TUNING STRATEGIES IN CONSTRAINED SIMULATED ANNEALING FOR NONLINEAR GLOBAL OPTIMIZATION*

BENJAMIN W. WAH AND TAO WANG
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, IL 61801, USA
{wah,wangtao}@manip.crhc.uiuc.edu

Received (23 May, 1999)
Revised (4 April, 2000)

This paper studies various strategies in *constrained simulated annealing* (CSA), a global optimization algorithm that achieves asymptotic convergence to *constrained global minima* (CGM) with probability one for solving discrete constrained *nonlinear programming problems* (NLPs). The algorithm is based on the necessary and sufficient condition for discrete *constrained local minima* (CLM) in the theory of discrete Lagrange multipliers and its extensions to continuous and mixed-integer constrained NLPs. The strategies studied include adaptive neighborhoods, distributions to control sampling, acceptance probabilities, and cooling schedules. We report much better solutions than the best-known solutions in the literature on two sets of continuous benchmarks and their discretized versions.

Keywords: Asymptotic convergence with probability one, constrained global minimum, constrained local minimum, constrained simulated annealing, discrete Lagrange multipliers, Metropolis probability, nonlinear programming, saddle points.

1 Problem Definition

A general *discrete constrained nonlinear programming problem* (NLP) is formulated as follows:

$$\begin{array}{ll} \text{minimize}_x & f(x) \\ \text{subject to} & g(x) \leq 0 \\ & h(x) = 0 \end{array} \tag{1}$$

where $x = (x_1, \ldots, x_n)$ is a vector of discrete variables, $f(x)$ is a lower-bounded objective function, $g(x) = [g_1(x), \cdots, g_k(x)]^T$ is a set of k inequality constraints,

*Research supported by National Science Foundation Grant NSF MIP 96-32316.

and $h(x) = [h_1(x), \cdots, h_m(x)]^T$ is a set of m equality constraints. Functions $f(x)$, $g(x)$, and $h(x)$ can be either linear or nonlinear, continuous or discrete (*i.e.* discontinuous), and analytic in closed forms or procedural.

Without loss of generality, we discuss our results with respect to minimization problems, knowing that maximization problems can be converted to minimization ones by negating their objectives. To characterize the solutions sought, we define the following terms.

Definition 1. $\mathcal{N}(x)$, the *neighborhood* of point x in discrete space X, is a user-defined set of points $x' \in X$ such that $x' \in \mathcal{N}(x) \Longleftrightarrow x \in \mathcal{N}(x')$, and that it is possible to reach every other x'' starting from any x in one or more steps through neighboring points.

Definition 2. Discrete point $x \in X$ is a *feasible point* if $g(x) \leq 0$ and $h(x) = 0$. Feasible space $\mathcal{F}$ is the set of all feasible points in X.

Definition 3. Point $x \in X$ is a discrete *constrained local minimum* (CLM) iff a) x is a feasible point, and b) for every feasible point $x' \in \mathcal{N}(x)$, $f(x') \geq f(x)$.

Note that since neighborhoods are user-defined, it is possible for point x to be a CLM in one definition of $N(x)$ but not for another. The choice of neighborhoods, however, does not affect the validity of a search as long as one definition is used consistently throughout [26].

Definition 4. Point $x \in X$ is a discrete *constrained global minimum* (CGM) iff a) x is a feasible point, and b) for every feasible point $x' \in X$, $f(x') \geq f(x)$. The set of all CGM is denoted by X_{opt}.

Finding CGM of (1) is challenging as well as difficult. First, $f(x)$, $g(x)$, and $h(x)$ may be 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 multiple feasible regions before finding a CGM. Second, $f(x)$, $g(x)$, and $h(x)$ may be discontinuous or may not have closed-form derivatives, rendering it impossible to apply existing theories and methods developed for continuous problems. Third, there may be a large number of CLM, trapping trajectories that utilize only local information and that are not able to escape from local minima.

In this paper, we tune strategies in *constrained simulated annealing* (CSA) [25] and apply them to solve discrete, continuous, and mixed-integer constrained NLPs. We first overview the theory of discrete Lagrange multipliers [20, 26, 28] in Section 2 and summarize the major steps of CSA in Section 3. We then focus on developing various strategies to efficiently solve general NLPs and report improvements over the best-known solutions in solving some benchmarks in Section 4.

2 Previous Work

Direct solution methods solve (1) directly. Some examples include reject/discard methods, repair methods, feasible-direction methods, and interval methods. However, these methods are very problem specific, or are unable to cope with nonlinear

constraints, or are computationally expensive. Therefore, the majority of methods to solve (1) first transform it into another form before solving it.

2.1 Penalty Methods

Static penalty methods transform (1) into a single unconstrained optimization problem [3, 15]. If the penalties are large enough, then the global minimum to the unconstrained problem is the CGM to the original problem. The use of large penalties, however, lead to rugged search terrains and deep local minima, making it difficult for global-search methods to escape from infeasible regions unless their starting points are close to one of the feasible regions.

Selecting a suitable penalty also proves to be difficult. If it is much larger than necessary, then the terrain will become too rugged to be searched by local- or global-search methods. If it is too small, then the solution found may either be a CLM or not even a feasible solution to (1).

Dynamic penalty methods address the difficulties of static penalty methods by increasing penalties gradually. They transform (1) into a sequence of unconstrained problems, and converge asymptotically with probability one if every unconstrained problem is solved optimally [3, 15]. The last requirement, however, is difficult to achieve in practice. If any of the unconstrained problems is not solved optimally, then the process is not guaranteed to find a CGM.

In addition to penalty formulations, many heuristics have been developed to handle constraints. These include constraint handling techniques in GA [16], such as annealing penalties, adaptive penalties, preserving feasibility with specialized genetic operators, searching along the boundary of feasible regions, death-penalty methods, behavioral memory with a linear order of constraints, repair of infeasible solutions, co-evolutionary methods, and strategic oscillation. These heuristics generally require domain-specific knowledge or problem-dependent genetic operators, have difficulties in finding feasible regions or in maintaining feasibility for nonlinear constraints, and get stuck in local minima easily.

2.2 Theory of Discrete Lagrange Multipliers

Lagrangian methods augment the original variable space X by a Lagrange-multiplier space Λ, and resolve constraints gradually. Although they are similar to dynamic-penalty methods, they are governed by strong mathematical conditions on optimality. Here, we summarize the theory of *discrete Lagrange multipliers* that works in discrete space [20, 26, 28].

Consider first a *discrete* equality-constrained NLP, a special case of (1):

$$\begin{array}{lll} \text{minimize}_x & f(x) & \text{where } x = (x_1, \ldots, x_n) \text{ is a vector} \\ \text{subject to} & h(x) = 0 & \text{of discrete variables.} \end{array} \tag{2}$$

Definition 5. The *generalized discrete augmented Lagrangian function* of (2) is:

$$L_d(x, \lambda) = f(x) + \lambda^T H\left(h(x)\right) + \frac{1}{2}||h(x)||^2, \tag{3}$$

where H is a continuous transformation function, and $||h(x)||^2 = \sum_{i=1}^{m} h_i^2(x)$.

Definition 6. Point (x^*, λ^*) is a *saddle point* in discrete space if it satisfies:

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*) \tag{4}$$

for all $x \in \mathcal{N}(x^*)$ and all possible λ. Note that the first inequality only holds when all constraints are satisfied and must be true for all λ.

Theorem 1. *First-order necessary and sufficient condition for discrete CLM* [26, 28]. In discrete space, if function H in (3) is a non-negative (or non-positive) continuous function that satisfies $H(x) = 0$ iff $x = 0$, then the set of CLM is the same as the set of discrete saddle points.

Requiring H to be non-negative (or non-positive) in Theorem 1 is easy to achieve. Two examples of H are the absolute function $H(h(x)) = |h(x)|$ and the square function $H(h(x)) = h^2(x)$.

For inequality constraint $g_j(x) \leq 0$ in (1), we first transform it into equivalent equality constraint $\tilde{g}_j(x) = \max(0, g_j(x)) = 0$. Since such a transformation is always possible, we only consider problems with equality constraints in the rest of this paper. Assuming H to be the absolution function, the *discrete augmented Lagrangian function* of (2) is:

$$L(x, \lambda) = f(x) + \lambda^T |h(x)| + \frac{1}{2} ||h(x)||^2. \tag{5}$$

The condition in Theorem 1 is stronger than its continuous counterpart. In continuous space, points that satisfy the first-order necessary and second-order sufficient conditions [15] are a *subset* of all the CLM. These conditions require the existence of derivatives of the objective and constraint functions and are not applicable when any of these functions is discontinuous. As a result, a CGM that satisfies these conditions is not necessarily a CGM of the original problem. In contrast, a saddle point in discrete space has a one-to-one correspondence with a CLM. Hence, finding a CGM amounts to finding the saddle point with the minimum objective.

Theorem 1 can be understood intuitively as follows. When x^* is a CLM in discrete space, it is always possible to enumerate all its neighboring points (since they are finite) and choose λ^* to be large enough so that (4) is satisfied. In addition, a saddle point must be a CLM. These two facts lead to Theorem 1.

Extensions to continuous and mixed-integer problems. Theorem 1 can be extended to apply to continuous and mixed-integer NLPs if all continuous variables are discretized. Intuitively, a discretized problem is a fairly good approximation to the original problem if the discretization level is sufficiently fine. The following theorem shows the problem-independent worst-case bound between f_c^*, the optimal solution in continuous space, and f_d^*, the best solution in discretized space.

Theorem 2. *Effect of discretization on solution quality* [27]. Let c^*, the optimal solution in continuous space, be located in a discretized space with distance $s_{grid,i}$ between adjacent grid points in the i^{th} dimension. Assume the following conditions.

```
1.  procedure CSA
2.      set starting point x = (x, λ);
3.      set starting temperature T = T_0 and cooling rate α;
4.      set N_T (number of trials per temperature);
5.      while stopping condition is not satisfied do
6.          for κ ← 1 to N_T do
7.              generate a trial point x' from N(x) using G(x, x');
8.              accept x' with probability A_T(x, x')
9.          end_for
10.         reduce temperature to T ←− α × T;
11.     end_while
12. end_procedure
```

Figure 1: CSA: the constrained simulated annealing algorithm (see text for the initial values of parameters).

a) Lipschitz condition [17] holds for all objective and constraint functions; that is, for any two points $x, y \in X$, $|f(x) - f(y)| \leq \ell_f \cdot \|x - y\|$, $|h_i(x) - h_i(y)| \leq \ell_{h_i} \cdot \|x - y\|$, $i = 1, \ldots, m$, where ℓ_f, ℓ_{h_i} are the corresponding Lipschitz constants (positive real numbers). We also define ℓ_f^{min}, $\ell_{h_i}^{min}, i = 1, \ldots, m$ to be the minimum positive real numbers that satisfy these conditions.

b) Constraint $h_i(x)$ is considered to be satisfied if $|h_i(x)| \leq \Phi$, where Φ is a prespecified maximum tolerance.

c) The interval size over all dimensions satisfy: $G = \frac{\sqrt{\sum_{i=1}^{n} s_{grid,i}^2}}{2} \leq \frac{\Phi}{max_i(\ell_{h_i}^{min})}$.

Then,

$$f_d^* - f_c^* \leq \ell_f^{min} \cdot G \tag{6}$$

Theorem 2 states a bound on the difference between the solution of a discretized NLP and that of the original NLP, and the asymptotic convergence of the discretized solution to the original one when the grid size approaches zero. It also shows trade-offs between the degree of discretization and solution quality: the finer is the discretization, the closer will be the solution to the original one. Assuming that the conditions of the theorem are satisfied, 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. In the rest of this paper, we do not distinguish solution methods for discrete, continuous, and mixed-integer NLPs.

3 Constrained Simulated Annealing

Based on the result in Theorem 1, constrained simulated annealing (CSA) [25] in Figure 1 looks for saddle points with the minimum objective value, *i.e.*, a CGM. It does probabilistic ascents in the Lagrange-multiplier space Λ and probabilistic descents in the original variable space X, with probabilities of acceptance governed by the Metropolis probability as a function of temperature T.

Line 2 initializes a starting point $\mathbf{x} = (x, \lambda)$, where $\lambda = 0$ and x can be user provided or randomly generated (*e.g.* using a fixed seed 123 in our experiments).

Line 3 chooses an initial control parameter, called *temperature*, to be large enough such that almost any trial point $\mathbf{x}'$ is accepted. The choice is based on the amount of initial violations observed in a problem. Here, we generate randomly 100 points of x and their corresponding neighboring points x', where each component $|x'_i - x_i| \leq 0.001$, then set $T_0 = max_{x,i}\{|L(x', 1) - L(x, 1)|, |h_i(x)|\}$. We also set α, the cooling rate of T, to be 0.8 or 0.9 in our experiments.

Line 4 sets N_T, the number of trials at each temperature. 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 [6], and uses $n + m$ instead of n because of the constraints.

Line 5 terminates CSA if $\mathbf{x}$ is not changed within some threshold for two successive temperatures, or the current T is small enough (*e.g.* $T < 10^{-6}$).

Line 7 generates a random trial point $\mathbf{x}'$ in $\mathcal{N}(\mathbf{x})$ from the current point $\mathbf{x} = (x, \lambda)$ in search space $S = X \times \Lambda$ according to generation probability $G(\mathbf{x}, \mathbf{x}')$, where

$$\mathcal{N}(\mathbf{x}) = \{(x', \lambda) \in S \text{ where } x' \in \mathcal{N}_1(x)\} \cup \{(x, \lambda') \in S \text{ where } \lambda' \in \mathcal{N}_2(\lambda)\} \quad (7)$$

and $\mathcal{N}_2(\lambda)$ at (x, λ) is the neighborhood of λ that satisfies the following property:

$$\begin{aligned} \mathcal{N}_2(\lambda) = &\{\mu \in \Lambda \mid \mu < \lambda, \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\} \\ &\cup \{\mu \in \Lambda \mid \mu > \lambda, \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\} \end{aligned} \quad (8)$$

Neighborhood $\mathcal{N}_2(\lambda)$ prevents λ_i from being changed when the corresponding constraint is satisfied, *i.e.*, $h_i(x) = 0$. As an example, $\mathcal{N}_2(\lambda)$ can be a function in which μ differs from λ in one variable (*e.g.* $\mu_i \neq \lambda_i$, and $\mu_j = \lambda_j$ for $j \neq i$), and $\{\mu_i\}$ is a set of values, some of which are larger than λ_i and some are smaller.

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

$$G(\mathbf{x}, \mathbf{x}') > 0 \quad \text{and} \quad \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} G(\mathbf{x}, \mathbf{x}') = 1. \quad (9)$$

The choice of $G(\mathbf{x}, \mathbf{x}')$ is arbitrary as long as it satisfies (9). In our experiments, we generate (x', λ) with higher probability than that of generating than (x, λ').

After generating $\mathbf{x}'$, Line 8 accepts $\mathbf{x}'$ with acceptance probability $A_T(\mathbf{x}, \mathbf{x}')$ that consists of two components, depending on whether x or λ is changed in $\mathbf{x}'$.

$$A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} e^{-[L(\mathbf{x}') - L(\mathbf{x})]^+ / T} & \text{if } \mathbf{x}' = (x', \lambda) \\ e^{-[L(\mathbf{x}) - L(\mathbf{x}')]^+ / T} & \text{if } \mathbf{x}' = (x, \lambda') \end{cases} \quad \text{where } a^+ = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

$A_T(\mathbf{x}, \mathbf{x}')$ differs from that used in conventional SA [1, 13] that only has the first part of (10). The goal in conventional SA is to look for global minima in the x space. Hence, it only needs probabilistic descents in that space and does not need the λ subspace.

In contrast, our goal is to look for saddle points in the joint space of x and λ that exist at local minima in the x subspace and at local maxima in the λ subspace. To this end, the first part of (10) carries out *probabilistic descents* of $L(x, \lambda)$ with

respect to x for fixed λ. That is, it accepts a new x' (under fixed λ) with probability one when $\delta_x = L(x', \lambda) - L(x, \lambda)$ is negative; otherwise, it accepts 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.

However, descents in the x subspace alone only lead to local/global minima of the Lagrangian function without satisfying the constraints. Hence, the second part of (10) 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, we generate a new λ' while x is fixed and 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 ascents in the λ subspace while allowing occasional descents. Note that, according to (8), a Lagrange multiplier does not change when its corresponding constraint in satisfied.

Finally, Line 10 reduces T after looping N_T times of generating $\mathbf{x}'$ and accepting them with probability $A_T(\mathbf{x}, \mathbf{x}')$ at a given T. Theoretically, if T is reduced slow enough, such as using a logarithmic cooling schedule, then CSA has been shown [25] to converge to a CGM of (1) with probability one as T approaches 0. The convergence property of CSA is stated without proof in the following theorem:

Theorem 3. Asymptotic convergence of CSA. [25] Under a logarithmically decreasing temperature schedule, CSA converges asymptotically to a CGM with probability one.

In practice, we reduce T using the following geometric cooling schedule,

$$T \longleftarrow \alpha \times T \tag{11}$$

where α is a constant smaller than 1 (typically between 0.5 and 0.99). At high T, any trial point is accepted with high probabilities, allowing the search to traverse a large space and overcome infeasible regions. As T is gradually reduced, its acceptance probability decreases, and at very low temperatures the algorithm behaves like a local search and looks for saddle points.

In summary, there are four major differences between SA [1, 13] and CSA [25]:

a) *Targeted problems:* SA was designed for solving *unconstrained* NLPs, whereas CSA was designed for *constrained* NLPs. In addition to minimizing objective function $f(x)$, CSA has to satisfy a set of nonlinear constraints $h(x) = 0$ and $g(x) \leq 0$. Hence, SA can be viewed as a special case of CSA in the absence of constraints.

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

c) *Search procedure:* SA does probabilistic descents in the 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 an objective function $f(x)$, whereas CSA is implicit in minimizing a virtual energy according to the GSA framework [23] instead of $L(x, \lambda)$, since minimizing $L(x, \lambda)$ cannot satisfy all the constraints.

d) *Converged solutions:* SA has asymptotic convergence to an *unconstrained* global minimum [1], while CSA has asymptotic convergence to a *CGM* [25].

4 Experimental Results on Constrained NLPs

In this section, we first evaluate various strategies to improve the performance of CSA in solving a set of continuous constrained NLPs. Then we compare it with evolutionary algorithms (EAs) [16, 14], a sequential-quadratic-programming (SQP) [5, 4, 21] package DONLP2 [22], as well as interval methods [8]. We also apply CSA to solve derived discrete and mixed-integer constrained NLPs that cannot be solved efficiently by existing methods.

4.1 Nonlinear Constrained Benchmarks

We chose two sets of continuous nonlinear constrained benchmarks in our experiments: a) ten problems G1-G10 [16, 14] and b) a collection of optimization benchmarks [9]. Problems G1-G10 [16, 14] were originally developed for testing and tuning various constraint handling techniques in evolutionary algorithms (EAs). Examples of techniques developed include keeping the search within feasible regions with some specific genetic operators and dynamic and adaptive penalty methods. The second set of benchmarks [9] were collected by Floudas and Pardalos and were derived from practical applications. All these problems have objective functions of various types (linear, quadratic, cubic, polynomial, and nonlinear) and linear/nonlinear constraints of equalities and inequalities. The number 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.

Due to a lack of large-scale discrete and mixed-integer benchmarks, we derive them from the two sets of continuous benchmarks [16, 14, 9] as follows. In generating a mixed-integer NLP, we assume that variables with odd indices are continuous and those with even indices are discrete. In discretizing continuous variable x_i in the 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:

$$A_i = \begin{cases} \{a_i + \frac{b_i - a_i}{s} j, j = 0, 1, \cdots, s\} & \text{if } b_i - a_i < 1 \\ \{a_i + \frac{1}{s} j, j = 0, 1, \cdots, (b_i - a_i)s\} & \text{if } b_i - a_i \geq 1 \end{cases} \quad \text{where } s = 10^7.$$

Hence, the discretized search spaces produced for discrete and mixed-integer NLPs are very huge, and it is impossible to enumerate all possible points. For example, for a problem with 10 discretized variables, the size of its 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 the continuous and the discretized versions, since a CLM in the continuous version should differ very little from the corresponding solution in the discretized version.

4.2 CSA Components and Strategies

In this subsection, we examine the strategies used in CSA that may affect its performance in solving discrete, continuous, and mixed-integer constrained NLPs. In our experiments, we assume that an equality constraint is satisfied if $\Phi = 10^{-5}$ (see Theorem 2).

Choice of neighborhoods. CSA consists of two major steps: generating trial points and accepting them based on an acceptance probability. In theory, any neighborhoods $\mathcal{N}_1(x)$ and $\mathcal{N}_2(\lambda)$ that satisfy (8) and Definition 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 when using finite cooling schedules.

In our implementation, we choose a simple neighborhood $\mathcal{N}_1(x)$ as the set of points x' that differ from x in one variable. Likewise, $\lambda' \in \mathcal{N}_2(\lambda)$ 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 (8) and Definition 1 are satisfied.

We characterize $\mathcal{N}_1(x)$ by a vector σ, where σ_i controls the size of the neighborhood along x_i. Similarly, we characterize $\mathcal{N}_2(\lambda)$ by a vector ϕ, where ϕ_i denotes the maximum possible perturbation along λ_i.

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

$$x' = x + \theta_i \, \mathbf{e}_i, \tag{12}$$

where $\mathbf{e}_i$ is a vector with its i^{th} component being 1 and the others 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 $[-\sigma_i, \sigma_i]$ [6]; b) *Gaussian*, where θ_i is generated from a Gaussian distribution with zero mean and variance σ_i [29]; and c) *Cauchy*, where θ_i is generated from a Cauchy density $f_d(x) = \frac{1}{\pi} \frac{\sigma_i}{\sigma_i^2 + x^2}$ [7, 29].

The major advantage [7, 29] 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 (12) 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 point (x, λ'). In generating $\mathbf{x}' = (x, \lambda')$ from $\mathbf{x} = (x, \lambda)$, we apply the following rule,

$$\lambda' = \lambda + \eta_j \, \mathbf{e}_j, \tag{13}$$

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 $[-\frac{2}{3}\phi_j, \frac{4}{3}\phi_j]$; 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.

Acceptance probabilities. After generating $\mathbf{x}' = (x', \lambda)$ or $\mathbf{x}' = (x, \lambda')$, $\mathbf{x}'$ is accepted according to the Metropolis acceptance probability (10).

Besides the Metropolis rule, we also evaluate three other possible acceptance rules studied in SA: Logistic acceptance rule [1, 19], Hastings' rule [12, 10], and Tsallis' rule [2, 11]. All these acceptance rules lead to asymptotic convergence [18], although they differ in solution quality when applied under a finite cooling schedule.

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:

$$\text{Logistic rule:} \quad A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} \frac{1}{1+e^{+\delta L_x/T}} & \text{if } \mathbf{x}' = (x', \lambda) \\ \frac{1}{1+e^{-\delta L_\lambda/T}} & \text{if } \mathbf{x}' = (x, \lambda'), \end{cases} \tag{14}$$

$$\text{Hastings' rule:} \quad A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} \frac{1+2\left[\frac{1}{2}\min\{e^{\delta L_x/T}, e^{-\delta L_x/T}\}\right]^\gamma}{1+e^{+\delta L_x/T}} & \text{if } \mathbf{x}' = (x', \lambda) \\ \frac{1+2\left[\frac{1}{2}\min\{e^{\delta L_\lambda/T}, e^{-\delta L_\lambda/T}\}\right]^\gamma}{1+e^{-\delta L_\lambda/T}} & \text{if } \mathbf{x}' = (x, \lambda'), \end{cases} \tag{15}$$

$$\text{Tsallis' rule:} \quad A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} \min\left\{1, [1-(1-q)\delta L_x/T]^{1/(1-q)}\right\} & \text{if } \mathbf{x}' = (x', \lambda) \\ \min\left\{1, [1+(1-q)\delta L_\lambda/T]^{1/(1-q)}\right\} & \text{if } \mathbf{x}' = (x, \lambda'), \end{cases} \tag{16}$$

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

Adaptive neighborhoods for x. During the course of CSA, we adaptively adjust $\mathcal{N}_1(x)$ by updating scale vector σ for x using a modified $1:1$ rate rule [6] in order to balance the ratio between accepted and rejected configurations:

$$\sigma_i = \begin{cases} \sigma_i \left[1 + \beta_0 (p_i - p_u)/(1 - p_u)\right] & \text{if } \ p_i > p_u \\ \sigma_i / \left[1 + \beta_1 (p_v - p_i)/p_v\right] & \text{if } \ p_i < p_v, \end{cases} \tag{17}$$

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:

$$\phi = w \otimes h(x) = [w_1 h_1(x), \cdots, w_m h_m(x)], \tag{18}$$

where $\otimes$ represents vector product. When $h_i(x)$ is satisfied, there is no need to update the corresponding λ_i, and thus $\phi_i = 0$. 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:

$$w_i = \begin{cases} \eta_0\, w_i & \text{if } h_i(x) > \tau_0 T \\ \eta_1\, w_i & \text{if } h_i(x) < \tau_1 T, \end{cases} \tag{19}$$

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.

Cooling schedules. In practice, we reduce T by a geometric cooling schedule (11), 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 $\alpha = 0.95$.

Performance comparisons of various 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$_1$; 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).

We use 12 difficult continuous benchmark problems: G1, G2 and G5 from [16, 14], and 2.1, 2.7.5, 5.2, 5.4, 6.2, 6.4, 7.2, 7.3 and 7.4 from [9], and their mixed-integer versions to test the various strategies.

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.

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:

$$r_t(i) = \begin{cases} t_x(i)/t_r(i) - 1 & \text{if } t_x(i) > t_r(i) \\ 1 - t_r(i)/t_x(i) & \text{if } t_x(i) < t_r(i), \end{cases} \tag{20}$$

$$r_f(i) = (f_x(i) - f_r(i))/|f_{best}|, \tag{21}$$

where f_{best} is the best-known solution for the problem. We use (20) to measure the symmetric speedup [24] in order to give equal weights to speedups and slowdowns but use (21) 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.

$$r_f = \frac{1}{R}\sum_{i=1}^{R} r_f(i), \qquad r_t = \frac{1}{R}\sum_{i=1}^{R} r_t(i). \tag{22}$$

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

Figure 2 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 generates more infeasible local points and gets stuck there. Among the four acceptance probabilities, Logistic, Hastings' and Tsallis' rules are worse, using either more running times or reaching worse solutions on the 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.

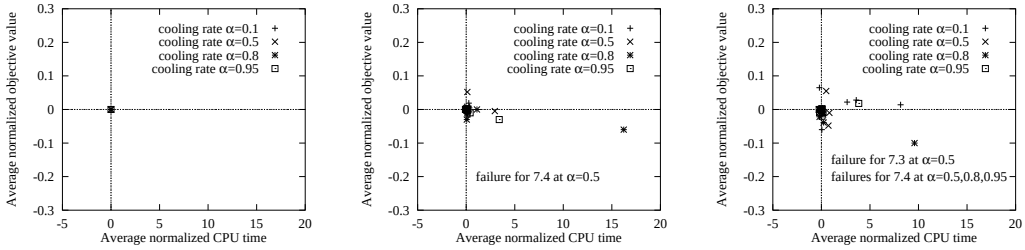
In short, CSA with (Cauchy_1, S-uniform, M) performs the best among all the combinations of strategies tested. Accordingly, we test CSA using (Cauchy_1, S-uniform, M) at cooling rate 0.8 in the following experiments. Note that the performance results may differ if different cooling rates are used.

Figures 3 thru 5 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 4l) in which CSA has much lower success ratio than that of solving the original continuous version (Figure 3l). 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.

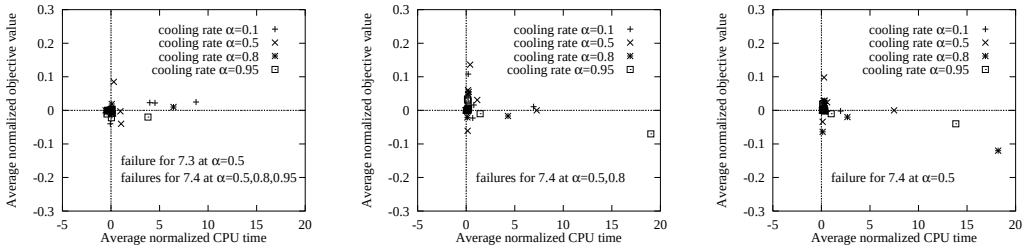
4.3 Comparison Results

In this subsection, we report experimental results of CSA based on (Cauchy_1, S-uniform, M) and $\alpha = 0.8$ on ten constrained NLPs G1-G10 [16, 14] and all of Floudas and Pardalos' benchmarks [9]. As a comparison, we also solved the continuous NLPs using DONLP2 [22], 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.

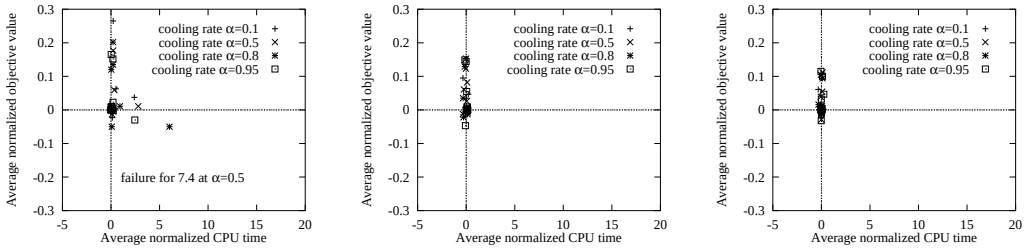
Table 1 compares the performance of CSA on *continuous* problems G1-G10. The first two columns show the problem IDs and the CGM (or constrained global maxima), if known. The next two columns show the best solutions obtained by EAs with specific constraint-handling techniques. The fifth thru eighth columns show



a) (Cauchy$_1$, S-uniform, M) b) (Cauchy$_0$, S-uniform, M) c) (Gauss, S-uniform, M)

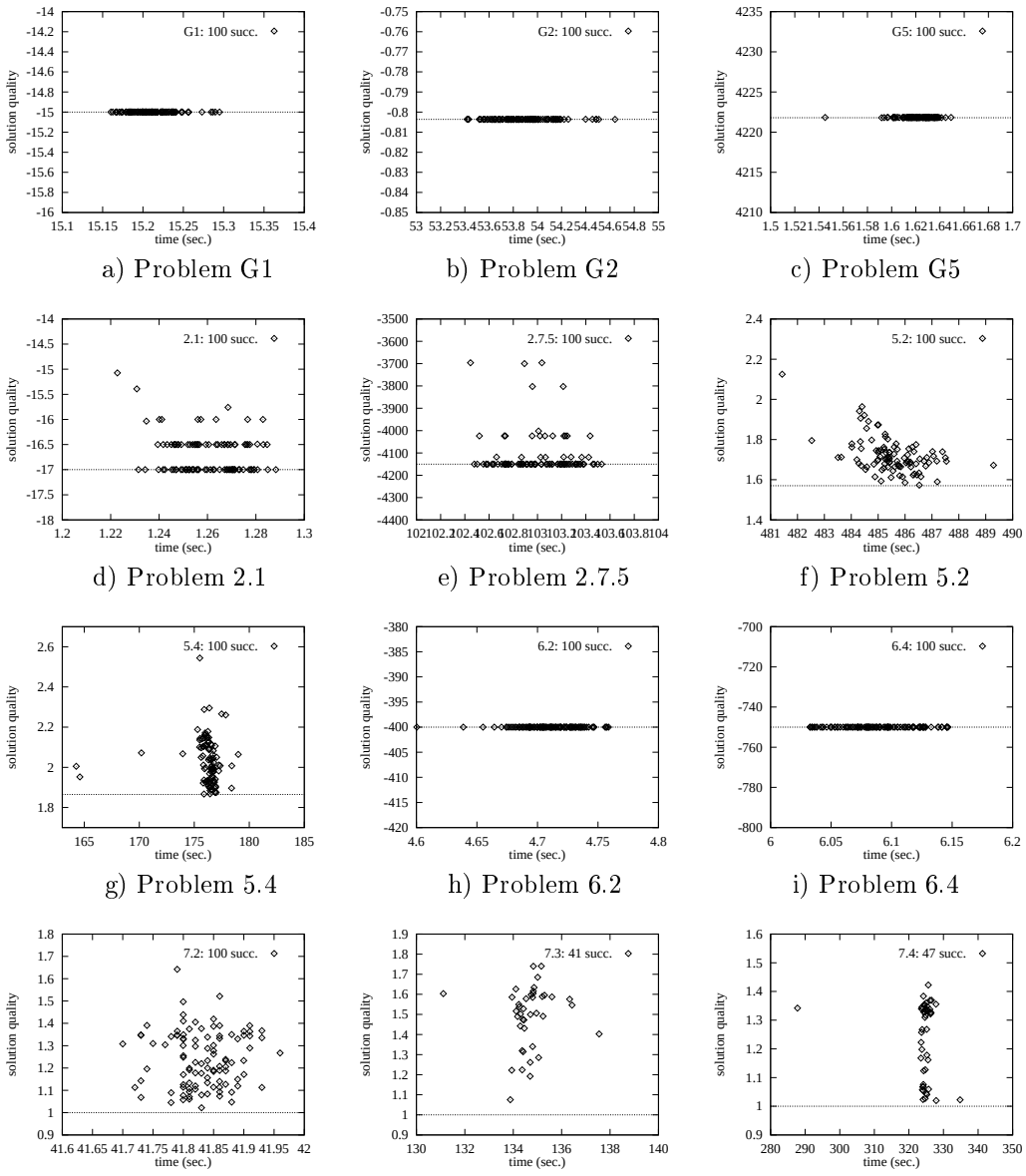


d) (Uniform, S-uniform, M) e) (Cauchy$_1$, S-uniform, L) f) (Cauchy$_1$, S-uniform, H)



g) (Cauchy$_1$, S-uniform, T) h) (Cauchy$_1$, NS-uniform, M) i) (Cauchy$_1$, Nonuniform,M)

Figure 2: 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.



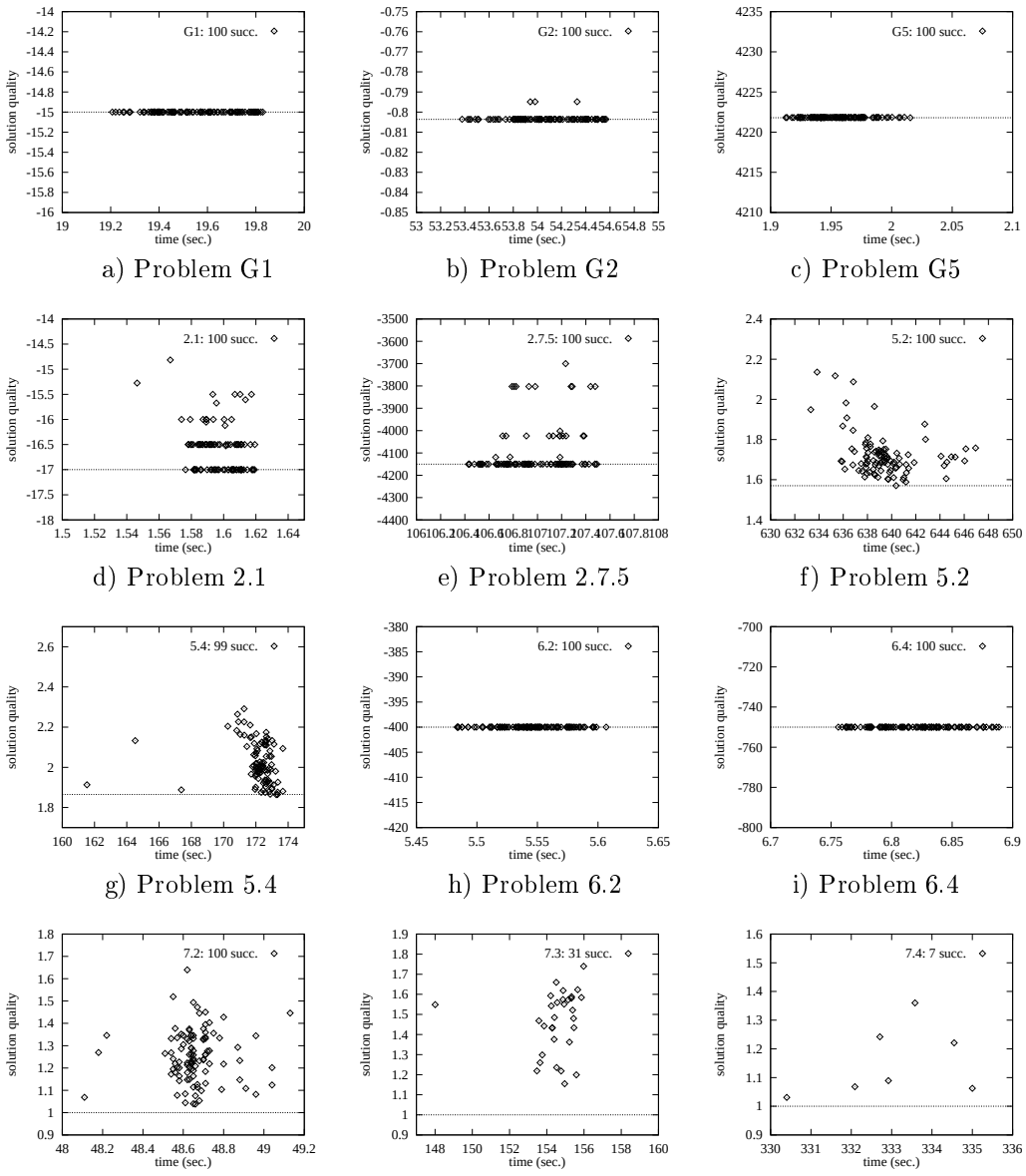
a) Problem G1 b) Problem G2 c) Problem G5

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 3: 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.)



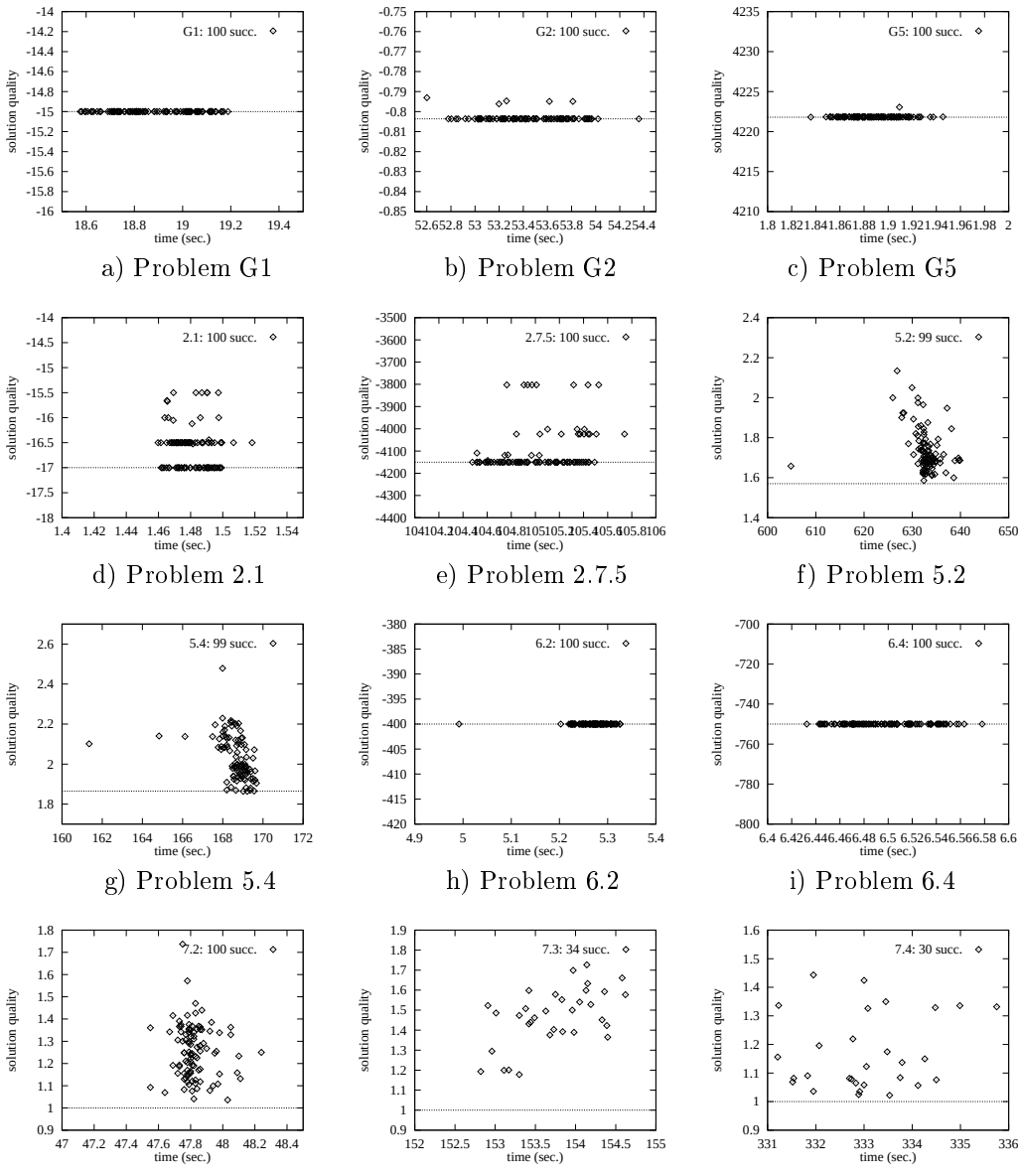
a) Problem G1 b) Problem G2 c) Problem G5

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: 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 5: 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.)

Table 1: Comparison results of DONLP2 (SQP), EA, and CSA in solving 10 continuous problems 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. Numbers in bold represent the best solutions. Both SQP and CSA use the same sequence of starting points.)

Problem ID	Global Solution	EAs		SQP: DONLP2 (100 runs)				CSA (100 runs)			
		best solution	specific method	best solution	average solution	fract. of best sol.	CPU time per run	best solution	average solution	fract. of best sol.	CPU time per run
G1 (min)	-15	**-15**	Genocop	**-15**	-12.83	11%	0.124	**-15**	-15	**100%**	15.2
G2 (max)	unknown	0.80355	S.T.	0.59701	0.24021	1.0%	0.663	**0.80362**	0.80362	**100%**	53.8
G3 (max)	1.0	0.999866	S.T.	**1.0**	1.0	92%	0.339	**1.0**	1.0	**100%**	37.2
G4 (min)	-30665.5	-30664.5	H.M.	**-30665.5**	-30665.5	66%	0.022	**-30665.5**	-30665.5	**100%**	1.76
G5 (min)	unknown	5126.498	D.P.	**4221.9**	4221.9	89.0%	0.021	**4221.9**	4221.9	**100%**	1.62
G6 (min)	-6961.81	**-6961.81**	Genocop	**-6961.81**	6961.81	91%	0.014	**-6961.81**	-6961.81	**100%**	0.518
G7 (min)	24.3062	24.62	H.M.	**24.3062**	24.3062	**100%**	0.047	**24.3062**	24.3062	**100%**	14.0
G8 (max)	unknown	**0.095825**	H.M.	**0.095825**	0.04485	25.0%	0.014	**0.095825**	0.095825	**100%**	0.798
G9 (min)	680.63	680.64	Genocop	**680.63**	680.63	**100%**	0.031	**680.63**	680.63	**100%**	4.48
G10 (min)	7049.33	7147.9	H.M.	**7049.33**	7049.33	63%	0.146	**7049.33**	7049.33	**100%**	4.69

Table 2: Performance of CSA in solving derived discrete and 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. Numbers in bold represent the best solutions.)

Problem ID	Global Solution	CSA for Discrete NLPs (100 runs)				CSA for Mixed-Integer NLPs (100 runs)			
		best solution	average solution	fract. of best sol.	CPU time per run	best solution	average solution	fract. of best sol.	CPU time per run
G1 (min)	-15	**-15**	-15	100%	19.5	**-15**	-15	100%	18.8
G2 (max)	unknown	**0.80362**	0.80334	97%	54.0	**0.80362**	0.80315	95%	53.4
G3 (max)	1.0	**1.0**	1.0	100%	37.1	**1.0**	1.0	100%	36.2
G4 (min)	-30665.5	**-30665.5**	-30665.5	100%	3.12	**-30665.5**	-30665.5	100%	2.98
G5 (min)	unknown	**4221.9**	4221.9	100%	1.95	**4221.9**	4221.9	100%	1.88
G6 (min)	-6961.81	**-6961.81**	-6961.81	100%	0.644	**-6961.81**	-6961.81	100%	0.605
G7 (min)	24.3062	**24.3062**	24.3062	100%	16.1	**24.3062**	24.3062	100%	15.6
G8 (max)	unknown	**0.095825**	0.095825	100%	0.792	**0.095825**	0.095825	100%	0.759
G9 (min)	680.63	**680.63**	680.63	100%	4.75	**680.63**	680.63	100%	4.61
G10 (min)	7049.33	**7049.33**	7049.33	100%	5.82	**7049.33**	7049.33	100%	5.43

Table 3: Performance comparison of Epperly's method [8] (an interval method), DONLP2 (SQP), and CSA in solving Floudas and Pardalos' continuous constrained benchmarks [9]. (All times are in CPU seconds on a Pentium-III 500-MHz computer running Solaris 7. CSA is based on (Cauchy$_1$, S-uniform, M) and $\alpha = 0.8$. Numbers in bold represent the best solutions. Both SQP and CSA use the same sequence of starting points.)

Problem ID No.	Global Solution	Epperly	SQP: DONLP2 (100 runs)				CSA (100 runs)			
		best solution	best solution	average solution	fract. of best sol.	CPU time per run	best solution	average solution	fract. of best sol.	CPU time per run
2.1 (min)	-17	**-17**	**-17**	-10.05	2%	0.017	**-17**	-16.68	**52%**	1.26
2.2 (min)	-213	**-213**	**-213**	-213	**100%**	0.022	**-213**	-213	**100%**	1.82
2.3 (min)	-15	**-15**	**-15**	-12.83	11%	0.124	**-15**	-15	**100%**	15.2
2.4 (min)	-11	**-11**	**-11**	-10.03	27%	0.039	**-11**	-11	**100%**	2.75
2.5 (min)	-268	**-268**	**-268**	-268	**100%**	0.085	**-268**	-268	**100%**	23.1
2.6 (min)	-39	**-39**	**-39**	-18.86	4%	0.095	**-39**	-38.9	**96%**	10.4
2.7.1 (min)	-394.75	**-394.75**	**-394.75**	-249.87	13%	0.702	**-394.75**	-394.75	**100%**	79.5
2.7.2 (min)	-884.75	**-884.75**	**-884.75**	-738.79	15%	0.744	**-884.75**	-884.75	**100%**	98.2
2.7.3 (min)	-8695.0	**-8695.0**	**-8695.0**	-5765.9	14%	0.516	**-8695.0**	-8666.1	**98%**	98.7
2.7.4 (min)	-754.75	**-754.75**	**-754.75**	-626.54	16%	0.716	**-754.75**	-754.75	**100%**	98.1
2.7.5 (min)	-4150.4	**-4150.4**	**-4150.4**	-3520.4	8%	0.483	**-4150.4**	-4112.8	**78%**	108.0
2.8 (min)	15990.0	15990.0	**15639.0**	18701.7	22%	1.51	**15639.0**	15639.0	**100%**	77.2
3.1 (min)	7049.33	-	**7049.33**	7049.33	63%	0.146	**7049.33**	7049.33	**100%**	4.69
3.2 (min)	-30665.5	**-30665.5**	**-30665.5**	-30665.5	66%	0.022	**-30665.5**	-30665.5	**100%**	1.76
3.3 (min)	-310.0	**-310.0**	**-310.0**	-222.0	3%	0.039	**-310.0**	-310.0	**100%**	3.04
3.4 (min)	-4.0	**-4.0**	**-4.0**	-3.94	24%	0.021	**-4.0**	-4.0	**100%**	0.859
4.3 (min)	-4.51	**-4.51**	**-4.51**	-4.47	32%	0.027	**-4.51**	-4.51	**100%**	1.51
4.4 (min)	-2.217	**-2.217**	**-2.217**	-2.20	14%	0.026	**-2.217**	-2.214	**92%**	1.68
4.5 (min)	-11.96	**-13.40**	**-13.40**	-13.10	2%	0.031	**-13.40**	-13.40	**100%**	3.63
4.6 (min)	-5.51	**-5.51**	**-5.51**	-4.25	28%	0.012	**-5.51**	-5.51	**100%**	0.538
4.7 (min)	-16.74	**-16.74**	**-16.74**	-16.74	**100%**	0.012	**-16.74**	-16.74	**100%**	0.481
5.2 (min)	1.567	-	**1.567**	1.63	**4%**	18.3	**1.567**	1.72	1%	485.1
5.4 (min)	1.86	-	**1.86**	1.89	**78%**	6.63	**1.86**	2.03	2%	176.1
6.2 (max)	400.0	**400.0**	**400.0**	395.5	88%	0.056	**400.0**	400.0	**100%**	4.71
6.3 (max)	600.0	**600.0**	**600.0**	560.4	89%	0.055	**600.0**	599.7	**98%**	5.37
6.4 (max)	750.0	**750.0**	**750.0**	664.3	79%	0.058	**750.0**	750.0	**100%**	5.95
7.2 (min)	1.0	-	**1.0**	1.02	**7%**	0.33	1.02	1.24	1%	41.83
7.3 (min)	1.0	-	**1.0**	1.13	**14%**	2.21	1.07	1.48	1%	134.5
7.4 (min)	1.0	-	**1.0**	1.01	**18%**	5.65	1.02	1.24	1%	443.1

the best and average solutions of DONLP2, the fraction of runs reaching the best solutions, and the average CPU time per run. The last four columns give the results on CSA.

Similarly, Table 3 reports the results on Floudas and Pardalos' *continuous* benchmarks [9]. The second column shows the global or the best-known solutions in [9] (normalized to one for problems 7.2, 7.3, and 7.4). The third column shows the best solutions obtained by one implementation of interval methods, called Epperly's method [8], whereas the other columns have the same meaning as that in Table 1.

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 discrete 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 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, CSA is the best in terms of both solution quality and ratio of reaching the best solutions. This is demonstrated in the solution of G2 that has a huge number of local optima. Although DONLP2 is generally very fast and works well if enough starting points were used, it has difficulty in solving G2. In 100 runs of DONLP2 to solve G2, only one was able to find the best solution of 0.59701, which is much worse than those obtained by EAs (0.803553) and CSA (0.803619). Even with 10,000 runs, DONLP2 was only able to find the best solution of 0.736554.

Another 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 to find CLM.

Third, interval methods, such as Epperly's implementation [8], 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. We plan to address this issue by techniques to generate better samples using Bayesian analysis, incorporate derivative information into sampling, and combine techniques in SQP into CSA.

Tables 2 and 4 show the results of applying CSA to solve our derived discrete and mixed-integer problems. The results show that CSA is robust as well as effective in solving continuous, discrete, and mixed-integer constrained NLPs.

Table 4: Performance results of CSA in solving derived discrete and mixed-integer NLPs based on Floudas and Pardalos' continuous constrained benchmarks [9]. (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. Numbers in bold represent the best solutions.)

Problem ID	Global Solution	CSA for Discrete Problems (100 runs)				CSA for Mixed-Integer Problems (100 runs)			
		best solution	average solution	fract. of best sol.	CPU time per run	best solution	average solution	fract. of best sol.	CPU time per run
2.1 (min)	-17	**-17**	-16.6	44%	1.59	**-17**	-16.6	41%	1.48
2.2 (min)	-213	**-213**	-213	100%	2.18	**-213**	-213	100%	2.01
2.3 (min)	-15	**-15**	-15	100%	17.4	**-15**	-15	100%	16.8
2.4 (min)	-11	**-11**	-11	100%	3.09	**-11**	-11	100%	2.89
2.5 (min)	-268	**-268**	-268	100%	24.2	**-268**	-268	100%	23.6
2.6 (min)	-39	**-39**	-38.9	96%	11.0	**-39**	-38.9	93%	10.6
2.7.1 (min)	-394.75	**-394.75**	-392.79	98%	105.1	**-394.75**	-394.75	100%	103.4
2.7.2 (min)	-884.75	**-884.75**	-878.23	92%	102.7	**-884.75**	-881.86	96%	101.2
2.7.3 (min)	-8695.0	**-8695.0**	-8695.0	100%	98.9	**-8695.0**	-8666.1	98%	97.3
2.7.4 (min)	-754.75	**-754.75**	-754.75	100%	102.7	**-754.75**	-754.75	100%	101.1
2.7.5 (min)	-4150.4	**-4150.4**	-4103.0	75%	115.6	**-4150.4**	-4119.3	81%	114.0
2.8 (min)	15990.0	**15639.0**	15639.0	100%	81.7	**15639.0**	15639.0	100%	79.5
3.1 (min)	7049.33	**7049.33**	7049.33	100%	5.82	**7049.33**	7049.33	100%	5.43
3.2 (min)	30665.5	**30665.5**	30665.5	100%	3.12	**30665.5**	30665.5	100%	2.98
3.3 (min)	-310.0	**-310.0**	-310.0	100%	2.70	**-310.0**	-310.0	100%	2.57
3.4 (min)	-4.0	**-4.0**	-4.0	100%	1.05	**-4.0**	-4.0	100%	1.01
4.3 (min)	-4.51	**-4.51**	-4.51	100%	1.78	**-4.51**	-4.51	100%	1.69
4.4 (min)	-2.217	**-2.217**	-2.214	92%	1.94	**-2.217**	-2.214	92%	1.86
4.5 (min)	-11.96	**-13.40**	-13.40	100%	3.95	**-13.40**	-13.40	100%	3.78
4.6 (min)	-5.51	**-5.51**	-5.51	100%	0.656	**-5.51**	-5.51	100%	0.618
4.7 (min)	-16.74	**-16.74**	-16.74	100%	0.601	**-16.74**	-16.74	100%	0.548
5.2 (min)	1.567	**1.567**	1.724	1%	639.0	1.586	1.734	1%	632.8
5.4 (min)	1.86	**1.86**	2.02	3%	172.0	**1.86**	2.03	3%	168.5
6.2 (max)	400.0	**400.0**	400.0	100%	5.54	**400.0**	400.0	100%	5.26
6.3 (max)	600.0	**600.0**	599.0	94%	5.82	**600.0**	599.2	94%	5.52
6.4 (max)	750.0	**750.0**	750.0	100%	6.92	**750.0**	750.0	100%	6.46
7.2 (min)	1.0	1.03	1.25	1%	48.6	1.03	1.26	1%	47.7
7.3 (min)	1.0	1.15	1.46	1%	153.8	1.17	1.46	1%	153.3
7.4 (min)	1.0	1.03	1.15	1%	332.3	1.02	1.17	1%	333.5

5 Conclusions

In this paper we have presented CSA, a general and robust algorithm for constrained global optimization. Based on the theory of discrete Lagrange multipliers, CSA is a powerful new method for solving discrete, continuous, and mixed-integer constrained nonlinear programming problems. The theory and algorithms proposed differ from conventional theory of Lagrange multipliers in four aspects.

a) The traditional theory in continuous space relies on regular points and the differentiability of (objective and constraint) functions and does not work when these conditions are not satisfied. In contrast, the theory of discrete Lagrange multipliers is based on saddle points defined using discrete neighborhoods and works for discontinuous as well as non-differentiable functions.

b) The first-order conditions in continuous space are necessary (augmented by second-order sufficient conditions), whereas the first-order condition in discrete space is necessary and sufficient and is much stronger.

c) Global minimization of constrained NLPs based on the first-order necessary and second-order sufficient conditions in continuous space may not lead to true global minima because a global minimum may not be a regular point or may not satisfy the first- and second-order conditions. Hence, traditional Lagrange-multiplier conditions are only useful for local optimization. In contrast, there is one-to-one correspondence between saddle points in discrete space and CLM, as shown in our first-order necessary and sufficient condition. Hence, global minimization can be achieved by searching in the space of saddle points. Our proposed CSA has asymptotic convergence to a discrete saddle point with the best objective value with probability one. The new CSA algorithm represents a major break-through over traditional simulated annealing that can achieve only asymptotic convergence with probability one in solving *unconstrained* optimization problems.

d) The theory in discrete space can be extended, under certain conditions, to continuous and mixed-integer spaces by discretizing continuous variables, whereas there is no such extension in the continuous Lagrange-multiplier theory.

We have studied in this paper various strategies in CSA. Based on discrete, continuous and mixed-integer benchmarks, we conclude that, when CSA is run under a finite cooling schedule, it should generate sample points based on a Cauchy distribution and use the Metropolis probability to accept newly generated points. In the future, we plan to study better cooling schedules and strategies to handle equality constraints.

References

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* J. Wiley and Sons, 1989.

[2] 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.

[3] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press, 1982.

[4] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, pages 1–52, 1995.

[5] 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.

[6] 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.

[7] P. Courrieu. The hyperbell algorithm for global optimization: A random walk using cauchy densities. *Journal of Global Optimization*, 10:37–55, 1997.

[8] T. Epperly. *Global Optimization of Nonconvex Nonolinear Programs Using Parallel Branch And Bound*. PhD thesis, University of Wisconsin-Madison, 1995.

[9] 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.

[10] B. Gidas. Non-stationary Markov chains and convergence of the annealing algorithm. *J. Statist. Phys.*, 39:73–131, 1985.

[11] U. H. E. Hansmann. Simulated annealing with tsallis weights: A numerical comparison. *Physica A*, 242:250–257, 1997.

[12] W. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[13] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[14] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[15] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, Reading, MA, 1984.

[16] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[17] J. D. Pinter. *Global Optimization in Action*. Nonconvex Optimization and Its Applications. Kluwer Academic, 1996.

[18] F. Romeo and A. L. Sangiovanni-Vincentelli. Probabilistic hill climbing algorithms: Properties and applications. In *Proc. Chapel Hill Conf. on VLSI*, pages 393–417, 1985.

[19] F. Romeo and A. L. Sangiovanni-Vincentelli. A theoretical framework for simulated annealing. *Algorithmica*, 6:302–345, 1991.

[20] 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.

[21] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Preprint, Department of Mathematics*, 1993.

[22] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming*, 82:413–448, 1998.

[23] A. Trouve. Cycle decomposition and simulated annealing. *SIAM Journal on Control and Optimization*, 34(3):966–986, 1996.

[24] 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.

[25] 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.

[26] 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.

[27] Z. Wu. *The Theory and Applications of Discrete Lagrange Multipliers in Constrained Nonlinear Programming.* Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, August 2000.

[28] Zhe 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.

[29] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Trans. on Evolutionary Computation*, 3(2):82–102, 1999.