Constrained Genetic Algorithms and their Applications in Nonlinear Constrained Optimization*

Benjamin W. Wah and Yi-Xin 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 problem-independent framework that unifies various mechanisms for solving discrete constrained nonlinear programming (NLP) problems whose functions are not necessarily differentiable and continuous. The framework is based on the first-order necessary and sufficient conditions in the theory of discrete constrained optimization using Lagrange multipliers. It implements the search for discrete-neighborhood saddle points (SP_{dn}) by performing ascents in the original-variable subspace and descents in the Lagrange-multiplier subspace. Our study on the various mechanisms shows that CSAGA, a combined constrained simulated annealing and genetic algorithm, performs well. Finally, we apply iterative deepening to determine the optimal number of generations in CSAGA.

1 Introduction

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

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

Here, $f(x)$ 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 equality constraints. $f(x)$, $g(x)$, and $h(x)$ are not necessarily differentiable and can be either linear or nonlinear, continuous or discrete, and analytic or procedural. X is the Cartesian product of all possible combinations of variables in x. Without loss of generality, we consider only minimization problems.

To characterize solutions sought in discrete space, we define the following concepts on discrete neighborhoods and constrained solutions in discrete space.

Definition 1. $\mathcal{N}_d(x)$, the *neighborhood* [1] of point x in discrete space X, is a *finite* user-defined set of points $\{x' \in X\}$ such that x' is reachable from x in one step, 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. 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 $x' \in \mathcal{N}_d(x)$ where x' is feasible.

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 in our previous work that the necessary and sufficient condition for a point to be a CLM_{dn} is that it satisfies the discrete-space saddle-point condition [8] (Section 2.1). We have also extended simulated annealing (SA) [7] and greedy search [9] to look for saddle points in discrete neighborhoods of x (SP_{dn}) (Section 2.2). At

*Proc. 12th International Conference on Tools with Artificial Intelligence, November 2000.

the same time, new problem-dependent constraint-handling heuristics have been developed in the genetic-algorithm (GA) community to handle nonlinear constraints [5] (Section 2.3). Up to now, there is no clear understanding on whether these algorithms can be unified and which strategy is the most effective in finding CGM_{dn}.

Based on our previous work, we develop in this paper a framework that unifies SA, greedy search, and GA in looking for SP_{dn}. We study *constrained genetic algorithm* (CGA) and combined constrained SA and GA (CSAGA) and ways to achieve optimal average completion times in finding CGM_{dn}.

The algorithms studied are all random searches that probe a search space in a random order, where a probe is a point examined by an algorithm, independent of whether it is accepted or not. The overhead of one run of such an algorithm is characterized by the number of probes made and the *reachability probability* that it hits a CGM_{dn} in any of its probes. Let p_j be the probability that an algorithm finds a CGM_{dn} in its j^{th} probe. Then $P_R(N)$ when the algorithm stops after N probes is:

$$P_R(N) = 1 - \prod_{j=1}^{N}(1 - p_j), \tag{1}$$

assuming independent probes (a simplifying assumption). Reachability can be maintained by keeping the best solution found at any time and by reporting the best solution when the algorithm stops.

Although it is hard to control $P_R(N)$ for a search, we can always improve its chance of finding a CGM_{dn} by running it multiple times from random starting points, each examining the search space by N probes. Given $P_R(N)$ for each run of the algorithm and that all runs are independent, $\bar{B}(N, P_R(N))$, the expected total number of probes to find a CGM_{dn}, is:

$$\sum_{j=1}^{\infty} P_R(N)(1 - P_R(N))^{j-1} N \times j = \frac{N}{P_R(N)}. \tag{2}$$

In general, there exists an optimal N that minimizes $\frac{N}{P_R(N)}$ when $P_R(N)$ satisfies the following sufficient conditions: a) $P_R(0) = 0$ and $\lim_{N\to\infty} P_R(N) = 1$, and b) $P_R''(0) > 0$ [6]. Using these conditions, we have developed an optimal strategy that minimizes (2) in searching for CGM_{dn} [6] (Section 2.2).

We present in Section 3.3 an extension of the optimal strategy in CGA and CSAGA that allows them to minimize $\bar{B}(N, P_R(N))$ in (2) in finding a CGM_{dn}. Finally, Section 4 shows the experimental results.

2 Previous Work

This section summarizes the theory of discrete constrained optimization using Lagrange multipliers and two algorithms developed based on the theory. We also describe work in GA to solve constrained NLPs.

2.1 Discrete Constrained Optimization using Lagrange multipliers

The theory is based on solving discrete equality-constrained NLPs represented as [8, 9]:

$$\begin{array}{lll} \text{minimize}_{\mathrm{x}} & f(x) & x = (x_1, \ldots, x_n) \text{ are} \\ \text{subject to} & h(x) = 0 & \text{finite discrete variables,} \end{array} \tag{3}$$

where $h(x) = [h_1(x), \cdots, h_m(x)]^T$ is a vector of m equality constraints. Both $f(x)$ and $h(x)$ may be analytic or procedural but not necessarily differentiable.

A *generalized augmented Lagrangian function* of (3) is defined as follows:

$$L_d(x, \lambda) = f(x) + \lambda^T H\left(h(x)\right) + \frac{1}{2}||h(x)||^2, \tag{4}$$

where H is a continuous transformation function satisfying $H(y) = 0$ iff $y = 0$, and $\lambda = \{\lambda_1, \cdots, \lambda_m\} \in R^m$ is a vector of Lagrange multipliers.

A *direction of maximum potential drop (DMPD)* for $L_d(x, \lambda)$ in discrete neighborhood of x for fixed λ is a vector[1] that points from x to $x' \in \mathcal{N}_d(x)$ with the minimum L_d:

$$\Delta_x L_d(x, \lambda) = \vec{\nu}_x = y \ominus x \text{ where } y = \min_{\substack{x' \in \mathcal{N}_d(x) \\ \cup \{x\}}} L_d(x', \lambda). \tag{5}$$

Here, $\ominus$ is the vector-subtraction operator for moving from x to a point in $\mathcal{N}_d(x) \cup \{x\}$. Intuitively, vector $\vec{\nu}_x$ points from x to y, the point with the minimum L_d among all discrete neighbors of x, including x itself.

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

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

for all $x \in \mathcal{N}_d(x^*)$ and all $\lambda \in R^m$. Note that although we use similar terminologies as in continuous space, SP_{dn} in (6) are different from those in continuous space because their definition is based on discrete neighborhoods of x.

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

Theorem 1. *First-order necessary and sufficient conditions for CLM_{dn}.* In discrete x space of (3), if H in (4) is a non-negative (or non-positive) continuous function satisfying $H(x) = 0$ iff $x = 0$, then point x of (3) is a CLM_{dn} iff:

- It satisfies (6) for any $\lambda \geq \lambda^*$, where $\lambda' \geq \lambda^*$ means that each element of λ' is not less than the corresponding element of λ^*; or
- It satisfies the following first-order conditions:

$$\Delta_x L_d(x, \lambda) = 0; \quad h(x) = 0. \tag{7}$$

Requiring H to be non-negative (or non-positive) is easy to achieve. Two such examples are $H(h(x)) = |h(x)|$ and $H(h(x)) = h^2(x)$. A similar transformations can be used to transform an inequality constraint, like $g_j(x) \leq 0$, into an equivalent equality constraint $\max(g_j(x), 0) = 0$. For this reason, we only consider problems with equality constraints in the rest of this paper.

The conditions in Theorem 1 are stronger than their counterparts in continuous space. In general, in the theory of Lagrange multipliers in continuous space [3], the set of solution points of an application problem that satisfy the first-order necessary and second-order sufficient conditions is not necessarily the same as the set of CLM_{dn} of the problem and the set of points satisfying the saddle-point condition. Only when the Lagrangian function is differentiable everywhere and all CLM_{dn} are regular points that the three sets are identical. Further, a CGM_{dn} at a point that is not regular or not differentiable cannot be found by existing methods in continuous space. These limitations are overcome in Theorem 1 because it does not require differentiability and implies that finding SP_{dn} amounts to finding CLM_{dn} of the original problem. Further, a strategy looking for SP_{dn} with the minimum objective value will result in a CGM_{dn} because a CGM_{dn} is also a SP_{dn}.

2.2 Algorithms Implementing Thm. 1

The conditions in (7) provide a stopping condition when a search finds SP_{dn} but do not present the mechanism to arrive at such points. In this section we review two methods to look for SP_{dn}.

The *discrete Lagrangian method* (DLM) is an iterative local-search method based on the first-order conditions, similar to those in continuous space:

$$x^{k+1} = x^k \oplus \Delta_x L_d(x^k, \lambda^k), \tag{8}$$
$$\lambda^{k+1} = \lambda^k + \varrho H(h(x^k)), \tag{9}$$

where $\oplus$ is the vector-addition operator, $x \oplus y = (x_1 + y_1, \ldots x_n + y_n)$, and ϱ is a positive real number controlling how fast the Lagrange multipliers change. 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 of (8) [9]. However, if the number of neighboring points is very large and hill-climbing is used to find the first point with a smaller Lagrangian value in each descent, then DLM will stop at a feasible point but not necessarily a saddle point.

```
1.  procedure CSA
2.      set initial x = (x, λ) with random x and λ ← 0;
3.      initialize T_0 and cooling rate 0 < α < 1;
4.      set N_T (number of probes per temperature);
5.      while stopping condition is not satisfied do
6.          for n ← 1 to N_T do
7.              generate x' from N_d(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: CSA: Constrained simulated annealing [8].

Constrained simulated annealing (CSA) [7], on the other hand, looks for SP_{dn} with the minimum objective (Figure 1). This is done by carrying out *probabilistic ascents* in the λ subspace, with a probability of acceptance governed by the Metropolis probability, and by *probabilistic descents* in the x subspace.

Using distribution $G(\mathbf{x}, \mathbf{x}')$ to generate trial point $\mathbf{x}'$ in neighborhood $\mathcal{N}_d(\mathbf{x})$, a Metropolis acceptance probability $A_T(\mathbf{x}, \mathbf{x}')$, and a logarithmic cooling schedule, CSA has been proven to have asymptotic convergence with probability one to a CGM_{dn} [7]. The result is of theoretical interest only as it assumes an infinitely long cooling schedule. In practice, we can only use a finite cooling schedule, leading to a reachability probability less than one.

As described in Section 1, there exists an optimal cooling schedule that minimizes the expected overhead in (2) when two sufficient conditions are satisfied. Experimentally, we have verified that reachability probabilities of CSA satisfy these conditions [6]. By exploiting this property, we have used iterative deepening to find the optimal schedule. The algo-

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

Figure 2: CSA_{ID}: CSA with iterative deepening [6].

rithm starts with a short cooling schedule and doubles it every time it fails to find a solution of desired quality. To reduce the chance for a search to overshoot into exceedingly long cooling schedules, CSA in Figure 2 is run K (= 3) times at each cooling schedule. It has been proved that the algorithm has a completion time of the same order of magnitude as that using the optimal schedule if K is set to be large enough [6] .

2.3 Genetic Algorithms for Solving Constrained NLP Problems

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 (3) into an unconstrained function $\mathcal{F}(x)$, consisting of a sum of the objective and the constraints weighted by penalties, and use GA to minimize $\mathcal{F}(x)$.

Examples of penalty formulations include static penalties, dynamic penalties, annealing penalties, and adaptive penalties [5]. In general, these problem-dependent methods may require extensive tuning and lack a strong mathematical foundation, making them hard to guarantee convergence [4].

In addition to penalty methods, other methods have been studied in GA for handling constraints. 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 methods require domain-specific knowledge or problem-dependent genetic operators, and have difficulties in finding feasible regions or in maintaining feasibility for nonlinear constraints.

3 Framework to look for SP_{dn}

Existing work today lacks a framework that unifies the various mechanisms to look for SP_{dn}, making it difficult to know whether different algorithms are actually variations of each other. In this section we present a framework that unifies SA, GA, and greedy searches in looking for SP_{dn}.

Based on Theorem 1, Figure 3 depicts a general stochastic optimization 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 original-variable subspace, and the λ loop that updates the λ variables, if there are unsatisfied constraints for any candidate in the list, in order to perform ascents in the Lagrange-multiplier subspace. The procedure quits when no new probes can be generated in both the x and λ subspaces.

The general procedure is guaranteed to terminate only at feasible points; otherwise, new probes will be generated in the λ subspace to suppress the unsatisfied constraints. Further, if the probe generator in the x subspace is able to enumerate all the points in $\mathcal{N}_d(x)$ for any point x in the original-variable subspace, then the point where the procedure stops must be a SP_{dn}, or equivalently, a CLM_{dn}.

Both DLM and CSA discussed in Section 2.2 can be made to fit into this framework, each maintaining a list of one candidate. DLM entails greedy generations in the x and λ subspaces, deterministic insertions into the list of candidates and deterministic acceptance of candidates, and stops updating λ when all the constraints are satisfied. In contrast, CSA generates new probes randomly along 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 the constraints are satisfied.

In the rest of this section, we extend the mechanisms to include genetic operators and present in Section 3.1 CGA and in Section 3.2 the combined CSAGA. Finally, we propose the optimal version of these algorithms in Section 3.3.

3.1 CGA: Constrained GA

CGA was developed based on the general framework in Figure 3 that looks for SP_{dn}. Similar to traditional GA, it organizes a search into a number of generations, each involving a population of candidate points in the Lagrangian space. It uses genetic oper-

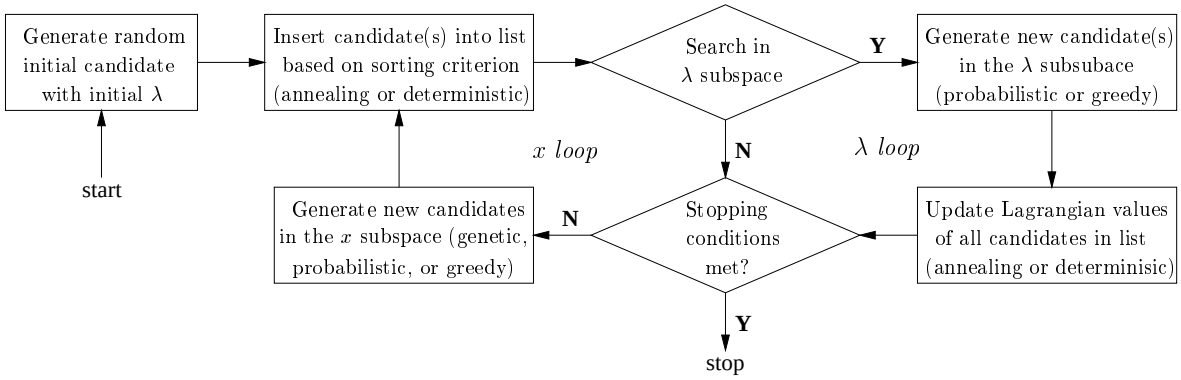


Figure 3: An iterative stochastic procedural framework to look for SP_{dn}.

```
1.  procedure CGA(P, N_g)
2.      set generation number t ← 0 and λ(t) ← 0;
3.      initialize random or user-provided 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 number of probes in x subspace;
10.         λ(t) ← λ(t) ⊕ c × H(h, P(t)); /* update λ */
11.         t ← t + 1;
12.     until (t > N_g)
13. end_procedure
```

Figure 4: CGA: Constrained genetic algorithm. P is the population size and N_g is the number of generations.

ators to generate new probes in the original-variable subspace, either greedy or probabilistic generations in the λ subspace, and deterministic organization of candidates according to their Lagrangian values. Figure 4 outlines the algorithm.

Line 4 terminates CGA when either the maximum number of allowed generations is exceeded or when no better feasible solution within a precision range is found in some successive generations. (The stopping condition is specified more precisely later in CGA with iterative deepening.)

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

Lines 6-9 search the x subspace by selecting from $\mathcal{P}(t)$ individuals to reproduce using genetic operators and by inserting the individuals generated into $\mathcal{P}(t)$ according to their fitness values. (In our experiments, we have used the seven operators in Genocop III [4].)

Line 10 updates λ according to the vector of maximum violations $\mathcal{H}(h, \mathcal{P}(t))$, where the maximum violation of a constraint is evaluated over all the individuals in $\mathcal{P}(t)$. That is,

$$\mathcal{H}_i(h, \mathcal{P}(t)) = \max_{x \in \mathcal{P}(t)} H(h_i(x)), \quad i = 1, 2, \cdots, m, \quad (10)$$

where $h_i(x)$ is the i^{th} constraint function, H is the non-negative transformation in (4), and c is a positive step-wise constant controlling how fast the Lagrange multipliers change (typically $c = 0.1$).

Operator $\oplus$ can be implemented in a deterministic or a probabilistic fashion. In a deterministic way, $\oplus$ can be implemented as simple vector addition. Alternatively, we implement $\oplus$ as vector addition based on an annealing rule: at each generation, we either increase or decrease λ. The probability to decrease λ, controlled by T, decreases from a high to a very low value during the evolution. In either case, a Lagrange multiplier will not be changed if its corresponding constraint is satisfied. Our evaluations have shown that Lagrange multipliers updated stochastically lead to shorter average completion times. For that reason, we have used stochastic updates in our experiments.

3.2 CSAGA: CSA + CGA

Based on the general framework in Figure 3, we design CSAGA by integrating CSA in Figure 1 and CGA in Figure 4 into a combined procedure. The new procedure differs from the original CSA in two aspects. First, by maintaining multiple candidates in a population, we need to decide how CSA should be applied to the multiple candidates in a population. Our evaluations show that, instead of running CSA corresponding to a candidate from a random starting point, it is best to run CSA sequentially, using the best solution found in one run as the starting point of the next run. Second, we need to determine the

1. **procedure** CSAGA(P, N_g)
2. set $t \leftarrow 0$, T_0, $0 < \alpha < 1$, and $\mathcal{P}(t)$;
3. **repeat** /* over multiple generations */
4. **for** $i \leftarrow 1$ **to** P **do** /* SA in Lines 5-11 */
5. **for** $j \leftarrow 1$ **to** $freq$ **do**
6. generate $\mathbf{x}'_j$ from $\mathcal{N}_d(\mathbf{x_j})$ using $G(\mathbf{x_j}, \mathbf{x'_j})$;
7. accept $\mathbf{x'_j}$ with probability $A_T(\mathbf{x_j}, \mathbf{x'_j})$
8. **end_for**
9. Assign the best $\mathbf{x}_j$ found as starting point
10. **end_for**
11. set $T \longleftarrow \alpha \times T$; /* set T for the SA part */
12. **repeat** /* by GA over probes in x subspace */
13. $y \leftarrow GA(select(\mathcal{P}(t)))$;
14. evaluate $L_d(y, \lambda)$ and insert y into $\mathcal{P}(t)$;
15. **until** sufficient number of probes in x subspace;
16. $t \leftarrow t + freq$; /* update generation number */
17. **until** $(t \geq N_g)$
18. **end_procedure**

Figure 5: CSAGA: Combined CSA and CGA. P is the population size and N_g is the number of generations.

duration of each run of CSA. This is controlled by parameter $freq$ that to set to $\frac{N_g}{6}$ after experimental evaluations.

Figure 5 shows the combined algorithm that uses both SA and GA to generate new probes in the original-variable subspace.

Line 2 initializes $\mathcal{P}(0)$. Unlike CGA, any $\mathbf{x}$ in $P(t)$ in CSAGA is defined in the joint x-λ space. Initially, x can be either user-provided or randomly generated, and λ is initialized to zero.

Lines 4-11 perform CSA using $freq$ probes on every individual in $P(t)$. Point $\mathbf{x}$ generated probabilistically is accepted based on annealing and the Metropolis probability. As discussed earlier, we use the best point of one run as the starting point of the next run.

Lines 12-15 start a GA search after the SA part is completed. The algorithm searches in the x subspace using GA and evaluates the L_d value of each individual in order to select the individuals with the best fitness. In ordering the individuals, since each individual has its own λ, we first get the average value of each Lagrange multiplier over the population and then calculate the L_d value of each individual using the average Lagrange multipliers.

3.3 Optimal CGA and CSAGA with Iterative Deepening

As discussed in Section 2.2 on CSA with iteration deepening [6], there exists an optimal N that minimizes $\bar{B}(N, P_R(N))$ in (2). In this section we extend this result to find the optimal number of generations in a run of CGA and CSAGA.

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** /*using iterative deepening to find CGM*/
4. **for** $i \leftarrow 1$ **to** K **do** call CGA(P, N_g) **end_for**
5. set $N_g \leftarrow \rho \times N_g$ (typically $\rho = 2$);
6. **until** N_g exceeds a maximum number allowed **or** (no better solution has been found in two successive increases of N_g **and** $N_g > \rho^5 N_0$ **and** a feasible solution has been found);
7. **end_procedure**

Figure 6: CGA_{ID}: CGA with iterative deepening.

The number of probes expended in CGA and CSAGA is $N = P \times N_g$. let $\hat{P}_R(N_g) = P_R(P \times N_g)$ be the reachability probability with N_g generations. Hence, the expected total number of probes using multiple runs with P and N_g under constant P is:

$$\frac{N}{P_R(N)} = \frac{P \times N_g}{P_R(P \times N_g)} = P\frac{N_g}{\hat{P}_R(N_g)} \tag{11}$$

In order to have an optimal $N_{g_{opt}}$ that minimizes (11), $\frac{N_g}{\hat{P}_R(N_g)}$ must have an absolute minimum in (0,∞). Such a minimum exists if $\hat{P}_R(N_g)$ satisfies the following sufficient conditions [6]: a) $\hat{P}_R(0) = 0$ and $\lim_{N_g \to \infty} \hat{P}_R(N_g) = 1$, and b) $\hat{P}''_R(0) > 0$ [6].

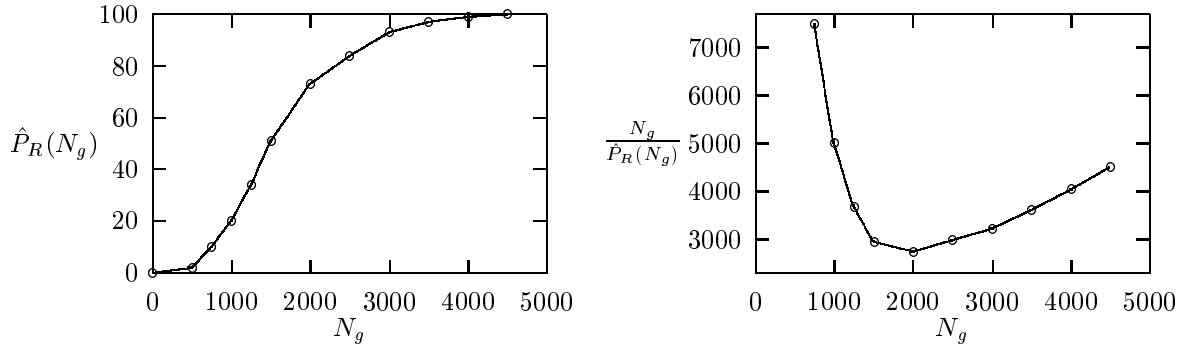
We have collected statistics on $\hat{P}_R(N_g)$ and N_g at various P by using CGA and CSAGA to solve ten discretized test problems G1-G10 [5]. The results indicate that, for both CGA and CSAGA, $\hat{P}_R(N_g)$ satisfies the two sufficient conditions. Figure 7 illustrates the existence of such an optimal N_g in applying CSAGA to solve discretized G1 with $P = 3$. The experimental results also show that $\hat{P}_R(N_g)$ is monotonically nondecreasing.

In a way similar to the design of CSA with iterative deepening, we apply iterative deepening to estimate $N_{g_{opt}}$. CGA_{ID} in Figure 6 uses a set of geometrically increasing N_g to find a CGM_{dn}:

$$N_{g_i} = \rho^i N_0, \qquad i = 0, 1, \ldots \tag{12}$$

where N_0 is the initial number of generations used.

Under each N_g, CGA is run for a maximum of K times but stops immediately when at least one feasible solution has been found, or when no better solution has been found in two successive generations and after the number of iterations has been increased



a) $\hat{P}_R(N_g)$ satisfies the two sufficient conditions b) Absolute minimum in $\frac{N_g}{\hat{P}_R(N_g)}$

Figure 7: An example showing the existence of a minimum in $\frac{N_g}{\hat{P}_R(N_g)}$ when CSAGA with a population size $P = 3$ was applied to solve G1 [5]. ($N_{g_{opt}} \approx 2000$.)

geometrically at least five times. These conditions are used to ensure that iterative deepening has been applied a sufficient number of times. For iterative deepening to work, $\rho > 1$.

Let $\hat{P}_R(N_{g_i})$ be the reachability probability of one run of CGA using N_{g_i} generations, $\beta(f')$ be the expected total number of probes taken by CGA_{ID} to find a CGM_{dn} starting from N_0 generations, and $B_{opt}(f')$ be the expected total number of probes taken by the original CGA with optimal $N_{g_{opt}}$ to find a solution of similar quality. According to (11),

$$B_{opt}(f') = P\frac{N_{g_{opt}}}{\hat{P}_R(N_{g_{opt}})} \tag{13}$$

Next, we show the sufficient conditions for $\beta(f')$ to be of the same order of magnitude as $B_{opt}(f')$.

Theorem 2. *Optimality of iterative deepening in CGA and CSAGA.* $\beta(f') = O(B_{opt}(f'))$ if

a) $\hat{P}_R(N_g)$ is monotonically non-decreasing for N_g in $(0, \infty)$;

b) $\hat{P}_R(0) = 0$, and $\lim_{N_g \to \infty} \hat{P}_R(N_g) = 1$;

c) $\hat{P}_R''(0) > 0$; and

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

The proof is not shown due to space limitations.

Typically, $\rho = 2$, and in all the benchmarks tested, $\hat{P}_R(N_{g_{opt}}) \geq 0.25$. Substituting these values into the last condition in Theorem 2 yields $K > 2.4$. In our experiments, we have used $K = 3$.

CSAGA with iterative deepening ($CSAGA_{ID}$) can be obtained by substituting CGA by CSAGA in Figure 6. Theorem 2 is also applicable to $CSAGA_{ID}$.

The only remaining issue left is in choosing a suitable population size P in each generation. Similar to the design of CGA, the range of optimal P in CGA_{ID} 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 because it performs worse than $CSAGA_{ID}$.

In selecting P for $CSAGA_{ID}$, we note in the design of CSA_{ID} that $K = 3$ parallel runs were made at each cooling rate in order to increase the corresponding probability of success. 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, the corresponding expected overhead to find a CGM_{dn} differs very little from that when a constant P is used.

4 Experimental Results

In this section, we show the results on testing our proposed algorithms on ten discretized constrained NLPs G1-G10 [5, 2]. These problems were originally designed to be solved by GA using problem-specific constraint handling techniques discussed in Section 2.3

Table 1 compares the various algorithms. We measure performance using $\mathcal{T}$, the expected overhead (2) of multiple runs of CSA with iterative deepening at the optimal cooling schedule and those of CGA and CSAGA with iterative deepening at the optimal number of generations.

The fifth and sixth columns of Table 1 show the performance of $\bar{B}(f^*)$: the first showing $\mathcal{T}$, the average time CSA_{ID} takes to find a CGM_{dn}, and the second showing $\mathcal{T}'$, the average time taken by a modified CSA_{ID} in which we feed the best point found

Table 1: Experimental results of EA, CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ in evaluating ten discretized constrained NLPs. (S.T. stands for strategic oscillation, H.M. for homomorphous mappings, and D.P. for dynamic penalty. All runs were done on a Pentinum III 500-MHZ computer with Solaris 7. The best $\bar{B}(f^*)$ is highlighted in bold font.)

Problem	Global	EAs		CSA_{ID}		CGA_{ID}		$CSAGA_{ID}$			
ID	Solution f^*	Best Sol.	Method	$\mathcal{T}(f^*)$	$\mathcal{T}'(f^*)$	P_{opt}	$\mathcal{T}(f^*)$	P	$\mathcal{T}(f^*)$	P_{opt}	$\mathcal{T}(f^*)$
G1 (min)	-15	-15	Genocop	6.92	6.25	40	5.49	3	3.31	2	**2.67**
G2 (max)	-0.80362	0.803553	S.T.	38.99	29.79	30	311.98	3	**20.64**	3	**20.64**
G3 (max)	1.0	0.999866	S.T.	8.09	7.45	30	14.17	3	3.40	2	**3.27**
G4 (min)	-30665.5	-30664.5	H.M.	1.94	1.06	5	3.95	3	**0.93**	3	**0.93**
G5 (min)	4221.9	5126.498	D.P.	2.97	2.16	30	68.9	3	2.32	2	**2.08**
G6 (min)	-6961.81	-6961.81	Genocop	3.26	2.20	4	7.62	3	1.41	2	**1.05**
G7 (min)	24.3062	24.62	H.M.	29.03	21.43	30	31.60	3	14.40	2	**12.82**
G8 (max)	0.095825	0.095825	H.M.	0.33	0.21	30	0.31	3	0.19	4	**0.17**
G9 (min)	680.63	680.64	Genocop	2.73	2.40	30	5.67	3	2.05	2	**1.87**
G10 (min)	7049.33	7147.9	H.M.	4.86	5.01	30	82.32	3	**3.25**	3	**3.25**

in one of the K runs as the starting point to the next run at each cooling schedule (Lines 5-9 of $CSAGA$). The results show that there can be significant improvements by using improved starting points.

The next two columns show the performance of CGA_{ID}: the first showing P_{opt}, the optimal population size obtained by enumeration, and the second showing the average time to find a CGM_{dn}. The results show that CGA_{ID} is not competitive as compared to CSA_{ID}, even using an optimal population size. The results on including an additional component in the algorithm to select a suitable population size at run time are worse and are not shown.

Finally, the last four columns show the performance of $CSAGA_{ID}$: the first two showing the average times using a constant population size, while the last two showing the average times using an optimal population size P_{opt} obtained by enumeration. The results show little improvements in using an optimal population size and improvements in $\mathcal{T}'$ ranging from 3% to 134% as compared to that of CSA_{ID}.

Comparing CGA_{ID} and $CSAGA_{ID}$ with EA, EA was only able to find CGM_{dn} in three of the ten problems, despite extensive tuning and using problem-specific heuristics, whereas both CGA and CSAGA can find CGM_{dn} 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.

Comparing the average completion times of CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$, we see that $CSAGA_{ID}$ outperforms CSA_{ID} and CGA_{ID} on all the ten test problems. This result shows that incorporating genetic search in CSA can achieve a higher reachability probability in a given search time, leading to a faster optimal-search algorithm.

References

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

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

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

[4] 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.

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

[6] B. W. Wah and Y. X. Chen. Optimal anytime constrained simulated annealing for constrained global optimization. In *Sixth International Conference on Principles and Practice of Constraint Programming.* Springer-Verlag, September 2000.

[7] 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.

[8] 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.

[9] Z. Wu. *The Theory and Applications of Nonlinear Constrained Optimization using Lagrange Multipliers.* Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, October 2000.