Hybrid Evolutionary and Annealing Algorithms for Nonlinear Discrete Constrained Optimization[1]

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 procedural framework that unifies various mechanisms to look for discrete-neighborhood saddle points in solving discrete constrained optimization problems (DCOPs). Our approach is based on the necessary and sufficient condition on local optimality in discrete space, which shows the one-to-one correspondence between the discrete-space constrained local minima of a problem and the saddle points of the corresponding Lagrangian function. To look for such saddle points, we study various mechanisms for performing ascents of the Lagrangian function in the original-variable subspace and descents in the Lagrange-multiplier subspace. Our results show that CSAEA, a combined constrained simulated annealing and evolutionary algorithm, performs well when using mutations and crossovers to generate trial points and accepting them based on the Metropolis probability. We apply iterative deepening to determine the optimal number of generations in CSAEA and show that its performance is robust with respect to changes in population size. To test the performance of the procedures developed, we apply them to solve some continuous and mixed-integer nonlinear programming (NLP) benchmarks and show that they obtain better results than those of existing algorithms.

Key words: Evolutionary algorithms, saddle points, iterative deepening, nonlinear discrete constrained optimization, simulated annealing.

1 Introduction

Many engineering applications can be formulated as *discrete constrained optimization problems* (DCOPs). Examples include production planning, computer integrated manufacturing, chemical control processing, and structure

[1]To appear in the *Central European Journal of Operations Research*, 2004

Table 1: List of acronyms used in this paper.

CLM_c	continuous neighborhood constrained local minimum
CGM_c	continuous neighborhood constrained global minimum
CEA	constrained evolutionary algorithm
CEA_{ID}	CEA with iterative deepening
CSA	constrained simulated annealing
CSA_{ID}	CSA with iterative deepening
CSAEA	Combined constrained SA and EA algorithm
CSAEA_{ID}	CSAEA with iterative deepening
CLM_d	discrete neighborhood constrained local minimum
CGM_d	discrete neighborhood constrained global minimum
DLM	discrete Lagrangian method
$\mathcal{N}_d$	discrete neighborhood
SP_d	discrete-neighborhood saddle point
DCOP	discrete constrained optimization problem
EA	evolutionary algorithm
MINLP	mixed-integer nonlinear programming problem
NLP	nonlinear programming problem
SA	simulated annealing
SSA	stochastic search algorithm

optimization.

In this paper we review the necessary and sufficient condition for local optimality in discrete constrained optimization, present procedures for solving these problems, and apply these procedures to solve some nonlinear programming problems (NLPs). The procedures presented are especially useful when an NLP is formulated by non-differentiable functions with discrete or mixed-integer variables. Table 1 summarizes the acronyms used.

A DCOP can be formulated as follows:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0 \ \text{ and } \ h(x) = 0, \end{aligned} \tag{1}$$

where $x = (x_1, \ldots, x_n)^T \in \mathcal{D}^n$ is a vector of bounded discrete variables. 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 constraint functions, and $h(x) = (h_1(x), \cdots, h_m(x))^T$ is a vector of m equality constraint functions. Functions $f(x)$, $g(x)$, and $h(x)$ are not necessarily differentiable and can be linear or nonlinear, continuous or discrete, and analytic or procedural. Without loss of generality, we consider only minimization problems.

Solutions to (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 whose radius approaches zero asymptotically. Such a concept does not exist in problems with discrete variables. To characterize solutions sought in discrete space, we define the following concepts on neighborhoods and constrained solutions in discrete space $X \subseteq \mathcal{D}^n$.

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

Definition 2. Point $x \in X$ is called a *discrete-neighborhood constrained local minimum* (CLM_d) if it satisfies two conditions: a) x is a feasible point and satisfies constraints $g(x) \leq 0$ and $h(x) = 0$; and b) $f(x) \leq f(x')$ for all feasible $x' \in \mathcal{N}_d(x)$. In particular, x is a CLM_d when x is feasible and all its neighboring points in $\mathcal{N}_d(x)$ are infeasible.

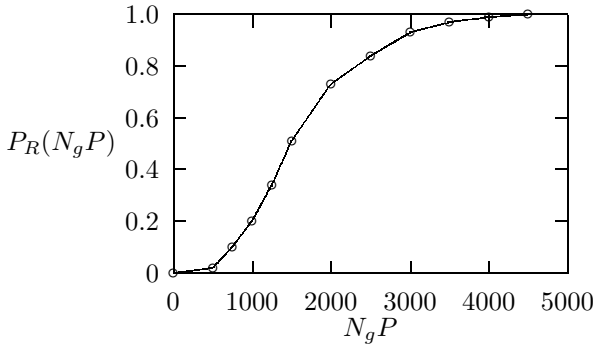
Definition 3. Point $x \in X$ is called a *constrained global minimum in discrete neighborhood* (CGM_d) iff a) x is feasible; and b) for every feasible point $x' \in X$, $f(x') \geq f(x)$. It is obvious that a CGM_d is also a CLM_d. The set of all CGM_d is X_{opt}.

There are corresponding definitions of constrained local minima (CLM_c) and constrained global minima (CGM_c) in continuous space, although they are not presented formally here.

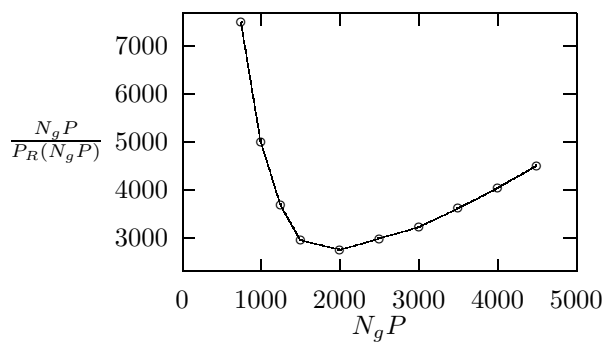
We have shown earlier [17] that the discrete-neighborhood extended saddle-point condition (Section 2.1) is necessary and sufficient for a point to be a CLM_d. We have also extended simulated annealing (SA) [16] and greedy search [17] to look for such saddle points SP_d (Section 2.2). At the same time, new problem-dependent heuristics have been developed in the evolutionary algorithm (EA) community to handle nonlinear constraints [12] (Section 2.3). Up to now, there is no clear understanding on how the various search mechanisms in SA, EA, and others can be unified effectively for solving a wide range of optimization problems.

Our primary goal in this paper is to show that discrete-neighborhood saddle points can be found effectively by integrating the various search mechanisms in SA and EA. We present a framework that employs these mechanisms to probe a search space, where a probe is a neighboring point examined, independent of whether it is accepted or not. We illustrate our approach on CSAEA, a combined constrained simulated annealing and evolutionary algorithm, that uses mutations and crossovers to generate trial points and that accepts the points generated according to the Metropolis probability. Finally, we use iterative deepening to determine the optimal number of generations in CSAEA and show that its performance is robust with respect to changes in population size. Note that our approach is general and works for other operators to probe a search space.

CSAEA is a *stochastic search algorithm* (SSA) that generates probes in a random fashion in order to find solutions. Among the various convergence conditions of SSAs to a solution of desired quality, the strongest one is



a) $P_R(N_gP)$ approaches one as N_gP is increased



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

Figure 1: An example showing the application of CSAEA with $P = 3$ to solve a discretized version of G1 [12] ($N_{opt}P \approx 2000$).

asymptotic convergence with probability one. In this case, the search stops at a desired solution in the last probe with probability one when the number of probes approaches infinity. This concept is of theoretical interest because any algorithm with asymptotic convergence does not imply that it will find better solutions with higher probabilities when terminated in finite time.

A weaker convergence condition is the following reachability condition:

Definition 4. The *reachability probability*, $P_R(N)$, of an SSA after making N probes is the probability that the SSA will find an incumbent solution of desired quality in any of its N probes.

Let p_j to be the probability that the SSA finds a solution of desired quality in its jth probe and all probes be independent (a simplifying assumption). The reachability probability after making N probes is:

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

As an example, Figure 1a plots $P_R(N_gP)$ when CSAEA (see Section 3.2)

was run under various number of generations N_g and fixed population size $P = 3$ (where $N = N_g P$). The graph shows that $P_R(N_g P)$ approaches one as $N_g P$ is increased.

Although it is hard to estimate $P_R(N)$ when a test problem is solved by an SSA, we can always improve its chance of finding a solution by running the same SSA multiple times from random starting points. Given $P_R(N)$ for one run of the SSA and that all runs are independent, the expected total number of probes to find a solution of desired quality 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 $P_R(N_g P)$ 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 some search strategies that minimize (3) in finding a solution of desired quality. Finally, Section 4 compares the performance of our proposed algorithms.

2 Previous Work

In this section, we summarize the theory and algorithms of Lagrange multipliers in discrete space and some related work in EA.

2.1 Extended Saddle-Point Condition for Discrete Constrained Optimization

Define an equality-constrained DCOP as follows [17, 18]:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & h(x) = 0, \end{aligned} \tag{4}$$

where x is a vector of bounded discrete variables.

A *transformed augmented Lagrangian function* of (4) is defined as [17]:

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

where $\lambda = (\lambda_1, \cdots, \lambda_m)^T$ is a vector of Lagrange multipliers, $|h(x)| = (|h_1(x)|, \ldots, |h_m(x)|)^T$, and $\| h(x) \|^2 = \sum_{i=1}^{m} h_i^2(x)$.

Here, we have used $|h(x)|$ to transform $h(x)$ into a non-negative function. Note that, although the transformed constraints are not differentiable at $h(x) = 0$, they are not an issue in our theory because we do not require their differentiability.

By applying a similar transformation, inequality constraint $g_j(x) \leq 0$ can be transformed into equivalent equality constraint $\max(g_j(x), 0) = 0$. Hence, we only consider equality constraints from now on.

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

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

for all $x \in \mathcal{N}_d(x^*)$ and all $\lambda \in R^m$. Note that saddle points defined in (6) are with respect to discrete neighborhoods and are different from those in continuous space, although they both satisfy the same inequalities in (6).

The concept of SP_d is very important in discrete searches because, starting from them, we can derive the necessary and sufficient condition for CLM_d that governs the correctness of all such search procedures. This is stated formally in the following theorem [17]:

Theorem 1. Necessary and sufficient extended saddle-point condition (ESPC) on CLM_d [17]. *A point in the discrete search space of (4) is a CLM_d iff there exists a finite λ^* such that (6) is satisfied for for any extended Lagrange multiplier $\lambda^{**} > \lambda^*$.*[2]

2.2 Iterative Procedures for finding SP_d

Theorem 1 proves that finding (x^*, λ^*) that satisfies (6) entails the search of some $\lambda^{**} > \lambda^*$ and a local minimum of $L_d(x, \lambda^{**})$ with respect to x for this λ^{**}. This requirement is different from that of conventional theory of Lagrange multipliers in continuous space, which requires the search of *unique* Lagrange multipliers that satisfy a system of nonlinear equations. Since there is no closed-form solutions to a system of nonlinear equations, solving them requires an iterative search in the joint (x, λ) space. In contrast, finding the local minimum of $L_d(x, \lambda^{**})$ with respect to x for $\lambda^{**} \geq \lambda^*$ can be carried out in a decoupled fashion, with the search of λ^{**} separate from the search of x^*. A possible implementation is to have an iterative process with an inner loop that performs descents in $L_d(x, \lambda)$, while keeping λ fixed, until either a local minimum of L_d has been found or a predetermined number of probes have been made. The outer loop then increases the λ's on those violated constraints. The following are two example algorithms designed using this approach.

The first algorithm, the *discrete Lagrangian method* (DLM) [18], is an iterative local search that uses (6) as the stopping condition. It updates the x variables in an inner loop in order to perform descents of L_d in the x subspace, while occasionally updating the λ variables of unsatisfied constraints in an outer loop in order to perform ascents in the λ subspace. Note that

[2]For two vectors v and w of the same number of elements, $v \leq w$ means that each element of v is not greater than the corresponding element of w; and $v < w$ means that each element of v is less than the corresponding element of w. 0, when compared to a vector, stands for a null vector.

```
procedure CSA (α, N_α)
    set initial x ← (x_1, ··· , x_n, λ_1, ··· , λ_k)^T
        with random x, λ ← 0;
    while stopping condition (6) is not satisfied do
        generate x' ∈ N_d(x) using G(x, x');
        accept x' with probability A_T(x, x')
        reduce temperature by T ← αT;
    end_while
end_procedure
```

a) CSA called with schedule N_α and cooling rate α

```
procedure CSA_ID
    set initial cooling rate α ← α_0 and N_α ← N_{α_0};
    set K ← number of CSA runs at fixed α;
    repeat
        for i ← 1 to K do call CSA(α, N_α); end_for;
        increase cooling schedule N_α ← ρN_α; α ← α^{1/ρ};
    until feasible solution has been found and no
        better solution in two successive increases of N_α;
end_procedure
```

b) CSA_{ID}: CSA with iterative deepening

Figure 2: Constrained simulated annealing algorithm (CSA) and its iterative-deepening extension.

the simple algorithm may get stuck in an infeasible region when the objective value is too small or when the Lagrange multipliers and/or constraint violations are too large. In this case, increasing the Lagrange multipliers will further deepen the infeasible region, making it impossible for a local-descent algorithm in the inner loop to escape from this region. One way to address this issue is to scale back λ periodically and to "lower" the barrier in the Lagrangian function in order for local descents in the inner loop to escape from an infeasible region.

The second algorithm is the *constrained simulated annealing* (CSA) [16] algorithm shown in Figure 2a. It looks for SP_d by probabilistic descents of the Lagrangian function in the x subspace in the inner loop and by probabilistic ascents in the λ subspace in the outer loop, with an acceptance probability governed by the Metropolis probability. The following theorem shows that the algorithm converges to a saddle point asymptotically with probability one [16].

Theorem 2. Asymptotic convergence of CSA [16]. *Suppose* $\mathbf{x}' \in \mathcal{N}_d(\mathbf{x})$ *is generated from* $\mathbf{x}$ *using generation probability* $G(\mathbf{x}, \mathbf{x}')$*, where* $G(\mathbf{x}, \mathbf{x}') > 0$ *and* $\sum_{\mathbf{x}' \in \mathcal{N}_d(\mathbf{x})} G(\mathbf{x}, \mathbf{x}') = 1$*. Further, assume a logarithmic cooling schedule*

on T and that $\mathbf{x}'$ is accepted with Metropolis probability $A_T(\mathbf{x}, \mathbf{x}')$:

$$A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} exp\left(-\frac{(L_d(\mathbf{x}')-L_d(\mathbf{x}))^+}{T}\right) & \textit{if } \mathbf{x}' = (x', \lambda) \\ exp\left(-\frac{(L_d(\mathbf{x})-L_d(\mathbf{x}'))^+}{T}\right) & \textit{if } \mathbf{x}' = (x, \lambda'), \end{cases} \tag{7}$$

where $(a)^+ = a$ if $a > 0$, and $(a)^+ = 0$ otherwise for all $a \in \mathcal{R}$. Then the Markov chain modeling CSA converges asymptotically to a CGM_d with probability one.

Theorem 2 extends a similar theorem in SA that proves the asymptotic convergence of SA to a global minimum when solving an unconstrained optimization problem. By looking for SP_d in Lagrangian space, Theorem 2 proves the asymptotic convergence of CSA to a CGM_d when solving a constrained optimization problem.

Theorem 2 is of theoretical interest because its convergence is only achieved as time approaches infinity. In practice, when CSA is run using a finite cooling schedule N_α, it finds a CGM_d with reachability probability $P_R(N_\alpha) < 1$. To increase its success probability, CSA with N_α can be run multiple times from random starting points. Assuming independent runs, a CGM_d can be found in finite average time defined by (3). The minimum of these average times is achieved when CSA is run using a cooling schedule of N_{opt}. (Figure 1b illustrates the existence of N_{opt} for CSAEA.) However, N_{opt} is problem-dependent and cannot be determined easily *a priori*.

To find N_{opt} at run time without using problem-dependent information, we have proposed to use *iterative deepening* [7] in order to determine N_{opt} iteratively. Figure 2b shows the pseudo code of CSA_{ID} (CSA with iterative deepening). The algorithm first runs CSA using a short duration. If the current run fails to find a CGM_d, it doubles the duration CSA is allowed and repeat the run [14]. This step is repeated until a CGM_d is found. Since the total overhead in iterative deepening is dominated by that of the last run, CSA_{ID} has a completion time of the same order of magnitude as that of the last run that succeeds to find a CGM_d. Figure 3 illustrates a case in which the total time incurred by CSA_{ID} is of the same order as the expected overhead at N_{opt}.

Since $P_R(N_{opt}) < 1$ for one run of CSA at N_{opt}, it is possible that CSA fails to find a solution when run with a duration close to N_{opt}. After doubling the cooling schedule, its duration in the next run will overshoot beyond N_{opt}. To reduce the chance of overshooting and to increase the success probability before its duration reaches N_{opt}, we have proposed to run CSA multiple times from random starting points at each duration in CSA_{ID}. For instance, CSA can be run $K = 3$ times at each duration before its duration is doubled (Figure 2b). Our results show that this strategy generally requires twice the average completion time with respect to multiple runs of CSA that use a duration of N_{opt} before it finds a CGM_d [14].

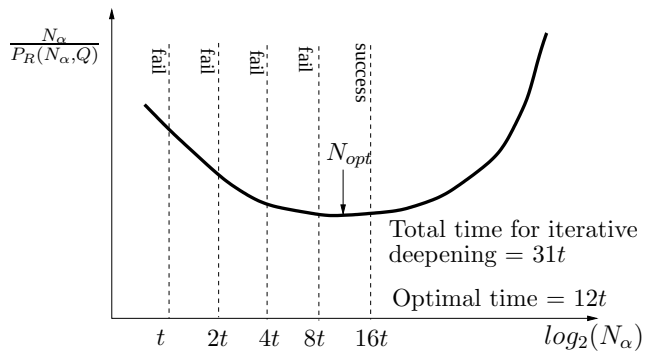


Figure 3: An application of iterative deepening in CSA_{ID}.

2.3 EA for Constrained Optimization

Evolutionary algorithm (EA) is a general optimization algorithm that incorporates aspects of natural selection or survival of the fittest. It maintains a population of alternative candidates (usually generated randomly initially) and probes the search space using genetic operators, such as crossovers and mutations, in order to find better candidates. The original EA was developed for solving unconstrained problems, using a single fitness function to rank candidates. Recently, many variants of EA have been developed for solving constrained NLPs. Most of these methods were based on penalty formulations that use EA to minimize an unconstrained penalty function. Similar to CSA, these methods do not require the differentiability or continuity of functions.

One approach is the *static-penalty formulation* with fixed penalties [2]:

$$F_s(x,\gamma) = f(x) + \sum_{i=1}^{m} \gamma_i |h_i(x)|^\rho, \quad \text{where } \rho > 0. \tag{8}$$

Penalty vector $\gamma = (\gamma_1, \cdots, \gamma_m)^T$ is *fixed* and chosen to be so large that:

$$F_s(x^*,\gamma) < F_s(x,\gamma) \quad \text{for all } x \in X - X_{opt} \text{ and } x^* \in X_{opt}. \tag{9}$$

Based on (9), an unconstrained global minimum of (8) over x is a CGM_d to (4); hence, it suffices to minimize (8) in solving (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.

A major issue of static-penalty methods is 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. If it is too small, then (9) does not hold and feasible solutions cannot be found by minimizing $F_s(x,\gamma)$.

Dynamic-penalty methods [6], on the other hand, address the difficulties of static-penalty methods by increasing penalties gradually in the following

fitness function:

$$F_d(x) = f(x) + (C\ t)^\alpha \sum_{j=1}^{m} |h_i(x)|^\beta, \tag{10}$$

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

Dynamic-penalty methods do not always guarantee convergence to CLM_d or CGM_d. 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 $x \in \mathcal{N}_d(x')$, $0 < |h_1(x')| < |h_1(x)|$, $|h_2(x')| > |h_2(x)| > 0$, 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 (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 other variants of penalty methods, such as annealing penalties, adaptive penalties [12] and self-adapting weights [4]. In addition, problem-dependent genetic operators for handling constraints have been studied. These include methods based on preserving feasibility with specialized 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.

In short, the success of penalty methods rely on choosing suitable penalties. Such penalties are difficult to pick *a priori* in static penalty methods and do not always work in dynamic penalty methods when a single penalty is used. Multiple penalties, one for each constraint as in Lagrangian methods, will be more flexible in guiding a search out of infeasible local minima.

3 A General Framework for finding SP_d

In this section, we propose a general procedural framework that unifies simulated annealing (SA), evolutionary algorithms (EA), and greedy searches in looking for discrete-neighborhood saddle points. Such a framework is important because it combines the best features from various mechanisms in order to arrive at better search algorithms.

Figure 4 depicts a unified problem-independent procedural framework to look for SP_d among a list of candidates maintained. It consists of two loops: the x loop that performs descents of $L_d(x, \lambda)$ in the x subspace, and the λ loop that performs ascents of $L_d(x, \lambda)$ in the λ subspace when there are unsatisfied constraints for any candidate in the list. The procedure stops when no new probes can be generated in both subspaces to improve $L_d(x, \lambda)$. By choosing different options or by tuning the parameters

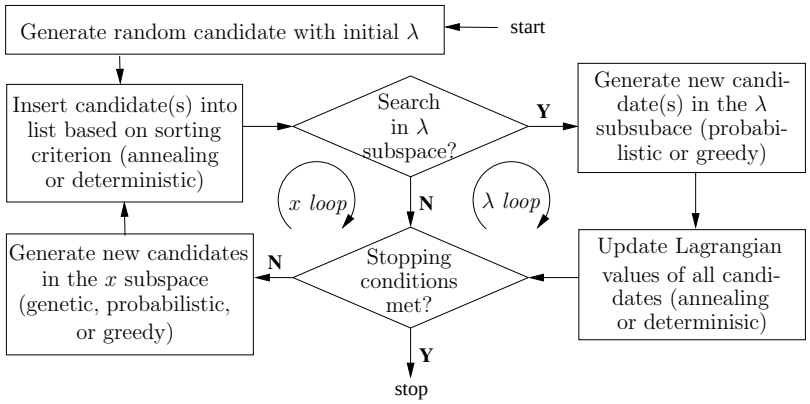


Figure 4: An iterative stochastic procedural framework to look for SP_d.

in the four components of the framework, we can design new constrained optimization algorithms or improve existing algorithms to look for CLM_d.

The general procedure is guaranteed to terminate only at feasible points; otherwise, new probes will be generated in the λ subspace to suppress any violated constraints. Further, if the procedure is able to enumerate all the points in $\mathcal{N}_{dn}(x')$ when trying to determine the direction to move from x' in the x subspace, then the point where the procedure stops must be a discrete-space saddle point, or equivalently, a CLM_d. This is true because the point where the procedure stops is a local minimum in the x subspace of $L_d(x, \lambda)$ and a local maximum in the λ subspace.

Both DLM and CSA in Section 2.2 fit into this procedural framework, each maintaining one candidate at a 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 the constraints are satisfied. On the other hand, CSA generates new probes randomly in one of the x and λ variables, accepts them based on the Metropolis probability (7) 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 this section, we present algorithms that use genetic operators to generate new probes and their iterative-deepening versions:

3.1 Constrained Evolutionary Algorithm (CEA)

CEA was developed based on the general framework in Figure 4 that looks for discrete-neighborhood saddle points. Similar to traditional EA, it organizes a search into a number of generations, each involving a population of candidates in the search space. It searches in the Lagrangian space, using genetic operators 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 5 outlines the pseudo code of the algorithm, and Table 2 shows how CEA fits

```
1.  procedure CEA(P, N_g)
2.      set generation number t ← 0 and λ(t) ← 0;
3.      initialize population P(t);
4.      repeat /* over multiple generations */
5.          eval. L_d(x, λ(t)) for all candidates in P(t);
6.          repeat /* over probes in x subspace */
7.              y ← EA(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) ⊕ c V(h(x), P(t)); /* update λ */
11.         t ← t + 1;
12.     until (t > N_g)
13. end_procedure
```

a) CEA called with population size P and N_g generations

```
1.  procedure CEA_ID
2.      set initial number of generations N_g = N_0;
3.      set K = number of CEA runs at fixed N_g;
3.      repeat /* iterative deepening to find CGM_d */
4.          for i ← 1 to K do call CEA(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) CEA_{ID}: CEA with iterative deepening

Figure 5: Constrained EA and its iterative deepening version.

Table 2: Both CEA and CSAEA fit into the procedural framework for finding discrete-space saddle points.

Framework Component	CEA	CSAEA
Generation of x probes	EA	SA & EA
Generation of λ probes	probabilistic	probabilistic
Insertion of x probes	annealing	annealing
Insertion of λ probes	annealing	annealing

into the general framework to look for discrete-space saddle points. We explain in the following the details of the algorithm.

Lines 2-3 initialize to zero the generation number t and the vector of Lagrange multipliers λ. A starting population of candidates in x can be either randomly generated or user provided.

Line 4 terminates CEA when either the maximum number of allowed generations is exceeded or when no better feasible solution with a given precision has been found in a number of generations. (The stopping condi-

tion is specified later in CEA with iterative deepening.)

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

Lines 6-9 explore the original-variable subspace by evolutionary search. They select from $\mathcal{P}(t)$ candidates to reproduce using genetic operators and insert the new candidates generated into $\mathcal{P}(t)$ according to their fitness.

Line 10 updates λ according to $\mathcal{V}(h, \mathcal{P}(t))$, the vector of maximum constraint violations over population $\mathcal{P}(t)$, where:

$$\mathcal{V}(h(x), \mathcal{P}(t)) = \left(\max_{x \in \mathcal{P}(t)} |h_1(x)|, \ldots, \max_{x \in \mathcal{P}(t)} |h_m(x)| \right)^T . \tag{11}$$

Here, $h_i(x)$ is the ith constraint function, and c is a positive step-wise parameter controlling how fast the Lagrange multipliers change.

There are two considerations in implementing operator $\oplus$ in Line 10: the generation of λ and its acceptance. We generate a new λ' either probabilistically from a uniform distribution in $(-c \times \mathcal{V}/2, c \times \mathcal{V}/2]$, or in a greedy fashion from a uniform distribution in $(0, c \times \mathcal{V}]$. The latter allows only ascents of the fitness function in the λ subspace. The new λ' can be accepted either deterministically when it leads to an improved fitness value, or stochastically according to the Metropolis probability (7). In any case, a Lagrange multiplier will not be changed if its corresponding constraint is satisfied.

Finally, Line 11 updates the generation number before advancing to the next generation.

It should be obvious that the necessary condition for CEA to converge is when $h(x) = 0$ for every candidate x in the population, implying that all the candidates are feasible solutions to the original problem. If any constraint in $h(x)$ is not satisfied by any of the candidates, then λ will continue to evolve in order to suppress the unsatisfied constraint. Note that although we only need the constraints for one (rather than all) candidate to be satisfied, it is difficult to design a penalty-update procedure that identifies a specific candidate to enforce constraint satisfaction. As a result, our penalty-update procedure tries to enforce constraint satisfaction for all the candidates.

One design consideration in applying CEA is in choosing a suitable population size. For the benchmark problems tested, the optimal population size ranges between 20 and 50. Although we have developed a dynamic procedure to select a suitable population size at run time, we do not present the procedure here because CEA performs worse than CSAEA described in the next subsection, which uses a small and fixed population size.

Another design consideration is in determining the proper number of generations to use in each run. It is not necessary to run CEA once for an exceedingly long duration in order for it to find a CGM_d. Similar to CSA, CEA is an SSA that can be run multiple times, each with a short duration, in order to minimize the expected overhead of finding a solution of desired quality. In Section 3.3, we use iterative deepening to determine the optimal number of generations.

```
1.  procedure CSAEA(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_d(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 EA over probes in x subspace */
12.             y ← EA(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: $CSAEA$: Combined CSA and CEA called with population size P and N_g generations.

3.2 Combined Constrained SA and EA (CSAEA)

Based on the general framework in Figure 4, Figure 6 shows a new algorithm CSAEA that integrates CSA in Figure 2a and CEA in Figure 5a. The algorithm uses both random probing in SA and genetic operators in EA to generate new probes in the x subspace. In each generation, CSA is performed on every candidate in the population for q probes, where q is set to $\frac{N_g}{6}$ after experimentation. Our evaluations have also shown that, instead of running CSA on a candidate anew from a random starting point each time the candidate is examined, the best solution point found in the previous run should be used as the starting point of the current run. Next, CSAEA performs an evolutionary search on the candidates by selecting candidates from the population, applying genetic operators on the candidates, and inserting the candidates back into the population. The following explains the actions performed in each step of the procedure.

Line 2 initializes $\mathcal{P}(0)$. Unlike CEA, $\mathbf{x} = (x_1, \cdots, x_n, \lambda_1, \cdots, \lambda_k)^T$ 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 $\mathbf{x}'_\mathbf{j}$ randomly and accept it based on (7). Experimentally, we have determined q to be $\frac{N_g}{6}$. As is discussed earlier, we use the best point of one run as the starting point of the next run.

Lines 11-15 perform an EA search by using genetic operators to generate probes in the x subspace and by sorting all the candidates according to their fitness value L_d after generating a probe. Since each candidate has its own vector of Lagrange multipliers, its Lagrangian value may be biased and cannot be compared directly with the Lagrangian value of another candidate.

As a result, the algorithm first computes the average value of Lagrange multipliers for each constraint over all the candidates in $\mathcal{P}(t)$ and then calculates L_d for each candidate using the average Lagrange multipliers.

Similar to CEA, we must choose a suitable number of candidates in the population of CSAEA and the duration of each run in order for the algorithm to perform well. We address these two issues in CSAEA_{ID} next.

3.3 CEA and CSAEA with an Optimal Schedule

In this section we present a method to determine the optimal number of generations in one run of CEA and CSAEA in order to find a CGM_d. The method is based on the use of iterative deepening [7] that determines an upper bound on N_g, the number of generations in one run of CEA or CSAEA, in order to minimize the expected total overhead in (3).

The number of probes expended in one run of CEA or CSAEA 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_d. From (3), the expected total number of probes using multiple runs and fixed P is:

$$\frac{N}{P_R(N)} = \frac{N_g P}{P_R(N_g P)} = P\frac{N_g}{\hat{P}_R(N_g)}. \tag{12}$$

We are interested to find $N_{g_{opt}}$, the optimal number of generations that minimizes (12). For the same reason as is indicated in Section 1, $N_{g_{opt}}$ exists and is bounded. Figure 1b illustrates the existence of such an absolute minimum when CSAEA with $P = 3$ was applied to solve G1 [12].

To estimate $N_{g_{opt}}$, we apply iterative deepening similar to that in CSA_{ID} (Section 2.2). Assume that CEA_{ID} in Figure 5b uses a set of geometrically increasing N_g to find a CGM_d:

$$N_{g_i} = \rho^i N_0, \qquad i = 0, 1, \ldots, \tag{13}$$

where N_0 is the (small) initial number of generations. Under each N_g, CEA is run a maximum of K times but stops immediately when the following three conditions hold: a) a feasible solution has been found, b) when no better solution has been found in two successive generations, and c) the number of iterations has been increased geometrically at least five times. These conditions are established in order 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 CEA under N_{g_i} generations, $B_{opt}(f')$ be the expected total number of probes taken by CEA with $N_{g_{opt}}$ to find a solution of quality f', and $\mathcal{B}_{ID}(f')$ be the expected total number of probes taken by CEA_{ID} in Figure 5b to find a solution of quality f' starting from N_0 generations. According to (12),

$$B_{opt}(f') = P\frac{N_{g_{opt}}}{\hat{P}_R(N_{g_{opt}})} \tag{14}$$

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

Theorem 3. CEA_{ID} and CSAEA_{ID} are optimal and can achieve $\mathcal{B}_{ID}(f') = O(B_{opt}(f'))$ when

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

Proof. Based on the schedule in (13) used by CEA_{ID}, define q in such a way that:

$$N_{g_{q-1}} < N_{g_{opt}} \leq N_{g_q}, \quad \text{where } N_{g_q} = \rho N_{g_{q-1}}. \tag{15}$$

If $\hat{P}_R(x)$ is monotonically non-decreasing in $(0, +\infty)$, we have:

$$\hat{P}_R(N_{g_{q-1}}) \leq \hat{P}_R(N_{g_{opt}}) \leq \hat{P}_R(N_{g_q}). \tag{16}$$

Let $\hat{P}_{NR}(N_g)$ be the probability that CEA does not succeed in K runs with N_g generations in each run. Since all runs are independent, we have:

$$\hat{P}_{NR}(N_g) = (1 - \hat{P}_R(N_g))^K. \tag{17}$$

From (16) and (17), we have:

$$\hat{P}_{NR}(N_{g_{opt}}) \geq \hat{P}_{NR}(N_{g_q}) \geq \hat{P}_{NR}(N_{g_{q+1}}) > \cdots \tag{18}$$

Let E_{B_i} be the expected total number of probes expended in Lines 3-6 in Figure 5b when CEA is run a maximum of K times at N_{g_i} probes each but stops immediately if it succeeds in any one of these runs. Then:

$$E_{B_i} = \sum_{j=0}^{K-1} [1 - \hat{P}_R(N_{g_i})]^j P \cdot N_{g_i} \leq K \cdot P \cdot N_{g_i} = O(N_{g_i}), \ i = 0, 1, 2, ... \tag{19}$$

When CEA in Line 4 of Figure 5b is called with $N_{g_0}, \ldots, N_{g_q}$ generations, its success probability is larger than the success probability of CEA called with N_{g_q} generations alone. Hence,

$$\mathcal{B}_{ID}(f') \leq [1 - \hat{P}_{NR}(N_{g_q})] E_{B_q} + \hat{P}_{NR}(N_{g_q}) \sum_{i=q+1}^{\infty} E_{B_i} \prod_{j=q+1}^{i-1} \hat{P}_{NR}(N_{g_j}). \tag{20}$$

From (18), (20) can be reduced to:

$$\begin{aligned} \mathcal{B}_{ID}(f') \ &\leq \ [1 - \hat{P}_{NR}(N_{g_{opt}})] E_{B_q} + \hat{P}_{NR}(N_{g_{opt}}) \sum_{i=q+1}^{\infty} E_{B_i} \prod_{j=q+1}^{i-1} \hat{P}_{NR}(N_{g_j}), \\ &\leq \ E_{B_q} + \hat{P}_{NR}(N_{g_{opt}}) \sum_{i=q+1}^{\infty} E_{B_i} \prod_{j=q+1}^{i-1} \hat{P}_{NR}(N_{g_j}). \end{aligned} \tag{21}$$

From (18), (19) and (21), we have:

$$\begin{aligned}\mathcal{B}_{ID}(f') &\leq \sum_{i=0}^{\infty}(\hat{P}_{NR}(N_{g_{opt}}))^i E_{B_{q+i}} = \sum_{i=0}^{\infty}(\hat{P}_{NR}(N_{g_{opt}}))^i O(N_{g_{q+i}}) \\ &= O(\rho^q N_{g_0}) \sum_{i=0}^{\infty}(\hat{P}_{NR}(N_{g_{opt}})\rho)^i. \qquad (22)\end{aligned}$$

For the search to converge in finite average time,

$$\hat{P}_{NR}(N_{g_{opt}})\rho = (1 - \hat{P}_R(N_{g_{opt}}))^K \rho < 1, \qquad (23)$$

which is condition (b) of the theorem. This proves the theorem:

$$\mathcal{B}_{ID}(f') = O(\rho^q N_{g_0}) = O(\rho N_{g_{opt}}) = O(B_{opt}(f')). \qquad (24)$$

■

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 $K > 2.4$. Hence, we have used $K = 3$ in our experiments. Since CEA 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 CEA with $N_{g_{opt}}$.

The only remaining issue is to choose a suitable population size P in each generation. In CEA_{ID}, we have 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 because the new CEA_{ID} performs worse than CSAEA_{ID}. In CSAEA_{ID}, $P = 3$ was found to perform well experimentally. Although the optimal P may be slightly different, the corresponding expected overhead to find a CGM_d differs very little from that when a constant P is used.

4 Experimental Results

We present in this section our experimental results on applying CSA_{ID}, CEA_{ID} and CSAEA_{ID} to evaluate some nonlinear benchmarks. We first determine the best combination of parameters to use in generating probes and in organizing candidates.

4.1 Implementation Details

An important element of our Lagrangian method is the frequency of updating λ. As in CSA [16], we set the ratio of generating trial points in x and λ subspaces to be $20n$ to m, where n is the number of variables and m is the number of constraints. This means that x is updated more often than λ.

In generating trial points in the x subspace, we use a dynamically controlled neighborhood size in the SA part [16] based on the 1:1 ratio rule [3],

whereas in the EA part, we use the seven operators in Genocop III [11] and L_d as our fitness function. We also use the default parameters of CSA [16] in the SA part and those of Genocop III [11] in the EA part.

The following rule generates trial point λ' in the λ subspace:

$$\lambda'_j = \lambda_j + r_1\ \phi_j, \qquad \text{where } j = 1, \cdots, m. \tag{25}$$

Here, r_1 is a random number in $[-1/2, +1/2]$ when λ is generated probabilistically, and is randomly generated in $[0, 1]$ when probes in λ are generated in a greedy fashion.

We adjust ϕ in (25) adaptively according to the degree of constraint violations:

$$\phi = w \otimes \mathcal{V}(h(x), \mathcal{P}(t)), \tag{26}$$

where $\otimes$ is the vector-product operator, and $\mathcal{V}$ is the vector of maximum violations defined in (11). When $\mathcal{V}_i(h(x), \mathcal{P}(x)) = 0$, λ_i does not need to be updated; hence, $\phi_i = 0$. In contrast, when $\mathcal{V}_i(h(x), \mathcal{P}(x)) > 0$, we adjust ϕ_i by modifying w_i according to how fast $\mathcal{V}_i(h(x), \mathcal{P}(x))$ is changing:

$$w_i = \begin{cases} \eta_0\ w_i & \text{if } \mathcal{V}_i(h(x), \mathcal{P}(x)) > \tau_0 T \\ \eta_1\ w_i & \text{if } \mathcal{V}_i(h(x), \mathcal{P}(x)) < \tau_1 T, \end{cases} \tag{27}$$

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 $\mathcal{V}_i(h(x), \mathcal{P}(x))$ is reduced too quickly (second case in (27)), $\mathcal{V}_i(h(x), \mathcal{P}(x))$ is over-weighted, leading to possibly poor objectives or difficulty in satisfying other under-weighted constraints. Hence, we reduce λ_i's neighborhood. In contrast, if $\mathcal{V}_i(h(x), \mathcal{P}(x))$ is reduced too slowly (first case in (27)), 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 with T.

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, the initial and the 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 [12, 8] and Floudas and Pardalos' Problems [5]. We have also tested our algorithms on MacMINLP [9], a collection of mixed-integer benchmarks.

In generating a discrete constrained NLP, we discretize continuous variables in the original continuous constrained NLP into discrete variables as follows. Continuous variable x_i in the range $[l_i, u_i]$ is forced to take values from the following set, where l_i and u_i are the lower and upper bounds of x_i, respectively, and $s = 1.0 \times 10^7$:

$$A_i = \begin{cases} \left(a_i + \frac{b_i - a_i}{s}\, j \mid j = 0, 1, \cdots, s\right) & \text{if } b_i - a_i < 1 \\ \left(a_i + \frac{1}{s}\, j \mid j = 0, 1, \cdots, \lfloor (b_i - a_i)s \rfloor\right) & \text{if } b_i - a_i \geq 1. \end{cases} \tag{28}$$

Table 3: Timing results on evaluating various combinations of strategies in $\mathrm{CSA_{ID}}$, $\mathrm{CEA_{ID}}$ and $\mathrm{CSAEA_{ID}}$ 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 ten runs and were collected on a Pentinum III 500-MHz computer with Solaris 7. '−' means that no solution with the desired quality can be found. The best result in each column is boxed.

Probe Generation Strategy		Insertion	Solution 1% off CGM_d			Solution 10% off CGM_d		
λ subspace	x subspace	Strategy	CSA_{ID}	CEA_{ID}	$CSAEA_{ID}$	CSA_{ID}	CEA_{ID}	$CSAEA_{ID}$
prob	prob	anneal.	6.91	23.99	4.89	1.35	–	1.03
prob	prob	determ.	9.02	–	6.93	1.35	2.78	1.03
prob	determ.	anneal.	–	18.76	–	89.21	2.40	–
prob	determ.	determ.	–	16.73	–	–	2.18	–
greedy	prob	anneal.	7.02	–	7.75	1.36	–	0.90
greedy	prob	determ.	7.02	–	7.75	1.36	–	0.90
greedy	determ.	anneal.	–	25.50	–	82.24	1.90	–
greedy	determ.	determ.	–	25.50	–	82.24	1.90	–

Table 3 shows the evaluation results on various strategies in $\mathrm{CSA_{ID}}$, $\mathrm{CEA_{ID}}$, and $\mathrm{CSAEA_{ID}}$ on a discretized version of G2 [12, 8]. We show the average time of ten runs for each combination in order to reach two solution quality levels (1% or 10% worse than CGM_d, assuming the value of CGM_d is known). Evaluation results on other benchmark problems are similar.

Our results show that $\mathrm{CEA_{ID}}$ usually takes longer than $\mathrm{CSA_{ID}}$ or $\mathrm{CSAEA_{ID}}$ to find a solution of similar quality. Further, $\mathrm{CSA_{ID}}$ and $\mathrm{CSAEA_{ID}}$ have 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 into their population by annealing rules lead 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 [12, 8], discretized according to (28). 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 [16].

Table 4: Results on CSA_{ID}, CEA_{ID} and CSAEA_{ID} in finding the best-known solution f^* for ten 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 ten 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		EAs		CSA_{ID}		CEA_{ID}		CSAEA_{ID}				
ID	Best 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.

Table 4 compares the performance of CSA_{ID}, CEA_{ID}, and CSAEA_{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 value of f^* was not provided to the algorithms before they were run.) The first four columns show the problem IDs, the corresponding known f^*, the best solutions obtained by EAs, and the specific constraint handling techniques used to generate the solutions. Since all CSA_{ID}, CEA_{ID} and CSAEA_{ID} can find a CGM_d in *all 10 runs*, we compare their performance with respect to $\mathcal{T}$, the average total time of multiple runs until a CGM_d is found. The fifth and sixth columns show, respectively, the average time and the number of $L_d(x, \lambda)$ function evaluations CSA_{ID} took to find f^*. The next two columns show the performance of CEA_{ID} with respect to P_{opt}, the optimal population size found by enumeration, and the average time for finding f^*. These results show that CEA_{ID} is not competitive as compared to CSA_{ID}, even when P_{opt} is used. The results on including additional steps in CEA_{ID} to select a suitable P at run time are worse and are not shown. Finally, the last five columns show the performance of CSAEA_{ID}. The first three present the average time and number of $L_d(x, \lambda)$ evaluations under constant P, whereas the last two show the average time using P_{opt} found by enumeration. These results show little improvements in using P_{opt}. Further, CSAEA_{ID} has between 9% and 38% in improvement in $\mathcal{B}_{ID}(f^*)$, when compared to those of CSA_{ID}, for the 10 problems except for G4 and G10.

Comparing CEA_{ID} and CSAEA_{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. In contrast, both CEA_{ID} and CSAEA_{ID} can find f^* for all these problems without problem-dependent strategies. We are not able to report the timing results of EA because the results in the literature are the best among many runs with extensive tuning.

Table 5 shows the results on selected discretized Floudas and Pardalos' NLP benchmarks [5] with more than ten variables. The first three columns show the problem IDs, the known f^*, and the number of variables (n_v) of each problem. The last two columns compare $\mathcal{B}_{ID}(f^*)$ of CSA_{ID} and CSAEA_{ID} with fixed $P = 3$. They show that CSAEA_{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 EA maintains more diversity in candidates by keeping a population, thereby allowing competition among the candidates and leading SA to explore more promising regions.

Finally, we compare our methods with MINLP_BB [9], an MINLP solver using the branch and bound method, in solving selected MINLP benchmark problems in MacMINLP with no more than 100 variables [13]. Since the benchmarks in these experiments were not discretized, CSA_{ID} and CSAEA_{ID} were used as heuristics in sampling the continuous subspace.

Table 6 summarizes the experimental results. The first six columns show, respectively, the problem ID, the type of objective function (linear, quadratic, or nonlinear), the number of variables (n_v), the number of integer variables (n_i), the type of constraint functions (linear, quadratic, or nonlin-

Table 5: Results on CSA_{ID} and $CSAEA_{ID}$ 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_d, we consider a CGM_d found when the solution quality is within 10% of the true CGM_d. All CPU times in seconds were averaged over 10 runs and were collected on a Pentium-III 500-MHz computer with Solaris 7.

Problem			CSA_{ID}	$CSAEA_{ID}$
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.

ear), and the number of constraints (n_c). The next six columns show the solution obtained and the CPU time expended, respectively, by MINLP_BB, CSA_{ID} and CSAEA_{ID}, where MINLP_BB was evaluated at the NEOS server (*http://www-neos.mcs.anl.gov/neos*).

CSA_{ID} and CSAEA_{ID} perform much better in terms of solution quality than MINLP_BB for the problems they can solve. For example, CSA_{ID} found a solution of objective value 1.856 and CSAEA_{ID} found a solution of objective value 1.877 for problem SPRING, but MINLP_BB failed to find a feasible solution. For problem TRIMLON4, MINLP_BB found a feasible solution of objective value 11.1, whereas CSA_{ID} and CSAEA_{ID} found solutions of objective value 8.8 and 8.5, respectively.

However, the running times of CSA_{ID} and CSAEA_{ID} are not competitive with those of MINLP_BB in solving large problems. This happens because CSA_{ID} and CSAEA_{ID} are sampling algorithms, whereas MINLP_BB utilizes derivative information of functions during its search. An advantage of sampling algorithms is that they do not rely on derivatives and can be applied to solve constrained NLPs whose functions are not differentiable or not in closed form.

5 Conclusions

In this paper we have presented new algorithms to look for discrete-neighborhood saddle points in the Lagrangian space of constrained optimization problems. Because the discrete-neighborhood extended saddle-

Table 6: Results comparing CSA_{ID}, $CSAEA_{ID}$ and MINLP_BB [9] in solving selected MINLPs from MacMINLP [10]. All times for CSA_{ID} and $CSAEA_{ID}$ are in seconds on a Pentium-III 500-MHz computer running Solaris 7. All times for MINLP_BB are in seconds on the NEOS server at http://www-neos.mcs.anl.gov/neos/. '-' means that no feasible solution can be found. Boxed numbers represent the best solutions among the three methods if they have different solutions.

Problem						MINLP_BB		CSA_{ID}		$CSAEA_{ID}$	
ID	objective	n_v	n_i	constraints	n_c	solution	T	solution	T	solution	T
BATCH	nonlinear	46	24	nonlinear	73	285507	0.58	287463	96.78	$\boxed{280374}$	41.82
C-SCHED1	nonlinear	73	60	linear	16	-30639.3	0.42	-	-	-	-
Ex12.6.3	linear	92	8	nonlinear	57	19.6	23	19.6	910.30	19.6	451.29
Ex12.6.4	linear	88	8	nonlinear	57	$\boxed{8.6}$	70	9.0	140.85	$\boxed{8.6}$	270.88
Ex12.6.5	linear	130	8	nonlinear	76	15.1	4	$\boxed{10.6}$	73.0	$\boxed{10.6}$	35.78
Ex12.6.6	linear	180	8	nonlinear	97	16.3	18	-	-	-	-
OPTPRLOC	quadratic	30	25	quadratic	29	$\boxed{-8.06414}$	0.78	-2.502	38.59	-2.502	21.95
SPRING	nonlinear	17	11	nonlinear	10	-	-	$\boxed{1.856}$	29.34	1.877	14.16
SYNTHES1	nonlinear	6	3	nonlinear	6	6.00976	0.01	$\boxed{2.3756}$	0.76	$\boxed{2.3756}$	0.60
SYNTHES2	nonlinear	11	5	nonlinear	14	73.0353	0.04	$\boxed{3.285}$	18.90	$\boxed{3.285}$	12.58
SYNTHES3	nonlinear	17	8	nonlinear	19	68.0097	0.10	32.659	204.53	$\boxed{32.657}$	370.67
TRIMLON2	linear	8	8	nonlinear	12	5.3	0.11	5.3	0.08	5.3	0.10
TRIMLON4	linear	24	24	nonlinear	26	11.1	6.70	8.8	23.19	$\boxed{8.5}$	289.21
TRIMLON5	linear	35	35	nonlinear	33	12.5	10.60	$\boxed{10.9}$	999.52	11.0	321.02
TRIMLON6	linear	48	48	nonlinear	41	27	13.71	47.5	665.26	$\boxed{19.2}$	432.02
TRIMLOSS2	linear	37	31	nonlinear	24	5.3	2.59	5.3	0.43	5.3	0.42
WIND-FAC	linear	15	3	nonlinear	14	$\boxed{0.254487}$	0.04	1.627	259.86	1.627	186.92

point condition only requires some Lagrange multipliers larger than a critical threshold to be found, the search of such saddle points can be implemented efficiently by an iterative search. Our main contribution in this paper is in developing an effective search that combines a) probes generated randomly and using genetic operators in the original variable space for performing descents of the Lagrangian function, b) random probing in the Lagrange-multiplier space for performing ascents, c) acceptance of probes generated using the Metropolis probability, and d) a schedule that minimizes the number of iterations (with the same order of magnitude as the optimal schedule). Our results also show that existing evolutionary approaches for solving constrained optimization problems, which only look for constrained solutions in the original variable space, can be improved significantly by looking for saddle points in the Lagrangian space.

One of the limitations of our approach is that the saddle-point condition only works in discrete neighborhoods. As a result, continuous variables in continuous or mixed-integer problems must be discretized before the condition can be applied. Furthermore, probes generated randomly or using genetic operators do not likely follow descent directions, and differentiation is more effective for finding gradient directions in differentiable functions. These limitations can be overcome by the extended saddle-point condition in mixed space we have developed recently [15]. Based on the new condition, descent directions in the continuous subspace can be found by differentiation, whereas descent directions in the discrete subspace can be found by enumeration. Results on solving mixed-integer optimization problems will be reported in a future paper.

Acknowledgments – This material is based upon work supported by the National Aeronautics and Space Administration under Grant NCC2-1230 and the National Science Foundation under Grant ITR 03-12084. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Aeronautics and Space Administration and the National Science Foundation.

References

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

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

[3] A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the simulated anneal-

ing algorithm. *ACM Trans. on Mathematical Software*, 13(3):262–280, 1987.

[4] A. E. Eiben and Z. Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. *Proc. of the 3rd IEEE Conf. on Evolutionary Computation*, pages 258–261, 1996.

[5] 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.

[6] J. Joines and C. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems. In *Proc. of the First IEEE Int'l Conf. on Evolutionary Computation*, pages 579–584, 1994.

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

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

[9] S. Leyffer. Mixed integer nonlinear programming solver. *http://www-unix.mcs.anl.gov/˜leyffer/solvers.html*, 2002.

[10] S. Leyffer. MacMINLP: AMPL collection of MINLP problems. *http://www-unix.mcs.anl.gov/˜leyffer/MacMINLP/*, 2003.

[11] Z. Michalewicz and G. Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. *Proc. of IEEE Int'l Conf. on Evolutionary Computation*, 2:647–651, 1995.

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

[13] R. Vanderbei. Nonlinear optimization models. *http://www.sor.princeton.edu/˜rvdb/ampl/nlmodels/*, 2003.

[14] 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*, pages 425–439, September 2000.

[15] B. W. Wah and Y. X. Chen. Partitioning of temporal planning problems in mixed space using the theory of extended saddle points. In *Proc. IEEE Int'l Conf. on Tools with Artificial Intelligence*, November 2003.

[16] 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.

[17] 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.

[18] 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, May 2001.