Hybrid Constrained Simulated Annealing and Genetic Algorithms for Nonlinear Constrained Optimization

Benjamin W. Wah and **Yixin Chen**
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
E-mail: {wah, chen}@manip.crhc.uiuc.edu
URL: http://manip.crhc.uiuc.edu

Abstract- This paper presents a framework that unifies various search mechanisms for solving constrained nonlinear programming (NLP) problems. These problems are characterized by functions that are not necessarily differentiable and continuous. Our proposed framework is based on the first-order necessary and sufficient condition for constrained local minimization in discrete space that shows the equivalence between discrete-neighborhood saddle points and constrained local minima. To look for discrete-neighborhood saddle points, we formulate a discrete constrained NLP in an augmented Lagrangian function and study various mechanisms for performing ascents of the augmented function in the original-variable subspace and descents in the Lagrange-multiplier subspace. Our results show that $CSAGA$, a combined constrained simulated annealing (SA) and genetic algorithm (GA), performs well. Finally, we apply iterative deepening to determine the optimal number of generations in $CSAGA$ and show that performance is robust with respect to changes in population size.

1 Introduction

A discrete constrained *nonlinear programming problem* (NLP) is formulated as follows:[1]

$$\begin{array}{lll} \min\limits_{x} & f(x) & \\ \text{subject to} & g(x) \leq 0 & x = [x_1, \ldots, x_n]^T \text{ is a vector} \\ & h(x) = 0 & \text{of bounded discrete variables.} \end{array} \quad (1)$$

Here, $f(x)$ is a lower-bounded objective function, $g(x) = [g_1(x), \cdots, g_k(x)]^T$ is a vector of k inequality constraints, $h(x) = [h_1(x), \cdots, h_m(x)]^T$ is a vector of m equality constraints. Functions $f(x)$, $g(x)$, and $h(x)$ are not necessarily differentiable and can be either linear or nonlinear, continuous or discrete, and analytic or procedural. Without loss of generality, we consider only minimization problems.

Let X be the Cartesian product of the discrete domains of all variables in x. We characterize solutions in discrete space as follows:

[1] For two vectors v and w of the same number of elements, $v \leq w$ means that each element of v is not less than the corresponding element of w. $v \geq w$ can be defined similarly. 0, when compared to a vector, stands for a null vector.

Definition 1. $\mathcal{N}_{dn}(x)$, the *discrete neighborhood* [1] of point $x \in X$ is a *finite* user-defined set of points $\{x' \in X\}$ such that $x' \in \mathcal{N}_{dn}(x) \iff x \in \mathcal{N}_{dn}(x')$, and that for any $y^1, y^k \in X$, it is possible to find a finite sequence of points in X, $y^1, \cdots, y^k$, such that $y^{i+1} \in \mathcal{N}_{dn}(y^i)$ for $i = 1, \cdots k-1$.

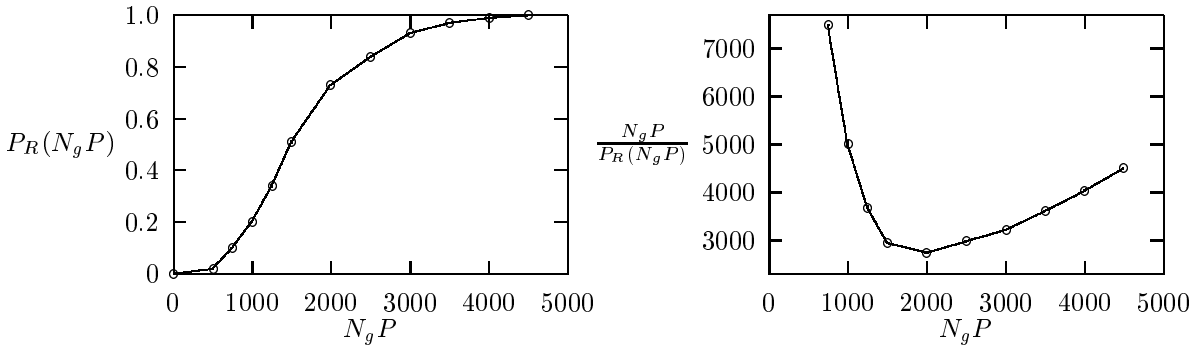
Definition 2. Point $x \in X$ is called a *constrained local minimum in discrete neighborhood* (CLM_{dn}) if it satisfies two conditions: a) x is feasible, and b) $f(x) \leq f(x')$, for all feasible $x' \in \mathcal{N}_{dn}(x)$.

Definition 3. Point $x \in X$ is called a *constrained global minimum in discrete neighborhood* (CGM_{dn}) iff a) x is feasible, and b) for every feasible point $x' \in X$, $f(x') \geq f(x)$. The set of all CGM_{dn} is X_{opt}.

We have shown earlier [10] that the necessary and sufficient condition for a point to be a CLM_{dn} is that it satisfies the discrete-neighborhood saddle-point condition (Section 2.1). We have also extended simulated annealing (SA) [9] and greedy search [10] to look for discrete-neighborhood saddle points SP_{dn} (Section 2.2). At the same time, new problem-dependent constraint-handling heuristics have been developed in the GA community to handle nonlinear constraints [7] (Section 2.3). Up to now, there is no clear understanding on how to unify these algorithms into one that can be applied to find CGM_{dn} for a wide range of problems.

Based on our previous work, our goal in this paper is to develop an effective framework that unifies SA, GA, and greedy search for finding CGM_{dn}. In particular, we propose *constrained genetic algorithm* (CGA) and *combined constrained SA and GA* ($CSAGA$) that look for SP_{dn}. We also study algorithms with the optimal average completion time for finding a CGM_{dn}.

The algorithms studied in this paper are all stochastic searches that probe a search space in a random order, where a probe is a neighboring point examined by an algorithm, independent of whether it is accepted or not. Assuming p_j to be the probability that an algorithm finds a CGM_{dn} in its jth probe and a simplistic assumption that all probes are independent, the performance of one run of such an algorithm can be characterized by N, the number of probes made (or CPU time taken), and $P_R(N)$, the *reachability probability* that a



a) $P_R(N_gP)$ approaches one asymptotically

b) Existence of absolute minimum $N_{opt}P$ in $\frac{N_gP}{P_R(N_gP)}$

Figure 1: An example showing the application of $CSAGA$ with $P = 3$ to solve a discretized version of G1 [7] ($N_{opt}P \approx 2000$).

CGM_{dn} is hit in any of the N probes:

$$P_R(N) = 1 - \prod_{j=1}^{N}(1 - p_j), \quad \text{where } N \geq 0. \tag{2}$$

Reachability can be maintained by reporting the best solution found by the algorithm when it stops.

As an example, Figure 1a plots $P_R(N_gP)$ when $CSAGA$ (see Section 3.2) was run under various number of generations N_g and fixed population size $P = 3$ (where $N = N_gP$). The graph shows that $P_R(N_gP)$ approaches one asymptotically as N_gP is increased.

Although it is hard to estimate the exact value of $P_R(N)$ when an algorithm is applied to solve a test problem, we can always improve the chance of finding a solution by running the same algorithm multiple times, each with N probes, from random starting points. Given $P_R(N)$ for one run of the algorithm and that all runs are independent, the expected total number of probes to find a CGM_{dn} is:

$$\sum_{j=1}^{\infty} P_R(N)(1 - P_R(N))^{j-1} N \times j = \frac{N}{P_R(N)}. \tag{3}$$

Figure 1b plots (3) based on $P_R(N_gP)$ in Figure 1a. In general, there exists N_{opt} that minimizes (3) because $P_R(0) = 0$, $\lim_{N_\alpha \to \infty} P_R(N_\alpha) = 1$, $\frac{N}{P_R(N)}$ is bounded below by zero, and $\frac{N}{P_R(N)} \to \infty$ as $N \to \infty$. The curve in Figure 1b illustrates this behavior.

Based on the existence of N_{opt}, we present in Section 3.3 search strategies in CGA and in $CSAGA$ that minimize (3) in finding a CGM_{dn}. Finally, Section 4 compares the performance of our algorithms.

2 Previous Work

We summarize the Lagrangian theory and associated algorithms for solving (1) and related work in GA.

2.1 Theory of Lagrange multipliers for solving discrete constrained NLPs

Define a discrete equality-constrained NLP as follows:

$$\begin{array}{lll} \min\limits_{x} & f(x) & x \text{ is a vector of bounded} \\ \text{subject to} & h(x) = 0 & \text{discrete variables,} \end{array} \tag{4}$$

A *generalized discrete augmented Lagrangian function* of (4) is defined as follows [10]:

$$L_d(x, \lambda) = f(x) + \lambda^T H\left(h(x)\right) + \frac{1}{2}||h(x)||^2, \tag{5}$$

where H is a non-negative continuous transformation function satisfying $H(y) \geq 0$, $H(y) = 0$ iff $y = 0$, and $\lambda = [\lambda_1, \cdots, \lambda_m]^T$ is a vector of Lagrange multipliers.

Function H is easy to design; examples include $H(h(x)) = [|h_1(x)|, \cdots, |h_m(x)|]^T$ and $H(h(x)) = [\max(h_1(x), 0), \cdots, \max(h_m(x), 0)]^T$.

Similar transformations can be used to transform inequality constraint $g_j(x) \leq 0$ into equivalent equality constraint $\max(g_j(x), 0) = 0$. Hence, we only consider problems with equality constraints from here on.

We define a *discrete-neighborhood saddle point* $SP_{dn}(x^*, \lambda^*)$ with the following property:

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*) \tag{6}$$

for all $x \in \mathcal{N}_{dn}(x^*)$ and all $\lambda, \lambda' \in R^m$. Note that although we use similar terminologies as in continuous space, SP_{dn} is different from SP_{cn} (saddle point in continuous space) because they are defined using different neighborhoods.

Theorem 1. *First-order necessary and sufficient condition on CLM_{dn}.* [10] A point in the discrete search space of (4) is a CLM_{dn} iff it satisfies (6) for any $\lambda \geq \lambda^*$.

Theorem 1 is stronger than its continuous counterparts. The first-order necessary conditions in continuous Lagrange-multiplier theory [5] require CLM_{cn} to be regular points and functions to be differentiable. In contrast, there are no such requirements for CLM_{dn}. Further, the first-order conditions

1. **procedure CSA** (α, N_α)
2. set initial $\mathbf{x} \leftarrow [x_1, \cdots, x_n, \lambda_1, \cdots, \lambda_k]^T$ with random x, $\lambda \leftarrow 0$;
3. **while** stopping condition is not satisfied **do**
4. generate $\mathbf{x}' \in \mathcal{N}_{dn}(\mathbf{x})$ using $G(\mathbf{x}, \mathbf{x}')$;
5. accept $\mathbf{x}'$ with probability $A_T(\mathbf{x}, \mathbf{x}')$
6. reduce temperature by $T \leftarrow \alpha T$;
7. **end_while**
8. **end_procedure**

a) CSA called with cooling schedule N_α and rate α

1. **procedure** CSA_{ID}
2. set initial cooling rate $\alpha \leftarrow \alpha_0$ and $N_\alpha \leftarrow N_{\alpha_0}$;
3. set $K \leftarrow$ number of CSA runs at fixed α;
4. **repeat**
5. **for** $i \leftarrow 1$ **to** K **do call** $CSA(\alpha, N_\alpha)$; **end_for**;
6. increase cooling schedule $N_\alpha \leftarrow \rho N_\alpha$;
7. **until** feasible solution has been found **and** no better solution in two successive increases of N_α;
8. **end_procedure**

b) CSA_{ID}: CSA with iterative deepening

Figure 2: CSA and its iterative-deepening extension

in continuous theory [5] are only necessary, and second-order sufficient condition must be checked in order to ensure that a point is actually a CLM_{cn} (CLM in continuous space). In contrast, the condition in Theorem 1 is necessary as well as sufficient.

2.2 Existing algorithms for finding SP_{dn}

Since there is a one-to-one correspondence between CGM_{dn} and SP_{dn}, it implies that a strategy looking for SP_{dn} with the minimum objective value will result in CGM_{dn}. We review two methods to look for SP_{dn}.

The first algorithm is the *discrete Lagrangian method* (DLM) [11]. It is an iterative local search that looks for SP_{dn} by updating the variables in x in order to perform descents of L_d in the x subspace, while occasionally updating the λ variables of unsatisfied constraints in order to perform ascents in the λ subspace and to force the violated constraints into satisfaction. When no new probes can be generated in both the x and λ subspaces, the algorithm has additional mechanisms to escape from such local traps. It can be shown that the point where DLM stops is a CLM_{dn} when the number of neighborhood points is small enough to be enumerated in each descent in the x subspace [10]. However, when the number of neighborhood points is very large and hill-climbing is used to find the first point with an improved L_d in each descent, then the point where DLM stops may be a feasible point but not necessarily a SP_{dn}.

The second algorithm is the *constrained simulated annealing* (CSA) [9] algorithm shown in Figure 2a. It looks for SP_{dn} by probabilistic descents in the x subspace and by probabilistic ascents in the λ subspace, with an acceptance probability governed by the Metropolis probability. Similar to DLM, if the neighborhood of every point is very large and cannot be enumerated, then the point where CSA stops may only be a feasible point but not necessarily a SP_{dn}.

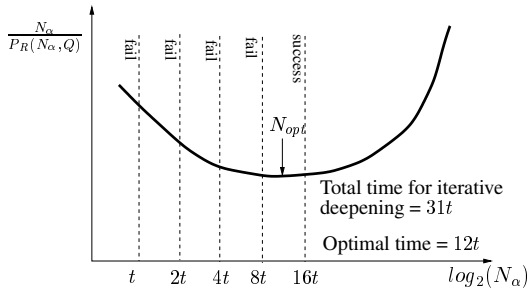


Figure 3: An application of iterative deepening in CSA_{ID}.

Theorem 2. *Asymptotic convergence of CSA.* [9] The Markov chain modeling CSA converges to a CGM_{dn} with probability one.

Theorem 2 extends a similar theorem for SA that proves its asymptotic convergence to unconstrained global minima of unconstrained optimization problems. By looking for SP_{dn} in the Lagrangian-function space, Theorem 2 shows the asymptotic convergence of CSA to CGM_{dn} in constrained optimization problems.

Theorem 2 implies that CSA is not a practical algorithm when used to find CGM_{dn} in one run with certainty because CSA will take infinite time.

In practice, when CSA is run once using a a finite cooling schedule N_α, it finds a CGM_{dn} with reachability probability $P_R(N_\alpha) < 1$. To increase its success probability, CSA with a finite N_α can be run multiple times from random starting points. Assuming that all the runs are independent, a CGM_{dn} can be found in finite average time defined by (3).

We have verified experimentally that the expected time defined in (3) has an absolute minimum at N_{opt}. (Figure 1b illustrates the existence of N_{opt} for $CSAGA$.) It follows that, in order to minimize (3), CSA should be run multiple times from random starting points using schedule N_{opt}.

To find N_{opt} at run time without using problem-dependent information, we have proposed to use *iterative deepening* [3] by starting CSA with a short schedule and by doubling the schedule each time the current run fails to find a CGM_{dn} [8]. Since the total overhead in iterative deepening is dominated by that of the last run, CSA_{ID} (CSA with iterative deepening in Figure 2b) has a completion time of the same order of magnitude as that using N_{opt} when the last schedule that CSA is run is close to N_{opt} and that this run succeeds. Figure 3 illustrates that the total time incurred by CSA_{ID} is of the same order as the expected overhead at N_{opt}.

Note that $P_R(N_{opt}) < 1$ for one run of CSA at N_{opt}. When CSA is run with a schedule close to N_{opt} and fails to find a solution, its cooling schedule will be doubled and overshoots beyond N_{opt}. To reduce the chance of overshooting into exceedingly long cooling schedules and to increase the success probability before its schedule reaches N_{opt}, we have

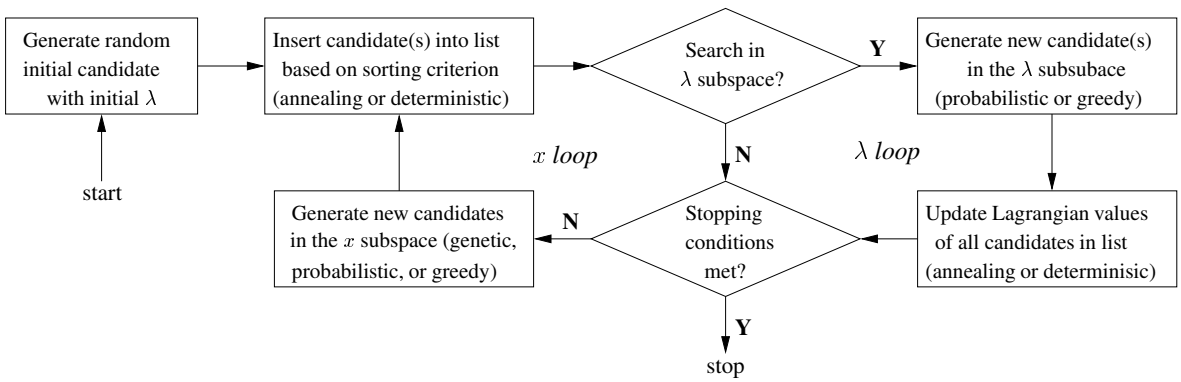


Figure 4: An iterative stochastic procedural framework to look for SP_{dn}.

proposed to run CSA multiple times from random starting points at each schedule in CSA_{ID}. Figure 2b shows CSA that is run $K = 3$ times at each schedule before the schedule is doubled. Our results show that such a strategy generally requires twice the average completion time with respect to multiple runs of CSA using N_{opt} before it finds a CGM_{dn} [8].

2.3 Genetic algorithms for solving constrained NLPs

Genetic algorithm (GA) is a general stochastic optimization algorithm that was originally developed for solving unconstrained problems. Recently, many variants of GA have been developed for solving constrained NLPs. Most of these methods were based on penalty formulations that transform (4) into an unconstrained function $\mathcal{F}(x)$, consisting of the sum of an objective and a set of constraints weighted by penalties, and use GA to minimize $\mathcal{F}(x)$. Penalties used can be static, dynamic, adaptive, or modified by annealing rules [7].

In general, unless suitable penalties are used, a local minimum of an unconstrained penalty function is only *a necessary but not a sufficient* condition for the point to be a CLM_{dn} of the original constrained NLP. Without a good method to select penalties, GA resorts to ad hoc tuning and has difficulty in achieving convergence [6].

Besides penalty methods, methods for handling nonlinear constraints directly have been studied. These include methods based on preserving feasibility with specialized genetic operators, methods searching along boundaries of feasible regions, methods based on decoders, repair of infeasible solutions, co-evolutionary methods, and strategic oscillation. These strategies cannot be generalized easily because they require domain-specific knowledge or are problem-dependent.

3 A General Framework to look for SP_{dn}

Although there are many methods for solving constrained NLPs, our survey in the last section shows a lack of a general framework that unifies these mechanisms. Without such a framework, it is difficult to know whether different algorithms are actually variations of each other. In this section we present a framework for solving constrained NLPs that unifies SA, GA, and greedy searches.

Based on the necessary and sufficient condition in Theorem 1, Figure 4 depicts a stochastic procedure to look for SP_{dn}. The procedure consists of two loops: the x loop that updates the variables in x in order to perform descents of L_d in the x subspace, and the λ loop that updates the λ variables of unsatisfied constraints for any candidate in the list in order to perform ascents in the λ subspace. The procedure quits when no new probes can be generated in both the x and λ subspaces.

The procedure will not stop until it finds a feasible point because it will generate new probes in the λ subspace when there are unsatisfied constraints. Further, if the procedure always finds a descent direction at x by enumerating all points in $\mathcal{N}_{dn}(x)$, then the point where the procedure stops must be a feasible local minimum in the x subspace of $L_d(x, \lambda)$, or equivalently, a CLM_{dn}.

Both DLM and CSA discussed in Section 2.2 fit into this framework, each maintaining a list of one candidate at any time. DLM entails greedy searches in the x and λ subspaces, deterministic insertions into the list of candidates, and deterministic acceptance of candidates until all constraints are satisfied. On the other hand, CSA generates new probes randomly in one of the x or λ variables, accepts them based on the Metropolis probability if L_d increases along the x dimension and decreases along the λ dimension, and stops updating λ when all constraints are satisfied.

In this section, we use genetic operators to generate probes and present in Section 3.1 CGA and in Section 3.2 $CSAGA$. Finally, we propose in Section 3.3 iterative-deepening versions of these algorithms.

3.1 Constrained genetic algorithm (CGA)

CGA in Figure 5a was developed based on the general framework in Figure 4. Similar to traditional GA, it organizes a search into a number of generations, each involving a population of candidate points in a search space. However, it searches in the L_d space using genetic operators to generate

```
1.  procedure CGA(P, N_g)
2.      set generation number t ← 0 and λ(t) ← 0;
3.      initialize population P(t);
4.      repeat /* over multiple generations */
5.          evaluate L_d(x, λ(t)) for all candidates in P(t);
6.          repeat /* over probes in x subspace */
7.              y ← GA(select(P(t)));
8.              evaluate L_d(y, λ) and insert into P(t)
9.          until sufficient probes in x subspace;
10.         λ(t) ← λ(t) ⊕ cH(h, P(t)); /* update λ */
11.         t ← t + 1;
12.     until (t > N_g)
13. end_procedure
```

a) CGA called with population size P and number of generations N_g.

```
1.  procedure CGA_ID
2.      set initial number of generations N_g = N_0;
3.      set K = number of CGA runs at fixed N_g;
3.      repeat /* iterative deepening to find CGM_dn */
4.          for i ← 1 to K do call CGA(P, N_g) end_for
5.          set N_g ← ρN_g (typically ρ = 2);
6.      until N_g exceeds maximum allowed or
            (no better solution has been found in two
            successive increases of N_g and N_g > ρ^5 N_0
            and a feasible solution has been found);
7.  end_procedure
```

b) CGA_{ID}: CGA with iterative deepening

Figure 5: Constrained GA and its iterative deepening version.

probes in the x subspace, either greedy or probabilistic mechanisms to generate probes in the λ subspace, and deterministic organization of candidates according to their L_d values.

Lines 4 and 12 terminate CGA when the maximum number of allowed generations is exceeded.

Line 5 evaluates in generation t all candidates in $\mathcal{P}(t)$ using $L_d(x, \lambda(t))$ as the fitness function.

Lines 6-9 explore the x subspace by selecting from $\mathcal{P}(t)$ candidates to reproduce using genetic operators and by inserting the new candidates generated into $\mathcal{P}(t)$ according to their fitness values.

After a number of descents in the x subspace (defined by the number of probes in Line 9 and the decision box "search in λ subspace?" in Figure 4), the algorithm switches to searching in the λ subspace. Line 10 updates λ according to the vector of maximum violations $\mathcal{H}(h(x), \mathcal{P}(t))$, where the maximum violation of a constraint is evaluated over all the candidates in $\mathcal{P}(t)$. That is,

$$\mathcal{H}_i(h(x), \mathcal{P}(t)) = \max_{x \in \mathcal{P}(t)} H(h_i(x)),\ i = 1, \cdots, m, \qquad (7)$$

where $h_i(x)$ is the ith constraint function, H is the non-negative transformation defined in (5), and c is a step-wise constant controlling how fast λ changes.

Operator $\oplus$ in Figure 5a can be implemented in two ways in order to generate a new λ. A new λ can be generated probabilistically based a uniform distribution in $(\frac{-c\mathcal{H}}{2}, \frac{c\mathcal{H}}{2}]$, or in a greedy fashion based on a uniform distribution in $(0, c\mathcal{H}]$. In addition, we can accept new probes deterministically by rejecting negative ones, or probabilistically using an annealing rule. In all cases, a Lagrange multiplier will not be changed if its corresponding constraint is satisfied.

```
1.  procedure CSAGA(P, N_g)
2.      set t ← 0, T_0, 0 < α < 1, and P(t);
3.      repeat /* over multiple generations */
4.          for i ← 1 to q do /* SA in Lines 5-10 */
5.              for j ← 1 to P do
6.                  generate x'_j from N_dn(x_j) using G(x_j, x'_j);
7.                  accept x'_j with probability A_T(x_j, x'_j)
8.              end_for
9.              set T ⟵ αT; /* set T for the SA part */
10.         end_for
11.         repeat /* by GA over probes in x subspace */
12.             y ← GA(select(P(t)));
13.             evaluate L_d(y, λ) and insert y into P(t);
14.         until sufficient number of probes in x subspace;
15.         t ← t + q; /* update generation number */
16.     until (t ≥ N_g)
17. end_procedure
```

Figure 6: $CSAGA$: Combined CSA and CGA called with population size P and number of generations N_g.

3.2 Combined Constrained SA and GA ($CSAGA$)

Based on the general framework in Figure 4, we design $CSAGA$ by integrating CSA in Figure 2a and CGA in Figure 5a into a combined procedure. The new algorithm shown in Figure 6 uses both SA and GA to generate new probes in the x subspace.

Line 2 initializes $\mathcal{P}(0)$. Unlike CGA, any candidate $\mathbf{x} = [x_1, \cdots, x_n, \lambda_1, \cdots, \lambda_k]^T$ in $\mathcal{P}(t)$ is defined in the joint x and λ subspaces. Initially, x can be user-provided or randomly generated, and λ is set to zero.

Lines 4-10 perform CSA using q probes on every candidate in the population. In each probe, we generate probabilistically $\mathbf{x}'_\mathbf{j}$ and accept it based on the Metropolis probability. Experimentally, we set q to be $\frac{N_g}{6}$.

Lines 11-15 start a GA search after the SA part has been completed. The algorithm searches in the x subspace by generating probes using GA operators, sorting all candidates according to their fitness values L_d after each probe is generated. In ordering candidates, since each candidate has its own vector of Lagrange multipliers, the algorithm first computes the average value of Lagrange multipliers for each constraint over all candidates in $\mathcal{P}(t)$ and then calculates L_d for each candidate using the average Lagrange multipliers.

3.3 CGA and $CSAGA$ with iterative deepening

In this section we present a method to determine the optimal number of generations in one run of CGA and $CSAGA$ in order to find a CGM_{dn}. The method is based on the use of iterative deepening [3] that determines an upper bound on N_g in order to minimize the expected total overhead in (3), where

N_g is the number of generations in one run of CGA.

The number of probes expended in one run of CGA or $CSAGA$ is $N = N_g P$, where P is the population size. For a fixed P, let $\hat{P}_R(N_g) = P_R(PN_g)$ be the reachability probability of finding CGM_{dn}. From (3), the expected total number of probes using multiple runs of either CGA or $CSAGA$ and fixed P is:

$$\frac{N}{P_R(N)} = \frac{N_g P}{P_R(N_g P)} = P\frac{N_g}{\hat{P}_R(N_g)} \tag{8}$$

In order to have an optimal number of generations $N_{g_{opt}}$ that minimizes (8), $\frac{N_g}{\hat{P}_R(N_g)}$ must have an absolute minimum in (0,∞). This condition is true since $\hat{P}_R(N_g)$ of CGA has similar behavior as $P_R(N_g)$ of CSA. It has been verified based on statistics collected on $\hat{P}_R(N_g)$ and N_g at various P when CGA and $CSAGA$ are used to solve ten discretized benchmark problems G1-G10 [7]. Figure 1b illustrates the existence of such an absolute minimum when $CSAGA$ with $P = 3$ was applied to solve G1.

Similar to the design of CSA_{ID}, we apply iterative deepening to estimate $N_{g_{opt}}$. CGA_{ID} in Figure 5b uses a set of geometrically increasing N_g to find a CGM_{dn}:

$$N_{g_i} = \rho^i N_0, \qquad i = 0, 1, \ldots \tag{9}$$

where N_0 is the (small) initial number of generations.

Under each N_g, CGA is run for a maximum of K times but stops immediately when a feasible solution has been found, when no better solution has been found in two successive generations, and after the number of iterations has been increased geometrically at least five times. These conditions are used to ensure that iterative deepening has been applied adequately. For iterative deepening to work, $\rho > 1$.

Let $\hat{P}_R(N_{g_i})$ be the reachability probability of one run of CGA under N_{g_i} generations, $B_{opt}(f')$ be the expected total number of probes taken by CGA with $N_{g_{opt}}$ to find a CGM_{dn}, and $\mathcal{B}_{ID}(f')$ be the expected total number of probes taken by CGA_{ID} in Figure 5b to find a solution of quality f' starting from N_0 generations. According to (8),

$$B_{opt}(f') = P\frac{N_{g_{opt}}}{\hat{P}_R(N_{g_{opt}})} \tag{10}$$

The following theorem shows the sufficient conditions in order for $\mathcal{B}_{ID}(f') = O(B_{opt}(f'))$.

Theorem 3. *Optimality of CGA_{ID} and $CSAGA_{ID}$.* $\mathcal{B}_{ID}(f') = O(B_{opt}(f'))$ if

a) $\hat{P}_R(0) = 0$; $\hat{P}_R(N_g)$ is monotonically non-decreasing for N_g in $(0, \infty)$; and $\lim_{N_g \to \infty} \hat{P}_R(N_g) \leq 1$;

b) $(1 - \hat{P}_R(N_{g_{opt}}))^K \rho < 1$.

The proof is not shown due to space limitations.

Typically, $\rho = 2$, and $\hat{P}_R(N_{g_{opt}}) \geq 0.25$ in all the benchmarks tested. Substituting these values into condition (b) in Theorem 3 yields $K > 2.4$. In our experiments, we have used $K = 3$. Since CGA is run a maximum of three times under each N_g, $B_{opt}(f')$ is of the same order of magnitude as *one* run of CGA with $N_{g_{opt}}$.

The only remaining issue left in the design of CGA_{ID} and $CSAGA_{ID}$ is in choosing a suitable population size P in each generation.

In designing CGA_{ID}, we found that the optimal P ranges from 4 to 40 and is difficult to determine a priori. Although it is possible to choose a suitable P dynamically, we do not present the algorithm here due to space limitations and because it performs worse than $CSAGA_{ID}$.

In selecting P for $CSAGA_{ID}$, we note in the design of CSA_{ID} in Figure 2b that $K = 3$ parallel runs are made at each cooling schedule in order to increase the success probability of finding a solution. For this reason, we set $P = K = 3$ in our experiments. Our experimental results in the next section show that, although the optimal P may be slightly different from 3, the corresponding expected overhead to find a CGM_{dn} differs very little from that when a constant P is used.

4 Experimental Results

We present in this section our experimental results in evaluating CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ on discrete constrained NLPs. In implementing CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$, we have used the defaualt parameters of CSA [9] in the SA part and those of Genocop III [6] in the GA part. In addition, for iterative deepening to work, we have set the following parameters: $\rho = 2$, $K = 3$, $N_0 = 10 \cdot n_v$, and $N_{max} = 1.0 \times 10^8 n_v$, where n_v is the number of variables, and N_0 and N_{max} are, respectively, initial and maximum number of probes.

Based on the framework in Figure 4, we first determine the best combination of strategies to use in generating probes and in organizing candidates. Using the best combination of strategies, we then show experimental results on some constrained NLPs.

Table 1 shows the results of evaluating various combinations of strategies in CSA_{ID}, CGA_{ID}, and $CSAGA_{ID}$ on a discretized version of G2 [7, 4]. We show the average time of 10 runs for each combination in order to reach two solution quality levels (1% or 10% worse than CGM_{dn}, assuming the value of CGM_{dn} is known). Evaluation results on other benchmark problems are similar and are not shown due to space limitations.

Our results show that CGA_{ID} is usually less efficient than CSA_{ID} or $CSAGA_{ID}$. Further, CSA_{ID} or $CSAGA_{ID}$ has better performance when probes generated in the x subspace are accepted by annealing rather than by deterministic rules (the former prevents a search from getting stuck in local minima or infeasible points). On the other hand, there is little

Table 1: Timing results on evaluating various combinations of strategies in CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ with $P = 3$ to find solutions that deviate by 1% and 10% from the best-known solution of a discretized version of G2. All CPU times in seconds were averaged over 10 runs and were collected on a Pentinum III 500-MHz computer with Solaris 7. '–' means that no solution with desired quality can be found.

Probe Generation Strategy		Insertion	Target Solution 1% off CGM_{dn}			Target Solution 10% off CGM_{dn}		
λ subspace	x subspace	Strategy	CSA_{ID}	CGA_{ID}	$CSAGA_{ID}$	CSA_{ID}	CGA_{ID}	$CSAGA_{ID}$
probabilistic	probabilistic	annealing	6.91 sec.	23.99 sec.	4.89 sec.	1.35 sec.	–	1.03 sec.
probabilistic	probabilistic	deterministic	9.02 sec.	–	6.93 sec.	1.35 sec.	2.78 sec.	1.03 sec.
probabilistic	deterministic	annealing	–	18.76 sec.	–	89.21 sec.	2.40 sec.	–
probabilistic	deterministic	deterministic	–	16.73 sec.	–	–	2.18 sec.	–
greedy	probabilistic	annealing	7.02 sec.	–	7.75 sec.	1.36 sec.	–	0.90 sec.
greedy	probabilistic	deterministic	7.02 sec.	–	7.75 sec.	1.36 sec.	–	0.90 sec.
greedy	deterministic	annealing	–	25.50 sec.	–	82.24 sec.	1.90 sec.	–
greedy	deterministic	deterministic	–	25.50 sec.	–	82.24 sec.	1.90 sec.	–

Table 2: Results on CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ in finding the best-known solution f^* for 10 discretized constrained NLPs and their corresponding results found by EA. (S.T. stands for strategic oscillation, H.M. for homomorphous mappings, and D.P. for dynamic penalty. $\mathcal{B}_{ID}(f^*)$, the CPU time in seconds to find the best-known solution f^*, were averaged over 10 runs and were collected on a Pentinum III 500-MHz computer with Solaris 7. $\#L_d$ represents the number of $L_d(x, \lambda)$-function evaluations. The best $\mathcal{B}_{ID}(f^*)$ for each problem is boxed.)

Problem	Best	EAs		CSA_{ID}		CGA_{ID}		$CSAGA_{ID}$				
ID	Solution f^*	Best Found	Method	$\mathcal{B}_{ID}(f^*)$	$\#L_d$	P_{opt}	$\mathcal{B}_{ID}(f^*)$	P	$\mathcal{B}_{ID}(f^*)$	$\#L_d$	P_{opt}	$\mathcal{B}_{ID}(f^*)$
G1 (min)	-15	-15	Genocop	1.65 sec.	173959	40	5.49 sec.	3	1.64 sec.	172435	2	1.31 sec.
G2 (max)	-0.80362	0.803553	S.T.	7.28 sec.	415940	30	311.98 sec.	3	5.18 sec.	261938	3	5.18 sec.
G3 (max)	1.0	1.0	S.T.	1.07 sec.	123367	30	14.17 sec.	3	0.89 sec.	104568	3	0.89 sec.
G4 (min)	-30665.5	-30664.5	H.M.	0.76 sec.	169913	5	3.95 sec.	3	0.95 sec.	224025	3	0.95 sec.
G5 (min)	4221.9	5126.498	D.P.	2.88 sec.	506619	30	68.9 sec.	3	2.76 sec.	510729	2	2.08 sec.
G6 (min)	-6961.81	-6961.81	Genocop	0.99 sec.	356261	4	7.62 sec.	3	0.91 sec.	289748	2	0.73 sec.
G7 (min)	24.3062	24.62	H.M.	6.51 sec.	815696	30	31.60 sec.	3	4.60 sec.	547921	4	4.07 sec.
G8 (max)	0.095825	0.095825	H.M.	0.11 sec.	21459	30	0.31 sec.	3	0.13 sec.	26585	4	0.10 sec.
G9 (min)	680.63	680.64	Genocop	0.74 sec.	143714	30	5.67 sec.	3	0.57 sec.	110918	3	0.57 sec.
G10 (min)	7049.33	7147.9	H.M.	3.29 sec.	569617	30	82.32 sec.	3	3.36 sec.	608098	3	3.36 sec.

difference in performance when new probes generated in the λ subspace are accepted by probabilistic or by greedy rules and when new candidates are inserted according to annealing or deterministic rules. In short, generating probes in the x and λ subspaces probabilistically and inserting candidates in both the x and λ subspaces by annealing rules leads to good and stable performance. For this reason, we use this combination of strategies in our experiments.

We next test our algorithms on ten constrained NLPs G1-G10 [7, 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 20, and that of constraints, including simple bounds, is up to 42. 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. These problems were originally designed to be solved by evolutionary algorithms (EAs) in which constraint handling techniques were tuned for each problem in order to get good results. Examples of such techniques include keeping a search within feasible regions with specific genetic operators and dynamic and adaptive penalty methods.

Table 2 compares the performance of CSA_{ID}, CGA_{ID}, and $CSAGA_{ID}$ with respect to $\mathcal{B}_{ID}(f^*)$, the expected total CPU time of multiple runs until a solution of value f^* is found. The first two columns show the problem IDs and the corresponding known f^*. The next two columns show the best solutions obtained by EAs and the specific constraint handling techniques used to generate the solutions. The fifth and sixth columns show, respectively, the average time and number of $L_d(x, \lambda)$ function evaluations CSA_{ID} takes to find f^*. The next two columns show the performance of CGA_{ID} with respect to P_{opt}, the optimal population size found by enumeration, and the average time to find f^*. These results show that CGA_{ID} is not competitive as compared to CSA_{ID}, even when P_{opt} is used. The results on including additional steps in CGA_{ID} to select a suitable P at run time are worse and are not shown due to space limitations. Finally, the last five columns show the performance of $CSAGA_{ID}$. The first three present the average times and number of $L_d(x, \lambda)$ evaluations using a constant P, whereas the last two show the average times using P_{opt} found by enumeration. These results show little improvements in using P_{opt}. Further, $CSAGA_{ID}$ has between 9% and 38% in improvement in $\mathcal{B}_{ID}(f^*)$, when compared to that of CSA_{ID},

Table 3: Results on CSA_{ID} and $CSAGA_{ID}$ with $P = 3$ in solving selected Floudas and Pardalos' discretized constrained NLP benchmarks (with more than $n_v = 10$ variables). Since Problem $5.*$ and $7.*$ are especially large and difficult and a search can rarely reach their true CGM_{dn}, we consider a CGM_{dn} found when the solution quality is within 10% of the true CGM_{dn}. All CPU times in seconds were averaged over 10 runs and were collected on a Pentium-III 500-MHz computer with Solaris 7.

Problem		$f(x)$	CSA_{ID}	$CSAGA_{ID}$
ID	Best f^*	n_v	$\mathcal{B}_{ID}(f^*)$	$\mathcal{B}_{ID}(f^*)$
2.7.1(min)	-394.75	20	35.11 sec.	14.86 sec.
2.7.2(min)	-884.75	20	53.92 sec.	15.54 sec.
2.7.3(min)	-8695.0	20	34.22 sec.	22.52 sec.
2.7.4(min)	-754.75	20	36.70 sec.	16.20 sec.
2.7.5(min)	-4150.4	20	89.15 sec.	23.46 sec.
5.2(min)	1.567	46	3168.29 sec.	408.69 sec.
5.4(min)	1.86	32	2629.52 sec.	100.66 sec.
7.2(min)	1.0	16	824.45 sec.	368.72 sec.
7.3(min)	1.0	27	2323.44 sec.	1785.14 sec.
7.4(min)	1.0	38	951.33 sec.	487.13 sec.

for the 10 problems except for G4 and G10.

Comparing CGA_{ID} and $CSAGA_{ID}$ with EA, we see that EA was only able to find f^* in three of the ten problems, despite extensive tuning and using problem-specific heuristics, whereas both CGA_{ID} and $CSAGA_{ID}$ can find f^* for all these problems without any problem-dependent strategies. It is not possible to report the timing results of EA because the results are the best among many runs after extensive tuning.

Finally, Table 3 shows the results on selected discretized Floudas and Pardalos' benchmarks [2] that have more than 10 variables and that have many equality or inequality constraints. The first three columns show the problem IDs, the known f^*, and the number of variables (n_v) in each. The last two columns compare $\mathcal{B}_{ID}(f^*)$ of CSA_{ID} and $CSAGA_{ID}$ with fixed $P = 3$. They show that $CSAGA_{ID}$ is consistently faster than CSA_{ID} (between 1.3 and 26.3 times), especially for large problems. This is attributed to the fact that GA maintains more diversity of candidates by keeping a population, thereby allowing competition among the candidates and leading SA to explore more promising regions.

Acknowledgments

Research supported by National Aeronautics and Space Administration Contract NAS2-1230.

Bibliography

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

[2] 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.

[3] R. E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

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

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

[6] Z. Michalewicz and G. Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. *Proceedings of IEEE International Conference on Evolutionary Computation*, 2:647–651, 1995.

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

[8] B. W. Wah and Y. X. Chen. Optimal anytime constrained simulated annealing for constrained global optimization. *Sixth Int'l Conf. on Principles and Practice of Constraint Programming*, September 2000.

[9] 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.

[10] 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.

[11] Z. Wu. *Discrete Lagrangian Methods for Solving Nonlinear Discrete Constrained Optimization Problems*. M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 1998.