.2	40.3	30.0	24.2	20.4
2.8 (min)	15639.0	-	-	-	-	-	-	62.5	27.8	27.8	27.7	27.7	27.7
3.1 (min)	7049.33	-	-	-	-	-	-	6.24	4.19	3.81	3.55	3.29	3.01
3.2 (min)	-30665.5	1.17	1.17	1.17	1.17	1.17	1.17	1.83	0.517	0.25	0.17	0.16	0.16
3.3 (min)	-310.0	-	-	-	-	-	-	2.05	0.48	0.46	0.46	0.46	0.46
3.4 (min)	-4.0	0.091	0.084	0.084	0.084	0.084	0.084	0.87	0.28	0.042	0.0075	0.002	0.0017
4.3 (min)	-4.51	0.18	0.18	0.18	0.18	0.15	0.123	1.29	0.53	0.48	0.47	0.47	0.47
4.4 (min)	-2.217	0.52	0.371	0.217	0.20	0.186	0.173	1.25	0.44	0.31	0.30	0.30	0.30
4.5 (min)	-13.40	-	-	-	-	3.10	3.10	3.61	1.92	1.84	1.84	1.84	1.84
4.6 (min)	-5.51	-	-	-	-	-	0.0429	0.603	0.0876	0.0081	0.0029	0.0021	0.0015
4.7 (min)	-16.74	0.0706	0.0461	0.0461	0.0461	0.0461	0.044	0.453	0.136	0.0735	0.0575	0.0388	0.0315
5.2 (min)	1.60	-	-	-	-	-	-	10360	5010	1358	510.0	472.9	455.7
5.4 (min)	1.86	-	-	-	-	-	-	5475	4140	660.0	305.7	202.7	198.6
6.2 (max)	400.0	-	-	-	-	-	-	4.91	3.07	3.02	3.02	3.02	3.02
6.3 (max)	600.0	-	-	-	-	-	-	4.76	3.65	3.45	3.40	3.37	3.37
6.4 (max)	750.0	-	-	-	-	-	-	5.28	4.31	4.31	4.31	4.31	4.31
7.2 (min)	1.05	-	-	-	-	-	-	3980	1327	137.5	101.9	51.2	26.9
7.3 (min)	1.51	-	-	-	-	-	-	13700	13650	6801	6801	6801	6801
7.4 (min)	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 5.3: Performance comparison of DONLP2 (SQP) and CSA in solving derived mixed-integer constrained NLPs G1-G10. CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. '-' stands for no feasible solution found for the specified solution quality within 100 runs. Both SQP and CSA use the same sequence of starting points.

Problem	Best-known	DONLP2 for Mixed-Integer NLPs (100 runs)						CSA for Mixed-Integer NLPs (100 runs)					
ID	Solutions	0%	1%	5%	10%	20%	50%	0%	1%	5%	10%	20%	50%
G1 (min)	-15	3.10	3.10	2.48	0.775	0.365	0.276	18.0	7.18	5.61	4.97	3.84	0.62
G2 (max)	0.8036	-	-	-	-	-	-	56.5	31.5	27.2	26.2	25.6	22.5
G3 (max)	1.0	0.405	0.405	0.405	0.405	0.405	0.405	37.0	22.7	22.7	22.7	22.7	22.7
G4 (min)	-30665.5	1.17	1.17	1.17	1.17	1.17	1.17	2.17	0.52	0.23	0.15	0.14	0.14
G5 (min)	4211	0.35	0.0309	0.0309	0.0309	0.0309	0.0309	1.65	0.69	0.66	0.66	0.66	0.66
G6 (min)	-6961.81	-	-	-	-	-	-	0.65	0.23	0.10	0.045	0.017	0.0046
G7 (min)	24.3062	-	-	-	-	-	-	16.8	8.64	7.25	6.56	5.72	4.47
G8 (max)	0.095825	0.056	0.056	0.056	0.056	0.056	0.056	0.79	0.15	0.085	0.060	0.038	0.018
G9 (min)	680.63	-	-	-	-	-	-	4.51	0.96	0.35	0.11	0.027	0.0053
G10 (min)	7049.33	-	-	-	-	-	-	5.82	5.80	3.41	3.25	2.97	2.69

as mixed-integer 2.7.2, where DONLP2 can only find feasible solutions within 20% to 10% worse than the best-known solutions.

Second, for those problems in which SQP and CSA are able to find feasible solutions, the average CPU time of SQP is generally shorter than that of CSA, because each run of SQP is much quicker than CSA although SQP has lower success ratio P_r than CSA.

Third, CSA spends more than 50% CPU time in improving the last 1% of solution quality. For example, in discrete problem G1, the average time of finding the best-known solution is 16.8 seconds, but the average time of finding feasible solutions within 1% worse than the best-known solution is 7.62 seconds. The reason is that CSA is sampling-based and it takes a long time to exactly hit the best-known solution.

Table 5.5 compares the performance of CSA on solving *continuous* problems G1-G10. The first two columns give the problem IDs and the best-known solutions. The next two columns

Table 5.4: Performance comparison of DONLP2 (SQP) and CSA in solving derived mixed-integer constrained NLPs from Floudas and Pardalos' continuous constrained benchmarks [68]. CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. '-' stands for no feasible solution found for the specified solution quality within 100 runs. Both SQP and CSA use the same sequence of starting points.

Problem	Best-known	DONLP2 for Mixed-Integer NLPs (100 runs)						CSA for Mixed-Integer NLPs (100 runs)					
ID	Solutions	0%	1%	5%	10%	20%	50%	0%	1%	5%	10%	20%	50%
2.1 (min)	-17	0.85	0.85	0.213	0.10	0.0708	0.0293	3.14	2.07	0.80	0.16	0.089	0.012
2.2 (min)	-213	0.022	0.022	0.022	0.022	0.022	0.022	1.84	0.38	0.045	0.0066	0.0031	0.0027
2.3 (min)	-15	3.10	3.10	2.48	0.775	0.365	0.276	18.0	7.18	5.61	4.97	3.84	0.62
2.4 (min)	-11	0.134	0.103	0.0848	0.0848	0.083	0.083	2.99	1.14	0.89	0.60	0.15	0.0066
2.5 (min)	-268	-	-	-	-	-	-	22.1	5.36	3.59	2.92	1.42	0.097
2.6 (min)	-39	9.50	2.38	2.38	1.36	1.19	0.288	10.5	3.52	2.56	2.19	1.84	1.36
2.7.1 (min)	-394.75	-	-	-	14.2	11.7	10.1	92.8	36.2	29.1	24.6	20.9	17.8
2.7.2 (min)	-884.75	-	-	-	-	37.2	24.8	99.1	40.2	32.8	22.1	18.4	6.80
2.7.3 (min)	-8695.0	-	-	-	-	-	-	92.4	34.6	28.1	23.9	18.9	14.7
2.7.4 (min)	-754.75	-	-	-	-	71.6	71.6	91.3	33.2	26.2	21.1	18.9	5.72
2.7.5 (min)	-4150.4	-	-	9.66	9.66	9.66	9.66	122.9	54.9	35.3	29.1	23.4	19.9
2.8 (min)	15639.0	-	-	-	-	-	-	82.8	36.6	36.6	26.7	26.7	26.7
3.1 (min)	7049.33	-	-	-	-	-	-	5.82	5.80	3.41	3.25	2.97	2.69
3.2 (min)	-30665.5	1.17	1.17	1.17	1.17	1.17	1.17	2.17	0.52	0.23	0.15	0.14	0.14
3.3 (min)	-310.0	1.30	1.30	0.229	0.139	0.0812	0.0494	2.80	0.45	0.43	0.43	0.42	0.42
3.4 (min)	-4.0	0.091	0.084	0.084	0.084	0.075	0.075	0.92	0.27	0.045	0.0074	0.002	0.0017
4.3 (min)	-4.51	0.18	0.18	0.18	0.18	0.15	0.123	1.73	0.49	0.44	0.44	0.44	0.44
4.4 (min)	-2.217	0.52	0.371	0.289	0.26	0.236	0.216	1.73	0.367	0.29	0.28	0.28	0.28
4.5 (min)	-13.40	-	-	-	-	3.10	3.10	4.16	1.80	1.75	1.75	1.75	1.75
4.6 (min)	-5.51	-	-	-	-	-	0.0429	0.63	0.077	0.0073	0.0029	0.0019	0.0014
4.7 (min)	-16.74	0.0667	0.0545	0.0545	0.0545	0.0545	0.0522	0.50	0.12	0.072	0.054	0.043	0.031
5.2 (min)	1.60	-	-	-	-	-	-	6063	3915	1391	476.6	366.7	329.0
5.4 (min)	1.86	-	-	-	-	-	-	7733	1521	618.7	210.4	132.9	129.2
6.2 (max)	400.0	-	-	-	-	-	-	5.44	2.85	2.84	2.84	2.84	2.84
6.3 (max)	600.0	-	-	-	-	-	-	6.15	3.65	3.18	3.16	3.16	3.16
6.4 (max)	750.0	-	-	-	-	-	-	6.05	4.08	4.08	4.08	4.08	4.08
7.2 (min)	1.04	-	-	-	-	-	-	4390	2190	621.4	180.0	60.4	24.8
7.3 (min)	1.09	-	-	-	-	-	-	13700	6850	6830	4307	1821	270.9
7.4 (min)	1.08	-	-	-	-	-	-	35900	35802	16700	8075	8053	4619

Table 5.5: Performance comparison of EAs, DONLP2 (SQP) and CSA in solving continuous constrained NLPs G1-G10. (S.T. stands for strategic oscillation, H.M. for homomorphous mappings, and D.P. for dynamic penalty). CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. Both SQP and CSA use the same sequence of starting points.

Problem	Best-known	EAs		DONLP2 for Continuous NLPs (100 runs)						CSA for Continuous NLPs (100 runs)					
ID	Solutions	best sol'n	method	0%	1%	5%	10%	20%	50%	0%	1%	5%	10%	20%	50%
G1 (min)	-15	-15	Genocop	1.13	1.13	1.03	0.365	0.163	0.124	15.2	6.36	4.88	4.20	3.68	0.58
G2 (max)	0.8036	0.80355	S.T.	-	-	-	-	-	33.2	53.8	31.3	26.4	24.6	23.6	21.7
G3 (max)	1.0	0.999866	S.T.	0.368	0.368	0.368	0.368	0.368	0.368	37.2	21.7	21.6	21.6	21.6	21.6
G4 (min)	-30665.5	-30664.5	H.M.	0.033	0.033	0.033	0.033	0.033	0.033	1.76	0.43	0.28	0.23	0.23	0.23
G5 (min)	4221.9	5126.498	D.P.	0.024	0.024	0.024	0.024	0.024	0.024	1.62	1.01	1.01	1.01	1.01	1.10
G6 (min)	-6961.81	-6961.81	Genocop	0.015	0.015	0.015	0.015	0.015	0.015	0.518	0.21	0.089	0.041	0.014	0.0048
G7 (min)	24.3062	24.62	H.M.	0.047	0.047	0.047	0.047	0.047	0.047	14.0	7.55	6.16	5.36	4.80	3.66
G8 (max)	0.095825	0.095825	H.M.	0.056	0.056	0.056	0.056	0.056	0.056	0.798	0.13	0.075	0.058	0.038	0.016
G9 (min)	680.63	680.64	Genocop	0.031	0.031	0.031	0.031	0.031	0.031	4.48	0.86	0.34	0.12	0.022	0.0048
G10 (min)	7049.33	7147.9	H.M.	2.32	2.32	2.32	2.32	2.32	2.32	4.69	3.60	3.23	3.08	2.87	2.52

show the best solutions obtained by EAs with specific constraint-handling techniques. The remaining columns are the results of DONLP2 and CSA.

Table 5.6 reports the results on Floudas and Pardalos' *continuous* benchmarks [68]. The third column shows the best solutions obtained by one implementation of interval methods, called Epperly's method [64], whereas the other columns have the same meaning as those in Table 5.5.

Our experimental results on continuous NLPs lead to the following observations.

First, EAs with various constraint handling techniques do not work well, even for simple problems like G7 where a local-search method like DONLP2 can find the optimal solution easily. The main reason is that these constraint handling techniques do not look for saddle points. Hence, they do not guarantee constraint satisfaction and have difficulty in finding a CGM. Another reason may be attributed to the difficulty of sampling methods in finding

Table 5.6: Performance comparison of Epperly's method [64] (an interval method), DONLP2 (SQP) and CSA in solving Floudas and Pardalos' continuous constrained NLPs [68]. CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. '-' stands for no feasible solution found for Epperly's method. Both SQP and CSA use the same sequence of starting points.

Problem	Best-known	Epperly	DONLP2 for Continuous NLPs (100 runs)						CSA for Continuous NLPs (100 runs)					
ID	Solutions	best sol'n	0%	1%	5%	10%	20%	50%	0%	1%	5%	10%	20%	50%
2.1 (min)	-17	-17	0.85	0.85	0.213	0.10	0.0708	0.0293	2.68	1.62	0.506	0.18	0.071	0.011
2.2 (min)	-213	-213	0.022	0.022	0.022	0.022	0.022	0.022	1.82	0.35	0.041	0.0066	0.003	0.0026
2.3 (min)	-15	-15	1.13	1.13	1.03	0.365	0.163	0.124	15.2	6.36	4.88	4.20	3.68	0.58
2.4 (min)	-11	-11	0.144	0.103	0.0848	0.0848	0.039	0.039	2.75	1.23	0.86	0.56	0.14	0.0074
2.5 (min)	-268	-268	0.085	0.085	0.085	0.085	0.085	0.085	23.1	4.95	3.35	2.76	1.34	0.087
2.6 (min)	-39	-39	2.38	2.38	2.38	1.36	1.15	0.183	10.8	3.2	2.31	1.99	1.69	1.18
2.7.1 (min)	-394.75	-394.75	5.40	5.40	5.40	5.40	2.70	0.975	79.5	34.4	27.7	22.7	19.4	16.5
2.7.2 (min)	-884.75	-884.75	4.96	4.96	4.96	2.66	1.16	0.744	98.2	30.3	24.3	19.1	16.58	6.13
2.7.3 (min)	-8695.0	-8695.0	3.69	3.69	3.69	3.69	1.66	0.707	100.7	33.8	27.9	23.1	16.9	13.2
2.7.4 (min)	-754.75	-754.75	4.48	4.48	4.48	2.39	1.21	0.716	98.1	30.9	24.3	20.0	17.8	5.55
2.7.5 (min)	-4150.4	-4150.4	6.04	2.20	1.07	0.833	0.644	0.514	138.5	57.8	34.9	28.9	21.8	18.5
2.8 (min)	15639.0	15990.0	6.86	6.86	3.08	3.08	2.85	1.98	77.2	53.8	53.8	53.8	53.8	53.8
3.1 (min)	7049.33	-	2.32	2.32	2.32	2.32	2.32	2.32	4.69	3.60	3.23	3.08	2.87	2.52
3.2 (min)	-30665.5	-30665.5	0.033	0.033	0.033	0.033	0.033	0.033	1.76	0.43	0.28	0.23	0.23	0.23
3.3 (min)	-310.0	-310.0	1.30	1.30	0.229	0.126	0.075	0.0488	3.04	0.41	0.40	0.40	0.39	0.36
3.4 (min)	-4.0	-4.0	0.0875	0.0875	0.0875	0.0875	0.078	0.078	0.86	0.24	0.039	0.0063	0.0028	0.0022
4.3 (min)	-4.51	-4.51	0.0844	0.0844	0.0844	0.0844	0.0844	0.0818	1.51	0.65	0.62	0.62	0.62	0.62
4.4 (min)	-2.217	-2.217	0.186	0.186	0.118	0.118	0.118	0.118	1.83	0.643	0.535	0.515	0.51	0.50
4.5 (min)	-13.40	-13.40	1.55	1.55	1.55	1.03	1.03	1.03	3.63	2.62	2.61	2.61	2.60	2.60
4.6 (min)	-5.51	-5.51	0.043	0.043	0.043	0.043	0.029	0.015	0.538	0.068	0.0071	0.0033	0.0028	0.0022
4.7 (min)	-16.74	-16.74	0.012	0.012	0.012	0.012	0.012	0.012	0.481	0.21	0.203	0.203	0.203	0.203
5.2 (min)	1.567	-	457.5	457.5	166.3	114.4	107.6	107.6	48510	24548	2140	656.8	464.5	390.5
5.4 (min)	1.86	-	8.50	8.50	8.50	8.50	8.39	8.39	8805	3360	615.2	255.9	159.5	159.5
6.2 (max)	400.0	400.0	0.064	0.060	0.059	0.059	0.059	0.059	4.71	3.77	3.77	3.75	3.75	3.75
6.3 (max)	600.0	600.0	0.062	0.062	0.061	0.061	0.061	0.061	5.48	3.90	3.88	3.81	3.81	3.81
6.4 (max)	750.0	750.0	0.073	0.072	0.072	0.072	0.072	0.072	5.95	4.47	4.47	4.47	4.47	4.47
7.2 (min)	1.0	-	4.71	4.71	0.569	0.569	0.569	0.569	-	-	2251	492.2	106.7	36.8
7.3 (min)	1.0	-	15.8	6.14	6.14	3.88	2.95	2.95	-	-	-	13540	6740	564.8
7.4 (min)	1.0	-	31.4	28.3	10.9	10.9	10.9	10.9	-	-	6438	2881	1835	651.1

exact solutions to continuous NLPs. EAs were only able to find the best solutions in three of the ten NLPs in G1-G10 despite extensive tuning.

Second, DONLP2 generally works better than CSA because SQP fully utilizes the derivative information in solving continuous constrained NLPs, whereas CSA depends only on sampling. Hence, each run of SQP is usually much shorter (one- to two-order faster) than CSA although SQP has lower success ratio P_r than CSA. In this situation, CSA can win over SQP only for those problems with huge numbers of local minima, such as G2. In 100 runs of DONLP2 to solve G2, only one run was able to find the best solution of 0.59701, which is much worse than those obtained by CSA (0.80362). Even with 10,000 runs, DONLP2 was only able to find the best solution of 0.736554.

The limitation of DONLP2 is that it requires the differentiability of the Lagrangian function; hence, it will not be able to solve NLPs whose derivatives are hard to calculate or are unavailable (such as discrete and mixed-integer NLPs). However, we must point out that CSA is generally not competitive with SQP in terms of execution time in solving continuous constrained NLPs with differentiable objective and constraint functions. Closed-form derivatives in these problems are very effective in SQP for finding feasible solutions.

Third, interval methods, such as Epperly's implementation [64], have difficulties in solving problems with nonlinear constraints whose lower bounds are difficult to determine. Examples include Problems 5.2, 5.4, 7.2, 7.3 and 7.4 in which feasible points were not found.

Last, our current CSA implementation is weak in solving problems with a large number of equality constraints, such as Problems 5.2, 5.4, 7.2, 7.3 and 7.4. Since CSA is sampling based, it has difficulty or takes a long time to exactly hit points that satisfy a lot of equality constraints.

5.3 Experimental Results on CUTE Benchmarks

Table 5.7 and Table 5.8, respectively, report comparison results in solving *discrete* and *mixed-integer* NLPs derived from some selected CUTE benchmarks [37] using the given starting point in each problem. The first column shows the problem IDs, and the next two columns give the number ($n_v = n$) of variables and the number ($n_c = m + k$) of constraints. The next five columns show the type of the objective function (linear, quadratic, or nonlinear), the number of linear equality constraints (n_{le}), the number of nonlinear equality constraints (n_{ne}), the number of linear inequality constraints (n_{li}), and the number of nonlinear inequality constraints (n_{ni}). The next six columns show the solutions and CPU times that we obtain by using LANCELOT [48, 121], DONLP2 [179], and CSA, respectively. For discrete and mixed-integer problems, both DONLP2 and LANCELOT first solve the corresponding continuous problems with relaxed equality constraints (see section 5.1), and then discretize their continuous solutions to the nearest discrete points defined by (5.1).

CSA is much better than two SQP packages, LANCELOT and DONLP2, in terms of its ability to solve discrete and mixed-integer NLPs. Both LANCELOT and DONLP2 are unable to find feasible solutions for most of the problems tested.

Table 5.9 shows comparison results in solving a large set of selected CUTE benchmarks [37] (with continuous variables) using the given starting point in each problem. Here, DONLP2 is of version 10/21/1997, and for LANCELOT, we use both the public version (01/05/2000) and the commercial version (by submitting problems through the Internet, http://www-neos.mcs.anl.gov/neos/solvers/NCO:LANCELOT/).

The running times of CSA are not competitive with those of LANCELOT and DONLP2 for solving continuous constrained NLPs, because CSA is sample-based whereas the latter two use information on derivatives. CSA, however, is much better in terms of solution quality.

Both LANCELOT and DONLP2 could not find feasible solutions for some problems, but CSA was able to solve all the problems and obtained better solutions than LANCELOT and DONLP2 in many problems. For example, CSA found a solution of 0.872 for Problem CRESC4, whereas both LANCELOT and DONLP2 could not find feasible solutions. For Problem HS20, LANCELOT found a solution of 40.2, DONLP2 found a better solution of 39.17, and CSA found the best solution of 38.19.

Table 5.11 shows the results of applying LANCELOT to solve the CUTE problems that cannot be solved by CSA at this time. These problems either are too large for CSA or have nonlinear constraints that make it very difficult for the sample-based CSA procedure to find feasible points.

Figure 5.1 depicts normalized solution quality and normalized CPU time of CSA with respect to LANCELOT for those CUTE benchmarks that are solvable by both CSA and LANCELOT. The smaller the value, the shorter the CPU time or the better solution of quality. The CPU time of CSA is always longer than that of LANCELOT, but the overall solution quality of CSA is better.

Table 5.7: Comparison results of LANCELOT, DONLP2, and CSA in solving discrete constrained NLPs that are derived from selected continuous problems from CUTE using the starting point specified in each problem. CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. $'-'$ means that no feasible solution can be found by both the public version (01/05/2000) and the commercial version of LANCELOT (by submitting problems through the Internet, http://www-neos.mcs.anl.gov/neos/solvers/NCO:LANCELOT/), and that no feasible solution can be found by DONLP2. Numbers in bold represent the best solutions among the three methods if they have different solutions.

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		DONLP2		CSA	
IDs				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
HS62	3	1	nonlinear	1	0	0	0	**-26273**	0.61	-25698	0.03	**-26273**	2.43
HS87	6	4	nonlinear	0	4	0	0	-	-	-	-	**8926**	7.32
HS99	7	2	nonlinear	0	2	0	0	-	-	-	-	$\mathbf{-8.31 \times 10^8}$	11.8
HS101	7	5	nonlinear	0	0	0	5	-	-	-	-	**1809.7**	69.3
HS102	7	5	nonlinear	0	0	0	5	-	-	-	-	**911.9**	73.7
HS103	7	5	nonlinear	0	0	0	5	-	-	**543.7**	0.13	**543.7**	67.8
HS104	8	5	nonlinear	0	0	0	5	-	-	**3.95**	0.03	**3.95**	30.1
NET1	48	57	nonlinear	21	17	16	3	-	-	-	-	-	-
NET2	144	160	nonlinear	64	59	32	5	-	-	-	-	$\mathbf{1.187 \times 10^6}$	209559

Table 5.8: Comparison results of LANCELOT, DONLP2, and CSA in solving mixed-integer constrained NLPs that are derived from selected continuous problems from CUTE using the starting point specified in each problem. CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. $'-'$ means that no feasible solution can be found by both the public version (01/05/2000) and the commercial version of LANCELOT (by submitting problems through the Internet, http://www-neos.mcs.anl.gov/neos/solvers/NCO:LANCELOT/), and that no feasible solution can be found by DONLP2. Numbers in bold represent the best solutions among the three methods if they have different solutions.

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		DONLP2		CSA	
IDs				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
HS62	3	1	nonlinear	1	0	0	0	**-26273**	0.61	-25698	0.03	**-26273**	2.28
HS87	6	4	nonlinear	0	4	0	0	-	-	-	-	**8926**	8.56
HS99	7	2	nonlinear	0	2	0	0	-	-	-	-	$\mathbf{-8.31 \times 10^8}$	11.8
HS101	7	5	nonlinear	0	0	0	5	-	-	-	-	**1809.7**	74.9
HS102	7	5	nonlinear	0	0	0	5	-	-	-	-	**911.9**	75.1
HS103	7	5	nonlinear	0	0	0	5	-	-	**543.7**	0.13	**543.7**	75.2
HS104	8	5	nonlinear	0	0	0	5	-	-	**3.95**	0.03	**3.95**	30.6
NET1	48	57	nonlinear	21	17	16	3	-	-	-	-	$\mathbf{9.41 \times 10^5}$	6349
NET2	144	160	nonlinear	64	59	32	5	-	-	-	-	$\mathbf{1.187 \times 10^6}$	196847

Table 5.9: Comparison results of LANCELOT, DONLP2, and CSA in solving selected continuous problems from CUTE using the starting point specified in each problem. CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$, and does not use derivative information in each run. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. $'-'$ means that no feasible solution can be found by both the public version (01/05/2000) and the commercial version of LANCELOT (by submitting problems through the Internet, http://www-neos.mcs.anl.gov/neos/solvers/NCO:LANCELOT/), and that no feasible solution can be found by DONLP2. $'*'$ means that solutions are obtained by the commercial version (no CPU time is available) but cannot be solved by the public version. Numbers in bold represent the best solutions among the three methods if they have different solutions.

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		DONLP2		CSA	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
ALJAZZAF	3	1	quadratic	0	1	0	0	**75.0**	0.46	-	-	**75.0**	1.38
ALLINITC	4	1	nonlinear	0	0	0	1	**30.44**	*	31.75	0.02	**30.44**	4.76
ALSOTAME	2	1	nonlinear	0	1	0	0	0.082	0.57	0.082	0.03	0.082	0.80
AVION2	49	15	nonlinear	15	0	0	0	-	-	-	-	$\mathbf{9.47 \times 10^7}$	946.7
BATCH	46	73	nonlinear	12	0	60	1	-	-	-	-	$\mathbf{2.59 \times 10^5}$	12465
BT11	5	3	nonlinear	1	2	0	0	0.825	0.62	0.825	0.02	0.825	3.73
BT12	5	3	quadratic	0	3	0	0	6.188	0.47	6.188	0.02	6.188	3.25
BT6	5	2	nonlinear	0	2	0	0	0.277	0.56	0.277	0.03	0.277	4.61
BT7	5	3	nonlinear	0	3	0	0	**306.5**	0.51	360.4	0.03	**306.5**	3.38
BT8	5	2	quadratic	0	2	0	0	1.0	0.57	1.0	0.02	1.0	2.93
CB2	3	3	linear	0	0	0	3	1.952	0.60	1.952	0.03	1.952	2.49
CRESC4	6	8	nonlinear	0	0	0	8	-	-	-	-	**0.872**	37.9
CSFI1	5	4	linear	0	2	0	2	**-49.07**	0.63	0.0	0.02	**-49.07**	4.56
DEMBO7	16	20	quadratic	0	0	0	20	**174.9**	*	-	-	**174.9**	342.9
DIPIGRI	7	4	nonlinear	0	0	0	4	680.6	0.68	680.6	0.03	680.6	9.19
DIXCHLNG	10	5	nonlinear	0	5	0	0	**0.0**	1.12	2471.9	0.04	**0.0**	44.12
DNIEPER	61	24	nonlinear	0	24	0	0	$\mathbf{1.87 \times 10^4}$	0.83	-	-	$\mathbf{1.87 \times 10^4}$	3703
ERRINBAR	18	9	linear	0	8	1	0	**28.05**	*	-	-	**28.05**	83.3
EXPFITA	5	22	nonlinear	0	0	22	0	1.13×10^{-3}	0.65	1.13×10^{-3}	0.07	1.13×10^{-3}	89.44
FLETCHER	4	4	quadratic	0	1	3	0	19.53	0.57	-	-	**12.18**	3.46
GAUSSELM	14	11	linear	0	5	6	0	**-2.25**	0.55	-	-	-2.007	56.19
GIGOMEZ2	3	3	linear	0	0	0	3	1.952	0.59	1.952	0.03	1.952	2.35
HIMMELBI	100	12	nonlinear	0	0	12	0	**-1735.6**	1.23	-	-	**-1735.6**	14114
HIMMELBJ	45	14	nonlinear	14	0	0	0	-	-	-	-	**-1910.3**	2001
HIMMELP2	2	1	nonlinear	0	0	0	1	-62.05	0.63	-62.05	0.03	-62.05	1.76

continued on next page

continued from previous page

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		DONLP2		CSA	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
HIMMELP6	2	5	nonlinear	0	0	2	3	**-59.01**	0.69	-57.85	0.02	**-59.01**	2.88
HONG	4	1	nonlinear	1	0	0	0	22.57	0.50	22.57	0.03	22.57	2.90
HS100	7	4	nonlinear	0	0	0	4	680.6	0.72	680.6	0.03	680.6	9.19
HS101	7	5	nonlinear	0	0	0	5	**1809.7**	*	-	-	**1809.7**	75.9
HS102	7	5	nonlinear	0	0	0	5	**911.9**	*	-	-	**911.9**	74.2
HS103	7	5	nonlinear	0	0	0	5	-	-	**543.7**	0.13	**543.7**	75.0
HS104	8	5	nonlinear	0	0	0	5	3.95	0.58	3.95	0.03	3.95	30.62
HS107	9	6	nonlinear	0	6	0	0	**5055**	0.59	5085.5	0.04	**5055**	42.6
HS108	9	13	quadratic	0	0	0	13	**-0.866**	0.58	-0.675	0.1	**-0.866**	52.69
HS109	9	10	nonlinear	0	6	2	2	-	-	-	-	**5362**	48.38
HS111	10	3	nonlinear	0	3	0	0	-47.76	0.83	-47.76	0.05	-47.76	79.4
HS114	10	11	quadratic	1	2	4	4	**-1768.8**	1.64	-	-	**-1768.8**	43.4
HS117	15	5	nonlinear	0	0	0	5	**32.35**	0.60	2400.0	0.04	**32.35**	54.7
HS119	16	8	nonlinear	8	0	0	0	**244.9**	0.54	-	-	**244.9**	421
HS12	2	1	quadratic	0	0	0	1	-30.0	0.46	-30.0	0.03	-30.0	0.95
HS18	2	2	nonlinear	0	0	0	2	5.0	0.65	5.0	0.03	5.0	0.96
HS19	2	2	nonlinear	0	0	0	2	-6961.8	0.58	-6961.8	0.03	-6961.8	1.27
HS20	2	3	nonlinear	0	0	0	3	40.2	0.52	39.17	0.03	**38.19**	1.64
HS23	2	5	quadratic	0	0	0	5	**2.0**	0.54	9.47	0.03	**2.0**	1.94
HS24	2	3	nonlinear	0	0	3	0	-1.0	0.55	-1.0	0.03	-1.0	1.33
HS26	3	1	nonlinear	0	1	0	0	0.0	0.65	0.0	0.03	0.0	1.28
HS27	3	1	nonlinear	0	1	0	0	0.04	0.49	0.04	0.03	0.04	1.27
HS29	3	1	nonlinear	0	0	0	1	-22.6	0.53	-22.6	0.03	-22.6	1.34
HS30	3	1	quadratic	0	0	0	1	**1.0**	0.52	3.0	0.02	**1.0**	1.44
HS32	3	2	nonlinear	1	0	0	1	**1.0**	0.54	1.02	0.03	**1.0**	1.72
HS33	3	2	nonlinear	0	0	0	2	-4.0	0.55	-3.0	0.03	**-4.59**	2.05
HS34	3	2	linear	0	0	0	2	-0.834	0.38	-0.834	0.03	-0.834	2.12
HS36	3	1	nonlinear	0	0	1	0	**-3300**	0.55	-1000	0.03	**-3300**	1.26
HS37	3	2	nonlinear	0	0	2	0	-3456	0.48	-3456	0.02	-3456	1.54
HS39	4	2	linear	0	2	0	0	-1.0	0.52	-1.0	0.03	-1.0	2.11
HS40	4	3	nonlinear	0	3	0	0	-0.25	0.58	-0.25	0.03	-0.25	2.83
HS41	4	1	nonlinear	1	0	0	0	1.926	0.52	1.926	0.03	1.926	1.37
HS42	4	2	nonlinear	1	1	0	0	13.86	0.56	13.86	0.02	13.86	2.36
HS43	4	3	quadratic	0	0	0	3	-44.0	0.49	-44.0	0.03	-44.0	4.48
HS46	5	2	nonlinear	0	2	0	0	0.0	0.54	0.0	0.02	0.0	4.28
HS54	6	1	nonlinear	1	0	0	0	0.0	0.58	-0.156	0.03	**-0.908**	3.87
HS55	6	6	nonlinear	6	0	0	0	6.667	0.49	-	-	**6.333**	6.91
HS56	7	4	nonlinear	0	4	0	0	**-3.456**	0.55	**-3.456**	0.06	-3.31	8.35
HS57	2	1	nonlinear	0	0	0	1	0.03065	0.57	**0.02846**	0.03	**0.02846**	20.45
HS59	2	3	nonlinear	0	0	0	3	**-7.803**	0.88	-6.75	0.03	**-7.803**	5.45
HS60	3	1	nonlinear	0	1	0	0	0.0326	0.62	0.0326	0.03	0.0326	1.65

continued on next page

continued from previous page

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		DONLP2		CSA	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
IDsHS61	3	2	quadratic	0	2	0	0	-143.65	0.57	-143.65	0.04	-143.65	1.54
HS62	3	1	nonlinear	1	0	0	0	-26273	0.61	-26273	0.03	-26273	2.20
HS63	3	2	quadratic	1	1	0	0	961.72	0.55	961.72	0.03	961.72	1.87
HS64	3	1	nonlinear	0	0	0	1	6299.8	0.43	6299.8	0.03	6299.8	1.68
HS68	4	2	nonlinear	0	2	0	0	**-0.9204**	0.72	-0.7804	0.04	**-0.9204**	6.76
HS69	4	2	nonlinear	0	2	0	0	-956.71	0.80	-956.71	0.04	-956.71	4.95
HS7	2	1	nonlinear	0	1	0	0	-1.732	0.56	-1.732	0.03	-1.732	0.94
HS71	4	2	nonlinear	0	1	0	1	**17.01**	0.62	31.64	0.04	**17.01**	2.60
HS73	4	3	linear	1	0	1	1	**29.9**	0.52	29.98	0.03	**29.9**	3.12
HS74	4	5	nonlinear	0	3	2	0	5126.5	0.50	5126.5	0.04	5126.5	5.25
HS75	4	5	nonlinear	0	3	2	0	5174.4	0.56	5174.4	0.04	5174.4	5.30
HS77	5	2	nonlinear	0	2	0	0	0.2415	0.56	0.2415	0.03	0.2415	4.09
HS78	5	3	nonlinear	0	3	0	0	-2.92	0.58	-2.92	0.03	-2.92	3.79
HS79	5	3	nonlinear	0	3	0	0	0.0788	0.57	0.0788	0.03	0.0788	4.10
HS80	5	3	nonlinear	0	3	0	0	0.054	0.58	0.054	0.03	0.054	4.25
HS83	5	3	quadratic	0	0	0	3	**-30666**	0.52	-	-	**-30666**	5.68
HS84	5	3	quadratic	0	0	0	3	-	-	-2.35×10^6	0.03	$\mathbf{-5.28 \times 10^6}$	7.69
HS87	6	4	nonlinear	0	4	0	0	-	-	8997	0.04	**8926**	8.43
HS93	6	2	nonlinear	0	0	0	2	-	-	**135.1**	0.03	**135.1**	4.96
HS99	7	2	nonlinear	0	2	0	0	-	-	$\mathbf{-8.31 \times 10^8}$	0.06	$\mathbf{-8.31 \times 10^8}$	12.8
HUBFIT	2	1	nonlinear	0	0	1	0	0.0169	0.46	0.0169	0.02	0.0169	1.21
LAUNCH	25	28	nonlinear	6	3	12	7	-	-	-	-	**9.0**	1941
LIN	4	2	nonlinear	2	0	0	0	**-0.02**	0.70	-0.0176	0.02	**-0.02**	6.69
LOADBAL	31	31	nonlinear	11	0	20	0	**0.453**	0.69	1.546	0.08	1.546	1712
LOOTSMA	3	2	nonlinear	0	0	0	2	-	-	-	-	**1.414**	2.10
MADSEN	3	6	linear	0	0	0	6	0.616	0.55	0.616	0.03	0.616	5.13
MARATOS	2	1	quadratic	0	1	0	0	-1.0	0.40	-1.0	0.02	-1.0	0.839
MATRIX2	6	2	quadratic	0	0	0	2	0.0	0.52	0.0	0.04	0.0	3.93
MESH	41	48	nonlinear	4	20	24	0	-	-	0.0	0.16	$\mathbf{-1.0 \times 10^5}$	4009
MISTAKE	9	13	quadratic	0	0	0	13	-1.0	0.58	-1.0	0.06	-1.0	55.0
MRIBASIS	36	55	linear	1	8	43	3	**18.218**	1.88	-	-	**18.218**	7612
MWRIGHT	5	3	nonlinear	0	3	0	0	24.97	0.56	24.97	0.02	**1.318**	3.92
NET1	48	57	nonlinear	21	17	16	3	-	-	-	-	$\mathbf{9.41 \times 10^5}$	6776
NET2	144	160	nonlinear	64	59	32	5	-	-	-	-	$\mathbf{1.187 \times 10^6}$	226271
NGONE	8	8	quadratic	0	0	2	6	**-0.5**	0.51	0.0	0.03	**-0.5**	24.7
ODFITS	10	6	nonlinear	6	0	0	0	-2380	0.50	-2380	0.04	-2380	26.8
OPTCNTRL	32	20	quadratic	10	10	0	0	**550**	0.51	-	-	**550**	432
OPTPRLOC	30	30	quadratic	0	0	5	25	**-16.42**	4.02	-	-	**-16.42**	1674
ORTHREGB	27	6	quadratic	0	6	0	0	0.0	0.76	0.0	0.04	0.0	218.3
PENTAGON	6	15	nonlinear	0	0	15	0	1.509×10^{-4}	0.56	$\mathbf{1.365 \times 10^{-4}}$	0.03	$\mathbf{1.365 \times 10^{-4}}$	32.1
POLAK1	3	2	linear	0	0	0	2	2.718	0.53	2.718	0.03	2.718	2.02

continued on next page

continued from previous page

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		DONLP2		CSA	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
POLAK3	12	10	linear	0	0	0	10	**5.933**	0.82	-	-	**5.933**	417.2
POLAK5	3	2	linear	0	0	0	2	50.0	0.52	50.0	0.02	50.0	1.87
POLAK6	5	4	linear	0	0	0	4	-44.0	0.74	-44.0	0.04	-44.0	11.8
QC	9	4	nonlinear	0	0	4	0	**-956.5**	0.58	-	-	**-956.5**	28.5
READING6	102	50	nonlinear	0	50	0	0	-	-	-	-	**-132.3**	28530
RK23	17	11	linear	4	7	0	0	**0.0833**	0.75	-	-	0.675	96.1
ROBOT	14	2	quadratic	0	2	0	0	**5.463**	0.55	-	-	**5.463**	34.8
S316-322	2	1	quadratic	0	1	0	0	334.3	0.48	334.3	0.02	334.3	0.83
SINROSNB	2	1	nonlinear	0	0	0	1	0.0	0.56	0.0	0.04	0.0	1.36
SNAKE	2	2	linear	0	0	0	2	-	-	**0.0**	0.02	**0.0**	1.43
SPIRAL	3	2	linear	0	0	0	2	**0.0**	0.71	0.121	0.31	**0.0**	3.46
STANCMIN	3	2	nonlinear	0	0	2	0	**4.25**	0.58	-	-	**4.25**	1.72
SVANBERG	10	10	nonlinear	0	0	0	10	**15.73**	0.59	16.5	0.03	**15.73**	85.3
SYNTHES1	6	6	nonlinear	0	0	4	2	**0.759**	0.55	10.0	0.04	**0.759**	9.97
SYNTHES2	11	14	nonlinear	1	0	10	3	**-0.554**	0.60	-	-	**-0.554**	94.3
SYNTHES3	17	23	nonlinear	2	0	17	4	**15.08**	0.51	-	-	**15.08**	261.7
TENBARS4	18	9	linear	0	8	1	0	**368.5**	*	-	-	**368.5**	84.9
TWOBARS	2	2	nonlinear	0	0	0	2	1.51	0.53	1.51	0.03	1.51	1.36
WOMFLET	3	3	linear	0	0	0	3	0.0	0.51	0.0	0.02	0.0	2.5
ZAMB2-8	138	48	nonlinear	0	48	0	0	**-0.153**	1.20	-	-	**-0.153**	46156
ZECEVIC3	2	2	quadratic	0	0	0	2	97.31	0.54	97.31	0.03	97.31	1.32
ZECEVIC4	2	2	quadratic	0	0	1	1	7.558	0.59	7.558	0.02	7.558	1.23
ZY2	3	2	nonlinear	0	0	0	2	**2.0**	0.46	7.165	0.03	**2.0**	2.35

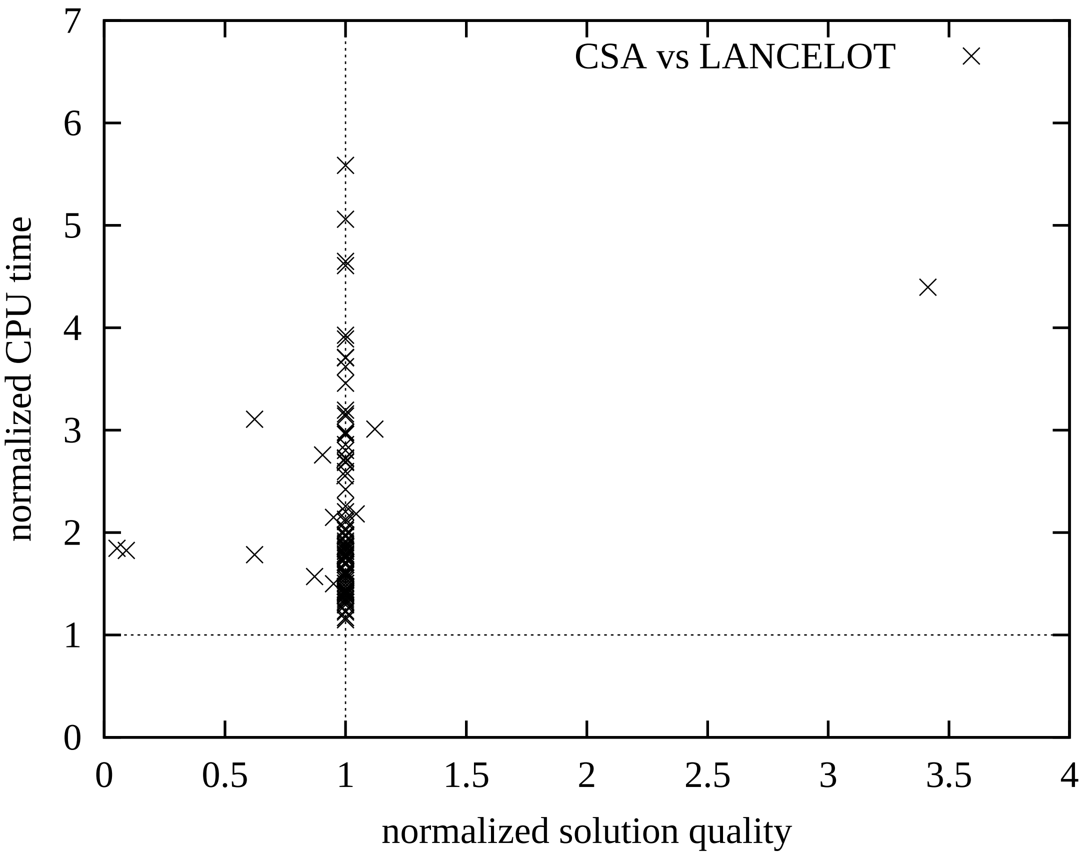


Figure 5.1: Normalized solution quality and normalized CPU time of CSA with respect to LANCELOT for those CUTE benchmarks that are solvable by both CSA and LANCELOT, where normalized CPU time is in log scale.

Table 5.11: Experimental Results of applying LANCELOT on selected CUTE problems that cannot be solved by CSA at this time. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7. $'-'$ means that no feasible solution can be found by both the public version (01/05/2000) and the commercial version of LANCELOT.

Problem IDs	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time
BRAINPC0	6907	6900	nonlinear	0	6900	0	0	1.5E-3	55.5
BRAINPC1	6907	6900	nonlinear	0	6900	0	0	0.0	84.8
BRAINPC2	13807	13800	nonlinear	0	13800	0	0	4.1E-8	93.2
BRAINPC3	6907	6900	nonlinear	0	6900	0	0	1.687E-4	89.4
BRAINPC4	6907	6900	nonlinear	0	6900	0	0	1.288E-3	79.1
BRAINPC5	6907	6900	nonlinear	0	6900	0	0	1.362E-3	143.7
BRAINPC6	6907	6900	nonlinear	0	6900	0	0	5.931E-5	85.2
BRAINPC7	6907	6900	nonlinear	0	6900	0	0	3.82E-5	109.4
BRAINPC8	6907	6900	nonlinear	0	6900	0	0	1.652E-4	112.8
BRAINPC9	6907	6900	nonlinear	0	6900	0	0	8.27E-4	68.2
BRIDGEND	2734	2727	linear	1304	1423	0	0	-	-
BRITGAS	450	360	nonlinear	0	360	0	0	0.0	8.3
C-RELOAD	342	284	linear	26	174	0	84	-1.027	51.1
FEEDLOC	90	259	linear	4	15	166	74	0.0	251.7
HELSBY	1408	1399	linear	658	741	0	0	-	-
HYDROELL	1009	1008	nonlinear	0	0	1008	0	-3.586E6	70.5
HYDROELM	505	504	nonlinear	0	0	504	0	-3.582E6	29.3
HYDROELS	169	168	nonlinear	0	0	168	0	-3.582E6	2.7
LAKES	90	78	quadratic	60	18	0	0	-	-
LEAKNET	156	153	linear	73	80	0	0	8.0	25.7
LHAIFAM	99	150	nonlinear	0	0	0	150	-	-

continued on next page

continued from previous page

Problem IDs	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time
NET3	464	521	nonlinear	195	199	110	17	-	-
READING7	1002	500	nonlinear	0	500	0	0	-	-
READING8	2002	1000	nonlinear	0	1000	0	0	-	-
SARO	4754	4015	linear	0	4015	0	0	252.3	3739.0
SAROMM	5120	5110	linear	365	4015	730	0	57.35	9147.5
TWIRISM1	343	313	nonlinear	50	174	5	84	-1.01	136.1
TWIRIMD1	1247	544	nonlinear	143	378	5	186	-1.034	10158
TWIRIBG1	3127	1239	nonlinear	292	630	5	312	-	-
ZAMB2-10	270	96	nonlinear	0	96	0	0	-1.58	2.99
ZAMB2-11	270	96	nonlinear	0	96	0	0	-1.116	1.83

5.4 Summary

In this chapter, we have applied CSA to solve many discrete, mixed-integer, and continuous constrained NLPs. Although CSA is able to solve problems without differentiability, the benchmarks that we evaluated here are all differentiable, making it possible to compare the performance of CSA with those methods that require derivatives such as SQP.

For discrete and mixed-integer problems, CSA shows much more advantages over SQP. Even though SQP can quickly solve continuous benchmarks, discretization of its continuous solutions sometimes may lead to either infeasible discrete solutions or worse solutions than CSA. Besides, because of the requirement of differentiability, SQP cannot be used to solve those problems whose derivatives are difficult to evaluate or are unavailable.

For continuous problems with differentiability, CSA is not competitive with SQP in terms of CPU time unless the problems have huge numbers of local minima such as G2. By using derivatives efficiently, SQP is normally one- to two-order faster than CSA in one run. Hence, one of future work is to develop some strategies to have CSA efficiently use derivatives without loss of its global convergence property.

Chapter 6

Filter-Bank Design for Subband Image Coding

Digital image processing has an enormous impact on a variety of industrial and real-life applications, especially for multimedia computing environments. To reduce image storage, image coding (or compression) has been a subject of great interest in both academia and industry. Subband/wavelet image coding has recently become a cutting edge technology for image compression [12, 54, 97, 192].

In this chapter, we address filter design issue in subband image coding, whose goal is to achieve a better objective measure called peak signal-to-noise ratio (PSNR) for compressed images. Because it is very time-consuming to directly compute this image-dependent PSNR, many heuristic objectives have been used in the literature [12, 54, 66, 188]. These include coding gain, frequency selectivity, perfect reconstruction (PR), linear phase (LP), and wavelet regularity. Hence, filter designs become multi-objective optimization problems.

We study different formulations for filter design, including two constrained NLPs and one unconstrained NLP, and then solve them using such optimization algorithms as FSQP [226] and CSA developed in Chapters 3 and 4. We demonstrate the performance of CSA

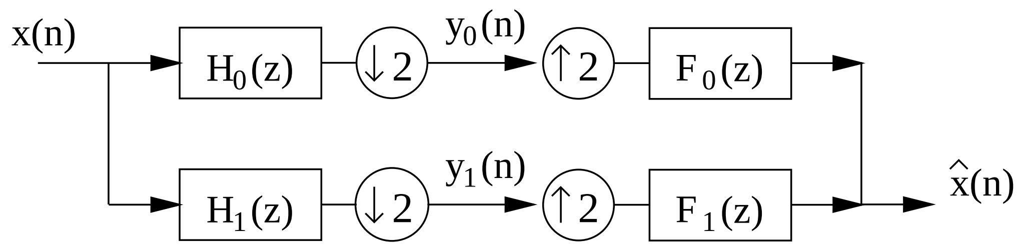


Figure 6.1: The structure of a two-band filter bank.

by improving the coding quality in terms of PSNRs, when compared with that of some best-known subband or wavelet filters.

6.1 Subband Transforms

Transform-based coding techniques have been the *de facto* standard for image compression, whose philosophy is that transformed images are easier to compress than original images. Such transforms can be either linear or nonlinear, but are usually linear due to their invertibility. Two commonly-used linear transforms are the *discrete cosine transform (DCT)* and *subband transform.*

JPEG [207], as an image coding standard, partitions images into nonoverlapping blocks. These image blocks are then transformed by DCT independently. When the transform coefficients are coded under low-bit-rate methods, blocking effects occur because the effects of adjacent image blocks are generally ignored.

Subband transform entails the overlapping of image data to avoid blocking artifacts. 1-D subband transform can be represented by a filter bank as shown in Figure 6.1. It is composed of an analysis subsystem, including filters $H_0(z)$, $H_1(z)$ and decimator, followed by a synthesis subsystem, including filters $F_0(z)$, $F_1(z)$ and interpolator. The analysis subsystem is used for image encoding, whereas the synthesis subsystem is used for image decoding.

Transformed subband signals $y_0(n)$ and $y_1(n)$ are obtained by first convoluting filters $h_0(n)$ and $h_1(n)$ with input signal $x(n)$, followed by down-sampling the signals. The sampling rates of $y_0(n)$ and $y_1(n)$ are half of that of the original signal $x(n)$. In general, filters $h_0(n)$ and $h_1(n)$ are carefully designed in such a way that subband signals $y_0(n)$ and $y_1(n)$ have better energy compaction and decorrelation than original signal $x(n)$.

In addition to subband transforms, *wavelet transforms* also use the filter bank shown in Figure 6.1, but were brought out from totally different perspectives [6]. Subband transforms are multi-rate digital signal processing systems, having three elements of filters, decimators and interpolators. They divide signals into different frequency bands or subbands, for example, low-pass and high-pass bands. Wavelet transforms, however, were motivated by approximation theory. This theory views images as locally smooth functions, and analyzes them using the notion of a multiresolution analysis or function approximations. Wavelet transforms work in spatial domains instead of frequency domains.

In the literature, filters used in subband transforms are called *subband filters*, whereas filters used in wavelet transforms are called *wavelet filters*. Subband filters are designed to optimize behavior in the frequency domain, whereas wavelet filters are designed to emphasize continuity or smoothness. When they are used in image coding, there is no big difference between them, except in the characteristics of the filters. We integrate them to establish the criteria of filter design in this chapter.

6.2 Procedure of Subband Image Coding

Subband image coding consists of the following phases: subband transform based on the filter bank, bit allocation and quantization, and lossless coding such as entropy coding.

6.2.1 Two-Dimensional Subband Transforms

Subband transforms usually require $h_0(n)$ and $f_0(n)$ to be low-pass filters, and $h_1(n)$ and $f_1(n)$ to be high-pass filters [97, 192]. Subband $y_0(n)$, obtained by filtering signal $x(n)$ with low-pass filter $h_0(n)$, is known as a *reference signal*. Subband $y_1(n)$, obtained by filtering signal $x(n)$ with high-pass filter $h_1(n)$, is known as a *detail signal*.

For any image $I(x, y)$, its 2-D subband transform is accomplished by two separate 1-D transforms shown in Figure 6.1, where (x, y) only takes values on integers defined on an image plane $M \times N$. Image $I(x, y)$ is first filtered along the x dimension, resulting in low-pass image $I_L(x, y)$ and high-pass image $I_H(x, y)$. Because down-sampling drops every other filtered value, the size of these two images is $M/2 \times N$. Both images $I_L(x, y)$ and $I_H(x, y)$ are then filtered along the y dimension, resulting in four subimages (or subbands): $I_{LL}(x, y)$, $I_{LH}(x, y)$, $I_{HL}(x, y)$, and $I_{HH}(x, y)$. Therefore, we have one reference image $I_{LL}(x, y)$ and three detail images that are directionally sensitive: $I_{LH}(x, y)$ emphasizing horizontal image features, $I_{HL}(x, y)$ vertical features, and $I_{HH}(x, y)$ diagonal features.

It is customary in subband image coding to recursively transform the reference image [97, 54]. In general, the number of such splits ranges from 3 to 5, depending on the compression ratio, the size of the original image $I(x, y)$, and the length of filters. The higher the compression ratio, the more times that subband transforms are performed. If the number of splits is L, then the total number of subimages obtained by recursive transforming the original image is $N_I = 3L + 1$.

Recursive applications of subband transform form a *splitting tree structure*, whose shape is determined by the splitting scheme. The above splitting scheme only constructs one of these trees, called the *dyadic tree structure*, and may not be the best for any images. After this sequence of subband transforms, one obtains a set of transform coefficients that comprise the reference subimage and detail subimages of various levels. No compression of the original image has been achieved yet, and further processing, such as quantization and coding, is needed.

6.2.2 Bit Allocation and Quantization

The purpose of subband transforms is to have different subimages embed different important information used by the human visual system [54]. If filters $h_0(n)$ and $h_1(n)$ are correctly determined, the subband transform decorrelates pixel values of original image $I(x, y)$ and concentrates most image information into a relatively small number of coefficients, such as those in reference subimages. These important coefficients have to be finerly quantized than others in order to efficiently compress images, while keeping most image information in encoded images. Hence, designing good filters is very important in subband image compression.

Given a total bit budget, bit allocation tries to assign different numbers of bits to different subimages based on their relative importance. More formally, the bit allocation problem entails finding the optimal distribution of the available bits among all subimages. To fit every subimage into the available bits, quantization is required that causes some distortion. In general, quantization is characterized by a *quantization step.* A larger quantization step causes a larger distortion but a smaller number of bits, whereas a smaller quantization step causes a smaller distortion but a larger number of bits.

6.2.3 Coding Quantized Coefficients

After bit allocation and quantization, these quantized coefficients are efficiently coded to achieve a high compression ratio [54, 97, 192]. Entropy coding achieves smaller images on average by assigning shorter code words to more likely values and longer code words to less likely values, where likeliness is usually measured by histograms. Arithmetic coding provides a near-optimal entropy coding for quantized coefficients. It estimates the distribution of these coefficients that is approximated by a generalized Gaussian or Laplacian distribution density. Most of the time, the density is estimated online, allowing the exploitation of local changes in image statistics.

Many efficient methods have recently been proposed to greatly reduce the number of bits needed for encoding quantized coefficients. The basic idea is to exploit the relationship of these coefficients across subbands, for example, some significant coefficients are similar in shape and location across different subbands. Embedded zero-tree [176] combines this idea with the notion of coding zeros jointly by encoding and decoding an image progressively, whereas set partitioning in hierarchical tree (SPHIT) [168] further enhances its implementation. The rate-distortion of bit allocation can also be incorporated into a zero-tree [220] to get better performance. Instead of using a *priori* knowledge about transform coefficients, finding an optimized significance tree (or map) is obtained from a set of training images [53]. Note that all these approaches improve coding quality a lot by reducing the number of bits for coding transform coefficients, instead of improving the filters by changing their coefficient distribution.

6.3 Subband-Filter Design Criteria

There are two strategies of filter design [17, 18]. The first one is to design subband filters that are independent of individual input images (but depend on general image statistics), splitting tree structure, and quantization. Only one pair of low-pass and high-pass filters h_0 and h_1 need to be determined. Its major advantages are (a) that the design procedure, the image encoder, and the image decoder are simple, (b) that the subband filters obtained can be applicable to any image, any tree structure, and any quantization method, and (c) that if filter design criteria are carefully chosen, the filters will perform well in general, such as Daubechies' 9/7 wavelet filter [12, 193]. The disadvantage is that such filters may not perform the best for any given image.

The second design strategy is to optimize a design by jointly designing subband filters, splitting tree structure, and quantizers [149]. This design procedure is usually image dependent. An engineering approach to solve such dependence is to design subband filters using some training images [149]. Its success, however, depends heavily on the training images selected, and may only work well for a small set of images. Independent of quantizers and images, subband filters and tree structures are jointly designed by maximizing coding gain [17, 19], where different stages of subband decomposition employ different subband filters. Such image-dependent designs are usually computationally expensive. They achieve high coding quality for a given image, but may not be generalized to other images.

In this thesis, we adopt the first design strategy and expect to get better coding results from two aspects: (a) Proper filter design criteria from subband coding, coding theory, and wavelet theory are integrated into our design criteria. (b) Since the filter design is formulated as NLPs, our global optimization method may find better solutions.

6.3.1 Choice of Filter-Bank Types

For the application of subband image coding, we select a filter bank based on two criteria: perfect reconstruction (PR) and linear phase (LP) of filters. While the former precludes any errors caused by the filter bank itself, the latter avoids introducing any phase errors into transform coefficients, since phase distortions around edges are very visible. A biorthogonal filter bank [66, 188] can meet both requirements if its filter parameters were chosen carefully. Therefore, we use biorthogonal filter banks for image coding to have both PR and LP.

6.3.2 Performance Metrics for Filter-Bank Design

In a biorthogonal filter bank, synthesis filters $F_0(z)$ and $F_1(z)$ relate to analysis filters $H_0(z)$ and $H_1(z)$ by setting $F_0(z) = 2H_1(-z)$ and $F_1(z) = -2H_0(-z)$ in order to cancel aliasing errors [188]. Hence, only two filters $H_0(z)$ and $H_1(z)$ need to be determined. In this subsection, we investigate the performance metrics for designing these two filters.

6.3.2.1 Criteria for Filter Bank

To achieve both perfect reconstruction (PR) and linear phase (LP), it is required that the sum of the lengths of filters $H_0(z)$ and $H_1(z)$ be a multiple of 4, $N_0 + N_1 = 4m$, where N_0 and N_1 are, respectively, the lengths of filters $H_0(z)$ and $H_1(z)$. Type-B filters [139] are used here for which both filters $H_0(z)$ and $H_1(z)$ are of odd length and are symmetric (namely, LP). Because of the symmetry of the filters,

$$\begin{aligned} h_0(n) &= h_0(N_0+1-n), \quad n = 1, 2, \cdots, (N_0-1)/2 \\ h_1(n) &= h_1(N_1+1-n), \quad n = 1, 2, \cdots, (N_1-1)/2, \end{aligned}$$

the number of the filter coefficients is reduced by almost half. The filter parameters are denoted by $h_0 = \{h_0(n),\ n = 1, 2, \cdots, (N_0+1)/2\}$ and $h_1 = \{h_1(n),\ n = 1, 2, \cdots, (N_1+1)/2\}$.

To avoid any errors caused by the filter bank itself, we require the filter bank to be PR, which is enforced by a set of quadratic equations [103], called *PR condition*,

$$\frac{1}{2}\theta\left(i - \frac{N_0+N_1}{4}\right) = \sum_{k=1}^{2i}(-1)^{k-1}h_0(2i+1-k)h_1(k), \qquad i = 1, 2, \cdots, \frac{N_0+N_1}{4}, \quad (6.1)$$

where $\theta(x) = 1$ if $x = 0$, and 0 otherwise.

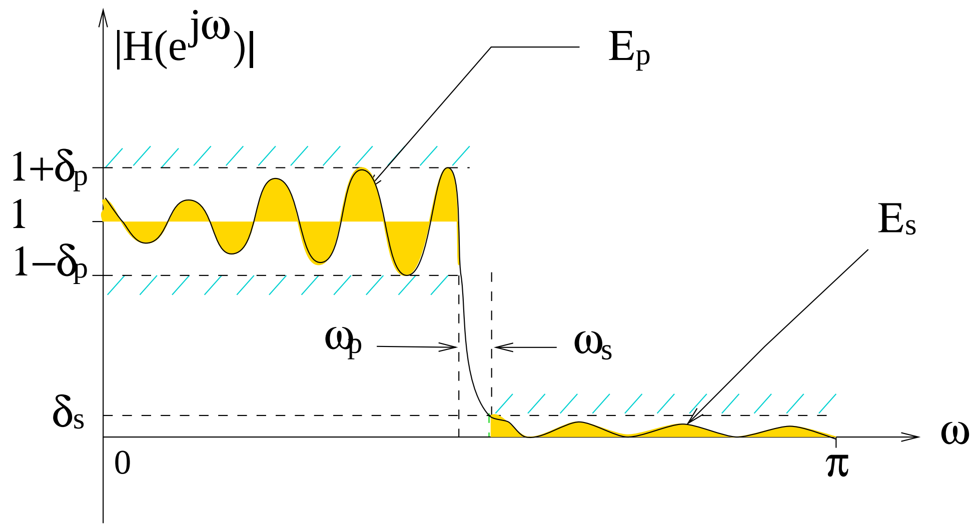


Figure 6.2: Performance metrics of a low-pass filter.

6.3.2.2 Individual Filter Criteria

As mentioned before, some important subimages containing low frequencies of the original image are finerly quantized than others containing high frequencies. If they are separated well by the analysis filters, one may achieve good quality of image coding. Otherwise, some important information may leak into unimportant subimages and may be degraded or lost by coarse quantization, leading to poor image quality.

To perfectly divide the frequency band into low and high frequencies, we require ideal filters. For low-pass filter h_0, its amplitude response $|H_0(\omega)| = 1$ if $0 \leq \omega \leq \pi/2$, and $|H_0(\omega)| = 0$ if $\pi/2 \leq \omega \leq \pi$. For high-pass filter h_1, its amplitude response $|H_1(\omega)| = 0$ if $0 \leq \omega \leq \pi/2$, and $|H_1(\omega)| = 1$ if $\pi/2 \leq \omega \leq \pi$. Since it is impossible to achieve these ideal filters using finite-length filters, we want to maximize their proximity to these ideal filters that is measured by *frequency metrics* (or *frequency selectivity*).

The frequency metrics of individual filters to approximate ideal filters include their stop-band energies $E_s(h_0)$ and $E_s(h_1)$, stopband ripples $\delta_s(h_0)$ and $\delta_s(h_1)$, passband energies

$E_p(h_0)$ and $E_p(h_1)$, passband ripples $\delta_p(h_0)$ and $\delta_p(h_1)$, and transition widths $\tau(h_0)$ and $\tau(h_1)$. Their meanings are depicted in Figure 6.2.

These performance metrics evaluate the degree of approximation to an ideal filter. However, they are time-consuming to evaluate, especially for metrics without closed-forms and are evaluated by numerical methods, such as stopband ripples $\delta_s(h_0)$ and $\delta_s(h_1)$, passband ripples $\delta_p(h_0)$ and $\delta_p(h_1)$, and transition widths $\tau(h_0)$ and $\tau(h_1)$. To greatly reduce computational costs, we only use passband and stopband energies given passband and stopband cut-off frequencies ω_p and ω_s [103, 188]. Let the Fourier transforms of filters h_0 and h_1 be, respectively,

$$FT_0(e^{j\omega}) = H_0(\omega)e^{-j\frac{N_0-1}{2}\omega} \qquad FT_1(e^{j\omega}) = H_1(\omega)e^{-j\frac{N_1-1}{2}\omega}$$

where the term $e^{-j\frac{N_0-1}{2}\omega}$ is the linear phase of filter $H_0(z)$ with delay $(N_0-1)/2$, and $e^{-j\frac{N_1-1}{2}\omega}$ is the linear phase of $H_1(z)$ with delay $(N_1-1)/2$, and

$$\begin{aligned} H_0(\omega) &= h_0((N_0+1)/2) + \sum_{n=1}^{(N_0-1)/2} 2h_0(n)cos\left(\frac{N_0+1}{2}-n\right)\omega \\ H_1(\omega) &= h_1((N_1+1)/2) + \sum_{n=1}^{(N_1-1)/2} 2h_1(n)cos\left(\frac{N_1+1}{2}-n\right)\omega \qquad (6.2) \end{aligned}$$

Here we only show the equations of stopband energy $E_s(h_0)$ and passband energy $E_p(h_0)$ for low-pass filter h_0, given its stopband and passband cut-off frequencies ω_{s0} and ω_{p0}. The stopband energy $E_s(h_0)$ is:

$$\begin{aligned}
E_s(h_0) &= \int_{\omega_{s0}}^{\pi} H_0^2(\omega)\, d\omega \qquad (6.3)\\
&= h_0^2((N_0+1)/2)(\pi-\omega_{s0}) - 4h_0((N_0+1)/2)\sum_{n=1}^{(N_0-1)/2} h_0(n)\frac{sin\left(\frac{N_0+1}{2}-n\right)\omega_{s0}}{\frac{N_0+1}{2}-n}\\
&\quad +2\sum_{n=1}^{(N_0-1)/2} h_0^2(n)\left[(\pi-\omega_{s0}) - \frac{sin(N_0+1-2n)\omega_{s0}}{N_0+1-2n}\right]\\
&\quad -2\sum_{n=1}^{(N_0-1)/2}\sum_{m=1,m\neq n}^{(N_0-1)/2} h_0(n)h_0(m)\left[\frac{sin(n-m)\omega_{s0}}{n-m} + \frac{sin(N_0+1-n-m)\omega_{s0}}{N_0+1-n-m}\right]
\end{aligned}$$

and the passband energy $E_p(h_0)$ is:

$$\begin{aligned}
E_p(h_0) &= \int_{0}^{\omega_{p0}} (H_0(\omega)-1)^2\, d\omega \qquad (6.4)\\
&= [h_0((N_0+1)/2)-1]^2\omega_{p0} + 4[h_0((N_0+1)/2)-1]\sum_{n=1}^{(N_0-1)/2} h_0(n)\frac{sin\left(\frac{N_0+1}{2}-n\right)\omega_{p0}}{\frac{N_0+1}{2}-n}\\
&\quad +2\sum_{n=1}^{(N_0-1)/2} h_0^2(n)\left[\omega_{p0} + \frac{sin(N_0+1-2n)\omega_{p0}}{N_0+1-2n}\right]\\
&\quad +2\sum_{n=1}^{(N_0-1)/2}\sum_{m=1,m\neq n}^{(N_0-1)/2} h_0(n)h_0(m)\left[\frac{sin(n-m)\omega_{p0}}{n-m} + \frac{sin(N_0+1-n-m)\omega_{p0}}{N_0+1-n-m}\right]
\end{aligned}$$

For high-pass filter h_1, we first transform it into a low-pass filter $H_1(\omega+\pi)$, and then follow the same steps from $H_1(\omega+\pi)$ to obtain its stopband energy $E_s(h_1)$ and passband energy $E_p(h_1)$, given its stopband and passband cut-off frequencies ω_{s1} and ω_{p1}.

6.3.2.3 Wavelet Criteria

Wavelet properties [54, 192] that are of particular interest in image coding are the *accuracy of approximation*, *regularity*, as well as the *support* of the wavelet basis. Approximation accuracy, as characterized by the number of *vanishing moments*, is not a problem here since the filter bank is guaranteed to be PR. The size of the support of wavelet basis should be small that can also be achieved by limiting filter lengths (i.e. N_0 and N_1) to be short. Very long filters tend to spread coding errors.

Regularity requires that iterated low-pass filters converge to continuous functions [193]. Intuitively, it can be understood to be smoothness and, thus, is also referred to as a *smoothness constraint*. If this is true, then any discontinuity in transform coefficients is caused by the input signal itself. Hence, regular basis functions lead to a better representation of the signal. In addition, quantization errors in transform coefficients produce reconstruction errors, and their smoothness is related to that of a wavelet transform, *i.e.*, filters. An argument for better regularity is that a smooth error signal is less annoying to the human eye than a discontinuous error signal, even if their actual distortion is the same [12].

Therefore, the concept of regularity seems to be a good criteria for subband image coding. The regularity order of function $\phi(x)$ is defined to be r if it is r times differentiable and its r-th derivative $d^r\phi(x)/dx^r$ is continuous, where r is an integer. In practice, r is between one and two, and additional smoothness does not appear to yield significant improvements for coding quality.

For low-pass filter h_0, the regularity of order r requires at least r zeros of its amplitude response $H_0(\omega)$ at $\omega = \pi$. Similarly, for high-pass filter h_1, the regularity of order r requires at least r zeros of its amplitude response $H_1(\omega)$ at $\omega = 0$. Hence, the regularity constraint can be written as

$$H_0(\omega)|_{\omega=\pi} = 0$$
$$d^n H_0(\omega)/d\omega^n|_{\omega=\pi} = 0 \qquad \text{for} \quad n = 1, 2, \cdots, r-1 \tag{6.5}$$

for low-pass filter $H_0(\omega)$. Similar conditions can be defined for high-pass filters $H_1(\omega)$ except that it is evaluated at $\omega = 0$.

The number of zeros determines the number of equalities in (6.5). Here we use regularity of order 2 because it is sufficient for subband coding [17, 19]. Because zeros appear in pairs, the 2-order regularity can be enforced by letting $H_0(\omega = \pi) = 0$ for low-pass filter h_0 and $H_1(\omega = 0) = 0$ for high-pass filter h_1. Substituting them into (6.2), we get two equations,

$$h_0((N_0+1)/2) + 2 \sum_{n=1}^{(N_0-1)/2} (-1)^{(\frac{N_0+1}{2}-n)} h_0(n) = 0$$
$$h_1((N_1+1)/2) + 2 \sum_{n=1}^{(N_1-1)/2} h_1(n) = 0. \tag{6.6}$$

There is one important point that needs to be emphasized. In the domain of signal processing, filter bank design [188, 66, 103] tries to achieve PR condition and approximate an ideal filter on every frequency. But in wavelet theory, filter design emphasizes smoothness near $\omega = 0$ or $\omega = \pi$ [12, 17, 52, 47]. Here, we formulate regularity as simple equations in (6.6) for 2-order regularity.

6.3.2.4 Coding Gain

Coding gain measures energy compaction, and high coding gains correlate consistently with high objective values [17, 19, 114, 193, 11]. However, it is image dependent and too expensive to be evaluated in filter designs. To greatly reduce its computation, one needs to make some simplifying assumptions. For instance, by modeling a natural image as a Markovian source with the nearest sample correlation ρ and by assuming uncorrelated quantization errors and high bit budget, Katto and Yasuda [114] derived a *filter-dependent* coding gain,

$$G(\rho, h_0, h_1) = \frac{1}{\prod_{k=0}^{M-1}(A_k B_k)^{1/M}} \tag{6.7}$$

for one level of subband decomposition, where M is the number of subbands ($M = 2$ here), and

$$A_k = \sum_i \sum_j h_k(i) h_k(j) \rho^{|j-i|} \qquad \text{and} \qquad B_k = \sum_i f_k(i)^2 = 4 \sum_i h_k(i)^2$$

where $\rho = 0.95$ for general images. For images that have low-pass characteristics, the Markovian source model for coding gain fits well. But this assumption is hardly valid [3] for those images that exhibit strong non-stationarities in the sense of varying local characteristics. In addition, a high-rate budget is violated for compression at a low-bit rate.

6.4 Optimization Formulations for Filter-Bank Design

To design analysis filters h_0 and h_1, we have identified four sets of performance metrics, PR condition (6.1) for the filter bank, frequency metrics of stopband and passband energies (6.3) and (6.4) for individual filters, wavelet 2-order regularity (6.6), and coding gain (6.7). In summary, it is a multi-objective problem with the following objectives:

- obj$_1$: satisfy PR condition (6.1);
- obj$_2$: minimize $E_s(h_0)$, $E_s(h_1)$, $E_p(h_0)$, and $E_p(h_1)$;
- obj$_3$: satisfy 2-order regularity (6.6);

- obj$_4$: maximize coding gain (6.7).

To solve this multi-objective problem, we reformulate it into three different formulations to see how they can achieve better PSNRs. The first one is a constrained formulation with the objective being a weighted sum of obj$_2$ and obj$_4$ and the constraints being obj$_1$ and obj$_3$,

$$\begin{aligned}
\min_{h_0,h_1} \quad & w_f \; (E_s(h_0) + E_p(h_0) + E_s(h_1) + E_p(h_1)) + w_c \; \tilde{G}(\rho, h_0, h_1) \\
\text{subject to} \quad & \frac{1}{2}\theta\left(i - \frac{N_0+N_1}{4}\right) = \sum_{k=1}^{2i} (-1)^{k-1} h_0(2i+1-k) h_1(k) \qquad i = 1, 2, \cdots, \frac{N_0+N_1}{4} \\
& h_0((N_0+1)/2) + 2 \sum_{n=1}^{(N_0-1)/2} (-1)^{(\frac{N_0+1}{2}-n)} h_0(n) = 0 \\
& h_1((N_1+1)/2) + 2 \sum_{n=1}^{(N_1-1)/2} h_1(n) = 0 \qquad (6.8)
\end{aligned}$$

where $\tilde{G}(\rho, h_0, h_1) = \left(\frac{1}{G(\rho,h_0,h_1)}\right)^M$, and w_f and w_c are relative weights between frequency metrics and the coding gain. This is a constrained NLP with a nonlinear objective, two linear equality constraint, and $\frac{N_0+N_1}{4}$ quadratic equality constraints. We denote this formulation by $A(w_f/w_c)$.

The second formulation is also a constrained formulation, whose objective function is obj_2 and whose constraints are obj_1, obj_3 and obj_4.

$$\begin{aligned}
\min_{h_0,h_1} \quad & E_s(h_0) + E_p(h_0) + E_s(h_1) + E_p(h_1) \\
\text{subject to} \quad & \frac{1}{2}\theta\left(i - \frac{N_0+N_1}{4}\right) = \sum_{k=1}^{2i}(-1)^{k-1}h_0(2i+1-k)h_1(k) \qquad i = 1, 2, \cdots, \frac{N_0+N_1}{4} \\
& h_0((N_0+1)/2) + 2\sum_{n=1}^{(N_0-1)/2}(-1)^{(\frac{N_0+1}{2}-n)}h_0(n) = 0 \\
& h_1((N_1+1)/2) + 2\sum_{n=1}^{(N_1-1)/2}h_1(n) = 0 \\
& G(\rho, h_0, h_1) \geq G_{ref} + \beta
\end{aligned} \tag{6.9}$$

where G_{ref} is the coding gain of a baseline filter, and β is an offset. We can obtain a filter with higher coding gain than G_{ref}, if $\beta > 0$ and (6.9) is solved successfully. This is a constrained NLP with a nonlinear objective, two linear equality constraint, $\frac{N_0+N_1}{4}$ quadratic equality constraints and one nonlinear constraint. This formulation is denoted by $B(\beta)$.

The third formulation is an unconstrained formulation, whose objective is defined as:

$$\begin{aligned}
\min_{h_0,h_1} \quad & E_s(h_0) + E_p(h_0) + E_s(h_1) + E_p(h_1) + \tilde{G}(\rho, h_0, h_1) \\
& + \frac{\gamma}{2}\sum_{i=1}^{(N_0+N_1)/4}\left[\frac{1}{2}\theta(i - (N_0+N_1)/4) - \sum_{k=1}^{2i}(-1)^{k-1}h_0(2i+1-k)h_1(k)\right]^2 \\
& + \frac{\gamma}{2}\left[h_0((N_0+1)/2) + 2\sum_{n=1}^{(N_0-1)/2}(-1)^{(\frac{N_0+1}{2}-n)}h_0(n)\right]^2 \\
& + \frac{\gamma}{2}\left[h_1((N_1+1)/2) + 2\sum_{n=1}^{(N_1-1)/2}h_1(n)\right]^2
\end{aligned} \tag{6.10}$$

where γ is the penalty to penalize the violations of PR condition (6.1) and 2-order regularity (6.6). This is an unconstrained NLP, denoted by $C(\gamma)$.

6.5 Implementation Issues

In this section, we discuss two approaches to solve optimization problems (6.8)-(6.10), and briefly review a wavelet image coding package [55] used to measure PSNRs of subband filters.

6.5.1 Solution Methods for Filter-Bank Design

We solve filter design problems (6.8)-(6.10) by two optimization methods.

- *CSA with (Cauchy$_1$,S-uniform,M):* The strategies used in CSA consist of generating trial points (x', λ) by the Cauchy$_1$ distribution, generating trial points (x, λ') by the symmetric uniform (S-uniform) distribution, and accepting trial points based on Metropolis rule (3.11). The cooling rate used in geometric cooling schedule (3.12) is $\alpha = 0.8$. See Chapter 4 for details on CSA.

- *FSQP:* Feasible SQP (FSQP) is one of the popular SQP packages [226]. It first converts nonlinear equality constraints into relaxed inequality constraints and adds penalties to the objective in order to penalize violations of these equality constrains. Afterwards, given a starting point, FSQP first generates a feasible point that satisfies all inequality as well as linear equality constraints, and then maintains feasibility of these inequality and linear equality constraints in successive iterations. Because it needs to use derivatives, FSQP can only be applied to solve continuous problems.

6.5.2 Wavelet Image Coding Package

To show the performance of filters on image coding, we use a wavelet image compression construction kit [55] that consists of the following three stages.

- *Invertible transform:* Invertible transforms allow a variety of transforms such as DCT, wavelets and wavelet packets. Since an image is finite, a symmetric extension is implemented in the package to cope with image boundaries. Some best-known filters, widely used in current wavelet image coding, are also included. This helps us compare the performance of our designed filters with that of existing ones.

- *Quantization:* Transform coefficients are uniformly discretized using a form of scalar quantization, and bit allocation is based on integer programming for near-optimal allocation in a simple quantization scheme.

- *Entropy coding:* Quantized coefficients are finally entropy coded to produce compressed images. The entropy coding uses simple histogram adaptation with escape codes.

Although this basic wavelet image coder employs simple strategies in each step, instead of state-of-the-art techniques, it is modular and easy to modify its quantizer, entropy coder, and wavelet filters. Therefore, we only need to change the wavelet-filter part in our experiments and comparisons.

6.5.3 Measurements of Coding Quality

There are two performance measures for image coding, compression ratio and peak signal-to-noise ratio (PSNR). The compression ratio is defined as:

$$\frac{\text{number of bits in the original image}}{\text{number of bits in the compressed image}}.$$

Here, we confine our experiments to 8 bit-per-pixel (bpp) grey-scale images. Thus, PSNR in decibels (dB) is computed as:

$$\text{PSNR} = 20\ \log_{10} \frac{255}{\text{RMSE}}$$

where RMSE is the root mean-squared error defined as:

$$\text{RMSE} = \sqrt{\frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} [f(i,j) - \hat{f}(i,j)]^2}$$

where N and M are the width and height of an image in pixels, f is the original image, and $\hat{f}$ is the decoded (or reconstructed) image.

6.6 Experimental Results

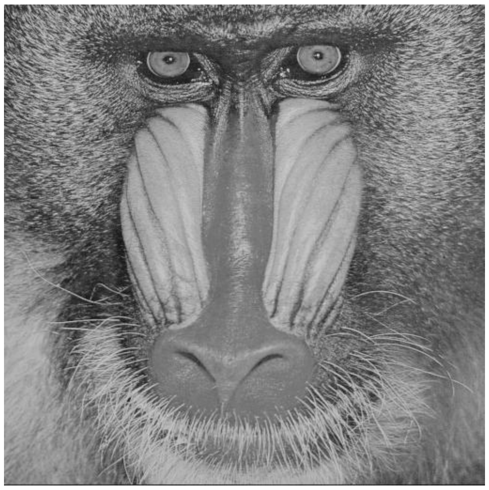
Two types of filters with different lengths N_0/N_1 are designed: 9/7 and 13/11 filters. Cut-off frequencies ω_s and ω_p control how close the designed filters are to the ideal filters. For 9/7 filter, we set $\omega_s = 0.7\pi$ and $\omega_p = 0.3\pi$, and for 13/11 filter, we set $\omega_s = 0.6\pi$ and $\omega_p = 0.4\pi$. For the purpose of comparison, we compare the coding quality of our designed 9/7 and 13/11 filters with that of Daubechies' 9/7 filter [12] and Villasenor's 13/11 filter [193], respectively. Both Daubechies' 9/7 and Villasenor's 13/11 filters were reported to perform very well for subband image coding [193].

We tested the performance of filters on four images shown in Figure 6.3: Lena is smooth, Barbara and Goldhill are more detailed with some textures, and Mandrill is totally of texture style. For a given test image, five compression ratios, 4:1, 8:1, 16:1, 32:1 and 64:1, are experimented.

(a) Lena image

(b) Barbara image

(c) Goldhill image



(d) Mandrill image

Figure 6.3: Four test images.

6.6.1 Control Parameters for Optimization Formulations

For constrained formulation $A(w_f/w_c)$, we study the effects of frequency metrics and coding gain on image coding quality by three choices of weights for w_f and w_c: (a) coding gain G and frequency metrics are equally weighted, namely, $w_f = w_c = 1$; (b) only frequency metrics are considered, namely, $w_f = 1$ and $w_c = 0$, and (c) only coding gain G is optimized, namely, $w_f = 0$ and $w_c = 1$.

For constrained formulation $B(\beta)$, we evaluate how the required minimum coding gain affects coding quality. In designing 9/7 filter, we select baseline G_{ref} to be the coding gain of Daubechies' 9/7 filter, and three choices of $\beta = 0$ dB, 0.5 dB and -0.5 dB are respectively tested. In designing 13/11 filter, we set baseline G_{ref} to be the coding gain of Villasenor' 13/11 filter, and examine three choices of $\beta = 0$ dB, 0.5 dB and -0.5 dB.

For unconstrained formulation $C(\gamma)$, we test three different values of penalty $\gamma = 1$, 100, 10000 that penalize the violations of PR condition (6.1) and 2-order regularity (6.6).

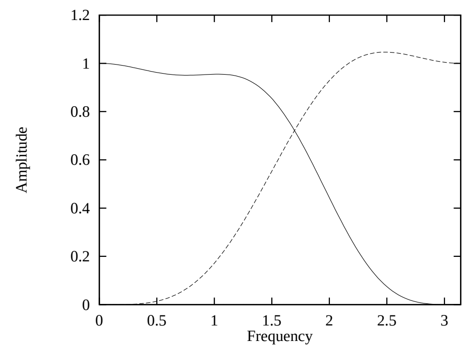
Due to the nonlinear objective and constraints, filter-design problems (6.8)-(6.10) may have many local minima with different objective values. To compare FSQP with CSA, we randomly generate 100 starting points $h_0(n), h_1(n) \in [-1, +1]$, and for each starting point $\left(h_0^{(t=0)}, h_1^{(t=0)}\right)$, we solve these optimization problems using both FSQP and CSA.

6.6.2 Experimental Results on 9/7 Filters

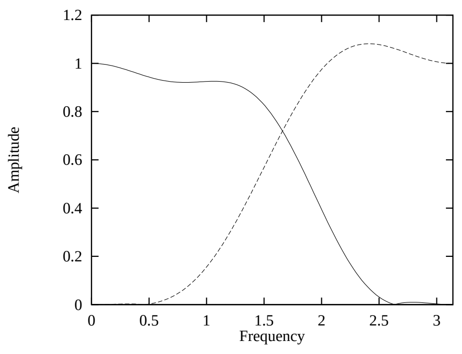
Table 6.1 reports the results of applying FSQP and CSA to design 9/7 filters with respect to different problem formulations and their control parameters. The first column shows the problem formulations and their control parameters. For unconstrained formulation $C(\gamma)$, we only show the result for $\gamma = 100$, because it is almost the same as those for $\gamma = 1$ and $\gamma = 10000$. The next four columns show the best solutions obtained by FSQP and CSA,

Table 6.1: Comparison results of applying FSQP and CSA to design 9/7 filters using different problem formulations and their control parameters. All runs were done on a Pentium-III 450-Mhz computer under Solaris 7. Numbers in bold represent the best solution in terms of average improved PSNR.

Methods	Formulations	best	coding	average	average	Average Improved PSNRs (dB)				
						compression ratio				
		sol'n	gain (dB)	time (sec.)	imp'd PSNR	4:1	8:1	16:1	32:1	64:1
	$\mathbf{A(w_f/w_c = 1.0/1.0)}$	**0.0174**	**5.822**	**0.095**	**0.063 dB**	0.066	0.041	0.132	0.047	0.029
	$A(w_f/w_c = 1.0/0.0)$	0.0131	5.816	0.150	0.059 dB	0.048	0.039	0.131	0.048	0.031
FSQP	$A(w_f/w_c = 0.0/1.0)$	0.00314	6.496	2.11	-1.49 dB	-1.84	-1.74	-1.59	-1.29	-0.99
	$B(\beta = 0.0\ dB)$	0.015	5.916	0.28	0.008 dB	0.012	0.029	0.008	0.008	-0.019
(100 runs)	$B(\beta = +0.5\ dB)$	0.0546	6.416	0.51	-0.94 dB	-1.22	-1.28	-0.94	-0.74	-0.49
	$B(\beta = -0.5\ dB)$	0.0131	5.816	0.12	0.06 dB	0.049	0.040	0.131	0.046	0.032
	$C(\gamma = 100)$	0.00805	6.139	0.028	-1.01 dB	-2.97	-1.03	-0.541	-0.305	-0.213
	$\mathbf{A(w_f/w_c = 1.0/1.0)}$	**0.0174**	**5.822**	**132.4**	**0.063 dB**	0.066	0.041	0.132	0.047	0.029
	$A(w_f/w_c = 1.0/0.0)$	0.0131	5.816	115.3	0.059 dB	0.048	0.039	0.131	0.048	0.031
CSA	$A(w_f/w_c = 0.0/1.0)$	0.00314	6.496	44.5	-1.49 dB	-1.84	-1.74	-1.59	-1.29	-0.99
	$B(\beta = 0.0\ dB)$	0.015	5.916	138.7	0.008 dB	0.012	0.029	0.008	0.008	-0.019
(100 runs)	$B(\beta = +0.5\ dB)$	0.0546	6.416	153.6	-0.94 dB	-1.22	-1.28	-0.94	-0.74	-0.49
	$B(\beta = -0.5\ dB)$	0.0131	5.816	154.1	0.06 dB	0.049	0.040	0.131	0.046	0.032
	$C(\gamma = 100)$	0.00805	6.139	85.35	-1.01 dB	-2.97	-1.03	-0.541	-0.305	-0.213



(a) Daubechies' 9/7 filter



(b) $A(w_f/w_c = 1.0/1.0)$

Figure 6.4: Frequency amplitudes of Daubechies' 9/7 filter and our best 9/7 filter obtained using formulation $A(w_f/w_c = 1.0/1.0)$.

the corresponding coding gains, average running time, and average improved PSNR over the four images and five compression ratios with respect to those of Daubechies' 9/7 filter. The next four columns detail average improved PSNRs at five compression ratios over the four test images.

For every problem formulation and every set of control parameters, FSQP and CSA locate the same best solutions. CSA is much slower than FSQP because CSA utilizes only sampling whereas FSQP utilizes derivatives. The best improved PSNR (0.063 dB) is achieved by the 9/7 filter designed using formulation $A(w_f/w_c = 1.0/1.0)$, whose coding gain is 5.822 dB. Here we call this filter our best 9/7 filter. Although the 9/7 filter obtained from $A(w_f/w_c = 0.0/1.0)$ has higher coding gain (6.496 dB) than our best 9/7 filter, its coding quality is much worse. This means that coding gain may not be the most important factor in subband image coding.

The amplitude responses of Daubechies' 9/7 filter and our best 9/7 filter are shown in Figure 6.4. Since Daubechies' 9/7 filter is designed based on wavelet theory, it has the most smoothness enforced by a high-order regularity. Our best 9/7 filter, however, has narrower transition bandwidths and is closer to the ideal filter, partly due to the optimization of frequency metrics. From the point of view of signal processing, the frequency response of our best 9/7 filter is better than Daubechies' 9/7 filter. Besides, the amplitude value of our filter equals zero at $\omega = \pi$ for low-pass filter h_0, and equals zero at $\omega = 0$ for high-pass filter h_1, showing some smoothness around $\omega = \pi$ and $\omega = 0$. This is enforced by the equality constraints of 2-order wavelet regularity.

Table 6.2 shows the coding quality on four images using Daubechies' 9/7 filter whose coding gain is 5.916 dB and our best 9/7 filter. For every image, we give PSNRs for both Daubechies' 9/7 and our best 9/7 filters and their difference. For Lena, Goldhill, and Mandrill, our filter has similar performance as Daubechies' 9/7 filter with the maximum improvement of 0.14 dB and the maximum degradation of -0.5 dB. For Barbara, our filter performs better, improving 0.58 dB at compression ratio 16:1.

6.6.3 Experimental Results on 13/11 Filters

Table 6.3 reports the comparison results of applying FSQP and CSA to design 13/11 filters with respect to different problem formulations and their control parameters. The first column shows the problem formulations and their control parameters. For unconstrained formulation $C(\gamma)$, we only show the result for $\gamma = 100$, since results for $\gamma = 1$ and $\gamma = 10000$ are almost the same. The next four columns show the best solutions obtained, the corresponding values of coding gain, average running time, and average improved PSNR over the four images and five compression ratios with respect to that obtained by Villasenor' 13/11 filter. The next four columns detail average improved PSNRs at five compression ratios over the four test images.

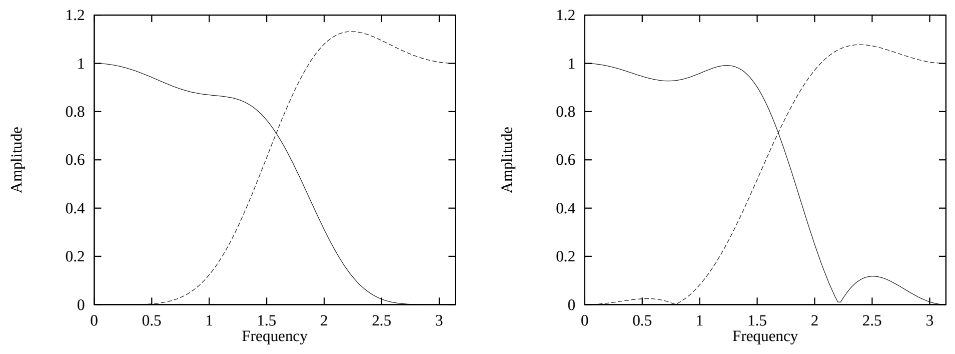
CSA is able to find better solutions than FSQP in terms of the objective values using formulation $A(w_f/w_c)$ of different weights w_f and w_c and for formulation $B(\beta = -1.0\ dB)$.

Table 6.2: PSNRs (dB) for Daubechies' 9/7 and our best 9/7 filters. Numbers in bold represent improved PSNRs.

Compression	Lena image			Barbara image			Goldhill image			Mandrill image		
Ratio	D's 9/7 PSNR	our 9/7 PSNR	Diff PSNR	D's 9/7 PSNR	our 9/7 PSNR	Diff PSNR	D's 9/7 PSNR	our 9/7 PSNR	Diff PSNR	D's 9/7 PSNR	our 9/7 PSNR	Diff PSNR
4:1	43.71	43.75	**0.04**	40.60	40.90	**0.30**	41.12	41.10	-0.02	35.35	35.30	-0.05
8:1	39.42	39.43	**0.01**	34.56	34.70	**0.14**	35.94	35.93	-0.01	29.76	29.78	**0.02**
16:1	36.18	36.19	**0.01**	29.54	30.12	**0.58**	32.62	32.58	-0.04	26.37	26.35	-0.02
32:1	33.17	33.19	**0.02**	26.65	26.69	**0.04**	30.08	30.07	-0.01	24.18	24.32	**0.14**
64:1	30.23	30.33	**0.10**	24.31	24.40	**0.09**	28.22	28.20	-0.02	22.94	22.89	-0.05

Table 6.3: Comparison results of applying FSQP and CSA to design 13/11 filters with respect to different problem formulations and their control parameters. All runs were done on a Pentium-III 450-Mhz computer under Solaris 7. Numbers in bold represent the best solution in terms of average improved PSNR.

Methods	Formulations	best sol'n	coding gain (dB)	average time (sec.)	average imp'd PSNR	Average Improved PSNRs (dB) compression ratio 4:1	8:1	16:1	32:1	64:1
FSQP (100 runs)	$A(w_f/w_c = 1.0/1.0)$	0.0514	5.961	0.39	0.029 dB	0.072	0.042	0.078	-0.006	-0.042
	$A(w_f/w_c = 1.0/0.0)$	0.0474	5.957	0.38	0.028 dB	0.068	0.041	0.079	-0.004	-0.043
	$A(w_f/w_c = 0.0/1.0)$	0.00326	6.414	5.08	-4.21 dB	-5.32	-4.57	-4.32	-3.52	-3.29
	$B(\beta = 0.0\ dB)$	0.0474	5.957	0.59	0.028 dB	0.068	0.041	0.079	-0.004	-0.043
	$B(\beta = +1.0\ dB)$	0.189	6.639	1.99	-0.54 dB	-0.49	-0.87	-0.54	-0.52	-0.30
	$B(\beta = -1.0\ dB)$	0.0474	5.957	0.47	0.028 dB	0.068	0.041	0.079	-0.004	-0.043
	$C(\gamma = 100)$	0.0165	6.045	0.143	-2.69 dB	-6.69	-3.33	-1.74	-1.06	-0.649
CSA (100 runs)	$A(w_f/w_c = 1.0/1.0)$	0.0488	5.923	610.9	0.10 dB	0.185	0.141	0.123	0.028	0.026
	$\mathbf{A(w_f/w_c = 1.0/0.0)}$	**0.0453**	**5.908**	**530.8**	**0.106 dB**	0.186	0.138	0.139	0.027	0.039
	$A(w_f/w_c = 0.0/1.0)$	0.00259	6.912	125.0	-2.18 dB	-2.44	-2.38	-2.40	-1.98	-1.71
	$B(\beta = 0.0\ dB)$	0.0474	5.957	797.9	0.028 dB	0.068	0.041	0.079	-0.004	-0.043
	$B(\beta = +1.0\ dB)$	0.189	6.639	792.7	-0.54 dB	-0.49	-0.87	-0.54	-0.52	-0.30
	$B(\beta = -1.0\ dB)$	0.0459	5.904	791.1	0.103 dB	0.186	0.132	0.139	0.019	0.041
	$C(\gamma = 100)$	0.0165	6.045	423.0	-2.69 dB	-6.69	-3.33	-1.74	-1.06	-0.649



(a) Villasenor' 13/11 filter (b) $A(w_f/w_c = 1.0/0.0)$

Figure 6.5: Frequency amplitudes of Villasenor's 13/11 filter and our best 13/11 filter obtained by solving $A(w_f/w_c = 1.0/0.0)$ using CSA.

The best improved PSNR (0.106 dB) is achieved by the 13/11 filter designed by CSA based on formulation $A(w_f/w_c = 1.0/0.0)$, whose coding gain is 5.908 dB. We call this filter our best 13/11 filter. Although the 13/11 filter obtained by CSA solving $A(w_f/w_c = 0.0/1.0)$ has much higher coding gain (6.912 dB), its coding quality is much worse.

The amplitude responses of Villasenor's 13/11 filter and our best 13/11 filter are shown in Figure 6.5. Table 6.4 shows the coding quality on four images using Villasenor's 13/11 filter whose coding gain is 5.639 dB and our best 13/11 filter. For every image, we give PSNRs for both Villasenor's 13/11 and our best 13/11 filters and their difference. Our best 13/11 filter is almost constantly better than Villasenor's 13/11 filter with improvement up to 0.25 dB.

6.7 Summary

In this chapter, we have studied filter design issue in subband image coding, whose goal is to achieve a better objective measure PSNR for compressed images. We integrated the design criteria from both signal processing and wavelet theory. Those criteria consist of coding gain,

Table 6.4: PSNRs (dB) for Villasenor's 13/11 and our best 13/11 filters. Numbers in bold represent improved PSNRs.

Compression	Lena image			Barbara image			Goldhill image			Mandrill image		
Ratio	V's 13/11 PSNR	our 13/11 PSNR	Diff PSNR	V's 13/11 PSNR	our 13/11 PSNR	Diff PSNR	V's 13/11 PSNR	our 13/11 PSNR	Diff PSNR	V's 13/11 PSNR	our 13/11 PSNR	Diff PSNR
4:1	43.72	43.93	**0.21**	40.96	41.12	**0.16**	40.86	41.05	**0.19**	35.21	35.40	**0.19**
8:1	39.40	39.44	**0.04**	34.82	35.02	**0.20**	35.67	35.92	**0.25**	29.87	29.94	**0.07**
16:1	36.16	36.16	0.00	30.18	30.37	**0.19**	32.48	32.61	**0.13**	26.17	26.41	**0.24**
32:1	33.14	33.11	-0.03	26.76	26.89	**0.13**	29.93	30.05	**0.12**	24.30	24.19	-0.11
64:1	30.14	30.24	**0.10**	24.43	24.36	-0.07	28.14	28.21	**0.07**	22.89	22.94	**0.05**

frequency selectivity, perfect reconstruction (PR), linear phase (LP), and wavelet regularity. Hence, filter designs are multi-objective optimization problems.

We have investigated different formulations for filter design that include two constrained NLPs and one unconstrained NLP, and then have solved them using FSQP [226] and CSA developed in Chapters 3 and 4. We have demonstrated the performance of CSA by improving the coding quality in terms of PSNRs, when compared with that of some best-known subband or wavelet filters.

Chapter 7

Conclusions and Future Work

7.1 Summary of Accomplished Research

A large number of applications in engineering, decision science, and operations research can be formulated as constrained *nonlinear programming problems* (NLPs). Improved solutions to these applications have significant impacts on system performances. The major work of this thesis is on the development of a stochastic global optimization method, called constrained simulated annealing (CSA), with asymptotic convergence for solving continuous, discrete, and mixed-integer constrained NLPs. Using this method, we have improved existing optimization methods in terms of solution quality for solving continuous NLPs and have also solved derived discrete and mixed-integer NLPs that cannot be solved efficiently by existing methods. We summarize the major contributions of this thesis as follows.

- We have developed constrained simulated annealing (CSA), a stochastic global optimization algorithm that converges to a CGM_{dn} with probability one in solving discrete constrained NLPs. This algorithm is based on the first-order necessary and sufficient conditions in the theory of discrete constrained optimization using Lagrange multipliers. These conditions are important because they establish a one-to-one correspondence between CLM_{dn} and discrete saddle points, thereby allowing constrained global optimization to be accomplished by finding a saddle point with the minimum objective value. To find such a saddle point, CSA carries out both probabilistic descents of the discrete augmented-Lagrangian function in the original-variable subspace and probabilistic ascents in the Lagrange-multiplier subspace. By modeling CSA as an inhomogeneous Markov chain, we have proved that CSA converges asymptotically to a CGM_{dn} with probability one. By achieving asymptotic convergence, CSA is one of the major developments in nonlinear *constrained* global optimization today and complements simulated annealing (SA) in nonlinear *unconstrained* global optimization.

- We have investigated various strategies used in CSA, consisting of neighborhood sizes and types, distributions to control the way that samples are drawn, acceptance probabilities, as well as cooling schedules, and have studied their tradeoffs for solving discrete, continuous, and mixed-integer constrained NLPs. Finally, we have implemented an optimization software package based on CSA and the best combination of strategies found in our evaluations.

- Finally, we have applied CSA to solve three sets of nonlinear constrained benchmarks and found improved solutions over existing optimization methods. We have also applied CSA to design filters in subband image coding and have obtained better quality in coding real images when compared with the results of some best-known filters.

7.2 Future Work

In this section, we describe some possible improvements and work for CSA in the future.

- *Handling difficult equality constraints.* As mentioned before, CSA is weak in solving problems with a large number of equality constraints. Since CSA randomly generates trial points and is sampling based, it has difficulty or takes a long time to exactly hit points that satisfy many equality constraints. We plan to address this issue by studying techniques to generate better samples using Bayesian analysis, incorporating derivative information into sampling, and combining techniques in SQP into CSA.

- *Development of any time algorithms.* Another difficulty in using CSA is to determine an optimal cooling schedule that allows a CGM_{dn} to be found in the shortest average amount of time. Such a schedule does not generally correspond to the case in which the success probability in one run of CSA is the highest, because the highest success probability in one run requires a cooling schedule that approaches infinity. Rather, the optimal schedule happens when CSA is allowed to be run multiple times, and the average total time of the multiple runs is the shortest. If each run is too short, it will have very little chance of finding a CGM_{dn}. This leads to a very large number of runs in order to find a CGM_{dn} and, thus, a long average total completion time. In contrast, if each run is too long, it will have a high probability of finding a CGM_{dn} but a very long completion time. Hence, the average total completion time of a few long runs of CSA will be still very long. An *optimal cooling schedule* is one that has a cooling rate in between and leads to the shortest average total completion time when CSA is run multiple times.

  We plan to design cooling schedules for CSA in such a ways that the average time spent in generating a solution of certain quality is of the same order of magnitude as that of multiple run of the original CSA with an *optimal* cooling schedule. We also plan to design a set of solution targets that allow CSA to generate improved solutions as more time is spent, eventually finding a CGM_{dn}. Both leads to an any time version of CSA. Wah and Chen [198] have developed one version of any-time CSA, called CSA_{AT-ID}, by applying iterative deepening to the cooling schedule. They proved that the total time spent on CSA_{AT-ID} is of the same order as the time of one CSA run with the optimal cooling schedule.

- *Extensions to optimization problems with noisy observations.* Up to now, it is implicitly assumed that the objective and constraint functions are evaluated accurately without noise. This, however, may not be the case in real applications whose observations of both the objective and constraints may be noisy. We plan to investigate different ways of generating trial points and accepting these points in such a way that the new algorithm will also converge asymptotically to optimal solutions.

Appendix A

Enhancing First-Order Methods for Continuous Constrained NLPs

In this appendix, we propose two strategies to improve the local search behavior of the first-order method, which is characterized by (2.7), for solving *continuous* constrained NLPs. The objective and constraint functions are assumed to be differentiable. In such a situation, we have $\mathbf{B} \subseteq \mathbf{C} \subseteq \mathbf{A}$, where $\mathbf{A}$ is the set of CLM_{cn}, $\mathbf{B}$ is the set of points satisfying the first-order necessary and second-order sufficient conditions, and $\mathbf{C}$ is the set of saddle points (see Section 2.1.3 for details). The first-order method aims to find a solution in $\mathbf{B}$.

First, we discuss methods to transform inequality constraints into equality constraints, since first-order methods cannot directly solve those NLPs with inequality constraints. A common method, called the *slack-variable method* [125], adds a slack variable to each inequality constraint to transform it into an equality constraint. Its main disadvantage is that, when a search trajectory is inside a feasible region, some satisfied constraints may still pose some effect on the Lagrangian function, leading to possible oscillations or even divergence when a CLM_{cn} lies on the boundary of the feasible region. To overcome this problem, we propose the *MaxQ* method that carries no effect on satisfied constraints. Hence, minimizing the Lagrangian function in the feasible region always leads to a CLM_{cn} of the objective function. We also study some techniques to speed up convergence of the *MaxQ* method.

Second, we develop a dynamic weighting strategy to improve the convergence speed of first-order methods without affecting their solution quality. First-order methods find CLM_{cn} in the Lagrangian space. When constraints are satisfied, they rely on gradient descents in the objective to find high-quality solutions. In contrats, when constraints are violated, these methods rely on gradient ascents in the Lagrange-multiplier space in order to increase the penalties on unsatisfied constraints and to force the constraints into satisfaction. The balance between gradient descents and ascents depends upon the relative weights between the objective and the constraints, which indirectly control the convergence speed and solution quality of first-order methods. Because the choice of proper weights depends on the problem instance and its initial starting points, it is difficult or impossible to determine them in advance. Our dynamic weighting algorithm monitors search progress and adaptively adjusts the weights once imbalance is detected. Using such a strategy, we are able to eliminate divergence, reduce oscillations, and speed up convergence.

A.1 Handling Inequality Constraints

The general approach in Lagrangian methods to handle inequality constraints is to transform them into equivalent equality constraints [31, 125]. A general continuous constrained NLP is defined as

$$\begin{array}{rll} \textit{minimize} & f(x) & \\ \textit{subject to} & h(x) = 0 & x = (x_1, \ldots, x_n) \in R^n \\ & g(x) \le 0 & \end{array} \tag{A.1}$$

where x is a vector of n continuous variables, $h(x) = [h_1(x), \cdots, h_m(x)]^T$ is a vector of m equality constraints, and $g(x) = [g_1(x), \cdots, g_k(x)]^T$ is a vector of k inequality constraints. The objective and constraint functions are assumed to be all differentiable.

A.1.1 Transformations Using Slack Variables

One possible transformation [31, 125] to handle inequality constraint $g_i(x) \leq 0$ is to add a slack variable z_i and transform it into an equality constraint $g_i(x) + z_i^2 = 0$. The corresponding augmented Lagrangian function is

$$L_z(x, z, \lambda, \mu) = f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2 + \sum_{i=1}^{k} \mu_i(g_i(x) + z_i^2) + \frac{1}{2}\sum_{i=1}^{k}[g_i(x) + z_i^2]^2 \quad \text{(A.2)}$$

where slack variables $z = [z_1, z_2, \cdots, z_k]^T$, $\lambda = [\lambda_1, \lambda_2, \cdots, \lambda_m]^T$ is the set of Lagrange multipliers for equality constraints, and $\mu = [\mu_1, \mu_2, \cdots, \mu_k]^T$ is the set of Lagrange multipliers for inequality constraints. $||h(x)||^2 = \sum_{i=1}^{m} h_i^2(x)$.

According to the first-order necessary conditions (2.6), given x and μ, the local minimum condition of L_z with respect to z_i, *i.e.*, $\nabla_{z_i} L_z(x, z, \lambda, \mu) = 0$, yields [31, 125]:

$$z_i^2 = \max(0, -g_i(x) - \mu_i). \quad \text{(A.3)}$$

After substituting it into (A.2) and simplifying it, we obtain

$$L_z(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2 + \frac{1}{2}\sum_{i=1}^{k}\left[\max^2(0, \mu_i + g_i(x)) - \mu_i^2\right] \quad \text{(A.4)}$$

where slack variables z have been removed.

Another possible transformation is to convert $g_i(x) \leq 0$ into $g_i(x) + z_i = 0$ with $z_i > 0$. Such a transformation is seldom used since the number of variables is increased and another form of inequality for slack variable z_i is produced. Therefore, we employ the former in the thesis.

Search Trajectories of the Slack-Variable Method. First-order methods find CLM_{cn} by performing descents in the x subspace and ascents in the λ and μ subspace. The balance between descents and ascents depends on the relative magnitudes of the Lagrange multipliers λ and μ with respect to the objective value. They play a role in balancing the objective $f(x)$ and constraints $h(x)$ and $g(x)$, and accordingly in controlling the convergence speed and the solution quality of first-order methods. At an equilibrium point, the forces due to descents and ascents reach a balance through appropriate Lagrange-multiplier values.

To emphasize how the relative weights affect the convergence speed and solution quality, we introduce an additional weight w into (A.4) and have

$$L_o(x, \lambda, \mu) = w\ f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2 + \frac{1}{2}\sum_{i=1}^{k}\left[\max^2(0, \mu_i + g_i(x)) - \mu_i^2\right] \qquad \text{(A.5)}$$

where $w > 0$ is a weight on the objective. When $w = 1$, $L_o(x, \lambda, \mu) = L_z(x, \lambda, \mu)$, which is the original Lagrangian function. The dynamic system used to solve (A.5) is as follows,

$$\begin{aligned} \frac{d}{dt}x(t) &= -\nabla_x L_o(x(t), \lambda(t), \mu(t)) \\ \frac{d}{dt}\lambda(t) &= \nabla_\lambda L_o(x(t), \lambda(t), \mu(t)) \\ \frac{d}{dt}\mu(t) &= \nabla_\mu L_o(x(t), \lambda(t), \mu(t)) \end{aligned} \qquad \text{(A.6)}$$

Starting from an initial point $(x(t = 0), \mu(t = 0))$, we solve (A.6) using the ordinary differential equation solver *LSODE* [98] and obtain a search trajectory $(x(t), \mu(t))$. When a CLM_{cn} is on the boundary of a feasible region, the dynamic equations (A.6) approach it from both the inside and outside of the feasible region. We observe three behaviors of the search trajectory:

- The trajectory gradually reduces its oscillations and eventually converges;

- The trajectory oscillates within some range but never converges;
- The magnitude of oscillations increases, and the trajectory finally diverges.

Example A.1 To illustrate these three behaviors, consider Problem 2.3 in [68], which is given as follows,

$$\begin{aligned}
\text{Minimize}\quad & 5x_2+5x_3+5x_4+5x_5-x_6-x_7-x_8-x_9-x_{10}-8x_{11}-8x_{12} && \text{(A.7)}\\
& -8x_{13}-x_{14}-5x_2^2-5x_3^2-5x_4^2-5x_5^2 && \\
\text{subject to}\quad & 2x_2+2x_3+8x_{11}+8x_{12}\le 10.0, && 0\le x_i\le 1.0, \quad i=2,3,\cdots,14,\\
& 2x_2+2x_4+8x_{11}+8x_{13}\le 10.0, && 2x_3+2x_4+8x_{12}+8x_{13}\le 10.0,\\
& 8x_{11}-8x_2\le 0.0, && 8x_{12}-8x_3\le 0.0,\\
& 8x_{13}-8x_4\le 0.0, && -2x_5-x_6+8x_{11}\le 0.0,\\
& -2x_7-x_8+8x_{12}\le 0.0, && -2x_9-x_{10}+8x_{13}\le 0.0
\end{aligned}$$

It is a quadratic problem with linear inequality constraints.

We set the initial point at $t=0$ as follows: $x(t=0)$ is at the middle of the search space, *i.e.,* $x_i(t=0)=0.5$ and $\mu(t=0)=0$. The total logical time used for *LSODE* is $t_{max}=10^5$, and is divided into small units of $\delta_t=1.0$, resulting in the maximum number of iterations to be 10^5 $(=t_{max}/\delta_t)$. The stopping condition for (A.6) is the Lyupunov condition,

$$||dx(t)/dt||^2+||d\mu(t)/dt||^2\le\delta=10^{-25} \tag{A.8}$$

The dynamic system stops as it converges or reaches the maximum number of iterations.

When $w=1$, (A.6) diverges quickly into infinity. If we scale the objective by 10 and set $w=1/10$, then the objective value $f(x(t))$ oscillates within the range -17 and -10, while the maximum violation $v_{max}(t)$ is between 0 and 0.4, as shown in Figure A.1. Here, $v_{max}(t)$

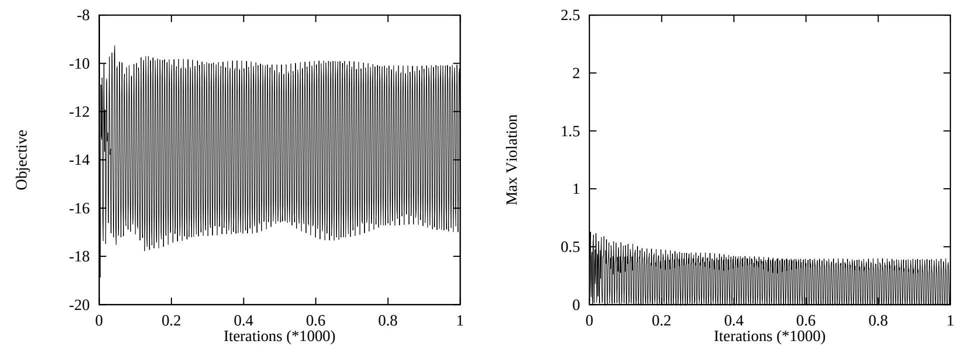


Figure A.1: The objective and maximum violation oscillate using the slack-variable method ($w = 1/10$).

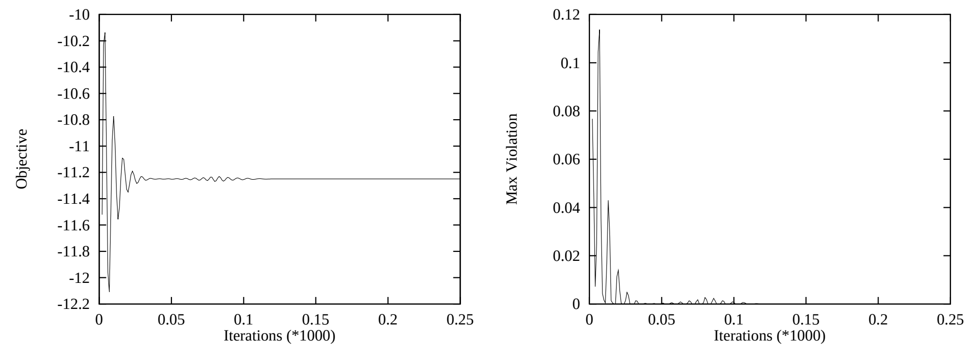


Figure A.2: The objective and maximum violation converge after oscillations subside using the slack-variable method ($w = 1/15$).

at time t is defined as

$$v_{max}(t) = \max_{1 \leq i \leq m, 1 \leq j \leq k} \{|h_i(x(t))|, \max[0, g_j(x(t))]\}. \tag{A.9}$$

If we further reduce w to $1/15$, then the oscillations subside, and the trajectory eventually converges (see Figure A.2).

Intuitively, occurrence of oscillations can be explained as follows. Suppose we start from an infeasible point initially ($t = 0$) where inequality constraint $g_i(x(t = 0))$ is violated, *i.e.*, $g_i(x(t = 0)) > 0$. As the search progresses, the corresponding $\mu_i(t)$ increases and pushes the trajectory towards a feasible region. At some time t, inequality constraint $g_i(x(t)) \leq 0$ is satisfied for current point $x(t)$. At this point, $\frac{d\mu_i(t)}{dt} = max(0, g_i(x) + \mu_i) - \mu_i$, and is negative when $g_i(x) < 0$. Hence, the trajectory decelerates but continues to move into the feasible region even when constraint $g_i(x(t))$ is satisfied. The movement of the trajectory inside the feasible region eventually stops because the CLM_{cn} is on the boundary, and the force due to descents in the objective pushes the trajectory outside the feasible region. Likewise, when the trajectory is outside the feasible region, a force due to the constraints pushes the trajectory inside the feasible region. If these two forces are not well balanced, the search may diverge or oscillate without convergence.

A.1.2 Transformations Using the *MaxQ* Method

To avoid oscillations in the slack variable method, we like the search to converge to a CLM_{cn} without oscillations when it is on the boundary of a feasible region and when the trajectory is outside the feasible region. This is done by our proposed *MaxQ* method that transforms an inequality constraint using

$$g_i(x) \leq 0 \iff p_i(x) = max^{q_i}(0, g_i(x)) = 0 \qquad q_i > 1, \;\; i = 1, \cdots, k \tag{A.10}$$

where q_i are control parameters to be determined later. For a given x, if $g_i(x) > 0$, $max^{q_i}(0, g_i(x)) = [g_i(x)]^{q_i} > 0$. Otherwise, $max^{q_i}(0, g_i(x)) = 0$. The augmented Lagrangian function then becomes

$$\begin{aligned} L_q(x, \lambda, \mu) &= w\ f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2 + \mu^T p(x) + \frac{1}{2}||p(x)||^2 \\ &= w\ f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2 + \sum_{i=1}^{k} \left[\mu_i p_i(x) + \frac{1}{2} p_i^2(x)\right] \end{aligned} \tag{A.11}$$

where $p(x) = [p_1(x), p_2(x), \cdots, p_k(x)]^T$, and w is an additional weight.

Choice of Parameter q_i. It is important to choose suitable control parameters q_i ($i = 1, 2, \cdots, k$), because they affect the convergence speed of first-order methods. One can easily show that, when $q_i \geq 1$, inequality constraint $g_i(x) \leq 0$ is equivalent to equality constraint $p_i(x) = max^{q_i}(0, g_i(x)) = 0$. Suppose that q_i is a constant. When solving (A.11), we need to evaluate partial derivative $\bigtriangledown_x p_i(x)$ where

$$\bigtriangledown_x p_i(x) = q_i max^{q_i - 1}(0, g_i(x)) \bigtriangledown_x g_i(x) = p_i'(x) \bigtriangledown_x g_i(x)$$

If $q_i \leq 1$, $p_i'(x)$ is not continuous, and the derivative of $L_q(x, \lambda, \mu)$ is then not continuous at $g_i(x) = 0$. Because the continuity of derivatives is required by most differential-equation solvers such as *LSODE*, we set $q_i > 1$ in (A.10).

One way of selecting q_i is to have it very close to 1, namely, $q_i \to 1$. At this time, the dynamic system will approach a feasible region slowly because $p_i'(x) \simeq 1$ if $g_i(x) > 0$. It is independent of how far the current point x is away from the feasible region. Thus, a larger control parameter q_i is needed for fast convergence if the current point x is far from the feasible region. In contrast, if we choose $q_i \gg 1$, then $p_i'(x) \simeq 0$ as $g_i(x) \to 0$, meaning slow convergence towards a CLM_{cn} on the boundary of the feasible region.

Taking these facts into account, in order to have fast convergence, we should adapt q_i dynamically as the search goes to a CLM_{cn}. Since different inequality constraints may have different convergence rates to a CLM_{cn}, we associate with each inequality constraint $g_i(x) \leq 0$ its own control parameter q_i. Parameter q_i is updated adaptively based on the value of $g_i(x)$: q_i is large if $g_i(x) \gg 0$, and q_i is gradually reduced to a value approaching 1 when the search is close to a CLM_{cn}. One possible choice of q_i is as follows,

$$q_i(g_i(x)) = \frac{s_0}{1 + exp(-s_1 g_i(x))} \tag{A.12}$$

where $s_0 = 2$ and $s_1 > 0$ are two parameters that control the shape of function $q_i(x)$. When $g_i(x)$ approaches 0, q_i will approach 1. The dynamic equations to solve (A.11) are defined as

$$\begin{aligned}
\frac{d}{dt}x(t) &= -\nabla_x L_q(x(t),\lambda(t),\mu(t)) \\
&= -\,w\nabla_x f(x) - \sum_{i=1}^{m}(\lambda_i + h_i(x))\nabla_x h_i(x) - \sum_{i=1}^{k}(\mu_i + p_i(x))\nabla_x p_i(x) \\
\frac{d}{dt}\lambda_i(t) &= \nabla_{\lambda_i} L_q(x(t),\lambda(t),\mu(t)) = h_i(x) \qquad \text{(A.13)} \\
\frac{d}{dt}\mu_i(t) &= \nabla_{\mu_i} L_q(x(t),\lambda(t),\mu(t)) = p_i(x)
\end{aligned}$$

where

$$\bigtriangledown_x p_i(x) = \left[q_i'(g_i(x))p_i(x)Ln\ max(0,g_i(x)) + q_i(g_i(x))max^{q_i(g_i(x))-1}(0,g_i(x))\right]\bigtriangledown_x g_i(x)$$

Note that (A.11) is similar to (A.5) in the sense that both use the max function. The main difference is that (A.11) avoids the case of (A.5) in which inequality constraint $g_i(x(t)) \le 0$ is satisfied at time t but $g_i(x(t))$ still appears in the Lagrangian function. When $g_i(x(t))$ is satisfied, it is meaningful to minimize $f(x)$ independent of the value of $g_i(x(t))$.

Convergence Properties. There are two kinds of CLM_{cn} x^* as shown in Figure A.3 based on their locations. When CLM_{cn} x^* is within a feasible region, *i.e.*, $g_i(x^*) < 0$ (see Figure A.3a), $p_i(x^*) = 0$, and $\bigtriangledown_x f(x^*) = 0$. This implies that x^*, an equilibrium point of dynamic system (A.13), is given by

$$\frac{d}{dt}x(t) = 0 \quad \text{and} \quad \frac{d}{dt}\lambda(t) = 0 \quad \text{and} \quad \frac{d}{dt}\mu(t) = 0 \qquad \text{(A.14)}$$

Thus, the trajectory controlled by (A.13) converges to this CLM_{cn} x^*.

When CLM_{cn} x^* is on the boundary of the feasible region, as shown in Figure A.3b, it will be approached asymptotically from outside the feasible region (*e.g.* from the right side

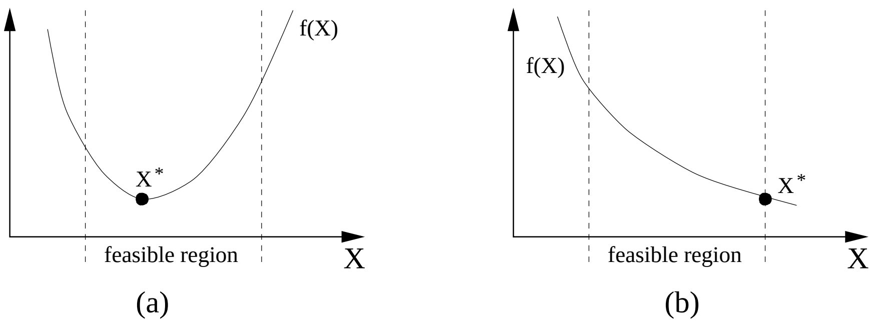


Figure A.3: Relationship between CLM_{cn} and feasible region. (a) CLM_{cn} is within the feasible region; (b) CLM_{cn} is on the boundary of feasible region.

of Figure A.3b). To prove this, we only need to show that x^* is asymptotically a regular point of constraints $p_i(x) = 0$ [31, 125], because inequality constraint $g_i(x) \le 0$ has been transformed into an equivalent equality constraint $p_i(x) = 0$.

Because x^* is on the boundary of the feasible region, $g_i(x^*) = 0$ and $p_i(x^*) = 0$. In addition, when $x \longrightarrow x^*$, $g_i(x) \longrightarrow g_i(x^*) = 0$, and $q_i(g_i(x)) \longrightarrow q_i(g_i(x^*)) = 1$. By taking limits, we obtain

$$\lim_{x \to x^*} q_i'(g_i(x))p_i(x)Ln\ max(0, g_i(x)) = 0$$

$$\lim_{x \to x^*} q_i(g_i(x))max^{q_i(g_i(x))-1}(0, g_i(x)) = 1.$$

Therefore, $\lim_{x \to x^*} \bigtriangledown_x p_i(x) = \bigtriangledown_x g_i(x^*)$, which means that asymptotically regular point x^* of $p_i(x)$ is the same as that of the original constraint $g_i(x) \le 0$. Since a CLM_{cn}, if exists, must be a regular point of $g_i(x)$ [125], x^* can be asymptotically reached by dynamic system (A.13).

Choice of Control Parameter for *LSODE*. When we prove convergence to a CLM_{cn} of the *MaxQ* method, we assume that q_i takes the form in (A.12) where $s_0 = 2$. With this choice, $\nabla_x p_i(x)$ changes very quickly from 1 to 0 near the CLM_{cn} as $g_i(x) \longrightarrow 0$, making it difficult for *LSODE* to find a suitable step size in order to reach the CLM_{cn}. To make the gradient change smoothly, we set $s_0 = 2.5$ or $s_0 = 3.0$, and set s_1 to satisfy $q_i(g_i(x)) = 2$ when $g_i(x) = 1$. Hence, $s_1 = -Ln[s_0/2 - 1]$.

Dynamic Switch of Inequality Constraints to Equality Constraints. As discussed above, if solution x^* is on the boundary of a feasible region, *i.e.*, when some $g_i(x^*)$ equals zero, then dynamic system (A.13) is unable to be at this point exactly. Although this solution can be approached with any precision, it may take a long time even if q_i is changed dynamically. This happens because the violation of constraint $g_i(x^*)$ is small when current point $x(t)$ is close to x^*. Consequently, increments of Lagrange multiplier μ_i will be small, leading to slow convergence towards the CLM_{cn}.

Suppose we know that some $g_j(x^*) = 0$ for a given solution x^*. In this case, faster convergence can be achieved if we treat $g_j(x)$ as an equality constraint, *i.e.,* $g_j(x) = 0$, rather than an inequality constraint, $g_j(x) \leq 0$. The difficulty, however, is that it is impossible to know in advance which inequality constraints $g_j(x) \leq 0$ will satisfy the boundary condition, $g_j(x) = 0$, for the solution on the boundary.

This suggests a good heuristic that, if $x(t)$ is very close to the boundary of a particular inequality constraint $g_j(x(t))$, *i.e.* $g_j(x(t))$ is positive and close to zero, and if the convergence of this constraint is slow, it is very likely that the CLM_{cn} is on the boundary of this constraint $g_j(x) \leq 0$. At this point, we can use equality constraint $g_j(x) = 0$ to replace inequality constraint $g_j(x) \leq 0$ to improve the convergence rate. This scheme is more than the concept of active sets [31, 125] because specific features of the *MaxQ* method are utilized.

Assuming that we solve the dynamic system using *LSODE*, let x and x_0 be the points of two successive iterations. Switching inequality constraint $g_j(x) \leq 0$ to equality constraint $g_j(x) = 0$ takes place if the following two conditions are satisfied.

- Point x changes very little for a window of iterations in the dynamic system. This can be measured by $\#\{max_i|x_i - x_{0i}|/max_j|x_j| < \delta\} \geq 10$ ($\delta = 10^{-4}$ in our experiments).

- The dynamic system converges to the boundary when $g_j(x)$ is very close to zero; that is, $0 < g_j(x) < \epsilon = 10^{-4}$.

Both conditions are very important. (a) Without the first condition, trajectory $x(t)$ that occasionally passes the boundary of $g_j(x(t))$ may cause some inequality constraints to be switched erroneously. Hence, it is important to make sure that the trajectory really changes by very little during a period of time. (b) The second condition guarantees that only those inequality constraints very close to the boundary can be switched into equality ones. Note that a dynamic switch may happen to many inequality constraints at the same time as long as they satisfy these two conditions.

Once inequality constraint $g_j(x) \leq 0$ is switched to an equality constraint, the terrain of the Lagrangian function $L_q(x, \lambda, \mu)$ is changed and may be totally different. In order to maintain the search direction in the x subspace at the right time of switch, we have to adjust Lagrange multiplier μ_j. Let the point just before the switch be (x, μ_j), and the Lagrangian term associated with inequality constraint $g_j(x) \leq 0$ is

$$L_j(x, \mu_j) = \mu_j max^{q_j}(0, g_j(x)) + \frac{1}{2} max^{2q_j}(0, g_j(x)) = \mu_j g_j^{q_j}(x) + \frac{1}{2} g_j^{2q_j}(x)$$

based on the second switch conditions. The derivative of $L_j(x, \mu_j)$ with respect to x and μ_j are

$$\begin{aligned}
\nabla_x L_j(x, \mu_j) &= \left[\mu_j g_j^{q_j-1}(x) + g_j^{2q_j-1}(x)\right] \left[q_j + q_j'(x) g_j(x) Ln\ g_j(x)\right] \nabla_x g_j(x) \\
\nabla_{\mu_j} L_j(x, \mu_j) &= g_j^{q_j}(x)
\end{aligned}$$

Let $(x, \hat{\mu}_j)$ be the point after the switch. This means that we apply equality constraint $g_j(x) = 0$ at the current point $(x, \hat{\mu}_j)$ where inequality constraint $g_j(x) \leq 0$ was before. The Lagrangian term related to $g_j(x) = 0$ is

$$\hat{L}_j(x, \hat{\mu}_j) = \hat{\mu}_j g_j(x) + \frac{1}{2} g_j^2(x)$$

and the derivative of $\hat{L}_j(x, \hat{\mu}_j)$ with respect to x and $\hat{\mu}_j$ are

$$\begin{aligned} \nabla_x \hat{L}_j(x, \hat{\mu}_j) &= (\hat{\mu}_j + g_j(x)) \nabla_x g_j(x) \\ \nabla_{\hat{\mu}_j} \hat{L}_j(x, \hat{\mu}_j) &= g_j(x) \end{aligned}$$

The search direction in the Lagrange-multiplier space changes very little, because control parameter q_j is close to 1 at the time of the switch, and then $\nabla_{\mu_j} L_j(x, \mu_j) \simeq \nabla_{\hat{\mu}_j} \hat{L}_j(x, \hat{\mu}_j)$, independent of the value μ_j. To retain the search direction in the x subspace, we set $\nabla_x L_j(x, \mu_j) = \nabla_x \hat{L}_j(x, \hat{\mu}_j)$ and get

$$\hat{\mu}_j = \left[\mu_j g_j^{q_j - 1}(x) + g_j^{2q_j - 1}(x) \right] \left[q_j + q_j'(x) g_j(x) Ln\ g_j(x) \right] - g_j(x). \tag{A.15}$$

Illustration of the MaxQ Method. To show how *MaxQ* avoids divergence and oscillations that occur in the slack-variable method, we consider the same problem 2.3 in [68]. The starting point is in the middle of the search space, the same as that used in the slack-variable method. The same three cases were tested here: no scaling, scaling the objective by 10 (*i.e.*, $w = 1/10$), and scaling the objective by 15. All of them converge with similar behavior. Figure A.4 shows the second case. Obviously, the *MaxQ* method has smoother and better convergence property as compared to the slack-variable method.

The solution to Problem 2.3 is on the boundary of a feasible region as shown in Figure A.3b. Since all the Lagrange multipliers are zero initially, only objective function $f(x)$ has

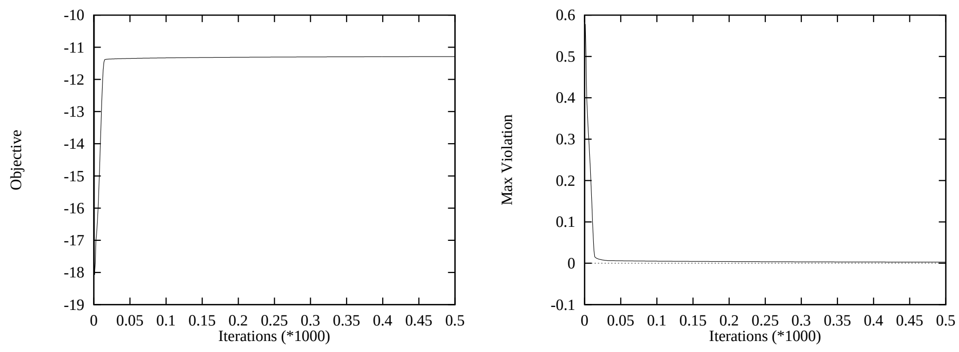


Figure A.4: The objective and maximum violation converge smoothly using *MaxQ* for Problem 2.3 with $w = 1/10$.

effect in the Lagrangian function, causing the trajectory to move away from the feasible region. The Lagrange multipliers then increase, pushing the trajectory back towards the boundary. Hence, the objective value increases, and the value of the maximum violation decreases (see Figure A.4).

Note that there is a gap between our current implementation of the *MaxQ* method and its theoretic result, in the sense that the analytic proof requires $s_0 = 2$ in (A.12) but *LSODE* uses $s_0 = 2.5$ or 3.0. This gap causes the *MaxQ* method to converge slowly sometimes, like the case shown in Figure A.4 that requires about 94,000 iterations. In this example, the *MaxQ* method reduces constraint violations very quickly in the beginning but slowly afterwards.

This problem can be solved by two approaches. The first is to use another differential equation solver that is insensitive to quick changes of gradients, making the analytic result hold during the search. But no such solver can be found. The second is to adaptively adjust the relative weight w between the objective and the constraints during the search. As soon as we detect slow convergence, w is adjusted accordingly.

A.1.3 Comparisons of the Slack-variable and MaxQ Methods

Two approaches have been investigated to cope with inequality constraints in first-order methods. For a general continuous constrained NLP (A.1), the augmented Lagrangian function using the slack-variable method is defined as

$$L_{slack}(x, \lambda, \mu) = w\ f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2 + \frac{1}{2}\sum_{i=1}^{k}\left[\max^2(0, \mu_i + g_i(x)) - \mu_i^2\right] \quad (A.16)$$

and that using the *MaxQ* method is defined as

$$L_{maxq}(x, \lambda, \mu) = w\ f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2 + \mu^T p(x) + \frac{1}{2}||p(x)||^2 \quad (A.17)$$

where $p(x) = [p_1(x), p_2(x), \cdots, p_k(x)]^T$, and $p_i(x)$ is given by (A.10).

The dynamic system searches solutions by performing descents in the variable x subspace and ascents in the Lagrange-multiplier λ and μ subspace. This search is governed by a system of ordinary differential equations,

$$\begin{aligned}\frac{d}{dt}x(t) &= -\nabla_x L_b(x(t), \lambda(t), \mu(t)) \\ \frac{d}{dt}\lambda(t) &= \nabla_\lambda L_b(x(t), \lambda(t), \mu(t)) \\ \frac{d}{dt}\mu(t) &= \nabla_\mu L_b(x(t), \lambda(t), \mu(t))\end{aligned} \quad (A.18)$$

where augmented Lagrangian function $L_b(x, \lambda, \mu)$ can be either $L_{slack}(x, \lambda, \mu)$ or $L_{maxq}(x, \lambda, \mu)$. The stopping condition is

$$||dx(t)/dt||^2 + ||d\lambda(t)/dt||^2 + ||d\mu(t)/dt||^2 \leq \delta = 10^{-25} \quad (A.19)$$

Therefore, dynamic system (A.18) stops when stopping condition (A.19) is satisfied or when it reaches the maximum number of iterations.

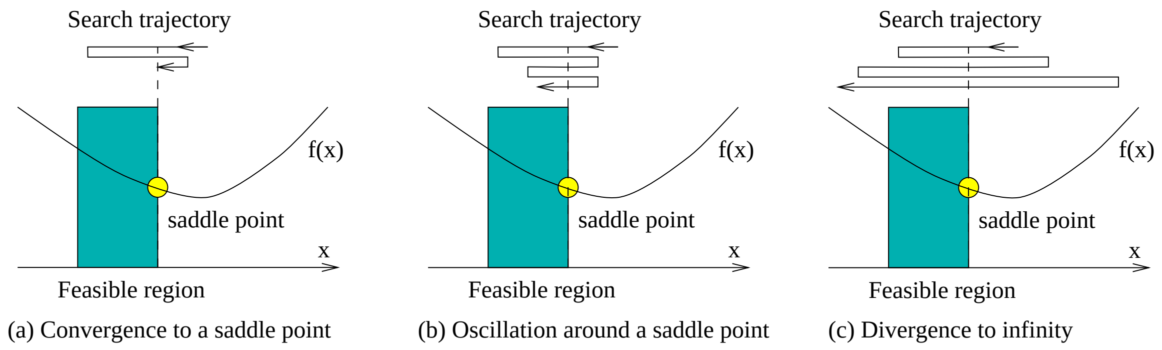


Figure A.5: Illustration of the search trajectories of the slack-variable method when CLM_{cn} is on the boundary of a feasible region. The search may (a) converge to a CLM_{cn}, (b) oscillate around a CLM_{cn}, or (c) diverge to infinity.

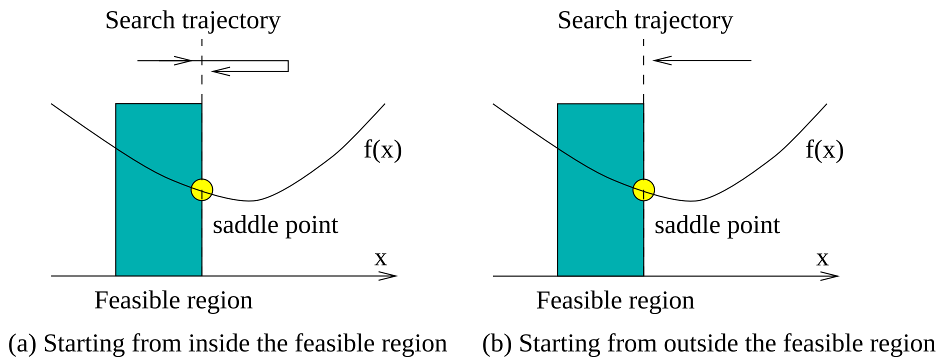


Figure A.6: Illustration of the search trajectory of the MaxQ method when CLM_{cn} is on the boundary of a feasible region. The search starts from inside (left) or outside (right) the feasible region. In both cases, the search trajectory eventually approaches the CLM_{cn} from outside the feasible region.

The convergence behavior of the slack-variable method is different from that of the *MaxQ* method when the CLM_{cn} is on the boundary of a feasible region, even if they use the same starting point $(x(t=0), \lambda(t=0), \mu(t=0))$. The slack-variable method approaches the CLM_{cn} from both inside and outside the feasible region. The search trajectory oscillates around the CLM_{cn} and converges if the magnitude of oscillations dies down. In some situations, the magnitude of oscillations does not reduce, or even increases, causing the search to diverge eventually. Depending on the initial point and the relative magnitude of the objective function and the constraints, the search trajectory could end up in one of three different states as illustrated in Figure A.5.

The *MaxQ* method, however, avoid the problems of oscillations and divergence when the CLM_{cn} is on the boundary of a feasible region. The *MaxQ* method approaches this CLM_{cn} asymptotically from outside the feasible region. The closer it gets to the boundary, the slower it goes. Figure A.6 illustrates the situations when the search starts from either inside or outside a feasible region. In both cases, the search trajectory eventually approaches the CLM_{cn} from outside the feasible region. The *MaxQ* method may converge slowly when the search is very close to the CLM_{cn} on the boundary of a feasible region.

We will show comparison results of the slack-variable method and the *MaxQ* method in Section A.3.

A.2 Dynamic Weighting Strategy

In the last section, we have studied two methods to handle inequality constraints. The slack-variable method is sensitive to the relative weights between the objective and the constraints, and may diverge, oscillate or converge. Although the *MaxQ* method does not diverge and oscillate, it may converge slowly. To solve the problems of divergence, oscillation, and slow convergence, we propose a dynamic weighting strategy in this section.

Combining Lagrangian functions (A.16) and (A.17), we have a general Lagrangian function as follows:

$$L_b(x, \lambda, \mu) = w\ f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2 + L_{ineq}(\mu, x) \tag{A.20}$$

where w is the relative weight, λ is the Lagrange multiplier for equality constraints, and μ for inequality constraints. $L_{ineq}(\mu, x)$ depends on the way to deal with inequality constraints. It can be either the slack-variable method or the *MaxQ* method. The corresponding dynamic system is defined by (A.18).

The convergence behavior can be affected significantly by the choice of w. The best choice is problem-instance dependent, and seems impossible to select a priori. Even for the same problem, it is depends on starting points. The dynamic weighting strategy tries to adapt w based on the behavior of dynamic system (A.18) in order to obtain high-quality solutions with short convergence time.

In general, before a trajectory reaches an equilibrium point, changing the weight of the objective may speed up or delay convergence. Further, when the trajectory is at an equilibrium point, changing the weight of the objective may bring the trajectory out of the equilibrium point. In this section, we exploit the first property by designing weight adaptation methods in order to speed up convergence without affecting solution quality.

A.2.1 General Weight-Adaptation Strategy

Figure A.7 outlines the algorithm, whose basic idea is to first estimate the initial weight $w(t = 0)$ (Step 1), measure the performance metrics of the search trajectory $(x(t), \lambda(t), \mu(t))$ periodically, and adapt $w(t)$ to improve convergence time or solution quality.

```
1.  Set control parameters:
        time interval $\delta_t$,
        initial weight $w(t = 0)$,
        maximum number of iterations $i_{max}$;
2.  Set window size $N_w = 100$ or 10;
3.  $j := 1$;        /* $j$ is the iteration number */
4.  while $j \le i_{max}$ and stopping condition is not satisfied do
5.      advance search trajectory by $\delta_t$ to get to $(x_j, \lambda_j, \mu_j)$;
6.      if trajectory diverges then
            reduce $w$; restart the algorithm by going to Step 2;
        end if
7.      record useful information for calculating performance metrics;
8.      if $((j \;\; mod \;\; N_w) == 0)$ then
            /* Test whether $w$ should be modified at the end of a window */
9.          compute performance metrics based on data collected;
10.         change $w$ when the conditions defined in Figure A.8 are satisfied;
        end if
11. end while
```

Figure A.7: Pseudo code for a new dynamic weight-adaptation algorithm

Let t_{max} be the total (logical) time for the search, and t_{max} be divided into small units of time δ_t so that the maximum number of iterations is t_{max}/δ_t. Further, assume a stopping condition if the search were to stop before t_{max} (Step 4). Given a starting point $x(t = 0)$, we set the initial values of the Lagrange multipliers to be zero, *i.e.*, $\lambda(t = 0) = \mu(t = 0) = 0$. Let (x_i, λ_i, μ_i) be the point of the i^{th} iteration, and $v_{max,i}$ be its maximum violation defined by (A.9).

To monitor the progress of the search trajectory, we divide time into non-overlapping windows of size N_w iterations each (Step 2). In each window, we compute some metrics to measure the progress of the search relative to that of previous windows. For the u^{th} window ($u = 1, 2, \cdots$), we calculate the average (or median) of $v_{max,i}$ over all the iterations in the window,

$$\bar{v}_u = \frac{1}{N_w} \sum_{j=(u-1)N_w+1}^{uN_w} v_{max,j} \qquad \text{or} \qquad \bar{v}_u = \underset{\substack{(u-1)N_w+1 \\ \leq j \leq uN_w}}{\text{median}} \{v_{max,j}\} \tag{A.21}$$

and the average (or median) of the objective $f(x)$.

$$\bar{f}_u = \frac{1}{N_w} \sum_{j=(u-1)N_w+1}^{uN_w} f(x_j) \qquad \text{or} \qquad \bar{f}_u = \underset{\substack{(u-1)N_w+1 \\ \leq j \leq uN_w}}{\text{median}} \{f(x_j)\} \tag{A.22}$$

During the search, we apply an algorithm (*e.g. LSODE*) to solve dynamic system (A.18), and advance the trajectory by time interval δ_t in each iteration in order to arrive at point (x_j, λ_j, μ_j) (Step 5).

At this point, we test whether the trajectory diverges or not (Step 6). Divergence happens when the maximum violation $v_{max,j}$ is larger than an extremely large value (e.g. 10^{20}). If it happens, we reduce w by a large amount, say $w \Longleftarrow w/10$, and restart the algorithm. In each iteration, we also record some statistics, such as $v_{max,j}$ and $f(x_j)$, which will be used to calculate the performance metrics for each window (Step 7). At the end of each window or every N_w iterations (Step 8), we decide whether to update w based on the performance metrics (A.21) and (A.22) (Step 9). In our current implementation, we use the averages (or medians) of maximum violation $v_{max,i}$ and objective $f(x_j)$. In general, other application-specific metrics can be used, such as the number of oscillations of the trajectory in continuous NLPs. Based on these measurements, we adjust w accordingly (Step 10).

A.2.2 Dynamic Weight Adaptation for Continuous Constrained NLPs

To understand how weights should be updated in Step 10, we examine all the possible behaviors of the resulting search trajectory in successive windows. We have identified three possible cases.

First, the trajectory does not stay within a feasible region, but goes from one feasible region to another through an infeasible region. During this time, the maximum violation $v_{max,i}$ is zero when the trajectory is in a feasible region, increased when it travels to an infeasible region, and decreased when entering another feasible region. No oscillations or very small number of oscillations will be observed because oscillations usually occur around an equilibrium point of the dynamic system. In this case, no change of w is required.

Second, the search trajectory oscillates around an equilibrium point of a feasible region for the slack variable method. This can be detected when the number of oscillations in each window is larger than some threshold. Figures A.1 and A.2 show two typical types of oscillations. To determine whether the oscillations will subside eventually, we compute $\bar{v}_u - \bar{v}_{u+1}$, the reduction of the average values of the maximum violation $v_{max,i}$, for two successive windows u and $u+1$. If the difference is not reduced reasonably, we assume that the trajectory does not converge, and decrease w to give the constraints more weights accordingly. For the case of using the *MaxQ* method, there is no oscillation (see Figure A.4 for example), and its slow convergence can be detected by measuring $\bar{v}_u - \bar{v}_{u+1}$.

Third, the search trajectory moves very slowly within a feasible region. This happens when w is very small, and the constraints dominate the search process. As a result, the objective value is improved very slowly and may eventually converge to a poor value. This situation is identified if the trajectory remains within a feasible region in two successive

```
Given performance measurements for the $m^{th}$ window
        average (or median) of the maximum violation: $\bar{v}_m$
        average (or median) of the objective: $\bar{f}_m$
        number of oscillations: $NO_m$
and application-specific constants $\alpha_0, \alpha_1, \beta_0, \beta_1, \beta_2, \delta, \epsilon$;
1. Increase weight $w \Longleftarrow w/\alpha_0$ **if**
        $(c_{1,1})$ $\bar{v}_{m-1}, \bar{v}_m < \delta$
        $(c_{1,2})$ $\beta_0|\overline{f}_{m-1}| > \overline{f}_{m-1} - \overline{f}_m > \beta_1|\overline{f}_{m-1}|$;
2. Decrease weight $w \Longleftarrow \alpha_1 w$ **if**
        $(c_{2,1})$ $\bar{v}_m \geq \delta$
        $(c_{2,2})$ $\bar{v}_{m-1} - \bar{v}_m \leq \beta_2 \bar{v}_{m-1}$
        $(c_{2,3})$ $NO_m \geq \epsilon$    (for slack variable method).
```

Figure A.8: Weight-adaptation rules (Step 10 of Figure A.7).

windows, and there is little improvement in the objective. Obviously, we need to increase w in order to speed up the improvement of the objective. If the objective remains unchanged, then the trajectory has converged, and no further modification of w is necessary.

Given the above three search behaviors, Figure A.8 shows a comprehensive list of rules to adapt weight w (Step 10). Scaling factors $0 < \alpha_0, \alpha_1 < 1$ control how fast w is updated. Because we use numerical methods to solve the dynamic system defined in (A.18), a trajectory in the u^{th} window is said to satisfy all the constraints when $\bar{v}_u < \delta$, where δ is related to the convergence condition and the required precision. Parameters $0 < \beta_0, \beta_1, \beta_2 < 1$ control, respectively, the degree of improvement over the objective and the degree of reduction of the maximum violation. The number of allowed oscillations is represented by ϵ.

There are two points that need to be emphasized. First, when comparing these metrics between two successive windows $u-1$ and u, both must use the same weight w. Otherwise, the comparison is not meaningful because the terrain may be totally different. Hence, after adapting w, we have to wait at least two windows before changing it again. Second, the weight-adaptation scheme can be applied to dynamic system (A.18) using either the slack-variable method or the *MaxQ* method to handle inequality constraints. The only difference lies in the testing of oscillations, because there are no oscillations in the *MaxQ* method.

Weight w should be increased (Rule 1) when the third convergence behavior occurs. In this case, the trajectory is within a feasible region, and the objective is improved in successive windows. Weight w will be increased when the improvement of the objective in a feasible region is not fast enough, but will remain unchanged when the improvement is beyond an upper bound.

Weight w should be decreased (Rule 2) when we observe the second convergence behavior. The trajectory oscillates around a CLM_{cn} (only for the slack-variable method), while the maximum violation is not reduced quickly or may even have increased.

A.2.3 Illustrative Examples

In this subsection, we show how the dynamic weighting scheme is capable of reducing oscillations and speed up convergence in both the slack-variable and *MaxQ* methods.

Slack-variable method. Recall in Figure A.1 that the trajectory oscillates when $w = \frac{1}{10}$ for Problem 2.3 in [68]. Figure A.9 shows the resulting trajectory and the maximum violation when the dynamic weight-adaptation algorithm is applied on the same problem. We started with $w(t = 0) = \frac{1}{5}$, $\alpha_0 = \alpha_1 = \frac{1}{2}$, $\delta = 10^{-8}$, $\beta_0 = 10^{-2}$, $\beta_1 = 10^{-3}$, and $\beta_2 = 10^{-2}$. In the first window (first $N_w = 100$ iterations), the average $\bar{v}_1 = 4.11$ that increases slightly to $\bar{v}_2 = 4.2$ in the second window. In addition, $\bar{v}_2 \geq \delta$ and $NO_2 \geq 5$. According to the conditions in Figure A.8, w is updated to $\frac{1}{10}$. This change in w leads to significant reduction

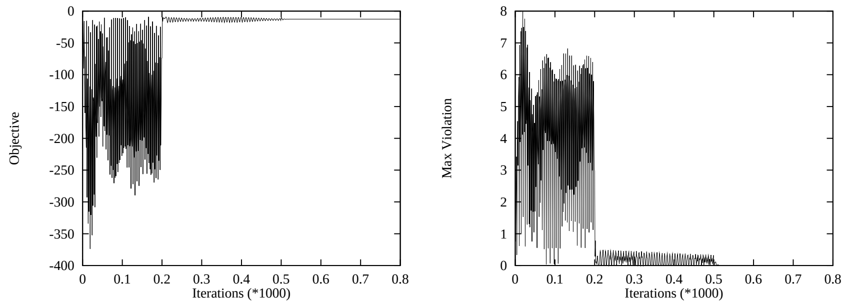


Figure A.9: The objective function and maximum violation first oscillate and then converge using dynamic weight adaptation.

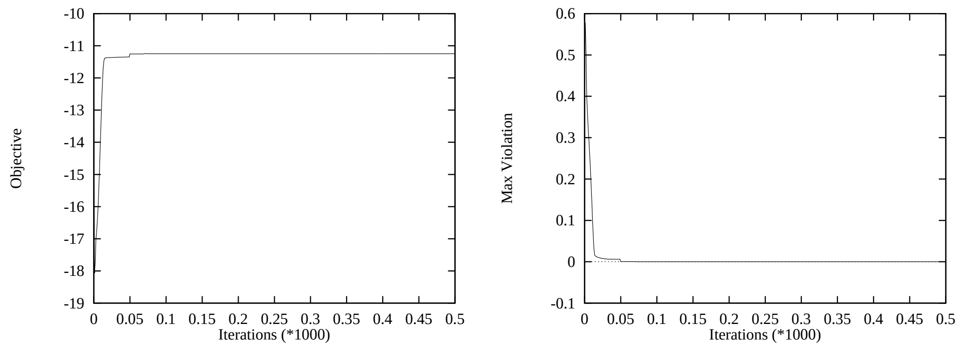


Figure A.10: The objective and maximum violation converge after 756 iterations for *MaxQ* using dynamic weight adaptation (initial $w = \frac{1}{10}$).

in the maximum violation in the third window. Weight w remains unchanged between the third and the fifth windows. At the end of the fifth window, the maximum violation changes very little. Hence, the algorithm reduces w to $\frac{1}{20}$, causing the trajectory to converge quickly.

***MaxQ* method.** Corresponding to Figure A.4, we start from the initial weight $w(t = 0) = 1/10$ and the same starting point $(x(0), \lambda(0), \mu(0))$. Figure A.10 shows the resulting search profile, in which the search converges using only 756 iterations, which is significantly better than the 94,000 iterations without weight adaptation. It is important to note that solution quality is the same as that in Figure A.4, and both obtain the objective value -11.25 when the search converges.

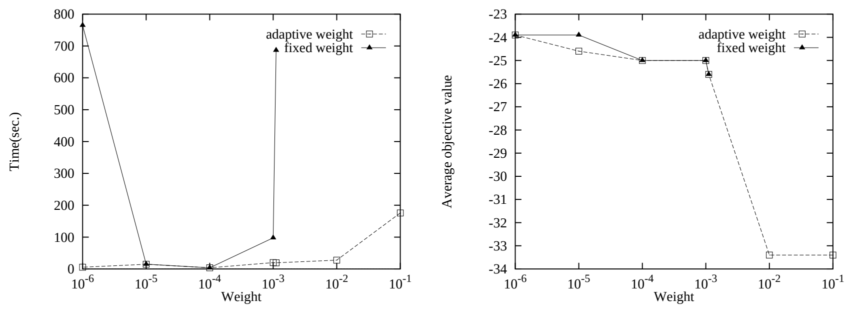
A.2.4 Comparison of Static and Dynamic Weighting

In a continuous constrained benchmark suite [68], we selected five problems with identifiers 2.6, 3.4, 4.6, 5.4 and 6.4 from different classes to test our algorithm. In each problem, we randomly generated 20 starting points $x(t = 0)$ uniformly distributed in its search range, and set the initial values for the Lagrange multipliers to be zero, *i.e.* $\lambda(t = 0) = \mu(t = 0) = 0$. For each starting point, we applied the first-order methods using both static weights and dynamically changed weights under the same convergence condition defined in (A.19).

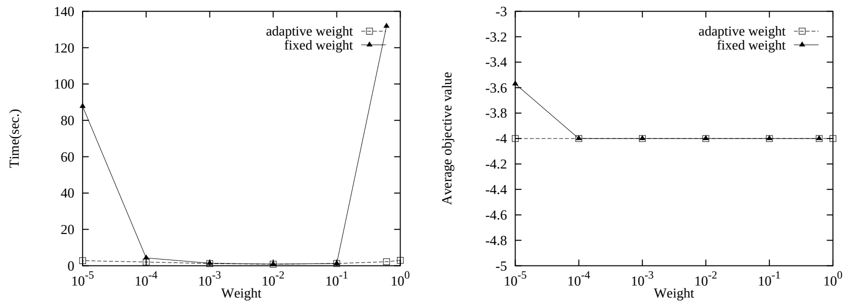
We chose the static weights in the range between 10^{-6} to 10^{2}, which were also used as initial weights $w(t = 0)$ in the adaptive algorithm. The weights were chosen from a wide range in order to test the robustness of our weight-adaptation algorithm. We would like our algorithm to adjust the weight so that the search will converge faster with a solution that is at least as good as that with static weights. In our experiments, we used the following control parameters: time unit $\delta_t = 1$, window size $N_w = 100$, $\alpha_0 = \alpha_1 = \frac{1}{2}$, $\delta = 10^{-8}$, $\beta_0 = 10^{-2}$, $\beta_1 = 10^{-3}$, and $\beta_2 = 10^{-2}$.

We used two performance metrics, convergence time and solution quality, in our experiments, where convergence time was measured by the average convergence time (in seconds) from 20 starting points, and solution quality was measured by the average objective value when the search converged.

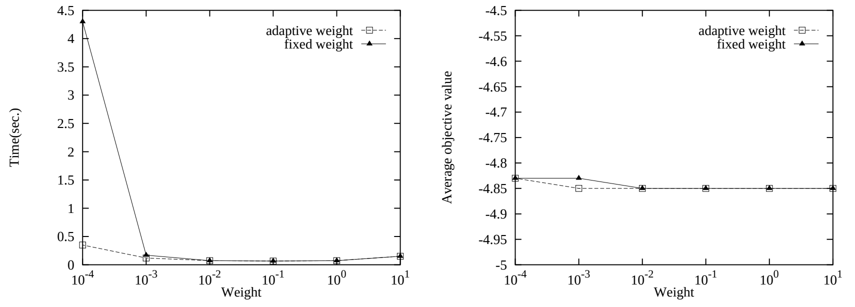
From the experimental results shown in Figure A.11, we have the following observations. First, different problems require different ranges of static weights in order to get fast convergence. For example, Problem 2.6 (Figure A.11a) converges the fastest using $w = 10^{-4}$, whereas Problem 5.4 (Figure A.11d) converges the fastest using $w = 1$. These weights differ by four orders of magnitude. Hence, it is difficult to choose a good static weight for a given problem instance in advance.



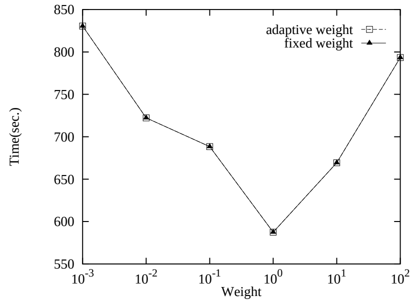
(a) Problem 2.6

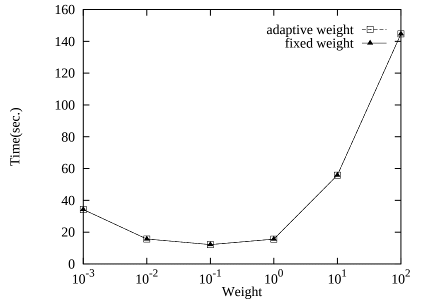


(b) Problem 3.4



(c) Problem 4.6





(d) Problem 5.4 (e) Problem 6.4

Figure A.11: Comparison of the first-order method with static weights and the adaptive first-order method in terms of convergence time and solution quality. All runs were done on a Sparc-10 computer under SunOS.

Second, our adaptive algorithm outperforms the first-order method with static weights. For example, in Problem 2.6 (Figure A.11a), the first-order method with a static weight is unable to converge (either diverge or oscillate forever) when $w = 10^{-2}$ or larger. However, our adaptive algorithm can detect this misbehavior and adjust w to allow the search to converge in 27.62 seconds on the average. Likewise, the first-order method with static weight $w = 10^{-6}$ converges using an average of 765.0 seconds, but the adaptive algorithm converges using an average of 5.64 seconds and with the same solution quality. These results show that our weight-adaptation algorithm is robust and insensitive to the initial weights $w(t = 0)$ in comparison to the first-order method with a static weight. In fact, our adaptive algorithm can get even better solutions in shorter time when the initial weight is small.

Third, when all the constraints are equality constraints, w is unchanged in the adaptive algorithm, resulting in the same performance for both methods. This is demonstrated in Problem 5.4 (Figure A.11d). In some cases, w is unchanged in the adaptive algorithm when there are both inequality and equality constraints, such as Problem 6.4 (Figure A.11e). This happens when the Lagrange multipliers already take care of the balance between the objective and the constraints.

A.3 Experimental Results

Table A.1 presents the comparison results of solving a collection of constrained NLP benchmarks [68]. The first and second columns show the problem IDs and the best-known results. The next three columns give the solutions, running times (S-Time) of the slack-variable method with static weight $w = 1$, and running times (D-Time) of the slack-variable method using dynamic weighting strategy with initial weight $w(t = 0) = 1$, respectively. The next

Table A.1: Results on a collection of constrained optimization benchmarks [68] comparing the *MaxQ* and the slack-variable methods. All times are in CPU seconds on a Pentium-III 500MHz computer under Solaris 7. Symbol '-' means that the method was not able to find a solution for the corresponding problem.

Problem	Best Known	Slack Variable			MaxQ Method			SQP: DONLP2	
ID	Solutions	Solutions	S-Time	D-Time	Solutions	S-Time	D-Time	Solutions	Time
2.1	-17.0	-17.0	-	0.40	-17.0	0.42	0.29	-17.0	0.017
2.2	-213.0	-213.0	0.034	0.034	-213.0	0.34	0.29	-213.0	0.022
2.3	-15.0	-15.0	0.17	0.17	-15.0	1.07	0.80	-15.0	0.124
2.4	-11.0	-11.0	-	1.24	-11.0	0.347	0.347	-11.0	0.039
2.5	-268.0	-268.0	0.13	0.13	-268.0	0.79	0.79	-268.0	0.085
2.6	-39.0	-39.0	-	13.8	-39.0	20.7	0.784	-39.0	0.095
2.7.1	-394.75	-394.75	1.35	1.35	-394.75	2.66	1.66	-394.75	0.702
2.7.2	-884.75	-884.75	1.51	1.51	-884.75	4.45	2.21	-884.75	0.744
2.7.3	-8695.0	-8695.0	1.33	1.33	-8695.0	2.58	1.58	-8695.0	0.516
2.7.4	-754.75	-754.75	1.46	1.46	-754.75	3.24	2.20	-754.75	0.716
2.7.5	-4150.4	-4150.4	1.53	1.53	-4150.4	2.76	1.70	-4150.4	0.483
2.8	15639.0	15639.0	1.59	1.59	15639.0	3.77	1.93	15639.0	1.51
3.2	-30665.5	-30665.5	0.128	0.126	-30665.5	0.167	0.114	-30665.5	0.022
3.3	-310.0	-310.0	0.047	0.047	-310.0	0.38	0.23	-310.0	0.039
3.4	-4.0	-4.0	0.031	0.031	-4.0	0.202	0.159	-4.0	0.021
4.3	-4.51	-4.51	0.022	0.022	-4.51	0.264	0.061	-4.51	0.027
4.4	-2.217	-2.217	0.018	0.018	-2.217	0.27	0.046	-2.217	0.026
4.5	-13.40	-13.40	0.030	0.030	-13.40	0.35	0.069	-13.40	0.031
4.6	-5.51	-5.51	0.01	0.01	-5.51	1.02	0.032	-5.51	0.012
4.7	-16.74	-16.74	0.01	0.01	-16.74	0.01	0.01	-16.74	0.012
5.2	1.567	1.567	31.3	31.3	1.567	15.9	15.9	1.567	18.3
5.4	1.86	1.86	11.4	11.4	1.86	7.49	7.49	1.86	6.63
6.2	400.0	400.0	0.105	0.105	400.0	0.40	0.249	400.0	0.056
6.3	600.0	600.0	0.124	0.124	600.0	0.57	0.405	600.0	0.055
6.4	750.0	750.0	0.084	0.084	750.0	0.392	0.216	750.0	0.058
7.2	56825.0	56825.0	2.08	2.08	56825.0	2.61	1.85	56825.0	0.33
7.3	44903.0	44903.0	4.39	4.39	44903.0	11.5	8.26	44903.0	2.21
7.4	35920.0	35920.0	12.2	12.2	35920.0	15.6	13.8	35920.0	5.65

three columns report the results for the *MaxQ* method, and the last two columns for a popular SQP package called DONLP2 [179].

From this table, we have the following observations. First, the slack-variable method may not converge if static weight w is not chosen correctly, as in Problems 2.1, 2.4 and 2.6. The *MaxQ* method can always converge, but is generally slower than the slack-variable method when the latter can converge. Second, dynamic weighting is able to avoid oscillations and divergence of the slack-variable method, and to greatly speed up the *MaxQ* methods. Last, all three methods find the same best solutions, but DONLP2 is the best in terms of solution time. This is because the first two methods converge linearly, whereas DONLP2 converges at least superlinearly by fully using Hessian information.

Bibliography

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* J. Wiley and Sons, 1989.

[2] E. H. L. Aarts. A new polynomial-time cooling schedule. *Proc. of the IEEE Int. Conf. on CAD-85*, pages 206–208, 1985.

[3] S. O. Aase. *Image Subband Coding Artifacts: Analysis and Remedies.* PhD thesis, Norwegian Institute of Technology, 1993.

[4] M. M. Ali and C. Storey. Modified controlled random search algorithms. *Int. Journal of Computer Mathematics*, 53:229–235, 1994.

[5] M. M. Ali and C. Storey. Aspiration based simulated annealing algorithms. *Journal of Global Optimization*, 11:181–191, 1997.

[6] M. M. Ali, A. Torn, and S. Viitanen. A direct search simulated annealing algorithms for optimization involving continuous variables. Technical report, Turku Centre for Computer Science, Abo Akademi University, Finland, 1997.

[7] M. M. Ali, A. Torn, and S. Viitanen. A numerical comparison of some modified controlled random search algorithms. *Journal of Global Optimization*, 11:377–385, 1997.

[8] I. Andricioaei and J. E. Straub. Generalized simulated annealing algorithms using tsallis statistics: Application to conformational optimization of a tetrapeptide. *Physical Review E*, 53(4):R3055–R3058, 1996.

[9] S. Anily and A. Federgruen. Ergodicity in parametric nonstationary Markov chains: An application to simulated annealing methods. *Operations Research*, 35(6):867–874, 1987.

[10] S. Anily and A Federgruen. Simulated annealing methods with general acceptance probabilities. *Journal of Appl. Prob.*, 24:657–667, 1987.

[11] M. Anitescu and F. A. Potra. An efficient procedure for maximizing the coding gain for PR filter banks. *IEEE Trans. on Circuits and Systems - II Analog and Digital Signal Processing*, page to appear, 1999.

[12] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Trans. on Image Processing*, 1(2):205–220, 1992.

[13] ASA. http://www.ingber.com.

[14] T. Back, F. Hoffmeister, and H. P. Schwefel. A survey of evolution strategies. In *Proc. of 4th Int'l Conf. on Genetic Algorithms*, pages 2–9, 1991.

[15] T. Back and H. P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[16] E. Balas. *Minimax and Duality for Linear and Nonlinear Mixed-Integer Programming.* North-Holland, Amsterdam, Netherlands, 1970.

[17] I. Balasingham and T. A. Ramstad. On the relevance of the regularity constraint in subband image coding. In *Proc. of 31st Asilomar Conf. on Signals, Systems, and Computers*, 1997.

[18] I. Balasingham, T. A. Ramstad, and J. M. Lervik. Survey of odd and even length filters in tree-structured filter banks for subband image compression. In *Proc. of Int. Conf. on Acoustics, Speech, and Signal Proc.*, pages 3073–3076, 1997.

[19] Ilangko Balasingham. *On Optimal Perfect Reconstruction Filter Banks for Image Compression.* PhD thesis, Norwegian University of Science and Technology, 1998.

[20] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical report, CMU-CS-95-193, Carnegie Mellon University, Pittsburgh, PA, 1995.

[21] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proc. of Int. Conf. on Machine Learning*, pages 30–38, 1997.

[22] S. Baluja and S. Davies. Fast probabilistic modeling for combinatorial optimization. In *Proc. of 15th National Conf. on Artificial Intelligence (AAAI)*, 1998.

[23] W. Baritompa. Accelerations for a variety of global optimization methods. *Journal of Global Optimization*, 4:37–45, 1994.

[24] W. Baritompa and A. Cutler. Accelerations for global optimization covering methods using second derivatives. *Journal of Global Optimization*, 4:329–341, 1994.

[25] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[26] M. S. Bazaraa and J. J. Goode. A survey of various tactics for generating Lagrangian multipliers in the context of Lagrangian duality. *European Journal of Operational Research*, 3:322–338, 1979.

[27] J. C. Bean and A. B. Hadj-Alouane. A dual genetic algorithm for bounded integer programs. *Technical Report 92-53, Department of Industrial and Operations Engineering*, 1992.

[28] C. J. Belisle. Convergence theorems for a class of simulated annealing algorithms in R^n. *Journal of Applied Prob.*, 29:885–895, 1992.

[29] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numer. Math*, pages 238–242, 1962.

[30] J. Bernasconi. Low autocorrelation binary sequences: Statistical mechanics and configuration space analysis. *J. Physique*, 48(4):559–567, 1987.

[31] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press, 1982.

[32] G. L. Bilbro and W. E. Snyder. Optimization of function with many minima. *IEEE Trans. on Systems, Man, and Cybern.*, 21(4):840–849, 1991.

[33] G. L. Bilbro, M. B. Steer, R. J. Trew, C. R. Chang, and S. G. Skaggs. Extraction of the parameters of equivalent circuits of microwave transistors using tree annealing. *IEEE Trans. on Microwave Theory Technol.*, 38(11):1711–1718, 1990.

[34] K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16(2):101–113, 1994.

[35] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, pages 1–52, 1995.

[36] J. S. De Bonet, C. L. Isbell, and J. Paul Viola. MIMIC: Finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems*, 1997.

[37] I. Bongartz, A. R. Conn, N. Gould, and Ph. L. Toint. Cute: Constrained and unconstrained testing environment. *ACM Trans. on Mathematical Software*, 21(1):123–160, 1995.

[38] J. A. Boyan and A. W. Moore. Using prediction to improve combinatorial optimization search. In *Proc. of 6th Int'l Workshop on Artificial Intelligence and Statistics*, 1997.

[39] J. A. Boyan and A. W. Moore. Learning evaluation functions for global optimization and Boolean satisfiability. In *Proc. of 15th National Conf. on Artificial Intelligence (AAAI)*, 1998.

[40] D. W. Bulger and G. R. Wood. Hesitant adaptive search for global optimization. *Mathematical Programming*, 81:89–102, 1998.

[41] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *Technical Report OTC97/05, Optimization Technology Center*, 1997.

[42] M. F. Cardoso, R. L. Salcedo, and S. F. de Azevedo. Non-equilibrium simulated annealing: A faster approach to combinatorial minimization. *Industrial Eng. Chemical Research*, 33:1908–1918, 1994.

[43] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[44] B. C. Cetin, J. Barben, and J. W. Burdick. Terminal repeller unconstrained subenergy tunneling (TRUST) for fast global optimization. *Journal of Optimization Theory and Applications*, 77, April 1993.

[45] K. M. Cheh, J. B. Goldberg, and R. G. Askin. A note on the effect of neighborhood structure in simulated annealing. *Computers and Operations Research*, 18:537–547, 1991.

[46] A. Cichocki and R. Unbehauen. *Neural Networks for Optimization and Signal Processing.* John Wiley & Sons, 1993.

[47] A. Cohen, I. Daubechies, and J. C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, XLV:485–560, 1992.

[48] A.R. Conn, N. Gould, and Ph. L. Toint. *LANCELOT, A Fortran Package for Large-Scale Nonlinear Optimization.* Springer Verlag, 1992.

[49] A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Trans. on Mathematical Software*, 13(3):262–280, 1987.

[50] J. P. Courat, G. Raynaud, I. Mrad, and P. Siarry. Electronic component model minimization based on log simulated annealing. *IEEE Trans. on Circuits and Systems - I Fundamental Theory and Applications*, 41(12):790–795, 1994.

[51] P. Courrieu. The hyperbell algorithm for global optimization: A random walk using cauchy densities. *Journal of Global Optimization*, 10:37–55, 1997.

[52] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, XLI:909–996, 1988.

[53] G. Davis and S. Chawla. Image coding using optimized significance tree quantization. In *Proc. of Data Compression Conf.*, pages 387–396, 1997.

[54] G. Davis and A. Nosratinia. Wavelet-based image coding: An overview. *Applied and Computational Control, Signals, and Circuits*, 1(1), 1998.

[55] G. M. Davis. The wavelet image compression construction kit. *http://www.cs.dartmouth.edu/gdavis/wavelet/wavelet.html*, 1997.

[56] A. Dekkers and E. Aarts. Global optimization and simulated annealing. *Mathematical Programming*, 50:367–393, 1991.

[57] I. Diener and R. Schaback. An extended continuous Newton method. *Journal of Optimization Theory and Applications*, 67(1):57–77, October 1990.

[58] DONLP2. Spellucci's mixed SQP/ECQP method for general continuous nonlinear programming problems. *ftp://plato.la.asu.edu/pub/donlp2*, 2000.

[59] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Trans. on Evolutionary Computation*, 1(1):53–66, 1997.

[60] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.

[61] M. A. Duran and I. E. Grossmann. A mixed-integer nonlinear programming algorithm for process systems synthesis. *Chemical Engineering J.*, pages 592–596, 1986.

[62] M. A. Duran and I. E. Grossmann. An outer approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, pages 306–307, 1986.

[63] R. W. Eglese. Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 46:271–281, 1990.

[64] T. Epperly. *Global Optimization of Nonconvex Nonolinear Programs Using Parallel Branch And Bound.* PhD thesis, University of Wisconsin-Madison, 1995.

[65] Y. G. Evtushenko, M. A. Potapov, and V. V. Korotkich. Numerical methods for global optimization. In C. A. Floudas and P. M. Pardalos, editors, *Recent Advances in Global Optimization*, pages 274–297. Princeton University Press, 1992.

[66] N. J. Fliege. *Multirate Digital Signal Processing.* John Wiley and Sons, 1994.

[67] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization.* Topics in Chemical Engineering. Oxford University Press, 1995.

[68] C. A. Floudas and P. M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*, volume 455 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.

[69] C. A. Floudas and P. M. Pardalos, editors. *Recent Advances in Global Optimization.* Princeton University Press, 1992.

[70] D. B. Fogel. An analysis of evolutionary programming. In *Proc. of First Annual Conf. on Evolutionary Programming*, pages 43–51, 1992.

[71] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Trans. on Neural Networks*, 5(1):3–14, January 1994.

[72] M. I. Freidlin and A. D. Wentzell. *Random perturbations of dynamical systems.* New York : Springer, 1984, 1984.

[73] FSQP. http://www.isr.umd.edu/labs/cacse/fsqp/fsqp.html.

[74] B. Gavish. On obtaining the 'best' multilpliers for a Lagrangean relaxation for integer programming. *Comput. & Ops. Res.*, 5:55–71, 1978.

[75] R. P. Ge and Y. F. Qin. A class of filled functions for finding global minimizers of a function of several variables. *Journal of Optimization Theory and Applications*, 54(2):241–252, 1987.

[76] GENOCOP. http://www.coe.uncc.edu/ zbyszek/evol-systems.html.

[77] A. M. Geoffrion. Generalized Benders decomposition. *J. Optim. Theory and Appl.*, pages 237–241, 1972.

[78] B. Gidas. Non-stationary Markov chains and convergence of the annealing algorithm. *J. Statist. Phys.*, 39:73–131, 1985.

[79] J. C. Gilbert, R. H. Byrd, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Technical Report OTC 96/02, Optimization Technology Center*, 1996.

[80] GlobSol. http://www.mscs.mu.edu/ globsol/.

[81] F. Glover and G. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In *Proc. of Int'l Conf. on Metaheuristics for Optimization*, pages 113–133, 1995.

[82] F. Glover and M. Laguna. Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems (C. R. Reeves ed.)*, 1993.

[83] F. Glover and E. Woolsey. Further reduction of zero-one polynomial programs to zero-one linear programming. *Operations Research*, 1(21):156–161, 1973.

[84] F. Glover and E. Woolsey. Converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22:180–182, 1975.

[85] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Pub. Co., 1989.

[86] D. Granot, F. Granot, and J. Kallberg. Covering relaxation for positive 0-1 polynomial programs. *Management Science*, 3(25):264–273, 1979.

[87] D. Granot, F. Granot, and W. Vaessen. An accelerated covering relaxation algorithm for solving positive 0-1 polynomial programs. *Mathematical Programming*, 22:350–357, 1982.

[88] A. O. Griewank. Generalized descent for global optimization. *Journal of Optimization Theory and Applications*, 34:11–39, 1981.

[89] A. B. Hadj-Alouane and J. C. Bean. Genetic algorithm for the multiple-choice integer program. *Technical Report 92-50, Department of Industrial and Operations Engineering*, 1992.

[90] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[91] E. R. Hansen. *Global optimization using interval analysis.* M. Dekker, New York, 1992.

[92] P. Hansen, B. Jaumard, and V. Mathon. Constrained nonlinear 0-1 programming. *ORSA Journal on Computing*, 5(2):97–119, 1993.

[93] U. H. E. Hansmann. Simulated annealing with tsallis weights: A numerical comparison. *Physica A*, 242:250–257, 1997.

[94] W. E. Hart. A theoretical comparison of evolutionary algorithms and simulated annealing. In *Proc. of 5th Annual Conf. on Evolutionary Programming (EP96)*, pages 147–154, 1996.

[95] W. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[96] L He and E. Polak. Multistart method with estimation scheme for global satisfying problems. *Journal of Global Optimization*, 3:139–156, 1993.

[97] M. L. Hilton, B. D. Jawerth, and A. Sengupta. Compression still and moving images with wavelets. *Multimedia Systems*, 2(2):218–227, 1994.

[98] A. C. Hindmarsh. ODEPACK, a systematized collection of ODE solvers. In R. S. Stepleman, editor, *Scientific Computing*, pages 55–64. North Holland, Amsterdam, 1983.

[99] K. Holmberg. On the convergence of the cross decomposition. *Mathematical Programming*, pages 269–316, 1990.

[100] K. Holmberg. Generalized cross decomposition applied to nonlinear integer programming problems. *Optimization J.*, pages 341–364, 1992.

[101] A. Homaifar, S. H. Y. Lai, and X. Qi. Constrained optimization via genetic algorithms. *Simulation*, 62:242–254, 1994.

[102] J. J. Hopfield and D. M. Tank. Neural computation of decisions in optimization. *Biological Cybern.*, 52:141–152, 1985.

[103] B. R. Horng and A. N. Willson Jr. Lagrange multiplier approaches to the design of two-channel perfect-reconstruction linear-phase FIR filter banks. *IEEE Trans. on Signal Processing*, 40(2):364–374, February 1992.

[104] R. Horst and P. M. Pardalos. *Handbook of Global Optimization*. Kluwer Academic Publishers, 1995.

[105] R. Horst, P. M. Pardalos, and N. V. Thoai. *Introduction to Global Optimization*. Kluwer Academic Publishers, Amsterdam, 1995.

[106] R. Horst and H. Tuy. *Global optimization: Deterministic approaches*. Springer-Verlag, Berlin, 1993.

[107] M. E. Hribar. *Large-Scale Constrained Optimization*. PhD thesis, Northwestern University, 1996.

[108] ILOG. http://www.ilog.com/.

[109] L. Ingber. Very fast simulated re-annealing. *J. Math. Comput. Modelling*, 12:967–973, 1989.

[110] L. Ingber. *Adaptive Simulated Annealing (ASA)*. Lester Ingber Research, 1995.

[111] L. Ingber and N. Rosen. Genetic algorithms and very fast simulated re-annealing: A comparison. *J. Math. Comput. Modelling*, 16:87–100, 1992.

[112] J. Joines and C. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems. In *Proc. of the First IEEE Int'l Conf. on Evolutionary Computation*, pages 579–584, 1994.

[113] A. E. W. Jones and G. W. Forbes. An adaptive simulated annealing algorithm for global optimization over continuous variables. *Journal of Optimization Theory and Applications*, 6:1–37, 1995.

[114] J. Katto and Y. Yasuda. Performance evaluation of subband coding and optimization of its filter coefficients. In *Proc. of SPIE Visual Communications and Image Processing*, pages 95–106, 1991.

[115] R. B. Kearfott. On proving existence of feasible points in equality constrained optimization problems. *Mathematical Programming*, 83:89–100, 1998.

[116] R. R. Kearfott. A review of techniques in the verified solution of constrained global optimization problems. In *Applications of Interval Computations, R. B. Kearfott and V. Kreinovich (Eds.), Kluwer Academic Publishers, Dordrecht, The Netherlands*, pages 23–60, 1996.

[117] J. Kim and H. Myung. Evolutionary programming techniques for constrained optimization problems. *IEEE Trans. on Evolutionary Computation*, 1(2):129–140, 1997.

[118] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[119] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[120] V. Kvasnicka and J. Pospichal. A hybrid of simplex method and simulated annealing. *Chemometrics and Intelligent Laboratory Systems*, 39:161–173, 1997.

[121] LANCELOT. http://www.dci.clrc.ac.uk/activity/lancelot.

[122] L. S. Lasdon, A. D. Warren, A. Jain, and M. Ratner. Design and testing a generalized reduced gradient code for nonlinear programming. *ACM Trans. Math. Software*, 4:34–50, 1978.

[123] W. E. Lillo, M. H. Loh, S. Hui, and S. H. Zak. On solving constrained optimization problems with neural networks: A penalty method approach. *IEEE Trans. on Neural Networks*, 4(6):931–940, 1993.

[124] S. Lucidi and M. Piccioni. Random tunneling by means of acceptance-rejection sampling for global optimization. *Journal of Optimization Theory and Applications*, 62:255–277, 1989.

[125] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Reading, MA, 1984.

[126] C. Y. Maa and M. A. Shanblatt. A two-phase optimization neural network. *IEEE Trans. on Neural Networks*, 3(6):1003–1009, 1992.

[127] Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs.* Springer-Verlag, Berlin, 1994.

[128] Z. Michalewicz. Genetic algorithsm, numerical optimization and constraints. In L. J. Eshelman, editor, *Proc. Int'l Conf. on Genetic Algorithms*, pages 98–108. Morgan Kaufmann, 1995.

[129] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In *Proc. of 4th Annual Conf. on Evolutionary Programming*, pages 135–155, 1995.

[130] Z. Michalewicz, D. Dasgupta, R. G. LeRiche, and M. Schoenauer. Evolutionary algorithms for constrained engineering problems. *Computers and Industrial Engineering Journal*, 30(2):851–870, 1996.

[131] Z. Michalewicz and C. Z. Janikow. Handling constraints in genetic algorithms. In *Proc. of 4th Int'l Conf. on Genetic Algorithms*, pages 151–157, 1991.

[132] Z. Michalewicz and G. Nazhiyath. GENOCOP III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Proc. of 2nd IEEE Int'l Conf. on Evolutionary Computation*, pages 647–651, 1995.

[133] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[134] J. Mockus. *Bayesian Approach to Global Optimization.* Kluwer Academic Publishers, Dordrecht-London-Boston, 1989.

[135] J. Mockus. Application of Bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4:347–365, 1994.

[136] P. Molitor. Layer assignment by simulated annealing. *Microprocessing and Microprogramming*, 16(4-5):345–349, 1985.

[137] R. Moore and E. Hansen amd A. Leclerc. Rigorous methods for global optimization. In C. A. Floudas and P. M. Pardalos, editors, *Recent Advances in Global Optimization*, pages 321–342. Princeton University Press, 1992.

[138] J. A. Nelder and R. Mead. Simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[139] T. Q. Nguyen and P. P. Vaidyanathan. Two-channel perfect reconstruction FIR QMF structure which yield linear-phase analysis and synthesis filters. *IEEE Trans. on Acoutics, Speech, and Signal Processing*, 37(5):676–690, May 1989.

[140] A. E. Nix and M. D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Math. and Artificial Intel.*, 5:79–88, 1992.

[141] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag New York, Inc., 1999.

[142] D. Orvosh and L. Davis. Shall we repair? genetic algorithms, combinatorial optimization, and feasibility constraints. In *Proc. of 5th Int'l Conf. on Genetic Algorithms*, 1993.

[143] E. R. Panier and A. L. Tits. A superlinearly convergence feasible method for the solution of inequality constrained optimization problems. *SIAM Journal on control and Optimization*, 25(4):934–950, 1987.

[144] E. R. Panier and A. L. Tits. On combining feasibility, descent and superlinear convergence in inequlaity constrained optimization. *Mathematical Programming*, 59(2):261–276, 1993.

[145] P. M. Pardalos and J. B. Rosen. *Constrained Global Optimization: Algorithms and Applications*, volume 268 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1987.

[146] J. Paredis. Co-evolutionary constraint satisfaction. In *Proc. of 3rd Conf. on Parallel Problem Solving from Nature*, pages 46–55, 1994.

[147] K. Park and B. Carter. On the effectiveness of genetic search in combinatorial optimization. In *Proc. of 10th ACM Symposium on Applied Computing, Genetic Algorithms and Optimization Track*, pages 329–336, 1995.

[148] N. R. Patel, R. L. Smith, and Z. B. Zabinsky. Pure adaptive search in Monte Carlo optimization. *Mathematical Programming*, 43:317–328, 1988.

[149] V. Pavlovic, P. Moulin, and K. Ramchandran. An integrated framework for adaptive subband image coding. *IEEE Trans. on Signal Processing*, page revised.

[150] M. Peng, N. K. Gupta, and A. F. Armitage. An investigation into improvement of local minima of the hopfield network. *Neural Network*, 9(7):1241–1253, 1996.

[151] V. Petridis, S. Kazarlis, and A. Bakirtzis. Varying fitness functions in genetic algorithm constarined optimization: The cutting stock and unit commitment problems. *IEEE Trans. on System, Man, and Cybern. - Part B: Cybernetics*, 28(5):629–640, 1998.

[152] M. Piccioni. A combined multistart-annealing algorithm for continuous global optimization. Technical Report 87-45, Systems and Research Center, The University of Maryland, College Park MD, 1987.

[153] C. N. Potts and L. N. Van Wassenhove. Single machine tardiness sequencing heuristics. *IIE Transactions*, 23:346–354, 1991.

[154] D. Powell and M. M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proc. of 5th Int'l Conf. on Genetic Algorithms*, pages 424–431, 1993.

[155] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in Fortran 77.* Cambridge University Press, 1992.

[156] W. L. Price. A controlled random search procedure for global optimization. In L. C. Dixon and G. P. Szego, editors, *Towards Global Optimization 2*, pages 71–84. North-Holland, Amsterdam, Holland, 1978.

[157] R. L. Rardin. *Optimization in Operations Research.* Upper Saddle River, N.J. : Prentice Hall, 1998.

[158] V. Ravi, B. S. N. Murty, and J. Reddy. Nonequilibrium simulated annealing algorithm applied to reliability optimization of complex systems. *IEEE Trans. on Reliability*, 46(2):233–239, 1997.

[159] S. Rees and B. C. Ball. Citeria for an optimal simulated annealing schedule for problems of the traveling salesman type. *J. Physics*, 20(5):1239–1249, 1987.

[160] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In *Proc. of 3rd Int'l Conf. on Genetic Algorithms*, pages 191–197, 1989.

[161] H. E. Romeijn and R. L. Smith. Simulated annealing for constrained global optimization. *Journal of Global Optimization*, 5(2):101–126, September 1994.

[162] H. E. Romeijn and R. L. Smith. Simulated annealing for constrained global optimization. *Journal of Global Optimization*, 5:101–126, 1994.

[163] F. Romeo and A. L. Sangiovanni-Vincentelli. Probabilistic hill climbing algorithms: Properties and applications. In *Proc. of Chapel Hill Conf. on VLSI*, pages 393–417, 1985.

[164] F. Romeo and A. L. Sangiovanni-Vincentelli. A theoretical framework for simulated annealing. *Algorithmica*, 6:302–345, 1991.

[165] I. Rosenberg. Minimization of pseudo-Boolean functions by binary development. *Discrete Mathematics*, 7:151–165, 1974.

[166] T. J. Van Roy. Cross decomposition for mixed integer programming. *Mathematical Programming*, pages 25–46, 1983.

[167] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Trans. on Neural Networks*, 5(1):96–101, 1994.

[168] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical tress. *IEEE Trans. on Circuits and Systems for Video Technology*, 6:243–250, 1996.

[169] M. S. Sarma. On the convergence of the Baba and Dorea random optimization methods. *Journal of Optimization Theory and Applications*, 66:337–343, 1990.

[170] S. Schäffler and H. Warsitz. A trajectory-following method for unconstrained optimization. *Journal of Optimization Theory and Applications*, 67(1):133–140, October 1990.

[171] F. Schoen. Stochastic techniques for global optimization: A survey on recent advances. *Journal of Global Optimization*, 1(3):207–228, 1991.

[172] M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In *Proc. of 4th Parallel Problem Solving from Nature*, 1996.

[173] M. Schoenauer and S. Xanthakis. Constrained GA optimization. In *Proc. of 5th Int'l Conf. on Genetic Algorithms*, 1993.

[174] Y. Shang. *Global Search Methods for Solving Nonlinear Optimization Problems.* Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, August 1997.

[175] Y. Shang and B. W. Wah. A discrete Lagrangian based global search method for solving satisfiability problems. *J. of Global Optimization*, 12(1):61–99, January 1998.

[176] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. on Signal Processing*, 41, 1993.

[177] R. Spaans and R. Luus. Importance of search-domain reduction in random optimization. *Journal of Optimization Theory and Applications*, 75(3):635–638, December 1992.

[178] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Preprint, Department of Mathematics*, 1993.

[179] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming*, 82:413–448, 1998.

[180] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application.* Krieger Publishing Company, 1989.

[181] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[182] E. G. Sturua and S. K. Zavriev. A trajectory algorithm based on the gradient method I. the search on the quasioptimal trajectories. *Journal of Global Optimization*, 1991(4):375–388, 1991.

[183] H. Szu and R. Hartley. Fast simulated annealing. *Phys. Lett. A*, 122(3-4):157–162, 1987.

[184] J. Tind and L. A. Wolsey. An elementary survey of general duality theory in mathematical programming. *Mathematical Programming*, pages 241–261, 1981.

[185] A. Törn and A. Žilinskas. *Global Optimization.* Springer-Verlag, Berlin, 1989.

[186] A. Trouve. Rough large deviation estimates for the optimal convergence speed exponent of generalized simulated annealing algorithms. Technical report, LMENS-94-8, Ecole Normale Superieure, France, 1994.

[187] A. Trouve. Cycle decomposition and simulated annealing. *SIAM Journal on Control and Optimization*, 34(3):966–986, 1996.

[188] P. Vaidyanathan. *Multirate Systems and Filter Banks.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[189] J. van Den Berg and J. C. Bioch. Constrained optimization with the Hopfield-Lagrange model. In *Proc. of 14th IMACS World Congress*, pages 470–473, 1994.

[190] D. Vanderbilt and S. G. Louie. A Monte Carlo simulated annealing approach to optimization over continuous variables. *Journal of Computational Physics*, 56:259–271, 1984.

[191] S. A. Vavasis. *Nonlinear Optimization, Complexity Issues.* Oxford Univerusity Press, Oxford, 1991.

[192] M. Vetterli and J. Kovacevic. *Wavelets and Subband Coding.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[193] J. D. Villasenor, B. Belzer, and J. Liao. Wavelet filter evaluation for image compression. *IEEE Trans. on Image Processing*, 2:1053–1060, 1995.

[194] T. L. Vincent, B. S. Goh, and K. L. Teo. Trajectory-following algorithms for min-max optimization problems. *Journal of Optimization Theory and Applications*, 75(3):501–519, December 1992.

[195] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[196] A. Žilinskas. A review of statistical models for global optimization. *Journal of Global Optimization*, 2:145–153, 1992.

[197] B. W. Wah and Y.-J. Chang. Trace-based methods for solving nonlinear global optimization problems. *J. of Global Optimization*, 10(2):107–141, March 1997.

[198] B. W. Wah and Y. X. Chen. Optimal anytime constrained simulated annealing for constrained global optimization. *Sixth Int'l Conf. on Principles and Practice of Constraint Programming*, September 2000.

[199] B. W. Wah, A. Ieumwananonthachai, L. C. Chu, and A. Aizawa. Genetics-based learning of new heuristics: Rational scheduling of experiments and generalization. *IEEE Trans. on Knowledge and Data Engineering*, 7(5):763–785, October 1995.

[200] B. W. Wah and T. Wang. Constrained simulated annealing with applications in nonlinear constrained global optimization. In *Proc. Int'l Conf. on Tools with Artificial Intelligence*, pages 381–388. IEEE, November 1999.

[201] B. W. Wah and T. Wang. Efficient and adaptive Lagrange-multiplier methods for nonlinear continuous global optimization. *J. of Global Optimization*, 14(1):1–25, January 1999.

[202] B. W. Wah and T. Wang. Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. *Principles and Practice of Constraint Programming*, pages 461–475, October 1999.

[203] B. W. Wah and T. Wang. Tuning strategies in constrained simulated annealing for nonlinear global optimization. *Int'l J. of Artificial Intelligence Tools*, 9(1):3–25, 2000.

[204] B. W. Wah, T. Wang, Y. Shang, and Z. Wu. Improving the performance of weighted Lagrange-multiple methods for constrained nonlinear optimization. In *Proc. 9th Int'l Conf. on Tools for Artificial Intelligence*, pages 224–231. IEEE, November 1997.

[205] B. W. Wah, T. Wang, Y. Shang, and Z. Wu. Improving the performance of weighted Lagrange-multiplier methods for nonlinear constrained optimization. *Information Sciences*, 124(1-4):241–272, May 2000.

[206] B. W. Wah and Z. Wu. The theory of discrete Lagrange multipliers for nonlinear discrete optimization. *Principles and Practice of Constraint Programming*, pages 28–42, October 1999.

[207] G. K. Wallace. The JPEG still picture compression standard. *Communications of ACM*, 34(4):30–44, 1991.

[208] T. Wang and B. W. Wah. Adaptive Lagrange-Multiplier methods for continuous nonlinear optimization. In *Proc. Symposium on Applied Computing*, pages 361–365, Atlanta, GA, February 1998. ACM.

[209] T. Wang and B. W. Wah. Constrained optimization of filter banks in subband image coding. In *Workshop on Multimedia Signal Processing*, pages 432–437, Monterey, CA, December 1998. IEEE Signal Processing Society.

[210] T. Wang and B. W. Wah. Handling inequality constraints in continuous nonlinear global optimization. *J. of Integrated Design and Process Science*, 2(3):1–10, 1998.

[211] X. D. Wang. An algorithm for nonlinear 0-1 programming and its application in structural optimization. *Journal of Numerical Method and Computational Applications*, 1(9):22–31, 1988.

[212] Y. Wang, W. Yan, and G. Zhang. Adaptive simulated annealing for optimal design of electromagnetic devices. *IEEE Trans. on Magnetics*, 32(3):1214–1217, 1996.

[213] V. Wilson and G. S. Pawley. On the stability of the traveling salesman problem algorithm of hopfield and tank. *Biological Cybern.*, 58:63–70, 1988.

[214] M. H. Wright. Interior methods for constrained optimization. In A. Iserles, editor, *Acta Numerica 1992*, pages 341–407. Cambridge University Press, 1992.

[215] S. J. Wright. *Primal-dual interior-point methods.* Philadelphia: Society for Industrial and Applied Mathematics, 1997.

[216] Z. Wu. *Discrete Lagrangian Methods for Solving Nonlinear Discrete Constrained Optimization Problems.* M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 1998.

[217] Z. Wu. *The Theory and Applications of Nonlinear Constrained Optimization using Lagrange Multipliers.* Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 2001.

[218] Z. Wu and B. W. Wah. Solving hard satisfiability problems: A unified algorithm based on discrete Lagrange multipliers. In *Proc. Int'l Conf. on Tools with Artificial Intelligence*, pages 210–217. IEEE, November 1999.

[219] Z. Wu and B. W. Wah. Trap escaping strategies in discrete Lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *Proc. 1999 National Conf. on Artificial Intelligence*, pages 673–678. AAAI, July 1999.

[220] Z. Xiong, K. Ramchandran, and M. T. Orchard. Space-frequency quantization for wavelet image coding. *IEEE Trans. on Image Processing*, 6:677–693, 1997.

[221] X. Yao. Simulated annealing with extended neighborhood. *Int. Journal of Computer Mathematics*, 40:169–189, 1991.

[222] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Trans. on Evolutionary Computation*, 3(2):82–102, 1999.

[223] Z. B. Zabinsky. Stochastic methods for practical global optimization. *Journal of Global Optimization*, 13:433–444, 1998.

[224] S. Zhang and A. G. Constantinides. Lagrange programming neural networks. *IEEE Trans. on Circuits and Systems - II Analog and Digital Signal Processing*, 39(7):441–452, 1992.

[225] A. A. Zhigliavskii. *Theory of Global Random Search.* Kluwer Academic Publishers, Boston, 1991.

[226] J. L. Zhou, A. L. Tits, and C. T. Lawrence. User's guide for FFSQP version 3.7: A Fortran code for solving optimization programs, possibly minimax, with general inequality constraints and linear equality constraints, generating feasible iterates. *Technical Report SRC-TR-92-107r5, Institute for Systems Research*, 1997.

Vita

Tao Wang received his B.E and M.E. degress in computer science from Zhejiang University, China, 1989 and 1991, respectively. He worked as a research assistant at Computer Vision Lab, Zhejiang University from Sept. 1989 to July 1991. He developed robust motion estimation algorithms of 3-D moving objects for scene analysis in computer vision. From Sept. 1991 to Dec. 1994, he worked as both a research assistant and lecturer for graduate courses Computer Vision and Neural Networks. During this period, he proposed some learning algorithms for associative neural networks, developed robust image processing methods for image segmentation and restoration of real images and dynamic object recognition and track schemes, and designed and implemented an automatic PCB defect inspection system. He was admitted to Computer Science Dept. of University of Illinois at Urbana-Champaign in Spring 1995, and worked as a research assistant until Aug. 1997. From Sept. 1997, he held one-year CSE fellowship, and worked as a research assistant since Sept. 1998. His recent work is on development of efficient nonlinear constrained global optimization methods and their applications to engineering design benchmarks, subband image coding, and robust image transmission over Internet.

His interests include optimization, efficient algorithm design, image coding, video coding, and computer networks.