EVOLUTIONARY OPTIMIZATION

Edited by
RUHUL SARKER
University of New South Wales

MASOUD MOHAMMADIAN
University of Canberra

XIN YAO
University of Birmingham

Kluwer Academic Publishers
Boston/Dordrecht/London

son of genetic algorithms
em. In et. al., D., editor,
Computation Conference

nal genetic algorithms for
r, L., editors, *Proceedings*
Genetic Algorithms, San

scheduling problems via
imization. Second World

ed 3-colouring algorithm.

observations about GA-
he Second Conference on
tabling, Toronto, Canada.
torial optimisation by ge-
phenotype distinction. In
., editors, *EvCA96: Pro-*
ce on Evolutionary Com-
ute for High-performance
ences.
Rendon, M. (1999). Evo-
examination timetabling.
M., Honavar, V., Jakiela,
e Genetic and Evolution-
San Mateo, CA. Morgan

Job shop scheduling by
ng, 8:302–317.

Chapter 10

CONSTRAINED GENETIC ALGORITHMS AND THEIR APPLICATIONS IN NONLINEAR CONSTRAINED OPTIMIZATION *

Benjamin W. Wah and
Yi-Xin Chen

Abstract This chapter 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 developed 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 and genetic algorithm, performs well when using crossovers, mutations, and annealing to generate trial points. Finally, we apply iterative deepening to determine the optimal number of generations in *CSAGA* and show that its performance is robust with respect to changes in population size.

1. Introduction

Many engineering applications can be formulated as constrained *nonlinear programming problems* (NLPs). Examples include production planning, computer integrated manufacturing, chemical control process-

*Research supported by National Aeronautics and Space Administration Contract NAS2-37143.

ing, and structure optimization. These applications can be solved by existing methods if they are specified in well-defined formulae that are differentiable and continuous. However, only special cases can be solved when they do not satisfy the required assumptions. For instance, sequential quadratic programming cannot handle problems whose objective and constraint functions are not differentiable or whose variables are discrete or mixed. Since many applications involving optimization may be formulated by non-differentiable functions with discrete or mixed-integer variables, it is important to develop new methods for handling these optimization problems. inxxconstrained nonlinear programming

The study of algorithms for solving a disparity of constrained optimization problems is difficult unless the problems can be represented in a unified way. In this chapter we assume that continuous variables are first discretized into discrete variables in such a way that the values of functions using discretized variables approach those of the original continuous variables. Such an assumption is valid when continuous variables are represented as floating-point numbers and when the range of variables is small (say between 10^{-5} and 10^{5}). Intuitively, if discretization is fine enough, then solutions found in discretized space are fairly good approximations to the original solutions. The accuracy of solutions found in discretized problems has been studied elsewhere (Wu, 2000).

Based on discretization, continuous and mixed-integer constrained NLPs can be represented as discrete constrained NLPs as follows:[1]

$$\begin{array}{lll} \text{minimize} & f(x) & \\ \text{subject to} & g(x) \leq 0 & x = [x_1, \ldots, x_n]^T \text{ is a vector} \\ & h(x) = 0 & \text{of bounded discrete variables.} \end{array} \tag{10.1}$$

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.

Solutions to (10.1) cannot be characterized in ways similar to those of problems with differentiable functions and continuous variables. In the latter class of problems, solutions are defined with respect to neighborhoods of open spheres with radius approaching zero asymptotically. Such a concept does not exist in problems with discrete variables.

[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.

:ions can be solved by
ined formulae that are
:ial cases can be solved
;. For instance, sequen-
ns whose objective and
ie variables are discrete
timization may be for-
crete or mixed-integer
ods for handling these
r programming
ty of constrained opti-
ns can be represented
it continuous variables
ı a way that the values
ı those of the original
when continuous vari-
ınd when the range of
ıtuitively, if discretiza-
:etized space are fairly
ie accuracy of solutions
lsewhere (Wu, 2000).
ed-integer constrained
NLPs as follows:[1]

(10.1)

is a vector
crete variables.

on, $g(x) = [g_1(x), \cdots,$
$= [h_1(x), \cdots, h_m(x)]^T$
$f(x)$, $g(x)$, and $h(x)$
ier linear or nonlinear,
l. Without loss of gen-

ways similar to those
ntinuous variables. In
with respect to neigh-
ıg zero asymptotically.
liscrete variables.

$\leq w$ means that each element
be defined similarly. 0, when

Let X be the Cartesian product of the discrete domains of all variables in x. To characterize solutions sought in discrete space, we define the following concepts on neighborhoods and constrained solutions in discrete space:

Definition 1. $\mathcal{N}_{dn}(x)$, the *discrete neighborhood* (Aarts and Korst, 1989) of point $x \in X$ is a *finite* user-defined set of points $\{x' \in X\}$ such that $x' \in \mathcal{N}_{dn}(x) \Longleftrightarrow x \in \mathcal{N}_{dn}(x')$, 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 $x' \in \mathcal{N}_{dn}(x)$.

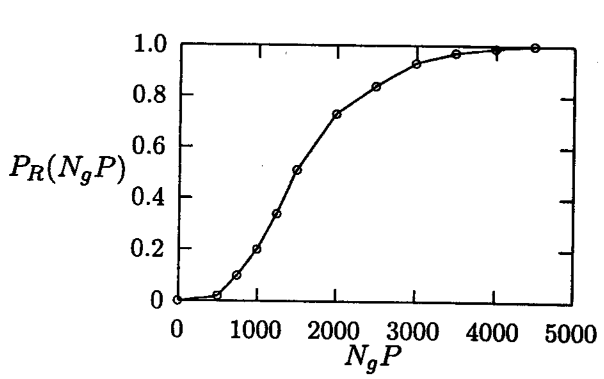
Definition 3. Point $x \in X$ is called a *constrained global minimum in discrete neighborhood* (CGM_{dn}) iff a) x is feasible, and b) for every feasible point $x' \in X$, $f(x') \geq f(x)$. The set of all CGM_{dn} is X_{opt}. According to our definitions, a CGM_{dn} must also be a CLM_{dn}.

In a similar way, there are definitions on continuous-neighborhood constrained local minima (CLM_{cn}) and constrained global minima (CGM_{cn}).

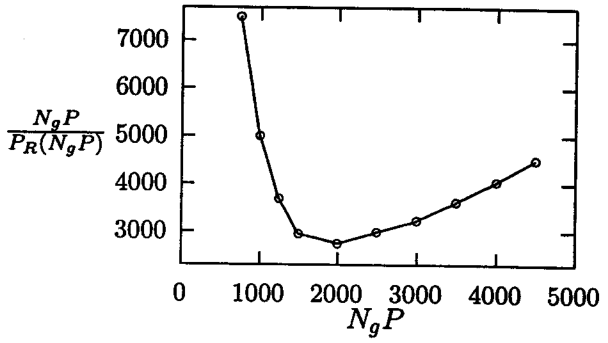
We have shown earlier (Wah and Wu, 1999) that the necessary and sufficient condition for a point to be a CLM_{dn} is that it satisfies the discrete-neighborhood saddle-point condition (Section 2.1). We have also extended simulated annealing (SA) (Wah and Wang, 1999) and greedy search (Wah and Wu, 1999) 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 (Michalewicz and Schoenauer, 1996) (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 chapter 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 chapter 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 j^{th} probe and a simplistic assumption that all probes are



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 10.1. An example showing the application of *CSAGA* with $P = 3$ to solve a discretized version of G1 (Michalewicz and Schoenauer, 1996) ($N_{opt}P \approx 2000$).

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

$$P_R(N) = 1 - \prod_{j=1}^{N}(1 - p_j), \quad \text{where } N \geq 0. \tag{10.2}$$

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

As an example, Figure 10.1a 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 value of $P_R(N)$ when a test problem is solved by an algorithm, 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:

$$\sum_{j=1}^{\infty} P_R(N)(1 - P_R(N))^{j-1} N \times j = \frac{N}{P_R(N)}. \tag{10.3}$$

Figure 10.1b plots (10.3) based on $P_R(N_gP)$ in Figure 10.1a. In general, there exists N_{opt} that minimizes (10.3) 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 10.1b 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 (10.3) in finding a CGM_{dn}. Finally, Section 4 compares the performance of our algorithms.

2. Previous Work

In this section, we first summarize the theory of Lagrange multipliers applied to solve (10.1) and two algorithms developed based on the theory. We then describe existing work in GA for solving constrained NLPs.

2.1 Theory of Lagrange multipliers for solving discrete constrained NLPs

Define a discrete equality-constrained NLP as follows (Wah and Wu, 1999; Wu, 2000):

$$\begin{array}{lll} \min\limits_{x} & f(x) & x \text{ is a vector of bounded} \\ \text{subject to} & h(x) = 0 & \text{discrete variables,} \end{array} \tag{10.4}$$

A *generalized discrete augmented Lagrangian function* of (10.4) is defined as follows (Wah and Wu, 1999):

$$L_d(x, \lambda) = f(x) + \lambda^T H(h(x)) + \frac{1}{2}||h(x)||^2, \tag{10.5}$$

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 $H(h(x)) = [\max(h_1(x), 0), \cdots, \max(h_m(x), 0)]^T$. Note that these transformations are not used in Lagrange-multiplier methods in continuous space because they are not differentiable at $H(h(x)) = 0$. However, they do not pose problems here because we do not require their differentiability.

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:

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*) \tag{10.6}$$

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.

The concept of SP_{dn} is very important in discrete problems because, starting from them, we can derive first-order necessary and sufficient condition for CLM_{dn} that leads to global minimization procedures. This is stated formally in the following theorem (Wah and Wu, 1999):

Theorem 1 First-order necessary and sufficient condition on CLM_{dn} *(Wah and Wu, 1999). A point in the discrete search space of (10.4) is a* CLM_{dn} *iff it satisfies (10.6) for any* $\lambda \geq \lambda^*$.

Theorem 1 is stronger than its continuous counterparts. The first-order necessary conditions in continuous Lagrange-multiplier theory (Luenberger, 1984) 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 in continuous theory (Luenberger, 1984) 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) (Wu, 1998). 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

```
1. procedure CSA $(\alpha, N_\alpha)$
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 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 $\alpha$;
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 $N_\alpha$;
8. end_procedure
```

b) CSA_{ID}: CSA with iterative deepening

Figure 10.2. Constrained simulated annealing algorithm (CSA) and its iterative-deepening extension

order to perform ascents in [illegible] space 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 (Wah and Wu, 1999; Wu, 2000). 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) (Wah and Wang, 1999) algorithm shown in Figure 10.2a. 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}.

Using $G(\mathbf{x}, \mathbf{x}')$ for generating trial point $\mathbf{x}'$ in $\mathcal{N}_{dn}(\mathbf{x})$, $A_T(\mathbf{x}, \mathbf{x}')$ as the Metropolis acceptance probability, and a logarithmic cooling schedule, CSA has been proven to have asymptotic convergence with probability one to CGM_{dn} (Wah and Wang, 1999). This property is stated in the following theorem:

Theorem 2 *Asymptotic convergence of CSA.* 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 (10.3).

We have verified experimentally that the expected time defined in (10.3) has an absolute minimum at N_{opt}. (Figure 10.1b illustrates the existence of N_{opt} for $CSAGA$.) It follows that, in order to minimize (10.3), 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* (Korf, 1985) by starting CSA with a short schedule and by doubling the schedule each time the current run fails to find a CGM_{dn} (Wah and Chen, 2000). Since the total overhead in iterative deepening is dominated by that of the last run, CSA_{ID} (CSA with iterative deepening in Figure 10.2b) 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 10.3 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 proposed to run CSA multiple times from random starting points at each schedule in CSA_{ID}. Figure 10.2b shows CSA that is run $K = 3$

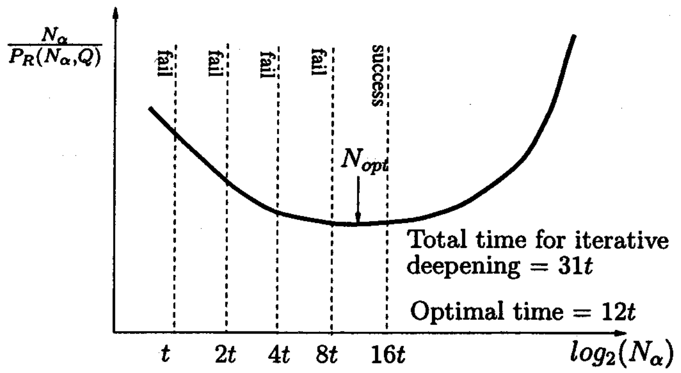


Figure 10.3. An application of iterative deepening in CSA_{ID}.

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} (Wah and Chen, 2000).

2.3 Genetic algorithms for solving constrained NLP problems

Genetic algorithm (GA) is a general stochastic optimization algorithm that maintains a population of alternative candidates and that probes a search space using genetic operators, such as crossovers and mutations, in order to find better candidates. The original GA was developed for solving unconstrained problems, using a single fitness function to rank candidates. Recently, many variants of GA have been developed for solving constrained NLPs. Most of these methods were based on penalty formulations that use GA to minimize an unconstrained penalty function $\mathcal{F}(x)$, consisting of a sum of the objective and the constraints weighted by penalties. Similar to CSA, these methods do not require the differentiability or continuity of functions.

One penalty formulation is the *static-penalty formulation* in which all penalties are fixed (Bertsekas, 1982):

$$\mathcal{F}_\rho(x,\gamma) = f(x) + \sum_{i=1}^{m} \gamma_i |h_i(x)|^\rho, \tag{10.7}$$

where $\rho > 0$, and penalty vector $\gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_m\}$ is *fixed* and chosen to be large enough so that

$$\mathcal{F}_\rho(x^*,\gamma) < \mathcal{F}_\rho(x,\gamma) \quad \forall x \in X - X_{opt} \text{ and } x^* \in X_{opt}. \tag{10.8}$$

Based on (10.8), an unconstrained global minimum of (10.7) over x is a CGM_{dn} to (10.4); hence, it suffices to minimize (10.7) in solving (10.4). Since both $f(x)$ and $|h_i(x)|$ are lower bounded and x takes finite discrete values, γ always exists and is finite, thereby ensuring the correctness of the approach. Note that other forms of penalty formulations have also been studied in the literature.

The major issue of static-penalty methods lies in the difficulty of selecting a suitable γ. If γ is much larger than necessary, then the terrain will be too rugged to be searched effectively by local-search methods. If it is too small, then feasible solutions to (10.7) may be difficult to find.

Dynamic-penalty methods (Joines and Houck, 1994), on the other hand, address the difficulties of static-penalty methods by increasing penalties gradually in the following fitness function:

$$\mathcal{F}(x) = f(x) + (C \times t)^{\alpha} \sum_{j=1}^{m} |h_i(x)|^{\beta}, \tag{10.9}$$

where t is the generation number, and C, α, and β are constants. In contrast to static-penalty methods, $(C \times t)^{\alpha}$, the penalty on infeasible points, is increased during evolution.

Dynamic-penalty methods do not always guarantee convergence to CLM_{dn} or CGM_{dn}. For example, consider a problem with two constraints $h_1(x) = 0$ and $h_2(x) = 0$. Assuming that a search is stuck at an infeasible point x' and that for all $x \in \mathcal{N}_{dn}(x')$, $0 < |h_1(x')| < |h_1(x)|$, $|h_2(x')| > |h_2(x)| > 0$, and $|h_1(x')|^{\beta} + |h_2(x')|^{\beta} < |h_1(x)|^{\beta} + |h_2(x)|^{\beta}$, then the search can never escape from x' no matter how large $(C \times t)^{\alpha}$ grows.

One way to ensure the convergence of dynamic-penalty methods is to use a different penalty for each constraint, as in Lagrangian formulation (10.5). In the previous example, the search can escape from x' after assigning a much larger penalty to $h_2(x')$ than that to $h_1(x')$.

There are many other variants of penalty methods, such as annealing penalties, adaptive penalties (Michalewicz and Schoenauer, 1996) and self-adapting weights (Eiben and Ruttkay, 1996). In addition, problem-dependent operators have been studied in the GA community 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. However, most methods require domain-specific knowledge or problem-dependent genetic operators, and have difficulties in finding feasible regions or in maintaining feasibility for nonlinear constraints.

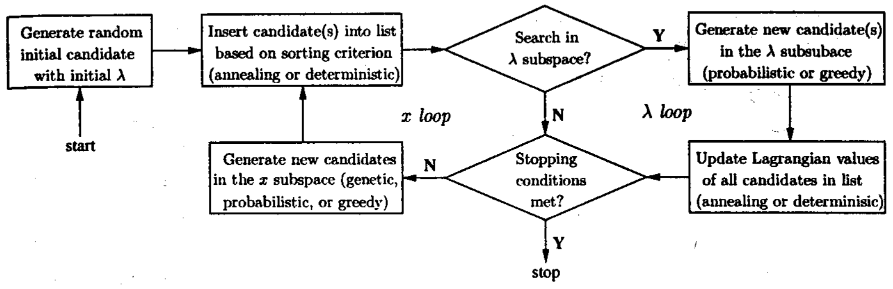

Generate random initial candidate with initial λ
Insert candidate(s) into list based on sorting criterion (annealing or deterministic)
Search in λ subspace?
Y
Generate new candidate(s) in the λ subsubace (probabilistic or greedy)
x loop
N
λ loop
start
Generate new candidates in the x subspace (genetic, probabilistic, or greedy)
N
Stopping conditions met?
Update Lagrangian values of all candidates in list (annealing or determinisic)
Y
stop

Figure 10.4. An iterative stochastic procedural framework to look for SP_{dn}.

In general, local minima of penalty functions are only necessary but not sufficient to be constrained local minima of the original constrained optimization problems, unless the penalties are chosen properly. Hence, finding local minima of a penalty function does not necessarily solve the original constrained optimization problem.

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 10.4 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

```
1. procedure CGA($P$, $N_g$)
2.     set generation number $t \leftarrow 0$ and $\lambda(t) \leftarrow 0$;
3.     initialize population $\mathcal{P}(t)$;
4.     repeat /* over multiple generations */
5.         evaluate $L_d(x, \lambda(t))$ for all candidates in $\mathcal{P}(t)$;
6.         repeat /* over probes in $x$ subspace */
7.             $y \leftarrow GA(select(\mathcal{P}(t)))$;
8.             evaluate $L_d(y, \lambda)$ and insert into $\mathcal{P}(t)$
9.         until sufficient probes in $x$ subspace;
10.        $\lambda(t) \leftarrow \lambda(t) \oplus c\mathcal{H}(h, \mathcal{P}(t))$; /* update $\lambda$ */
11.        $t \leftarrow t + 1$;
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 $i \leftarrow 1$ to $K$ do call $CGA(P, N_g)$ end_for
5.         set $N_g \leftarrow \rho N_g$ (typically $\rho = 2$);
6.     until $N_g$ exceeds maximum allowed or
           (no better solution has been found in two
           successive increases of $N_g$ and $N_g > \rho^5 N_0$
           and a feasible solution has been found);
7. end_procedure
```

b) CGA_{ID}: *CGA* with iterative deepening

Figure 10.5. Constrained *GA* and its iterative deepening version.

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 10.5a was developed based on the general framework in Figure 10.4. 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 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 2-3 initialize to zero the generation number t and the vector of Lagrange multipliers λ. The initial population $\mathcal{P}(t)$ can be either randomly generated or user provided.

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 10.4), 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,

$$\mathcal{H}_i(h(x), \mathcal{P}(t)) = \max_{x \in \mathcal{P}(t)} H(h_i(x)), \; i = 1, \cdots, m, \tag{10.10}$$

where $h_i(x)$ is the i^{th} constraint function, H is the non-negative transformation defined in (10.5), and c is a step-wise constant controlling how fast λ changes.

Operator $\oplus$ in Figure 10.5a 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.

3.2 Combined Constrained *SA* and *GA* (*CSAGA*)

Based on the general framework in Figure 10.4, we design *CSAGA* by integrating *CSA* in Figure 10.2a and *CGA* in Figure 10.5a into a combined procedure. The new procedure differs from the original *CSA* in two respects. 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

```
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 $q$ do /* $SA$ in Lines 5-10 */
5.              for $j \leftarrow 1$ to $P$ do
6.                  generate $\mathbf{x}'_j$ from $\mathcal{N}_{dn}(\mathbf{x}_j)$ 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.              set $T \longleftarrow \alpha T$; /* set $T$ for the $SA$ part */
10.         end_for
11.         repeat /* by $GA$ over probes in $x$ subspace */
12.             $y \leftarrow GA(select(\mathcal{P}(t)))$;
13.             evaluate $L_d(y, \lambda)$ and insert $y$ into $\mathcal{P}(t)$;
14.         until sufficient number of probes in $x$ subspace;
15.         $t \leftarrow t + q$; /* update generation number */
16.     until $(t \geq N_g)$
17. end_procedure
```

Figure 10.6. $CSAGA$: Combined CSA and CGA called with population size P and N_g generations.

CSA corresponding to a candidate from a random starting point, it is best to run CSA sequentially, using the best solution point found in one run as the starting point of the next run. Second, we need to determine the duration of each run of CSA. This is controlled by parameter q that was set to be $\frac{N_g}{6}$ after experimental evaluations. The new algorithm shown in Figure 10.6 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 $\mathbf{x}'_j$ and accept it based on the Metropolis probability. Experimentally, we set q to be $\frac{N_g}{6}$. As discussed earlier, we use the best point of one run as the starting point of the next run.

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.

```
0 */
g G(xj, x'j);
(xj, x'j)
part */
bspace */
P(t);
subspace;
er */
```

h population size P and

starting point, it is
n point found in one
e need to determine
by parameter q that
The new algorithm
erate new probes in

$\mathbf{x} = [x_1, \cdots, x_n, \lambda_1,$
bspaces. Initially, x
is set to zero.
y candidate in the
cally $\mathbf{x}'_j$ and accept
ally, we set q to be
run as the starting

as been completed.
ng probes using GA
ness values L_d after
nce each candidate
thm first computes
constraint over all
andidate using the

Note that $CSAGA$ has difficulties similar to those of CGA in determining a proper number of candidates to use in its population and the duration of each run. We address these two issues in the $CSAGA_{ID}$ in the next subsection.

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 (Korf, 1985) that determines an upper bound on N_g in order to minimize the expected total overhead in (10.3), 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 (10.3), the expected total number of probes using multiple runs of either CGA or $CSAGA$ and fixed P is:

$$\frac{N}{P_R(N)} = \frac{N_g P}{P_R(N_g P)} = P\frac{N_g}{\hat{P}_R(N_g)} \tag{10.11}$$

In order to have an optimal number of generations $N_{g_{opt}}$ that minimizes (10.11), $\frac{N_g}{\hat{P}_R(N_g)}$ must have an absolute minimum in $(0,\infty)$. 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 (Michalewicz and Schoenauer, 1996). Figure 10.1b 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 10.5b uses a set of geometrically increasing N_g to find a CGM_{dn}:

$$N_{g_i} = \rho^i N_0, \qquad i = 0, 1, \ldots \tag{10.12}$$

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 10.5b to find a solution of quality f' starting from N_0 generations. According to (10.11),

$$B_{opt}(f') = P\frac{N_{g_{opt}}}{\hat{P}_R(N_{g_{opt}})} \tag{10.13}$$

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 $\lim_{N_g \to \infty} \hat{P}_R(N_g) \le 1$;

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

The proof is not shown due to space limitations.

Typically, $\rho = 2$, and $\hat{P}_R(N_{g_{opt}}) \ge 0.25$ 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 10.2b 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 our experimental results in evaluating CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ on discrete constrained NLPs. Based on the framework in Figure 10.4, 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.

Due to a lack of large-scale discrete benchmarks, we derive our benchmarks from two sets of continuous benchmarks: Problem G1-G10 (Michalewicz and Schoenauer, 1996; Koziel and Michalewicz, 1999) and Floudas and Pardalos' Problems (Floudas and Pardalos, 1990).

4.1 Implementation Details

In theory, algorithms derived from the framework, such as CSA, CGA, and $CSAGA$, will look for SP_{dn}. In practice, however, it is important to choose appropriate neighborhoods and generate proper trial points in x and λ subspaces in order to solve constrained NLPs efficiently.

An important component of these methods is the frequency at which λ is updated. Like in CSA (Wah and Wang, 1999), we have set experimentally in CGA and $CSAGA$ the ratio of generating trial points in x and λ subspaces from the current point to be $20n$ to m, where n is the number of variables and m is the number of constraints. This ratio means that x is updated more often than λ.

In generating trial points in the x subspace, we have used a dynamically controlled neighborhood size in the SA part (Wah and Wang, 1999) based on the 1:1 ratio rule (Corana et al., 1987), whereas in the GA part, we have used the seven operators in Genocop III (Michalewicz and Nazhiyath, 1995) and L_d as our fitness function. In implementing CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$, we have used the default parameters of CSA (Wah and Wang, 1999) in the SA part and those of Genocop III (Michalewicz and Nazhiyath, 1995) in the GA part.

The generation of trial point λ' in the λ subspace is done by the following rule:

$$\lambda'_j = \lambda_j + r_1\,\phi_j \qquad \text{where } j = 1, \cdots, m. \tag{10.14}$$

Here, r_1 is randomly generated in $[-1/2, +1/2]$ if we choose to generate λ probabilistically, and is randomly generated in $[0, 1]$ if we choose to generate probes in λ in a greedy fashion.

We adjust ϕ adaptively according to the degree of constraint violations, where

$$\phi = w \otimes \mathcal{H}(x) = [w_1\mathcal{H}_1(x), w_2\mathcal{H}_2(x), \cdots, w_m\mathcal{H}_m(x)], \tag{10.15}$$

$\otimes$ represents vector product, and $\mathcal{H}$ is the vector of maximum violations defined in (10.10). When $\mathcal{H}_i(x)$ is satisfied, λ_i does not need to be

Table 10.1. 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	Sol. 1% off CGM_{dn}			Sol. 10% off CGM_{dn}		
λ subspace	x subspace	Strategy	*CSA*	*CGA*	*CSAGA*	*CSA*	*CGA*	*CSAGA*
probabilistic	probabilistic	annealing	6.91	23.99	4.89	1.35	–	1.03
probabilistic	probabilistic	deterministic	9.02	–	6.93	1.35	2.78	1.03
probabilistic	deterministic	annealing	–	18.76	–	89.21	2.40	–
probabilistic	deterministic	deterministic	–	16.73	–	–	2.18	–
greedy	probabilistic	annealing	7.02	–	7.75	1.36	–	0.90
greedy	probabilistic	deterministic	7.02	–	7.75	1.36	–	0.90
greedy	deterministic	annealing	–	25.50	–	82.24	1.90	–
greedy	deterministic	deterministic	–	25.50	–	82.24	1.90	–

updated; hence, $\phi_i = 0$. In contrast, when a constraint is not satisfied, we adjust ϕ_i by modifying w_i according to how fast $\mathcal{H}_i(x)$ is changing:

$$w_i = \begin{cases} \eta_0 \, w_i & \text{if } \mathcal{H}_i(x) > \tau_0 T \\ \eta_1 \, w_i & \text{if } \mathcal{H}_i(x) < \tau_1 T \end{cases} \qquad (10.16)$$

where T is the temperature, and $\eta_0 = 1.25$, η_1=0.8, $\tau_0 = 1.0$, and $\tau_1 = 0.01$ were chosen experimentally. When $\mathcal{H}_i(x)$ is reduced too quickly (*i.e.*, $\mathcal{H}_i(x) < \tau_1 T$), $\mathcal{H}_i(x)$ is over-weighted, leading to possibly poor objective values or difficulty in satisfying other under-weighted constraints. Hence, we reduce λ_i's neighborhood. In contrast, if $\mathcal{H}_i(x)$ is reduced too slowly (*i.e.*, $\mathcal{H}_i(x) > \tau_0 T$), we enlarge λ_i's neighborhood in order to improve its chance of satisfaction. Note that w_i is adjusted using T as a reference because constraint violations are expected to decrease when T decreases.

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.

4.2 Evaluation Results

Due to a lack of large-scale discrete benchmarks, we derive our benchmarks from two sets of continuous benchmarks: Problem G1-G10 (Michalewicz and Schoenauer, 1996; Koziel and Michalewicz, 1999) and Floudas and Pardalos' Problems (Floudas and Pardalos, 1990).

In generating a discrete constrained NLP, we discretize continuous variables in the original continuous constrained NLP into discrete variables as follows. In discretizing continuous variable x_i in range $[l_i, u_i]$, where l_i and u_i are lower and upper bounds of x_i, respectively, we force x_i to take values from the set:

$$A_i = \begin{cases} \left\{a_i + \frac{b_i - a_i}{s} j,\ j = 0, 1, \cdots, s\right\} & \text{if } b_i - a_i < 1 \\ \left\{a_i + \frac{1}{s} j,\ j = 0, 1, \cdots, \lfloor (b_i - a_i)s \rfloor\right\} & \text{if } b_i - a_i \geq 1, \end{cases} \tag{10.17}$$

where $s = 1.0 \times 10^7$.

Table 10.1 shows the results of evaluating various combinations of strategies in CSA_{ID}, CGA_{ID}, and $CSAGA_{ID}$ on a discretized version of G2 (Michalewicz and Schoenauer, 1996; Koziel and Michalewicz, 1999). 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 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 (Michalewicz and Schoenauer, 1996; Koziel and Michalewicz, 1999). 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 10.2. 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. The best $\mathcal{B}_{ID}(f^*)$ for each problem is boxed.)

Prob.	Best	EAs		CSA_{ID}	CGA_{ID}		$CSAGA_{ID}$			
ID	Sol. f^*	Best f	Method	$\mathcal{B}_{ID}(f^*)$	P_{opt}	$\mathcal{B}_{ID}(f^*)$	P	$\mathcal{B}_{ID}(f^*)$	P_{opt}	$\mathcal{B}_{ID}(f^*)$
G1	-15	-15	Genocop	1.65	40	5.49	3	1.64	2	$\boxed{1.31}$
G2	-0.80362	0.803553	S.T.	7.28	30	311.98	3	$\boxed{5.18}$	3	$\boxed{5.18}$
G3	1.0	1.0	S.T.	1.07	30	14.17	3	$\boxed{0.89}$	3	$\boxed{0.89}$
G4	-30665.5	-30664.5	H.M.	$\boxed{0.76}$	5	3.95	3	0.95	3	0.95
G5	4221.9	5126.498	D.P.	2.88	30	68.9	3	2.76	2	$\boxed{2.08}$
G6	-6961.81	-6961.81	Genocop	0.99	4	7.62	3	0.91	2	$\boxed{0.73}$
G7	24.3062	24.62	H.M.	6.51	30	31.60	3	4.60	4	$\boxed{4.07}$
G8	0.095825	0.095825	H.M.	0.11	30	0.31	3	0.13	4	$\boxed{0.10}$
G9	680.63	680.64	Genocop	0.74	30	5.67	3	$\boxed{0.57}$	3	$\boxed{0.57}$
G10	7049.33	7147.9	H.M.	$\boxed{3.29}$	30	82.32	3	3.36	3	3.36

Table 10.2 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. Since all CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ can find a CGM_{dn} in *all 10 runs*, we compare their performance with respect to $\mathcal{T}$, the average total overhead of multiple runs until a CGM_{dn} is found. 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}, for the 10 problems except for G4 and G10.

Table 10.3. 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.	$\boxed{14.86\text{ sec.}}$
2.7.2(min)	-884.75	20	53.92 sec.	$\boxed{15.54\text{ sec.}}$
2.7.3(min)	-8695.0	20	34.22 sec.	$\boxed{22.52\text{ sec.}}$
2.7.4(min)	-754.75	20	36.70 sec.	$\boxed{16.20\text{ sec.}}$
2.7.5(min)	-4150.4	20	89.15 sec.	$\boxed{23.46\text{ sec.}}$
5.2(min)	1.567	46	3168.29 sec.	$\boxed{408.69\text{ sec.}}$
5.4(min)	1.86	32	2629.52 sec.	$\boxed{100.66\text{ sec.}}$
7.2(min)	1.0	16	824.45 sec.	$\boxed{368.72\text{ sec.}}$
7.3(min)	1.0	27	2323.44 sec.	$\boxed{1785.14\text{ sec.}}$
7.4(min)	1.0	38	951.33 sec.	$\boxed{487.13\text{ sec.}}$

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 10.3 shows the results on selected discretized Floudas and Pardalos' benchmarks (Floudas and Pardalos, 1990) 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.

5. Conclusions

In this chapter we have presented new algorithms to look for discrete-neighborhood saddle points in discrete Lagrangian space of constrained optimization problems. Our results show that genetic algorithms, when combined with simulated annealing, are effective in locating saddle points. Future developments will focus on better ways to select appropriate heuristics in probe generation, including search direction control and neighborhood size control, at run time.

References

Aarts, E. and Korst, J. (1989) *Simulated Annealing and Boltzmann Machines.* J. Wiley and Sons.

Bertsekas, D. P. (1982) *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press.

Corana, A., Marchesi, M., Martini, C. and Ridella, S. (1987) Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Trans. on Mathematical Software*, 13(3):262–280.

Eiben, A. E. and Ruttkay, Zs. (1996) Self-adaptivity for constraint satisfaction: Learning penalty functions. *Proceedings of the 3rd IEEE Conference on Evolutionary Computation*, 258–261.

Floudas, C. A. and Pardalos, P. M. (1990) *A Collection of Test Problems for Constrained Global Optimization Algorithms*, volume 455 of *Lecture Notes in Computer Science.* Springer-Verlag.

Joines, J. and Houck, C. (1994) On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. *Proceedings of the First IEEE International Conference on Evolutionary Computation*, 579–584.

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

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

Luenberger, D. G. (1984) *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Reading, MA.

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

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

netic algorithms, when
n locating saddle points.
to select appropriate
direction control and

ng and Boltzmann Ma-

n and Lagrange Multi-

ı, S. (1987) Minimizing
with the simulated an-
al Software, 13(3):262–

vity for constraint sat-
lings of the 3rd IEEE
–261.
ollection of Test Prob-
orithms, volume 455 of
Verlag.
ıon-stationary penalty
ization problems with
al Conference on Evo-

An optimal admissible

ary algorithms, homo-
r optimization. Evolu-

'rogramming. Addison-

III: A co-evolutionary
s with nonlinear con-
ıference on Evolution-

tionary algorithms for
Evolutionary Compu-

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

Wah, B. W. and Wang, T. (1999) Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. *Principles and Practice of Constraint Programming*, 461–475.

Wah, B. W. and Wu, Z. (1999) The theory of discrete Lagrange multipliers for nonlinear discrete optimization. *Principles and Practice of Constraint Programming*, 28–42.

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

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