Constrained Simulated Annealing with Applications in Nonlinear Continuous Constrained Global Optimization*

Benjamin W. Wah and Tao Wang
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
E-mail: {wah, wangtao}@manip.crhc.uiuc.edu

Abstract

This paper improves constrained simulated annealing (CSA), a discrete global minimization algorithm with asymptotic convergence to discrete constrained global minima with probability one. The algorithm is based on the necessary and sufficient conditions for discrete constrained local minima in the theory of discrete Lagrange multipliers. We extend CSA to solve nonlinear continuous constrained optimization problems whose variables take continuous values. We evaluate many heuristics, such as dynamic neighborhoods, gradual resolution of nonlinear equality constraints and reannealing, in order to greatly improve the efficiency of solving continuous problems. We report much better solutions than the best-known solutions in the literature on two sets of continuous optimization benchmarks.

1. Problem Definition

A general constrained minimization problem is formulated as follows:

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

where $x = (x_1, \ldots, x_n)$ is a vector of variables, $f(x)$ is a lower-bounded objective function, $h(x) = [h_1(x), \cdots, h_m(x)]^T$ 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 $g(x)$ 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 space X, is a user-defined set of all points $x' \in X$ such that $x \notin \mathcal{N}(x)$ and $x' \in \mathcal{N}(x) \Longleftrightarrow x \in \mathcal{N}(x')$, and that it is possible to reach every other x'' starting from any x in one or more steps through neighboring points.

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

Definition 3. Point $x \in X$ is called a *constrained local minimum* 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 point x may be a local minimum to one definition of neighborhood but may not be for another. The choice of neighborhood, however, does not affect the validity of a search as long as one definition is used consistently throughout.

Definition 4. Point $x \in X$ is called a *constrained global minimum* 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 constrained global minima is denoted by X_{opt}.

Finding constrained global minima of (1) is challenging as well as difficult. First, $f(x)$, $h(x)$ and $g(x)$ 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,

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

Proceedings 11th IEEE International Conference on Tools with Artificial Intelligence, November 1999.

as feasible regions may be disjoint and the search may need to visit multiple feasible regions before finding the global minimum. Second, $f(x)$, $h(x)$ and $g(x)$ may be discontinuous or may not have closed-form derivatives, rendering it impossible to apply existing theories and methods in continuous space. Third, there may be a large number of constrained local minima, trapping trajectories that only utilize local information and limiting their solution quality.

In this paper, we apply *constrained simulated annealing* (CSA) [11] to solve nonlinear continuous constrained optimization problems. We first overview the theory of discrete Lagrange multipliers [8, 12, 13] in Section 2, summarize the major steps of CSA in Section 3, develop heuristics to efficiently solve continuous optimization problems and report improvements over the best known solutions in solving some continuous constrained benchmark problems in Section 4.

2 Previous Work

Direct methods solve (1) directly. Examples include reject/discard methods, repair methods, feasible-direction methods, and interval methods. However, these methods are very problem specific, or unable to cope with nonlinear constraints, or computationally expensive. Therefore, the majority of methods to solve (1) first transform it into another form before solving it.

Static penalty methods transform (1) into a single unconstrained optimization problem [1, 6]. Due to the use of large penalties, their major limitations are the ruggedness of their search terrains and the depth of unconstrained local minima. Unless starting points are close to one of the unconstrained global minima, it is very unlikely for these methods to traverse large search spaces and find global solutions.

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. If it is too small, then the solution found may either be a constrained local minimum or even not be 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 when every unconstrained problem is solved optimally [1, 6]. The last requirement, however, is difficult to achieve in practice. If any one unconstrained problem is not solved optimally, then the process is not guaranteed to find a constrained global minimum.

In addition to penalty formulations, many heuristics have been developed to handle constraints. These include constraint handling techniques in GA [7], such as annealing penalties, adaptive penalties, preserving feasibility with specialized genetic operators, searching along 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. However, most heuristics 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 at local minima easily.

2.1 Discrete Lagrangian Theory

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 [8, 12, 13].

We first consider a discrete equality-constrained minimization problem, a special case of (1):

$$\begin{aligned} &\text{minimize}_x \quad f(x) \\ &\text{subject to} \quad h(x) = 0 \end{aligned} \tag{2}$$

where $x = (x_1, \ldots, x_n)$ is a vector of discrete variables.

Definition 5. The *generalized discrete Lagrangian function* of (2) is defined as follows:

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

where H is a continuous transformation function.

Definition 6. A point (x^*, λ^*) is *discrete 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 conditions for discrete constrained local minima* [12, 13]. In discrete space, if function H in (3) is a non-negative (or non-positive) continuous function satisfying $H(x) = 0$ iff $x = 0$, then the set of constrained local minima 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 $H(h(x)) = h^2(x)$.

In a similar way, we transform inequality constraint $g_j(x) \leq 0$ into equivalent equality constraint $\tilde{g}_j(x) = \max(0, g_j(x)) = 0$. As (1) can always be transformed into (2) with equality constraints, we only consider constrained problem (2) in this paper. The *discrete Lagrangian function* for (2), after using the absolute function as H, is

$$L(x, \lambda) = f(x) + \lambda^T |h(x)|. \quad (5)$$

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 [6] are a *subset* of all the constrained local minima. Hence, the global minima of points satisfying these conditions are not necessarily the constrained global minima of the original problem. Further, these conditions require the existence of derivatives of the objective and constraint functions and are not applicable when any one of these functions is discontinuous. In contrast, finding a saddle point in discrete space always leads to a constrained local minimum. Further, if one can find the saddle point with the minimum objective value, then it is the constrained global minimum.

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

3 Constrained Simulated Annealing

Constrained simulated annealing (CSA) in Figure 1 was developed [11] based on Theorem 1. It does probabilistic ascents in the Lagrange multiplier space Λ as well as probabilistic descents in the original variable space X, with probabilities of acceptance governed by temperature T. Its goal is to reach a saddle point that, according to (4), is at a local maximum in the Lagrange-multiplier space and at a local minimum in the original-variable space.

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

Line 3 chooses an initial control parameter, called *temperature*, to be large enough such that almost all

```
1.  procedure CSA
2.      set starting point x = (x, λ);
3.      set starting temperature T = T0 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 by T ← α × T;
11.     end_while
12. end_procedure
```

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

trial points $\mathbf{x}'$ are accepted. Here, we generate the initial temperature by first randomly generating 100 points of x and their corresponding neighboring points x', where each component $|x'_i - x_i| \leq 0.001$, then by setting $T_0 = max_{x,i}\{|L(x', 1) - L(x, 1)|, |h_i(x)|\}$. The rationale for this temperature is based on the amount of initial violations observed in a problem. 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 one temperature. In our implementation, we select $N_T = \zeta(20n+m)$ with the maximal value of ζ being $10(n+m)$, where n is the number of variables, and m is the number of constraints. This setting is based on the heuristic rule in [2], and uses $n + m$ instead of n because of the constraints.

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

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

$$\begin{aligned}\mathcal{N}(\mathbf{x}) = &\{(x', \lambda) \in S \text{ where } x' \in \mathcal{N}_1(x)\} \\ &\cup \{(x, \lambda') \in S \text{ where } \lambda' \in \mathcal{N}_2(\lambda)\}\end{aligned} \quad (6)$$

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

$$\begin{aligned}\mathcal{N}_2(\lambda) = &\{\mu \in \Lambda \mid \mu < \lambda \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\} \\ &\cup \{\mu \in \Lambda \mid \mu > \lambda \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\}\end{aligned} \quad (7)$$

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:

$$G(\mathbf{x}, \mathbf{x}') > 0 \quad \text{and} \quad \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} G(\mathbf{x}, \mathbf{x}') = 1 \tag{8}$$

The choice of $G(\mathbf{x}, \mathbf{x}')$ can be arbitrary as long as it satisfies (8). We use a nonuniform distribution with higher probability of generating (x', λ) than (x, λ') in our experiments.

After generating trial point $\mathbf{x}'$, 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 $\mathbf{x}'$.

$$A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} e^{-[L(\mathbf{x}')-L(\mathbf{x})]^+/T} & \text{if } \mathbf{x}' = (x', \lambda) \\ e^{-[L(\mathbf{x})-L(\mathbf{x}')]^+/T} & \text{if } \mathbf{x}' = (x, \lambda') \end{cases} \tag{9}$$

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

The acceptance probabilities $A_T(\mathbf{x}, \mathbf{x}')$ differ from those used in conventional SA that only has the first part of (9). The goal in conventional SA is to look for global minima in the x space. Without the λ space, only probabilistic descents in the x space are needed.

In our case, our goal is to look for saddle points in the joint space of x and λ, which exist at local minima in the x space and at local maxima in the λ space. To this end, the first part of (9) carries out *probabilistic descents* of $L(x, \lambda)$ with respect to x for fixed λ. That is, when we generate a new point x' while λ is fixed, we accept the new point 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 an occasional ascents in the x space as done in conventional SA.

However, descents in the x space alone only lead to local/global minima of the Lagrangian function without satisfying the constraints. Hence, the second part of (9) 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 new λ' while x is fixed, 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 ascents in the λ space while allowing occasional descents (to reduce the ruggedness of the terrain and the depth of local minima) as in conventional SA. Note that when a constraint in satisfied, its Lagrange multiplier will not change according to (7).

Finally, Line 10 reduces T after looping N_T times of generating trial points $\mathbf{x}'$ and accepting them with probabilities $A_T(\mathbf{x}, \mathbf{x}')$ at a given T. Theoretically, if T is reduced slow enough, for example, using a logarithmic cooling schedule, then CSA has been shown [11] to converge to a constrained global minimum of (1) with probability one as T approaches 0.

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

$$T \longleftarrow \alpha \times T \tag{10}$$

where α is a constant smaller than 1 (typically between 0.8 and 0.99). At high temperature 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, the acceptance probability decreases, and at very low temperatures the algorithm behaves like a local search and looks for saddle points.

Note that CSA does not minimize $L(x, \lambda)$ due to its probabilistic descents in the x space and probabilistic ascents in the λ space. This is quite different from SA for unconstrained problems, whose explicit objective is to minimize a single objective by performing probabilistic descents. In fact, CSA [11] minimizes an implicit virtual energy according to the framework of GSA [10], and asymptotically converges to the constrained global minimum with probability one.

4 Experimental Results on Continuous Constrained Problems

In this section, we evaluate CSA's performance on solving some continuous benchmark problems and compare it with evolutionary algorithms, sequential quadratic programming and interval methods. We first discuss the extension of CSA to continuous problems and show the evaluation results in Section 4.2.

4.1 Extensions to Continuous Constrained Optimization Problems

As numerical evaluations of continuous problems using digital computers can be considered as discrete approximations of the original problems up to the precision of computers, there should be little changes when CSA is applied to solve continuous constrained problems. In the following, we discuss various issues involved in designing an efficient CSA.

Characteristics of neighborhood. 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 x_i. 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 (7) and Definition 1 are satisfied.

Generation of trial points. We characterize $\mathcal{N}_1(x)$ (resp. $\mathcal{N}_2(\lambda)$) by a step vector θ (resp. ϕ), where θ_i (resp. ϕ_i) denotes the maximum possible perturbation along x_i (resp. λ_i). We generate a trial point $\mathbf{x}' = (x', \lambda)$ from $\mathbf{x} = (x, \lambda)$ as follows:

$$x' = x + r_0\, \theta_i\, \mathbf{e}_i \tag{11}$$

where r_0 is a random variable uniformly distributed in $[-1, +1]$, $\mathbf{e}_i$ is a vector with its i^{th} component being 1 and the other components being 0, and i is randomly generated from $\{1, 2, \cdots, n\}$. Similarly, we generate a trial point $\mathbf{x}' = (x, \lambda')$ from $\mathbf{x} = (x, \lambda)$ as follows:

$$\lambda' = \lambda + r_1\, \phi_j\, \mathbf{e}_j \tag{12}$$

where r_1 is randomly distributed in $[-1, +1]$, and j is uniformly distributed in $\{1, 2, \cdots, m\}$.

After generating trial point $\mathbf{x}' = (x', \lambda)$ or $\mathbf{x}' = (x, \lambda')$, $\mathbf{x}'$ is accepted according to (9). We set the ratio of generating (x', λ) and (x, λ') from (x, λ) to be $20n$ to m, *i.e.*, every variable x_i is tried 20 times more often than each Lagrange multiplier variable λ_j. Hence, x is updated more frequently than λ.

Adaptive neighborhoods. The neighborhoods are dynamically adjusted in CSA to achieve high precision. This is done by dynamically modifying step vector θ for x according to the so-called 1:1 rate rule [2] that maintains equal ratio between accepted and rejected configurations. When the ratio is low, θ should be reduced because too many trials of (x', λ) are rejected. In contrast, when the ratio is high, θ should be increased because trial points (x', λ) are too close to (x, λ).

We adjust ϕ according to the degree of constraint violations. Here, we decompose ϕ as:

$$\phi = w \otimes h(x) = [w_1 h_1(x), \cdots, w_m h_m(x)] \tag{13}$$

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:

$$w_i = \begin{cases} \eta_0\, w_i & \text{if } h_i(x) > \tau_0 T \\ \eta_1\, w_i & \text{if } h_i(x) < \tau_1 T \end{cases} \tag{14}$$

where $\eta_0 = 1.5$, η_1=0.9, $\tau_0 = 1.0$, 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 overweighted, 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.

Handling nonlinear equalities and reannealing. When the number of nonlinear equalities is large, the ratio of feasible space $\mathcal{F}$ to the whole search space X is almost 0. At this time, randomly generated trial points may have difficulty to hit a feasible region even if they are very close to it, because the search space is continuous and random sampling is discrete.

To overcome this problem, we propose to first relax some or all linear/nonlinear equalities into inequalities, and to gradually restrict the degree of relaxation until the original problem (2) is solved. Let the set of relaxed equalities be I_R and the remaining set of non-relaxed equalities be I_O. Then the relaxed problem becomes:

$$\begin{array}{lll} \text{minimize}_x & f(x) & x = (x_1, \ldots, x_n) \\ \text{subject to} & h_i(x) = 0 & i \in I_O \\ & |h_i(x)| \le \delta & i \in I_R \end{array} \tag{15}$$

where δ controls the degree of relaxation. When $\delta = 0$, the relaxed problem (15) becomes the original problem (2). Transforming inequalities $|h_i(x)| \le \delta$ into equivalent equalities $\tilde{h}_i(x) = \max(0, |h_i(x)| - \delta)$ as before, we define the corresponding Lagrangian function:

$$L(x, \lambda) = f(x) + \sum_{i \in I_O} \lambda_i |h_i(x)| + \sum_{i \in I_R} \lambda_i \tilde{h}_i(x). \tag{16}$$

We solve this relaxed problem using CSA in Figure 1 with the following modifications:

a) When the current point $\mathbf{x} = (x, \lambda)$ satisfies all the constraints of (15), we set $\delta \longleftarrow 0.95 \times \delta$ if $\delta > 10^{-6}$, and redo the loop in Line 6 of Figure 1, *i.e.* prevent loop index κ from increment by one. This step allows equalities to be gradually resolved.

b) Reannealing is needed when the search is trapped in the relaxed problem and cannot proceed with smaller δ. In this case, the search stops at a region where all the constraints of (15) for a given δ are satisfied, but the region does not contain any points that satisfy these constraints with smaller δ. If T is already low, the chance of escaping from this region is small, and reannealing is required. Empirically, we do reannealing by increasing the current temperature to:

$$T = \min(T_0, 100 \times \delta) \tag{17}$$

In our experiments, if $\delta < 10^{-6}$, we consider the relaxed problem (15) to be equal to the original

Table 1. Comparison results of DONLP2 (SQP), GA, and CSA for 10 test problems. (S.T. stands for strategic oscillation, H.M. for homomorphous mappings, and D.P. for dynamic penalty. All times are seconds on a Sun SparcStation Ultra 5 computer. Numbers in bold represent the best solutions.)

Problem ID	Global Solution	EAs		SQP: DONLP2 (100 runs)				CSA (100 runs)			
		best solution	specific method	best solution	average solution	fract. of best sol.	time per run	best solution	average solution	fract. of best sol.	time per run
G1 (min)	-15	**-15**	Genocop	**-15**	-12.65	12%	0.083	**-15**	-15	**100%**	3.45
G2 (max)	unknown	0.803553	S.T.	0.391153	0.234977	1.0%	0.429	**0.803619**	0.802829	**91%**	46.8
G3 (max)	1.0	0.999866	S.T.	**1.0**	1.0	89%	0.309	**1.0**	1.0	**100%**	7.56
G4 (min)	-30665.5	-30664.5	H.M.	**-30665.5**	-30665.5	63%	0.0208	**-30665.5**	-30665.5	**100%**	1.19
G5 (min)	unknown	5126.498	D.P.	**4221.956**	4221.956	96.0%	0.0193	**4221.956**	4221.956	**100%**	7.39
G6 (min)	-6961.81	**-6961.81**	Genocop	**-6961.81**	6961.81	89%	0.0164	**-6961.81**	-6961.81	**100%**	0.384
G7 (min)	24.3062	24.62	H.M.	**24.3062**	24.3062	99%	0.0361	**24.3062**	24.3062	**100%**	3.40
G8 (max)	unknown	**0.095825**	H.M.	**0.095825**	0.046660	30.0%	0.0157	**0.095825**	0.095825	**100%**	0.248
G9 (min)	680.63	680.64	Genocop	**680.63**	680.63	**100%**	0.0293	**680.63**	680.63	**100%**	0.956
G10 (min)	7049.33	7147.9	H.M.	**7049.33**	7049.33	70%	0.134	**7049.33**	7049.33	**100%**	4.08

problem (2), and report the results obtained by solving (15).

Adaptive N_T. In order for CSA to converge asymptotically, enough trial points must be generated before reducing T. This requires succinct choice of N_T, because small N_T leads to poor solutions, whereas large N_T leads to wasted evaluations. In our experiments, we choose N_T in an adaptive fashion as follows:

1. Start from small $N_T = \zeta(20n + m)$ by setting $\zeta = 5$.

2. Solve the problem by CSA using given N_T.

3. If the same solution is obtained for two consecutive N_T, or if $\zeta > \zeta_{max}$ where $\zeta_{max} = 10(n + m)$, then stop and output the solution; otherwise, set $\zeta \longleftarrow 2 \times \zeta$, and go to Step 2.

In the worst case, this procedure has twice the computational complexity of CSA with the maximal value of ζ_{max}. In practice, the procedure can finish with small N_T in most of our experiments, and is much faster than CSA with ζ_{max}. Note that this procedure is not used to solve the relaxed problem (15), because the relaxed problem requires a large N_T in order to allow enough time for δ to be reduced from a large value of 1.0 to a small value of 10^{-6}.

4.2 Evaluation Results

In this section, we show experimental results of testing CSA on 10 constrained optimization problems G1-G10 [7, 5] and a collection of constrained optimization benchmarks [4]. These problems have objective functions of various types (linear, quadratic, cubic, polynomial, and nonlinear) and constraints of linear inequalities, nonlinear equalities, and nonlinear 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.

Problems G1-G10 [7, 5] were originally invented for evolutionary algorithms (EAs) in which various constraint handling techniques were developed and well tuned for each problem in order to get good results. Examples of these techniques include keeping the search within feasible regions with some specific genetic operators, and dynamic and adaptive penalty methods. The second set of benchmarks [4] were collected by Floudas and Pardalos and were derived from practical applications.

We also solved the problems using DONLP2 [9], a popular sequential-quadratic-programming (SQP) package. SQP is an efficient local-search method widely used for solving constrained optimization problems. Its quality depends heavily on its starting points since it is a local search.

Table 1 shows the comparison results on G1-G10. The first two columns show the problem IDs and the constrained global minima (or maxima), if known. The third and fourth columns show the best solutions obtained by EAs and the specific constraint handling techniques used to generate the solutions. The fifth thru eighth columns show 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 of CSA.

Table 2. Comparison results of Epperly's method [3](an interval method), DONLP2 (SQP), and CSA on a collection of constrained optimization benchmarks [4]. All times are CPU seconds on a Sun SparcStation Ultra 5 computer. Numbers in bold represent the best solution.

Problem ID No.	Global Solution	Epperly	SQP: DONLP2 (100 runs)				CSA (100 runs)			
		best solution	best solution	average solution	fract. of best sol.	time per run	best solution	average solution	fract. of best sol.	time per run
2.1 (min)	-17	**-17**	**-17**	-10.53	4%	0.0156	**-17**	-16.84	**74%**	2.09
2.2 (min)	-213	**-213**	**-213**	-213	**100%**	0.0203	**-213**	-213	**100%**	0.712
2.3 (min)	-15	**-15**	**-15**	-12.65	12%	0.0835	**-15**	-15	**100%**	3.46
2.4 (min)	-11	**-11**	**-11**	-10.05	33%	0.0315	**-11**	-11	**100%**	0.921
2.5 (min)	-268	**-268**	**-268**	-268	**100%**	0.053	**-268**	-268	**100%**	3.89
2.6 (min)	-39	**-39**	**-39**	-20.75	11%	0.0619	**-39**	-39	**100%**	3.55
2.7.1 (min)	-394.75	**-394.75**	**-394.75**	-259.86	19%	0.414	**-394.75**	-390.39	**95%**	17.2
2.7.2 (min)	-884.75	**-884.75**	**-884.75**	-757.41	25%	0.416	**-884.75**	-881.86	**96%**	16.59
2.7.3 (min)	-8695.0	**-8695.0**	**-8695.0**	-5700.3	17%	0.34	**-8695.0**	-8695.0	**100%**	13.8
2.7.4 (min)	-754.75	**-754.75**	**-754.75**	-622.37	16%	0.415	**-754.75**	-719.34	**69%**	13.3
2.7.5 (min)	-4150.4	**-4150.4**	**-4150.4**	-3456.3	4%	0.436	**-4150.4**	-4116.9	**81%**	67.6
2.8 (min)	15990.0	15990.0	**15639.0**	24321.3	15%	1.29	**15639.0**	15639.0	**100%**	27.9
3.1 (min)	7049.33	-	**7049.33**	7049.33	70%	0.134	**7049.33**	7049.33	**100%**	4.08
3.2 (min)	-30665.5	**-30665.5**	**-30665.5**	-30665.5	63%	0.020 8	**-30665.5**	-30665.5	**100%**	1.19
3.3 (min)	-310.0	**-310.0**	**-310.0**	-215.3	11%	0.0318	**-310.0**	-310.0	**100%**	0.686
3.4 (min)	-4.0	**-4.0**	**-4.0**	-3.96	24%	0.0216	**-4.0**	-4.0	**100%**	0.279
4.3 (min)	-4.51	**-4.51**	**-4.51**	-4.47	44%	1.12	**-4.51**	-4.51	**100%**	2.71
4.4 (min)	-2.217	**-2.217**	**-2.217**	-2.20	12%	1.49	**-2.217**	-2.217	**100%**	4.03
4.5 (min)	-11.96	**-13.40**	**-13.40**	-13.40	3%	1.94	**-13.40**	-13.40	**100%**	7.95
4.6 (min)	-5.51	**-5.51**	**-5.51**	-4.42	34%	0.0126	**-5.51**	-5.51	**100%**	0.146
4.7 (min)	-16.74	**-16.74**	**-16.74**	-16.74	**100%**	0.0131	**-16.74**	-16.74	**100%**	0.228
5.2 (min)	1.567	-	1.576	1.725	1%	12.2	**1.567**	1.593	**31%**	3950.0
5.4 (min)	1.86	-	**1.86**	1.95	69%	4.28	**1.86**	1.88	**93%**	705.8
6.2 (max)	400.0	**400.0**	**400.0**	399.5	89%	0.0417	**400.0**	398.5	**98%**	10.5
6.3 (max)	600.0	**600.0**	**600.0**	574.8	87%	0.0468	**600.0**	591.8	**89%**	25.3
6.4 (max)	750.0	**750.0**	**750.0**	648.3	73%	0.0464	**750.0**	750.0	**100%**	11.3

Tables 2 reports the results on Floudas and Pardalos' benchmark collection [4]. The second column shows the global or the best known solutions in [4]. The third column shows the best solutions obtained by one implementation of interval methods, called Epperly's method [3]. The other columns have the same meanings as those in Table 1.

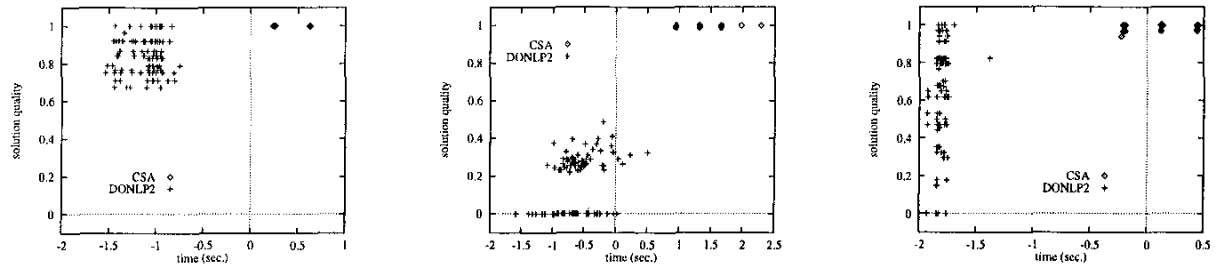
Based on our evaluation results, we have the following observations.

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 easily find the optimal solution. The main reason is that those constraint handling techniques do not guarantee constraint satisfaction and have difficulty in finding a global minimum while satisfying all the constraints. Another reason may be due to sampling that may not be able to find exact solutions to continuous problems. EAs were only able to find the best solutions in three of the ten problems in G1-G10 despite extensive tuning.

Second, CSA is the best in terms of both solution quality and ratio of reaching the best solutions. As DONLP2 is a local search algorithm, it worked well for problems with a small number of local minima if enough starting points were used. For these problems, DONLP2 was generally very fast and was able to complete within one second in many runs. However, DONLP2 has difficulty in solving problems with a huge number of local optima, such as G2. In 100 runs of DONLP2 to solve G2, only one was able to find the best solution of 0.391153, 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 major disadvantage of DONLP2 is that it needs derivatives of the Lagrangian function, making it suitable only for continuous problems with derivatives. This seriously limits its applicability to real engineering optimization problems, whose derivatives may be hard to calculate or unavailable (*e.g.* discrete problems and combinatorial optimization problems).

To illustrate the trade-offs between solution time and solution quality, Figure 2 shows typical quality-time distributions for CSA and DONLP2 where time is in logarithmic scale. Obviously, SQP is very efficient in solving some continuous problems from multiple start-



(a) Problem G1 (b) Problem G2 (c) Problem 2.1

Figure 2. Typical distributions of solution quality and evaluation time for CSA and SQP.

ing points, especially those with polynomial objective and constraints such as G1 and 2.1.

Third, interval methods, such as Epperly's method [3], have difficulties in solving problems with nonlinear equalities whose lower bounds are difficult to determine. Examples include Problems 5.2 and 5.4 in which feasible points were not found.

References

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

[2] 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.

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

[4] 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.

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

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

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

[8] Y. Shang and B. W. Wah. A discrete Lagrangian based global search method for solving satisfiability problems. *J. Global Optimization*, 12(1):61–99, January 1998.

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

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

[11] B. W. Wah and T. Wang. Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. *Principles and Practice of Constraint Programming*, (accepted to appear) October 1999.

[12] B. W. Wah and Z. Wu. The theory of discrete lagrange multipliers for nonlinear discrete optimization. *Principles and Practice of Constraint Programming*, (accepted to appear) October 1999.

[13] 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.