IMPROVING CONSTRAINED NONLINEAR SEARCH ALGORITHMS THROUGH CONSTRAINT RELAXATION

BY

HONGHAI ZHANG

B.S., University of Science and Technology of China, 1998

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2001

Urbana, Illinois

Abstract

In this thesis we study constraint relaxations of various nonlinear programming (NLP) algorithms in order to improve their performance. For both stochastic and deterministic algorithms, we study the relationship between the expected time to find a feasible solution and the constraint relaxation level, build an exponential model based on this relationship, and develop a constraint relaxation schedule in such a way that the total time spent to find a feasible solution for all the relaxation levels is of the same order of magnitude as the time spent for finding a solution of similar quality using the last relaxation level alone.

When the objective and constraint functions are stochastic, we define new criteria of constraint satisfaction and similar constraint relaxation schedules. Similar to the case when functions are deterministic, we build an exponential model between the expected time to find a feasible solution and the associated constraint relaxation level. We develop an anytime constraint relaxation schedule in such a way that the total time spent to solve a problem for all constraint relaxation levels is of the same order of magnitude as the time spent for finding a feasible solution using the last relaxation level alone. Finally, we study the asymptotic behavior of our new algorithms and prove their asymptotic convergence, based on the theory of asymptotic convergence in simulated annealing and constrained simulated annealing.

To my wife and my parents.

Acknowledgments

First, I would like to thank my research advisor, Professor Benjamin Wah, for his guidance, encouragement and valuable ideas in this work. He introduced me to this area and inspired me all the time.

I would like to thank all the member in my research group for their ideas and comments in this work. Special thanks go to Dr. Tao Wang for his research on constrained simulated annealing (CSA), Dr. Zhe Wu for his research on Lagrange multipliers for discrete constrained optimization, and Mr. Yixin Chen for his research on anytime CSA.

Finally and most specially, I like to thank my wife for her loving me, supporting me, and keeping my life colorful.

This research was supported by National Aeronautics and Space Administration Grant NAG 2-1230.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A large variety of applications in engineering, decision science and operations research can be formulated as constrained nonlinear programming problems (NLPs), whose variables can be continuous or discrete but bounded and whose objective and constraint functions are nonlinear.

1.1 Definition of NLP Problems

Generally, nonlinear programming problems can be classified into two classes: optimization with deterministic functions and optimization with stochastic functions (or optimization under uncertainty). These are defined as follows.

1.1.1 Optimization with deterministic functions

A traditional constrained nonlinear programming problem can be formulated as follows:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g(x) \le 0 \\ & h(x) = 0, \end{aligned} \tag{1.1}$$

where $x = (x_1, x_2, \ldots, x_n)$ is a vector of (continuous or discrete) variables, $f(x)$ is a lower-bounded objective function, $g(x) = [g_1(x), \ldots, g_k(x)]^T$ is a set of k inequality constraints, $h(x) = [h_1(x), \ldots, h_m(x)]^T$ is a set of m equality constraints, and all the variables in x are

Table 1.1: Definitions of the 12 classes of constrained NLPs

Constrained NLP Class	Variable Type	Function Type
C1	continuous	continuous and differentiable
C2	continuous	continuous and non-differentiable
C3	continuous	discontinuous
C4	discrete	continuous and differentiable
C5	discrete	continuous and non-differentiable
C6	discrete	discontinuous
C7	mixed-integer	continuous and differentiable
C8	mixed-integer	continuous and non-differentiable
C9	mixed-integer	discontinuous
C10	continuous	stochastic objective and constraints
C11	discrete	stochastic objective and constraints
C12	mixed-integer	stochastic objective and constraints

lower- and upper-bounded. It is also assumed that functions $f,\ g,\ h$ can be evaluated exactly given x.

Depending on whether x is discrete or continuous and whether $f,\ g$ and h are differentiable, continuous or non-continuous, NLPs can be classified into nine classes, as showed in the first nine rows of Table 1.1.

1.1.2 Optimization with stochastic functions

In general, objective and constraint functions are deterministic. However, in some applications, it is too expensive or even impossible to evaluate them exactly. For example, there may be noise in evaluating a function or the evaluation is is done by Monte Carlo simulations and can only achieve a certain degree of precision in limited time. Generally, such optimization problems can be formulated as follows:

$$\begin{aligned} \text{minimize} \quad & E(f(x)) \\ \text{subject to} \quad & E(g(x)) \leq 0 \\ & E(h(x)) = 0, \end{aligned} \tag{1.2}$$

where $E(f(x))$ is the expected value of objective $f(x)$ at point x, $x = (x_1, x_2, \cdots, x_n)^T,\ g = (g_1, g_2, \cdots, g_k)^T,\ h = (h_1, h_2, \cdots, h_m)^T$. Note that at any given point x, only $f,\ g,\ h$ can be observed, but we have no way of evaluating the exact value of $E(f(x)),\ E(g(x)),\ E(h(x))$.

We can observe a function multiple times in order to get an approximate value of its expected value. The more observations we take at a given point x, the more accurate is the estimated expected value, although more observations at a given point will incur more cost.

When constraint functions can be evaluated accurately, we assume constraint $h_i(x) = 0$ is satisfied when $|h_i(x)| < \tau$, where τ is a small positive constant. However, if $h_i(x)$ is stochastic, then we cannot get the exact value of $E(h_i(x))$ unless we take infinitely many samples. Hence, using the same criteria $h_i(x) = 0$ in the stochastic situation may require too many samples at each point. Instead, we propose the following criterion in constraint satisfaction.

To simplify the discussion, we assume both g and h are one-dimensional vectors. Further, we assume that the random noise in each observation of a function has a normal distribution with mean 0 and a fixed standard deviation. For an equality constraint $h(x) = 0$, suppose we have observed N samples $h^i(x),\ i = 1, \cdots, N$ at point x, where $\bar{h}$ is the sample mean and $\hat{\sigma}_{\bar{h}} = \hat{\sigma}_h/\sqrt{N}$ is the standard deviation of $\bar{h}$ and $\hat{\sigma}_h$ is the standard deviation of each sample. We can construct a $(1-\alpha)100\%$ confidence interval (LL, UU) of $\bar{h}$, where

$$\begin{aligned} LL &= \bar{h} - t_{1-\alpha/2,N-1}\hat{\sigma}_{\bar{h}} \\ UU &= \bar{h} + t_{1-\alpha/2,N-1}\hat{\sigma}_{\bar{h}}, \end{aligned} \tag{1.3}$$

and $t_{1-\alpha/2,N-1}$ is the $1-\alpha/2$ quantile of the t distribution with $N-1$ degree of freedom. If the confidence interval (LL, UU) does not cover 0, we will reject the hypothesis test H_0: $E(h(x)) = 0$. Otherwise we, accept it and assume that the constraint $h(x) = 0$ is satisfied.

Similarly, for an inequality constraint $g(x) \leq 0$, suppose we observe N samples $g^i(x),\ i = 1, \cdots, N$, at point x, where $\bar{g}$ is the sample mean, $\hat{\sigma}_g^2$ is the sample variance, and $\hat{\sigma}_{\bar{g}} = \hat{\sigma}_g/\sqrt{N}$

is the standard deviation of the mean. We can construct a $(1-\alpha)100\%$ confidence interval (LL, UU) where:

$$\begin{aligned} LL &= -\infty \\ UU &= \bar{g} + t_{1-\alpha,N-1}\hat{\sigma}_{\bar{g}}. \end{aligned} \tag{1.4}$$

If the confidence interval (LL, UU) does not cover 0, we reject the hypothesis test H_0: $E(g(x)) \leq 0$. Otherwise we accept it and assume that the constraint $E(g(x)) \leq 0$ is satisfied. When confidence level α is fixed and N is large enough, $\hat{\sigma}_{\bar{h}}$ and $\hat{\sigma}_{\bar{g}}$, the estimated standard deviations of the means $\bar{h}$ and $\bar{g}$, will decide the width of the confidence interval and the accuracy in estimating $E(h)$ and $E(g)$. In other words, if $\hat{\sigma}_{\bar{h}}$ (resp. $\hat{\sigma}_{\bar{g}}$) is large, then the confidence interval is wide, and it is very difficult to estimate $E(h)$ (resp. $E(g)$) using sample average $\bar{h}$ (resp. $\bar{g}$). When $\hat{\sigma}_{\bar{h}}$ (resp. $\hat{\sigma}_{\bar{g}}$) is small enough, then the estimation of $E(h(x))$ (resp. E(g(x))) using $\bar{h}$ (resp. $\bar{g}$) is accurate, and the solution can be regarded as that to the original problem (1.2). When $\hat{\sigma}_{\bar{h}}$ (resp. $\hat{\sigma}_{\bar{g}}$) becomes 0, the confidence interval for h (resp. g) becomes a single point $\bar{h}$ (resp. an interval $(0, \bar{g})$). In this case $\bar{h}$ (resp. $\bar{g}$) becomes the exact estimate of $E(h)$ (resp. $E(g)$).

Ideally, we hope that the width of a confidence interval can go to 0 when time goes to infinity. In practise, we require that $\hat{\sigma}_{\bar{h}} = \sigma_h^f$ and $\hat{\sigma}_{\bar{g}} = \sigma_g^f$ when an algorithm finishes, where σ_h^f and σ_g^f are predefined small constants.

In summary, we consider a constraint is satisfied with confidence $(1-\alpha)$ iff the confidence interval for the constraint covers 0 and the width of the confidence interval is less than or equal to a prescribed value σ^f.

Depending on whether $x = (x_1, x_2, \ldots, x_n)$ is discrete or continuous, stochastic NLPs can be classified into three classes, as showed in the last three lines of Table 1.1.

1.2 Basic Definitions and Concepts

Since we are plan to apply constraint relaxation to both continuous and discrete NLP solvers, we introduce some basic concepts in this section for both continuous and discrete problems. We start with traditional definitions in continuous space.

1.2.1 Concepts in continuous space

Definition 1.1 The neighborhood of a point x, $\mathcal{N}_{dn}(x)$, is defined to be $\{x' : |x' - x| \leq \epsilon\}$ where ϵ is a small constant.

Definition 1.2 [106] Point x in continuous space is a constrained local minimum, CLM_{cn}, if and only if there exists a small $\varepsilon > 0$ such that for all x' that satisfy $|x' - x| < \varepsilon$ and that x' is also a feasible point, $f(x') \geq f(x)$ holds true.

Definition 1.3 [106] Point x in continuous space is a constrained global minimum, CGM_{cn}, if and only if a) x is a feasible point, b) for any other feasible point x', $f(x') \geq f(x)$.

Definition 1.4 [106] The *Lagrangian function* of the general continuous constrained optimization problem defined in (1.1) (assuming only equality constraints) is $L(x, \lambda) = f(x) + \lambda^T h(x)$, where $\lambda \in R^m$ is a vector of Lagrange multipliers.

Definition 1.5 [106] A *saddle-point* (x^*, λ^*) of $L(x, \lambda)$ in continuous space, SP_{cn}, is defined as a point such that, for all (x^*, λ) and (x, λ^*) that satisfies $|x^* - x| < \epsilon$ and $|\lambda^* - \lambda| < \epsilon$,

$$L(x^*, \lambda) \leq L(x^*, \lambda^*) \leq L(x, \lambda^*) \tag{1.5}$$

where ϵ is a small positive constant.

The following are two general results on Lagrangian formulations in continuous space [106].

Theorem 1.1 (*First-Order Necessary Conditions for Continuous Problems*) [106] Let x be a local extremum point of $f(x)$ subject to $h(x) = 0$. Further, assume that $x = (x_1, \ldots, x_n)$ is a regular point of these constraints. Then there exists $\lambda \in R^m$ such that

$$\nabla_x f(x) + \lambda^T \nabla_x h(x) = 0. \tag{1.6}$$

Based on Definition 1.4, the necessary conditions can be expressed as follows:

$$\begin{aligned} \nabla_x L(x, \lambda) &= 0 \qquad (1.7) \\ \nabla_\lambda L(x, \lambda) &= 0. \end{aligned}$$

Theorem 1.2 [173] The relationships among solutions sets **A**, **B**, and **C** are as follows: (a) $\mathbf{B} \subset \mathbf{A}$, (b) $\mathbf{C} \subset \mathbf{A}$, and $\mathbf{B} \neq \mathbf{C}$, where **A** is the set of local minima subject to constraints, **B** is the set of solutions satisfying the first-order necessary and second-order sufficient conditions, and **C** is the set of saddle points.

1.2.2 Concepts in discrete space

Definition 1.6 [16] The *neighborhood* of point x in discrete search space X, $\mathcal{N}_{dn}(x)$, is a user-defined set of points $x' \in X$ such that $x' \in \mathcal{N}_{dn}(x) \Longleftrightarrow x \in \mathcal{N}_{dn}(x')$ (i.e. x and x' can reach each other in one step), and that it is possible to reach every other x'' from any x in a finite number of steps through neighboring points.

Definition 1.7 Point $x \in X$ is called a *feasible point* iff all the constraints are satisfied at x; i.e. $g_i(x) \geq 0$ and $h_j(x) = 0$ for all i and j. A *feasible region* is a set of feasible points.

Definition 1.8 [174] Point $x \in X$ is called a *constrained local minimum in discrete space* (CLM_{dn}) iff a) x is a feasible point, and b) for every feasible point $x' \in \mathcal{N}_{dn}(x), f(x') \geq f(x)$.

Definition 1.9 [174] Point $x \in X$ is called a *constrained global minimum in discrete space (CGM_{dn})* iff a) x is a feasible point, b) for every feasible point $x' \in X, f(x') \geq f(x)$. According to our definition, a CGM_{dn} is also a CLM_{dn}.

In the following, we summarize the theory of Lagrange multipliers for discrete constrained optimization. For simplicity, we assume only equality constraints in (1.1), knowing that inequality constraint $g_i(x) \leq 0$ can be transformed into an equivalent equality constraint $\max(g_i(x), 0) = 0$.

Definition 1.10 A *generalized augmented Lagrangian function* [161] of (1.1) with only equality constraints is defined as:

$$L(x, \lambda) = f(x) + \lambda^T H(h(x)) + \frac{1}{2} h^T(x) h(x), \tag{1.8}$$

where H is a continuous transformation function that satisfies: a) $H(y)$ is nonnegative or non-positive, and b) $H(y) = 0$ iff $y = 0$, where $\lambda = (\lambda_1, \ldots, \lambda_m)$ is a vector of Lagrange multipliers. The quadratic term in (1.8) makes the search of the Lagrangian function more stable.

Definition 1.11 [174] We define a *saddle point* (x^*, λ^*) in discrete space, SP_{dn}, with the following property:

$$L(x^*, \lambda) \leq L(x^*, \lambda^*) \leq L(x, \lambda^*), \tag{1.9}$$

for all $x \in \mathcal{N}_{dn}(x^*)$ and all possible $\lambda's$.

The concept of saddle points is very important in discrete problems because, starting from them, we can derive first-order necessary and sufficient conditions for CLM_{dn} that lead to global minimization procedures. This is stated formally in the following theorem [165].

Theorem 1.3 [165] *First-order necessary and sufficient conditions for CLM_{dn}.* In discrete variable space, a point is a constrained local minimum (CLM_{dn}) if and only if it is a saddle point.

1.2.3 Concepts in mixed-integer space

In mixed integer space, the problem formulation is the same as that in continuous or discrete space, except that some variables take integer values while others take continuous values. To differentiate the two kinds of variables, we re-write the formulation as follows:

$$\begin{aligned} \text{minimize} \quad & f(x, y) & & \\ \text{subject to} \quad & h(x, y) = 0 & \qquad & x = (x_1, x_2, \dots, x_{n_1}) \\ & g(x, y) \leq 0 & \qquad & y = (y_1, y_2, \dots, y_{n_2}), \end{aligned} \tag{1.10}$$

where $f(x, y)$ is the objective function, $g(x, y) = [g_1(x, y), \dots, g_k(x, y)]^T$ is a k-component vector representing inequality constraints, $h(x, y) = [h_1(x, y), \dots, h_m(x, y)]^T$ is a m-component vector representing equality constraints, x is a vector of continuous variables, and y is a vector of discrete variables.

Definition 1.12 Given point (x, y) in mixed-integer space, where x represents the continuous subspace and y the discrete subspace, the *neighborhood* of (x, y) is defined as :

$$\mathcal{N}_{mn}(x, y) = \mathcal{N}_{cn}(x) \ \cup \ \mathcal{N}_{dn}(y), \tag{1.11}$$

where $\mathcal{N}_{cn}$ and $\mathcal{N}_{dn}$ are, respectively, the sets of neighboring points of x in continuous subspace defined in Definition 1.1 and the sets of neighboring points of y in discrete subspace defined in Definition 1.6.

The definitions of mixed-integer-space saddle points and constrained local minima are the same as those in discrete or continuous space, except that $\mathcal{N}_{mn}(x, y)$ is used instead.

1.3 Constraint Relaxation

For small and continuous problems with differentiable functions, we often can find feasible solutions quickly. However, for large problems or those with non-differentiable functions, it is often difficult to find feasible solutions. We have found that, for large hard-to-satisfy optimization problems, if we relax their constraints, they are often easier to solve. We define the relaxation of a constraint as follows.

Definition 1.13 The *relaxed form of an equality constraint* $h(x) = 0$ is:

$$-r \le h(x) \le r, \tag{1.12}$$

where r is the relaxation level. Similarly, the *relaxed form of an inequality constraint* $g(x) \le 0$, is:

$$g(x) \le r. \tag{1.13}$$

It often requires much less time to solve a relaxed problem than the original problem. We illustrate such phenomenon in this section with respect to deterministic and stochastic optimization problems.

1.3.1 Constraint relaxation for NLPs with deterministic functions

We have investigated a variety of nonlinear programming algorithms, including Lancelot [52], KNITRO [44], SNOPT [71] and CSA [164]. Figure 1.1 plots the relationship between relaxation level r and the expected time to find a feasible solution by each algorithm. For CSA, we have used the optimal cooling schedule (that is discussed later) for each relaxation level.

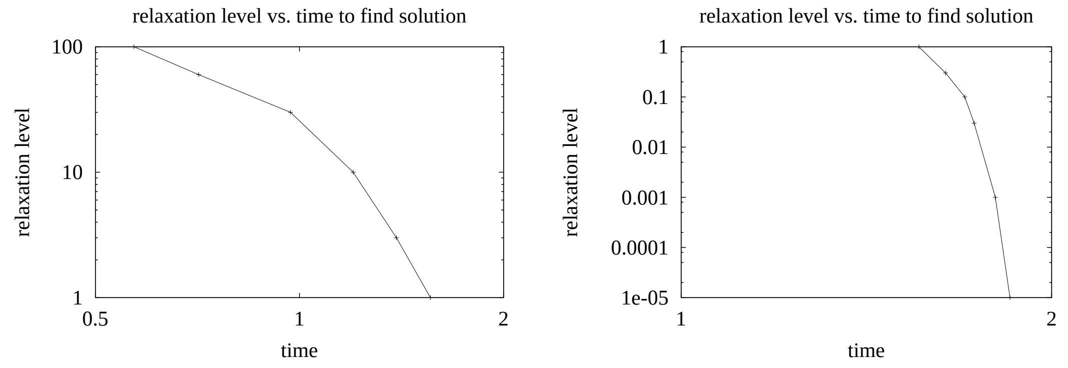
The graphs show that in some of the algorithms (SNOPT and CSA), $\log r$ and $\log T$ are linearly related, whereas in other algorithms (KNITRO and LANCELOT), $\log r$ and $\log T$ are piecewise linearly related. In general, we conclude that, for most of the NLP algorithms, $\log r$ and $\log T$ are linearly related.

An issue to be studied is how to choose an appropriate relaxation level r. If we choose a high relaxation level, then it will be easy to find a feasible solution for the relaxed problem. However this feasible solution may have large violations that are usually close to the relaxation level r. On the other hand, if we choose a tight relaxation level r, then an algorithm may fail to find a feasible solution within finite time.

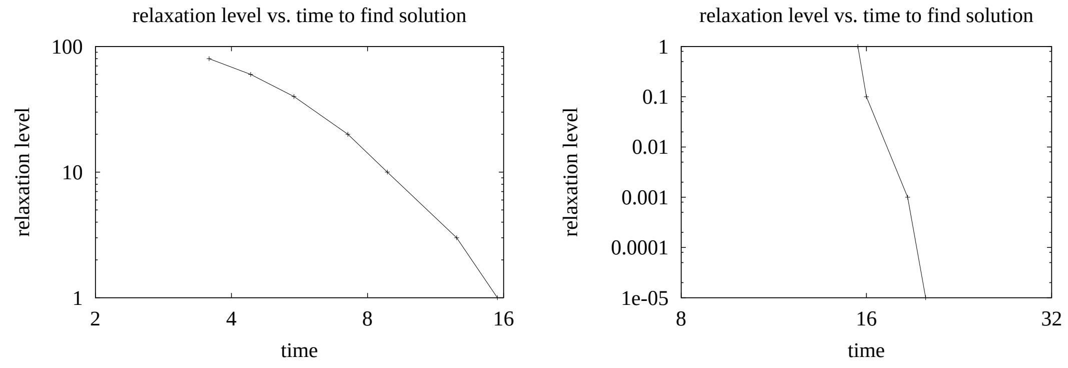
Definition 1.14 *The minimal feasible relaxation level (r_{MFRL})* for a NLP is the minimal relaxation level r for which the relaxed problem has a feasible solution. If the original NLP has feasible solution, then r_{MFRL} is 0.

If we know r_{MFRL} for an NLP, then it will be best to solve the NLP with relaxation level r_{MFRL}. Choosing a larger relaxation level will likely lead to solutions with larger violations, whereas choosing a smaller relaxation level will make the problem unsolvable. The issue is that we usually don't know r_{MFRL} for a given problem instance.

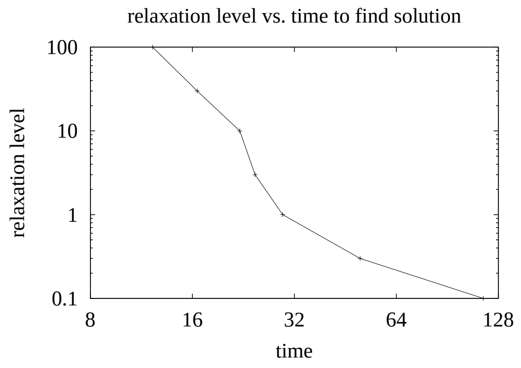
Definition 1.15 A *schedule of relaxation* is defined as a sequence of decreasing relaxation levels, $(r_1, r_2, \cdots, r_n)$. For each relaxation level r_i, the expected time to find a feasible solution for the relaxed problem instance with relaxation level r_i is T_i.

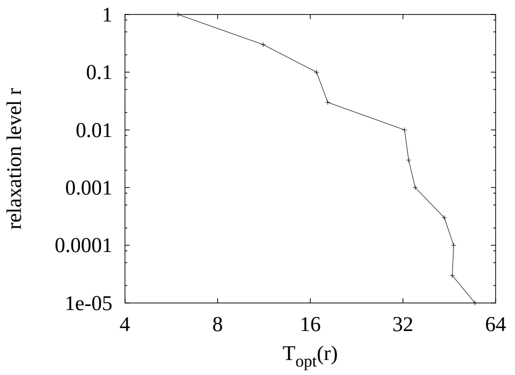


a) Problem *trimloss* using *KNITRO*



b) Problem *trimloss* using *LANCELOT*





c) Problem *britgas* using *SNOPT*

d) Problem *launch* using *CSA*

Figure 1.1: Log-log relationship between relaxation levels and expected times to find a feasible solution using Algorithm KNITRO, LANCELOT, SNOPT, and CSA.

Our goal is to verify statistically the relationship we have demonstrated in Figure 1.1 and design a schedule $(r_1, r_2, \cdots, r_n)$ such that the total time $\sum_{i=1}^{n} T_i$ will be of the same order of magnitude as T_n. In the illustrative example shown in Figure 1.2, we firstly apply CSA to solve CUTE problem LAUNCH. For each of the following relaxation levels: 1, 0.1, 0.01, 1E-3, 1E-4, 1E-5, we apply CSA with different cooling schedule until we find the optimal schedule. We then run CSA 100 times using the optimal cooling schedule and compute the optimal expected time as the average time of the 100 runs divided by the probability of finding feasible solution among the 100 runs. Second, we design a relaxation schedule $(r_1, r_2, \cdots, r_n)$, apply CSA_{ID} for each of the relaxation levels in the schedule, and record the accumulated time. We then compare the optimal expected time using CSA with the time of our proposed anytime algorithm ($CSA_{ID-ATCR}$). Our results demonstrate that the time spent using our optimal anytime schedule is of the same order of magnitude of the optimal expected time.

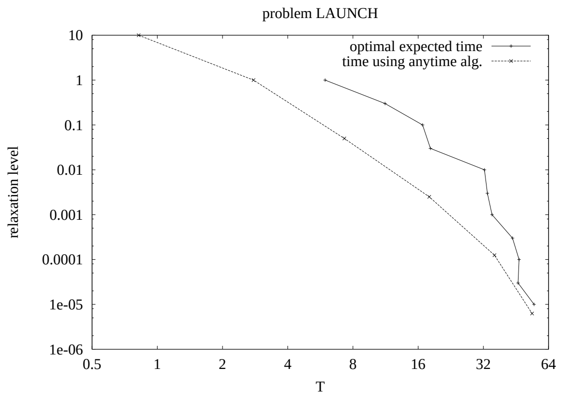


Figure 1.2: Comparison of CPU time using $CSA_{ID-ATCR}$ and the expected time using CSA with an optimal cooling schedule in solving CUTE problem LAUNCH in mixed-integer version

1.3.2 Constraint relaxation for NLPs with stochastic functions

When the constraints and objective are stochastic functions, a constraint $h(x) = 0$ is considered satisfied iff its confidence interval $(\bar{h}(x) - t_{1-\alpha/2,N-1}\sigma, \bar{h}(x) - t_{1-\alpha/2,N-1}\sigma)$ covers 0 and $\sigma \leq \sigma_f$. Hence the relaxed form of a stochastic constraint can be defined using σ_f as its relaxation level.

Given a desired σ_f, one way to achieve $\sigma < \sigma_f$ when the algorithm terminates is that we draw enough samples in each iteration such that $\sigma \leq \sigma_f$. However, as observed in Chapter

5, the entire run will require less samples if we draw a smaller number of samples in earlier iterations and sufficient samples such that $\sigma \leq \sigma_f$ when the algorithm finishes. Based on this strategy, we measure the expected number of samples for different σ_f using CSA_{ID} [160] to solve an optimization problem, whose objective is the expectation of a truncated 10-dimensional Rastrigin function plus a random noise with normal distribution:

$$\begin{aligned} \text{minimize} \quad & E(f(x)) = E(F(10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)), 190) + \epsilon(x)) \\ \text{subject to} \quad & E(\sum_{i=1}^{n}|x_i - 3.8| + \epsilon(x)) \leq 0.1 \text{ for } n = 10 \end{aligned} \tag{1.14}$$

where

$$F(y, f') = \begin{cases} f', & \text{if } y \leq f' \\ y, & \text{otherwise} \end{cases}.$$

Figure 1.3 plots the relationship between the expected number of samples drawn and relaxation level σ_f. Once again, we observe an log-log relationship between the expected number of samples drawn and σ_f.

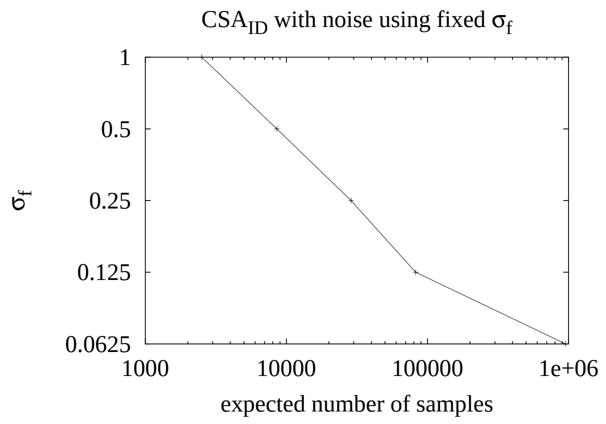


Figure 1.3: Relationship between the expected number of samples and the final confidence width σ_f using CSA_{ID} in solving (1.14)

Based on the above observation, our goal is to statistically verify the relationship between the expected time and σ_f and design a schedule of decreasing σ_f such that the total expected number of samples is of the same order of magnitude as the expected number of samples required using the last σ_f alone.

Figure 1.4 illustrates an example of applying an anytime schedule on CSA_{ID} to solve (1.14). As a reference, the lower curve is the expected number of samples using CSA with an optimal cooling schedule for each level of σ_f. The upper curve shows a schedule of σ_f generated by our $CSA_{ID-ATCR}$ algorithm and its corresponding number of samples for each σ_f. Our results demonstrate that, by choosing an appropriate schedule of σ_f, we can generate improved solution when more samples are drawn, and the total number of samples required for the whole schedule of σ_f is of the same order of magnitude as the number of samples required for the last σ_f alone.

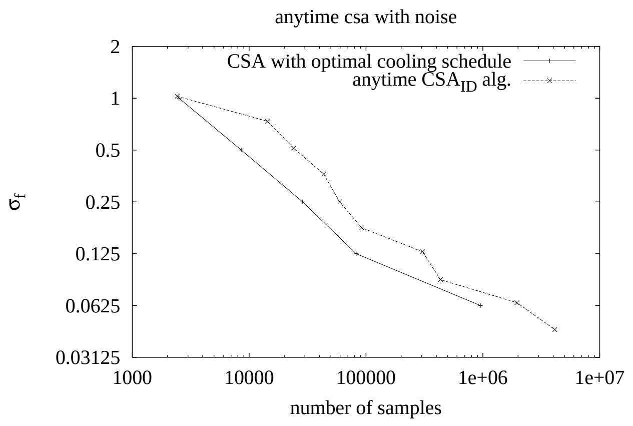


Figure 1.4: Comparison of the expected number of samples drawn using CSA with an optimal cooling schedule and the total number of samples using a schedule generated by $CSA_{ID-ATCR}$ in solving (1.14).

1.4 Outline of the Thesis

This thesis has six chapters. In the next chapter, we describe the previous work in the area. In Chapter 3, we develop the strategy of scheduling relaxation levels in CSA. In Chapter 4, we design an algorithm for solving mixed-integer NLPs and apply our strategy of relaxation scheduling on the algorithm. In Chapter 5, we develop relaxation schedules to solve constrained NLPs with stochastic functions. Finally we conclude the thesis in Chapter 6.

Chapter 2

Previous Work

In this chapter, we summarize previous work in the literature on the different classes of problems formulated in Chapter 1. We classify existing algorithms based on the class of NLPs it can solve. Some algorithms can solve more than one class of problems. For example, any algorithm that can solve mixed-integer problems can also solve continuous and discrete problems, although using them to solve continuous or purely discrete problems may not be as efficient as those algorithms that are specially designed for continuous or discrete problems. We find that for most of those algorithms, especially for those transformation-based algorithms, relaxation is very helpful in solving difficult problems.

2.1 Methods for Discrete Optimization

Methods in this class assume that all variables are discrete and functions are not differentiable. Since any general problem in this class is NP-hard, existing approaches aim at finding constrained local minima that satisfy all the constraints. They can be broadly classified into four categories: direct solution methods, transformations into constrained 0-1 NLP problems, Lagrangian relaxation, and transformations into unconstrained problems using penalty formulations.

2.1.1 Direct methods for solving discrete constrained NLPs

Direct solution methods for solving discrete constrained NLPs without any transformation on their objective and constraint functions can be classified into two categories. One major method is based on rejecting, discarding [27, 28, 126] or repairing [97] in order to avoid infeasible points. Constraint relaxation is especially helpful for this kind of approach because after relaxation, the feasible region is enlarged, and it is much easier to find a feasible points.

The other approach is based on enumeration or randomized search techniques [92]. Enumerative algorithms using branch-and-bound [104, 168] decompose a search space and estimate the bound for each in order to eliminate infeasible subspaces. In these algorithms, branching variables are chosen according to a priori ordering, and functions may be approximated by piecewise linear functions.

Constraint relaxation has two effects on this kind of methods. One effect is that it can enlarge the feasible region and thus increase the number of feasible points. This makes it easier to find a feasible point and helps reduce the search time. The other effect is that it may increase the number of branches that need to be explored that may not contain feasible points. In contrast, without constraint relaxation, tight constraints may help eliminate infeasible regions early in a search. Hence, whether constraint relaxation can help improve a search is problem dependent. If constraint relaxation leads to search regions with new feasible points, then it is helpful to use constraint relaxation. However, if the increased search regions that need to be explored do not contain feasible points and cannot be eliminated by other means, then constraint relaxation is not helpful. Generally, when lower bounds of a subproblem cannot be estimated accurately, constraint relaxation should not be used.

2.1.2 Transformations into constrained 0-1 NLPs

One major approach is to rewrite a discrete constrained NLP into a constrained nonlinear 0-1 programming problem before solving it. This rewriting process is simple because an integer variable can naturally be expressed as the summation of several binary bits or 0-1 variables. Existing nonlinear 0-1 integer programming algorithms can be classified into three categories [85].

First, a nonlinear problem can be linearized by replacing each distinct product of variables by a new 0-1 variable and by adding some new constraints [169, 75, 76]. Second, algebraic methods [83, 134] express an objective function as a polynomial function of its variables and their complements. Last, cutting-plane methods [78, 79] reduce a constrained nonlinear 0-1 problem into a generalized covering problem.

Constraint relaxation is often helpful for these methods because, after constraint relaxation, constraints become easy to satisfy, leading to easier-to-solve transformed 0-1 problems.

2.1.3 Lagrangian relaxation

A class of algorithms is called *Lagrangian relaxation* [69, 70, 65, 146, 33] that are often used in linear integer programming. Lagrangian relaxation transforms an *linear* integer minimization problem:

$$\begin{aligned} z = \text{minimize} \quad & Cx \\ \text{subject to} \quad & Gx \le b \qquad \text{where } x \text{ is an integer vector of variables} \\ & x \ge 0 \qquad\quad \text{and } C \text{ and } G \text{ are constant matrices} \end{aligned} \tag{2.1}$$

into the following form:

$$\begin{aligned} L(\lambda) = \text{minimize} \quad & (Cx + \lambda^T(b - Gx)) \\ \text{subject to} \quad & x \ge 0. \end{aligned} \tag{2.2}$$

The new relaxed problem is easy to solve for any given vector λ. A general relationship between the solution to the original minimization problem and the solution to the relaxed problem can be deduced based on Lagrangian Duality theory [152] Some research [29] addresses nonlinear optimization problems using this method. Here, Lagrangian relaxation should not be confused with our constraint relaxation.

After constraint relaxation, feasible regions are enlarged. Hence, it is is easier to find a feasible point. In general, constraint relaxation is helpful for this kind of methods.

2.1.4 Penalty formulations and methods

Penalty Formulations. This approach transforms (1.1) into an unconstrained problem, consisting of a sum of its objective and its constraints weighted by penalties, before solving the penalty function by unconstrained methods. It often uses an exact penalty function, since no derivative is required in discrete NLPs. The new penalty formulation is as follows:

$$cost(x) = f(x) + \sum_{i=1}^{n} w_i |h_i(x)|, \tag{2.3}$$

where $f(x)$ is the objective function, and w_i is the i^{th} weight coefficient. The problem is how to determine the $w_i's$.

A simple solution is to set w_i to be a large constant positive value, resulting in a *static-penalty formulation* [38, 106]. If the w_i's are large enough, a local minimum of $cost(x)$ is most likely a CLM_{dn}, and a global minimum of $cost(x)$ is a CGM_{dn}. However, if the w_i's are too large, the search space is very rugged. Consequently, it is difficult to locate feasible solutions using local-search methods because these methods have difficulty in escaping from deep local minima after getting there and in moving from one feasible region to another when feasible regions are disconnected. On the other hand, if the w_i's are too small, then local minima or global minima of $cost(x)$ may not be feasible solutions to the original constrained problem.

Dynamic-penalty methods overcome the difficulties in static-penalty methods by gradually increasing penalty parameter w_i's. Dynamic-penalty methods transform (1.1) into a sequence of unconstrained subproblems with increasing penalties, and use the solution of a previous subproblem as a starting point for the next subproblem. Dynamic-penalty methods have asymptotic convergence if each unconstrained subproblem in the sequence can be solved optimally [38, 106]. However, it is difficult to achieve optimality in each subproblem in practice, given finite time to solve each subproblem. This results in suboptimal solutions when the solutions in at least one subproblem are not optimal. Moreover, the solutions to the first few subproblems may not be related to the final goal of finding CLM_{dn} or CGM_{dn} since penalties in those subproblems are not large enough. Approximations to the process of sacrificing global optimality of solutions have been developed [98, 107].

A variety of constraint handling techniques have been developed based on dynamic-penalty formulations in [91, 96, 109, 125, 110, 73, 27, 142, 122, 141, 34]. Most of these techniques require domain-specific knowledge. The main difficulties of these heuristics are in finding feasible regions, maintaining feasibility for nonlinear constraints, or getting stuck easily in local minima [112, 109].

In general, methods based on penalty formulations have no guarantee to find CLM_{dn}. Consider a problem with only one constraint $h_1(x)$ and an objective $f(x)$. If a penalty-based algorithm starts from x^* and $|h_1(x^*)| = min_{x \in \mathcal{N}_{dn}(x^*) \cup \{x^*\}}\{|h_1(x)|\} > 0$ and $f(x^*) = min_{x \in \mathcal{N}_{dn}(x^*) \cup \{x^*\}}\{f(x)\}$ where $\mathcal{N}_{dn}(x^*)$ is the discrete neighborhood of x^*, then no matter how large the penalty becomes, no feasible solution can be found.

Next, we review methods for solving problems based on penalty formulations. These methods can be classified into local search, global search and stochastic global search.

Local Search Based on Penalty Formulations. Local search methods based on penalty formulations use local probes or perturbations to generate trial points and advance their search trajectories by either accepting or rejecting the trial points. Typical methods in this class include greedy search and hill climbing [26, 106]. These methods usually define their stopping conditions as to be closely related to their algorithmic steps. However, they have problems in finding feasible solutions and get stuck easily in infeasible local minima when the penalty parameters in (2.3) are not chosen properly. For this reason, local search methods are often used as a component in global search techniques to find high-quality solutions.

Global Search Based on Penalty Formulations. Global search methods based on penalty formulations normally use various techniques to escape from local minima in a search space. Typical methods in this class include tabu search [72, 74, 32], break-out strategies [117], guided local search (GLS) [157], random walk [145, 144], multi-start [140, 139, 87, 149], heuristic repair methods [51], learning-based approaches [42, 30, 31, 40], and Bayesian methods [114, 158, 153].

Generally speaking, global search methods based on penalty formulations can at most find constrained local minima (CLM_{dn}) without guarantee. However, as mentioned earlier, it is difficult to select suitable penalties, and most current methods use heuristics to select penalties.

Stochastic Global Optimization Based on Penalty Formulations. Based on penalty transformations for solving discrete constrained problems, stochastic global optimization methods can find $CGM_{dn}$when given sufficient time that approaches infinity. However, it is difficult to achieve global optimality in practice, when given finite time, because a search may commit too quickly to an infeasible region or a region with only CLM_{dn}.

Simulated annealing (SA) [99] and genetic algorithms (GA) [88] are two well-known stochastic global optimization algorithms for solving unconstrained NLPs.

The basic idea of SA is to not only accept better trial points, but also accept inferior trial points probabilistically. It also uses a parameter T called temperature to control the acceptance probability of inferior trial points.

One of the most important properties of SA is that it can achieve asymptotic convergence to an unconstrained global optimum with probability one, when it uses a logarithmic cooling schedule to decrease temperature T [67]. This important property makes SA popular, and it has been applied successfully to solve unconstrained optimization problems [128, 101, 53, 21].

However, when applied to solve constrained NLPs by using penalty formulations, SA only ensures global convergence to the optimal solution of each unconstrained sub-problems. That solution may be an infeasible point for the constrained optimization problems when penalties are not chosen properly. That is, the success of SA in constrained optimization depends heavily on the proper choice of penalties. Moreover, SA requires a very slow cooling schedule in order to converge to an optimal solution with high probabilities.

Genetic algorithm (GA) [112, 108, 77, 62, 109, 136, 118, 123, 86, 58, 35] is a stochastic global optimization algorithm . It maintains a population of candidate points in each generation. In each generation, it uses some genetic operators, such as crossover and mutation, to generate new candidate points. All the old and new candidate points are then ranked according to a fitness function, which is the objective function in case of unconstrained optimization problems. It then selects candidates to comprise a new generation from the candidate pools in the last generation. After a sufficient number of generations, it is expected to converge to the best candidates.

Similar to SA, GA requires a good choice of penalties in a penalty formulation in order for a search to converge to a CGM_{dn} of the original constrained problem. Otherwise, the search may end up with only CLM_{dn} or even infeasible solutions. Some variants of penalty formulations have been applied in GAs to handle constraints, including methods with multi-level static penalties [91], generation-based dynamic penalties [96], annealing penalties [109], and adaptive penalties [34, 82, 125]. However, those methods still have difficulties in choosing appropriate penalties for different kinds of constrained NLPs.

Besides SA and GA, some other stochastic global optimization strategies have also been applied to solve discrete constrained NLPs using penalty formulations, although they are not as popular as SA and GA. These include random search [177, 148], adaptive search [124, 43], controlled random search (CRS) [22, 18] and improved hit-and-run (IHR) [178, 177] methods. Most of these random search methods do not work well for general problems because: a) they depend heavily on choosing good penalty parameters in order to find feasible solutions; and b) the trial point generator, usually based on some random distribution function, has no implications of the final goal of finding CGM_{dn} or CLM_{dn}. Therefore, these random search methods are usually not very efficient.

Constraint relaxation in penalty methods Constraint relaxation is often very useful in this kind of methods. Usually, a penalty function is weighted by unsatisfied constraints. Before constraints are relaxed, the penalty function could be very complex because many constraints are not satisfied. After relaxation, only a few of the highly unsatisfied constraints are still in the penalty function; hence, the search direction will be focused on decreasing those highly unsatisfied constraints. During the process of tightening the constraints, more constraints will be considered and satisfied.

2.2 Methods for Continuous Constrained NLPs

Continuous constrained NLPs defined in (1.1) have continuous variables. Usually, solvers in this class will require the differentiability of functions in order to better exploit the continuity of functions. These methods can be further classified into three categories: direct search methods, penalty formulations and Lagrangian formulations.

2.2.1 Direct solutions for solving continuous constrained NLPs

Direct solution methods solve (1.1) directly without any transformation on the objective and constraint functions. Direct solution methods can be classified into local search, global search, and global optimization methods.

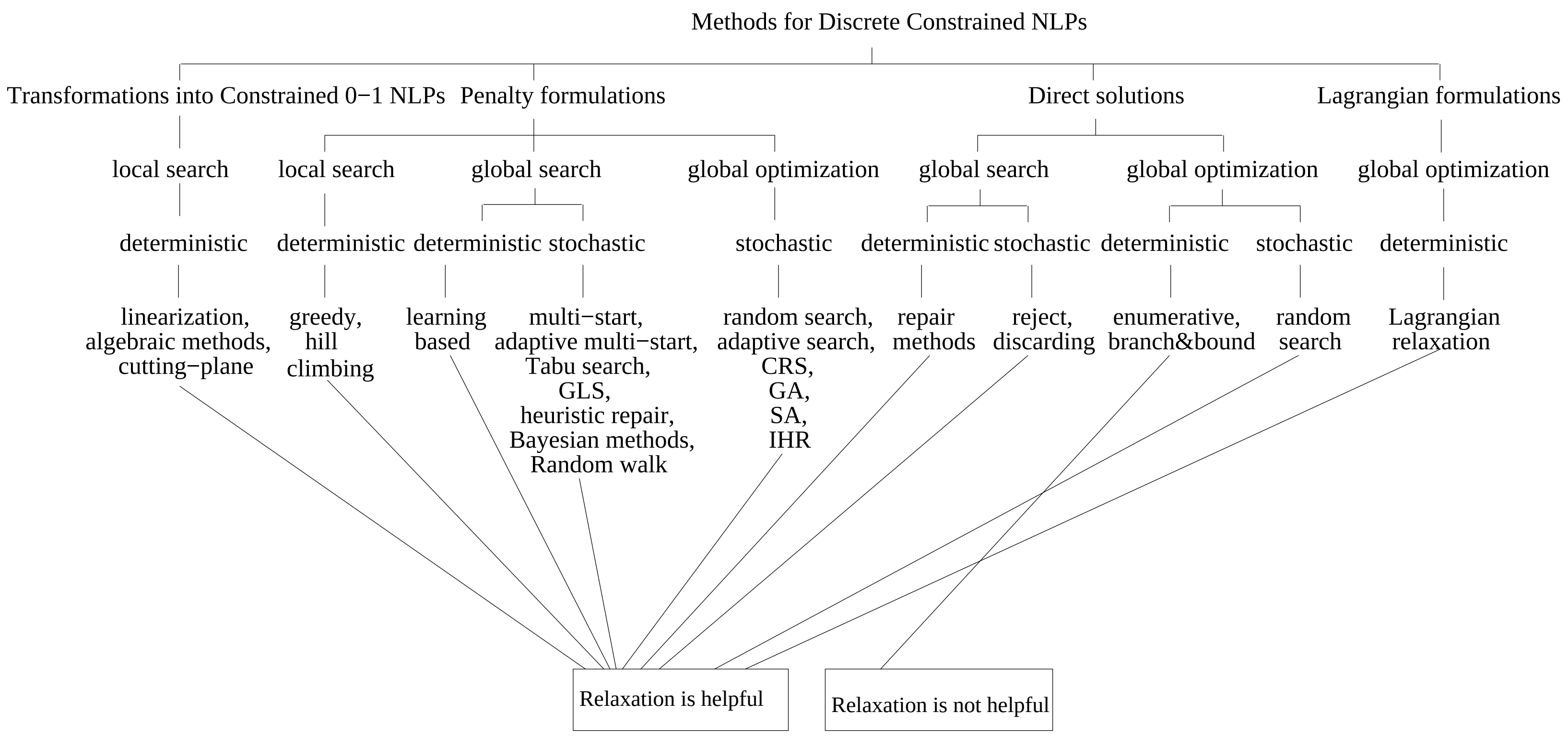


Figure 2.1: Classification of methods for solving discrete constrained NLPs

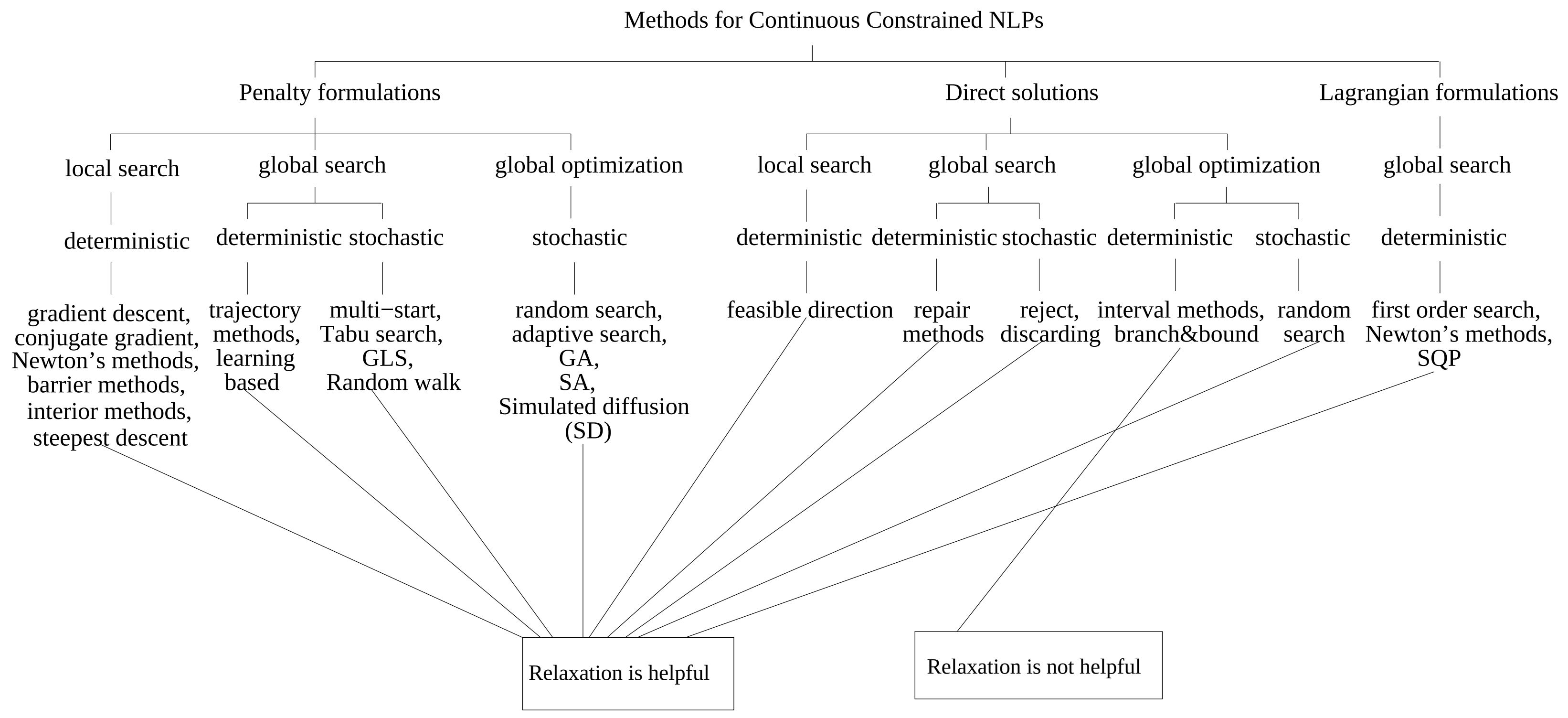

Methods for Continuous Constrained NLPs
Penalty formulations
Direct solutions
Lagrangian formulations
local search
global search
global optimization
local search
global search
global optimization
global search
deterministic
deterministic
stochastic
stochastic
deterministic
deterministic
stochastic
deterministic
stochastic
deterministic
gradient descent, conjugate gradient, Newton's methods, barrier methods, interior methods, steepest descent
trajectory methods, learning based
multi−start, Tabu search, GLS, Random walk
random search, adaptive search, GA, SA, Simulated diffusion (SD)
feasible direction
repair methods
reject, discarding
interval methods, branch&bound
random search
first order search, Newton's methods, SQP
Relaxation is helpful
Relaxation is not helpful

Figure 2.2: Classification of methods for solving continuous constrained NLPs

Local Search methods, such as feasible-direction methods [97, 109], require a search to start from a feasible point. Each search step will remain in a feasible region, while trying to improve the objective function at the same time.

Global Search methods use techniques to escape from local minima. Typical global search methods include rejecting methods, discarding methods [131, 126], repair methods [97, 120] and preserving feasibility [110, 73]. However, these techniques are often of limited use and have difficulties in handling nonlinear constraints.

Global Optimization methods use either deterministic techniques, like interval methods, or stochastic techniques, like randomized search, to look for a CGM_{cn}. Deterministic methods [93, 36] are often too expensive to apply for large problems except for linear problems and problems that can be linearized effectively because they often require some kind of enumerations of a search space. A typical deterministic approach divides a search space recursively into subregions, estimates the range of objective and constraints for each subregion, excludes infeasible subregions, keeps feasible ones, and further divides undecided ones. Examples include branch-and-bound and interval methods [84, 115]. These methods are computationally expensive and have difficulties in handling highly nonlinear constraints when it is not easy to estimate the range of nonlinear functions. When constraints are highly nonlinear, lower bounds cannot be found accurately, and the search space cannot be reduced effectively.

On the other hand, stochastic approaches, such as random search and adaptive random search [92], probe a search space using random sampling. Although optimality can be achieved when given enough time, the probability of hitting a feasible point by random sampling is very small, especially for large nonlinear constrained problems. Moreover, random sampling in a search space has little bearing to constraint satisfaction. Hence, stochastic approaches are generally not popular in solving continuous constrained NLPs.

As we have discussed in Section 2.1.1, except those enumerative methods like branch and bound, constraint relaxation is very helpful in direct search methods, since it can enlarge possible feasible regions that makes it much easier to find a feasible point.

2.2.2 Penalty formulations

By combining the objective function and constraints into a penalty function, penalty methods solve (1.1) by solving a sequence of unconstrained problems. However, unlike what have been done in discrete problems, penalty formulations in continuous problems often choose to add a continuous differentiable penalty term for each constraint. For example, when only equality constraints are present in (1.1), the penalty function can be written as [119]:

$$f(x) + \frac{1}{2\mu} \sum h_i^2(x), \tag{2.4}$$

where μ is called a penalty parameter. A search then minimizes the unconstrained function, for a series of decreasing values of μ, until the solution to the constrained optimization problem is found. Depending on the strategies used to handle local minima, these penalty methods can further be classified into local search, global search and global optimization.

Local Search Methods Based on Penalty Formulations. These methods attempt to solve each unconstrained continuous minimization problems using only local information, such as gradients. Typical local search methods include gradient descent, Newton descent [106, 121, 133], conjugate gradient methods, barrier methods, and interior methods [61, 121, 171, 172]. Since they have difficulty in escaping from local minima and their solution quality depends heavily on their starting points, they are often used as components of other global search methods.

Global Search Methods Based on Penalty Formulations. These methods improve local search methods by choosing multiple starting points, by using some heuristics to jump out of local minima.The best solution is returned as the search result from all the local minima found. Examples of global search methods include multi-start [140, 139, 87, 149], trace or trajectory methods [159, 156], tabu search [74, 32], guided local search (GLS) [157], learning-based approaches [42], PBIL [30], MIMIC [40], COMIT [31], and random walk [145, 144]. Although some of these methods were designed originally to solve discrete problems, they can also be used to solve continuous problems after some modifications. For example, random walk and GLS can solve continuous problems when provided with an appropriate trial-point generator. As long as the trial-point generator can generate points in continuous space, random walk and GLS can actually search in a continuous space.

Global Optimization Methods Based on Penalty Formulations. These methods can be classified into unconstrained algorithms with reachability and those with asymptotic convergence. Examples in the first category include random search [124, 43, 22] and GA [77, 123, 86]. Typical methods in the second category include covering methods [93, 59, 84, 115], simulated annealing (SA) and its variants [175, 128, 101, 177, 53, 20, 21, 77, 62, 109, 136, 118, 123, 86] As discussed in the discrete case, if penalties are not chosen appropriately and time is not enough, these methods may not be able to find feasible solutions to the original constrained optimization problem.

Constraint Relaxation in Penalty Formulations. As we have discussed in Section 2.1.4, constraint relaxation can reduce the number of unsatisfied constraints and thus focus the search on hard-to-satisfy constraints. Therefore, constraint relaxation is very helpful for methods based on penalty formulations.

2.2.3 Lagrangian formulations

Methods based on Lagrangian formulation generally work on equality constraints. Before applying Lagrangian methods, inequality constraints are often first transformed into equivalent equality ones by adding slack variables [106] or by using the *MaxQ* method [166, 167, 163]. A general continuous equality-constrained minimization problem is formulated as follows:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & h(x) = \ [h_1(x), \ldots, h_m(x)]^T = 0, \end{aligned} \tag{2.5}$$

where $x = (x_1, x_2, \ldots, x_n)$ is a vector of continuous variables. Both $f(x)$ and $h(x)$ are assumed to be continuous functions that are at least first-order differentiable.

The *augmented Lagrangian function* in continuous space of (2.5) is defined as:

$$L_c(x, \lambda) = f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2, \tag{2.6}$$

where λ is a vector of Lagrange multipliers. Compared to the conventional *Lagrangian function* in continuous space defined as $L_c(x, \lambda) = f(x) + \lambda^T h(x)$, the *augmented Lagrangian function* reduces the possibility of ill conditioning and is, therefore, more stable.

Various continuous Lagrangian methods have been developed to locate CLM_{cn}. They are all based on the first-order necessary conditions [106]. These [38, 106] include the first-order method, Newton's method, modified Newton's methods, quasi-Newton methods, and sequential quadratic programming (SQP) [39, 54, 94].

Note that only unsatisfied constraints are actually present in a Lagrangian formulation. After constraint relaxation, fewer constraints exist in the Lagrangian function and, thus, simplifies the search in Lagrangian space and make it easier to satisfy the first order necessary conditions.

2.3 Methods for Mixed-Integer NLPs

Mixed-integer NLPs involve variables that are continuous as well as discrete. Methods for mixed-integer NLPs can also be classified as direct search methods, penalty formulations, and Lagrangian formulations.

2.3.1 Direct solutions for MINLPs

Typical direct solution methods for solving MINLPs include: a) reject/discarding [95, 27, 131, 126] or repair methods [97, 120] that try to avoid infeasible points or repair infeasible points into feasible ones and that can at best find CLM_{mn}; b) random search techniques, like pure random/adaptive search [124], hesitant adaptive search [43], CRS [129, 19, 22] and IHR [177], that try to satisfy all the constraints and improve the objective by random sampling; and c) branch-and-bound based methods, like FATCOP [49, 48], that utilize linear programming relaxation, depth-first-search, and cutting planes, to handle nonlinear constraints in problems that can be modeled effectively by linear relationships [49]. Most of the methods in this class, except branch-and-bound based methods, can utilize constraint relaxation to make the search easier, especially in the initial step.

2.3.2 Penalty formulations

Penalty-based methods transform a constrained MINLP into an unconstrained optimization, minimize the sum of objective and constraints weighted by penalties, and solve it using

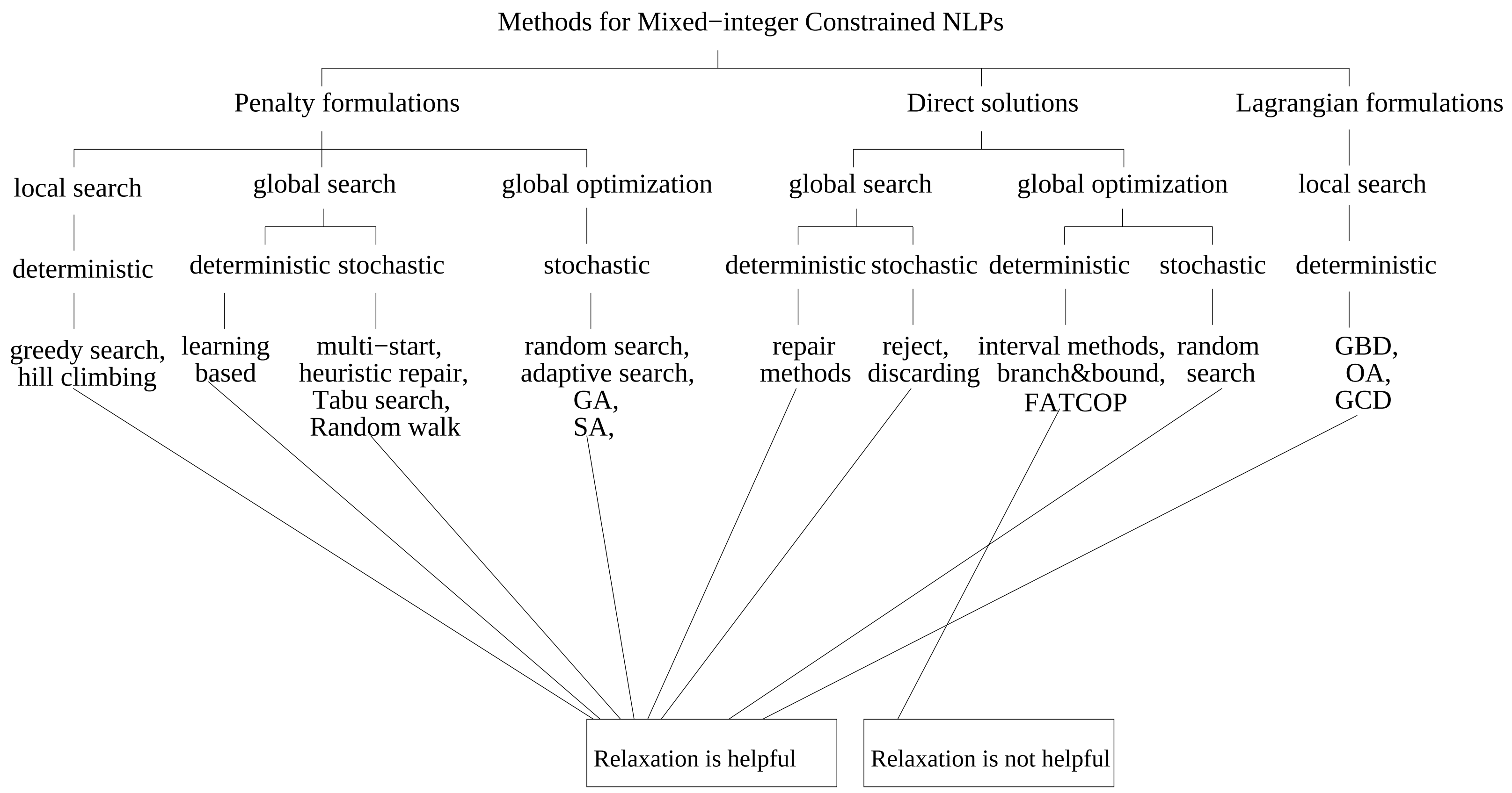


Figure 2.3: Classification of methods for solving mixed-integer constrained NLPs

existing unconstrained search algorithms. After the formulation, many local search, global search, and global optimization techniques can be applied to solve MINLPs.

Local Search Methods Based on Penalty Formulations. Typical methods include greedy descent and hill climbing [106, 130]. They probe in the joint space of discrete and continuous variables, while trying to decrease the objective and reduce constraint violations at the same time. Convergence to CLM_{mn} may not be guaranteed if penalties were not suitably chosen. Similar to that in the discrete and continuous cases, The search may be easily trapped in local minima in its variable space.

Global Search Methods Based on Penalty Formulations. These techniques, like Tabu search [72, 74], GLS [157], heuristic repair methods [51], learning-based approach [41, 42], Bayesian methods [113, 158, 114], multi-start [140, 139, 87, 149], and random walk [145, 144], can be applied to a constrained MINLP after it is transformed into an unconstrained problem using penalty formulations. Similar to the discrete and continuous cases, the success of these methods depends heavily on proper choices of penalties.

Global Optimization Methods Based on Penalty Formulations. Typical methods in this class include those that can ensure reachability, like GA [105, 176, 138, 138, 47, 45], random search [124], adaptive search [43], CRS [129, 19, 22] and IHR [177], and those that can guarantee asymptotic convergence, like SA [179, 16, 57] and many of its variants [175, 128, 177, 53, 20, 21]. See Section 2.2 for detailed explanations on these methods.

As we have discussed in the discrete and continuous cases, constraint relaxations are helpful for most of the search methods based on penalty formulation.

2.3.3 Lagrangian formulations

Methods in this class usually exploit the continuity and even the convexity of functions. These methods generally transform a MINLP into a Lagrangian formulation and then decompose it into subproblems such that, after fixing some variables, the resulting subproblem is convex in a subspace and can be solved easily. Although these methods may not require the differentiability of functions, derivative information can help improve solution time and quality significantly. There are three types of these methods.

Generalized Benders decomposition (GBD) [60, 68, 37] generates in each iteration an upper bound on the solution sought by solving a primal problem and a lower bound on a master problem. The primal problem corresponds to the original problem with fixed discrete variables and provides upper bound information and Lagrange multipliers for each constraint. The master problem is derived through nonlinear duality theory and provides lower bound information and the set of fixed discrete variables to be used in the next primal problem. Its main disadvantage is that it restricts their variable space to be nonempty and convex.

Outer approximation (OA) [56, 55] solves MINLPs by iterating a number of approximations. In each iteration the original problem is approximated by a subproblem containing the original feasible region. It is similar to GBD except that the master problem is formulated via primal information and outer linearization. This method requires the continuous subspace to be a nonempty, compact, and convex, and the objective and constraint functions to be convex.

Generalized cross decomposition (GCD) [60, 89, 90, 135] iterates alternately between two phases: Phase 1 solving the primal and dual subproblems and Phase 2 solving the master problem. Similar to OA and GBD, GCD also requires the objective and constraint functions to be properly convex.

Constraint relaxation is beneficial to methods in this class since it can reduce the number of unsatisfied constraints and simplify their Lagrangian functions.

2.4 Methods for Constrained NLPs with Uncertainty

There are generally two classes of problems with uncertainty. According to the source of uncertainty, they can be classified into two categories: stochastic programming and noisy cost optimization.

Stochastic Programming Problems [46, 143, 103] involve data that represent future information (e.g. the oil price of next year) and cannot be known with certainty. Stochastic programming often uses a multi-stage recourse model. For example, in a two-stage recourse model, the first stage has no uncertainty, using some variable x to control what happens in the first stage. The second stage may fall into one of a set of scenarios, each with a probability ω_i. Some variable y decides the actions in the second stage. The problem can be formulated to choose x and y in order to minimize the cost of the first-stage decision and the expected cost of the second-stage decision. More formally, the problem can be formulated as:

$$\begin{array}{rl} \text{minimize} & C^T x + E_\omega Q(x, y, \omega) \\ \text{subject to} & Ax = b, \\ & x \geq 0, \\ \text{where} & Q(x, y, \omega) = \min d(\omega)^T y, \\ \text{subject to} & T(\omega)x + W(\omega)y(\omega) = h(\omega). \end{array} \tag{2.7}$$

Since the second stage has a finite number of scenarios, each with a fixed probability, its expectation can be re-written as a weighted sum of the cost in each scenario and the associated probability of the scenario. Thus, the original problem can be transformed into a deterministic equivalent that can be solved as a general MINLP.

However, this kind of formulations has the following limitations. a) They are often formulated as linear programming problems. b) More importantly, they require the set of scenarios to be finite and the probability for each scenario to be known, which are often impractical. c) If the number of scenarios is large, the transformed MINLP is huge and is difficult to solve. In short, constraint relaxations are not helpful for reducing the complexity of this class of methods because it cannot reduce the number of scenarios.

Noisy Cost Optimization [81, 147, 17, 132, 66, 80, 23] involve uncertainties and random noise in function evaluations. To reduce the noise level at a point, multiple evaluations will have to be carried out. The problems can generally be formulated as follows:

$$\begin{array}{rl} \text{minimize} & E(f(x)) \\ \text{subject to} & E(h(x)) = 0 \\ & E(g(x)) \leq 0. \end{array} \tag{2.8}$$

Note that $f,\ g,\ h$ all contain some random noise, and multiple samples have to be drawn in order to achieve better precision. Simulated annealing [81, 147, 17, 132, 66, 80, 23] and direct search [25] have been used to solve versions of unconstrained problems. When constraints have noise, constraint relaxation is extremely useful to reduce their search complexity, since there is no need to ensure that constraint are satisfied early in a search.

2.5 Existing Constrained NLP Solvers

We summarize existing constrained NLP solvers in Table 2.1. The first column shows the name of each solver. Columns 2-13 indicate whether a solver can solve each corresponding class of constrained NLPs. A checkmark for solver X and problem class Y indicates that solver X can solve problems in class Y. Column 14 indicates whether a solver requires the convexity of functions; Column 15 shows the principal method used in each solver; Column 16 shows the input format, and Column 17 shows some other special requirement for each solver. We can see from the table that DONLP2 [150, 6], LANCELOT [52, 10, 102]. LOQO [155, 14], MINOS [151, 15], KNITRO [44], SNOPT [71], FSQP [5], HQP/OMUSES [2], and MOSEK [24, 13] are mainly used for solving continuous constrained NLPs with differentiable functions. Genocop [111, 9] and COBYLA2 [127] can solve constrained NLPs whose variables are continuous and whose functions are not differentiable or continuous. BARON [3, 137], BNB [1], MINLP_BB [12], SBB [11], Mittlp [4], and AlphaEcp [170, 8] can solve all continuous, discrete and mixed-integer constrained NLPs with continuous and differentiable functions. Finally, AUGMENTED [7], MSLIP [64] can solve constrained NLPs whose variables are continuous and whose functions are stochastic.

2.6 Summary

We have surveyed in this chapter existing work for solving discrete, continuous, mixed-integer constrained NLPs. Major existing approaches to these problems fall into one of the following three classes.

The first class of methods try to solve constrained NLPs directly. Except interval and branch and bound methods, most algorithms in this class can utilize constraint relaxation that can enlarge feasible regions and reduce solution time.

The second class of methods are based on penalty formulations. By combining both the objective and constraint functions through penalty coefficients, many unconstrained optimization techniques can be applied. Constraint relaxation is especially useful for these methods because, after constraint relaxation, the number of unsatisfied constraints is reduced and, thus, the penalty function is simplified.

The third class of methods are based on Lagrangian formulations. Since a Lagrangian formulation is essentially a type of penalty formulations, constraint relaxations are also helpful.

When constraints and objectives have noise in their evaluations, usual methods are to make multiple samples in order to control the noise to certain levels and achieve better precision. Since constraint relaxations can significantly reduce the number of samples at initial stages of a search, they can help greatly reduce the time for algorithms solving these problems.

From the survey, we conclude that constraint relaxations are helpful for most of the search methods, since it can the enlarge feasible region of a search problem, and reduce the number of unsatisfied constraints. In the next three chapters, we elaborate on the use of constraint relaxations in methods for solving mixed-integer problems and problems with noisy functions.

Table 2.1: Summary of existing constrained NLP solvers

Constrained	Applicability												Convexity	Principal	Input	Special
NLP solver	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	Required	Methods	Formats	Requirement
DONLP2	✓												No	sequential quadratic programming	Fortran,AMPL	
LANCELOT	✓												No	sequential quadratic programming	SIF,AMPL	
LOQO	✓												Yes	infeasible primal-dual interior-point	AMPL, Matlab	
MINOS	✓												No	sequential linearly constrained algorithm	GAMS, AMPL	modest nonlinearity
KNITRO	✓												No	prime-dual interior-point	AMPL	
SNOPT	✓												No	sequential quadratic programming	Fortran,GAMS,AMPL	modest free variables
CONOPT	✓												No	feasible path method	GAMS	
FSQP	✓												No	sequential quadratic programming	AMPL	
HQP/OMUSES	✓												No	integer-point, Newton-type SQP	SIF, C++	
MOSEK	✓												Yes	best interior-point method	AMPL	
Genocop	✓	✓	✓										No	genetic algorithm	C	
COBYLA2	✓	✓	✓										No	SLP method with estimation of gradient	Fortran	inequality constraints only
BARON	✓			✓			✓						No	branch and reduce	Baron model	functions are factorable
BNB	✓			✓			✓						No	branch and bound	Matlab	
MINLP_BB	✓			✓			✓						No	branch and bound, SQP	AMPL	
SBB	✓			✓			✓						No	branch and bound	GAMS	
Mittlp	✓			✓			✓						Yes	extended cutting plane	C	
AlphaEcp	✓			✓			✓						No	extended cutting plane	Fortran, LP	functions are pseudo-convex
AUGMENTED										✓			No	interior-point, augmented system	standard SLP	linear objective and constraints
MSLIP										✓			No	nested Benders decomposition	standard SLP	linear objective and constraints

Chapter 3

Constraint Relaxations in CSA

In this chapter we first introduce the general search framework for a general discrete or mixed-integer optimization problem. We then summarize the CSA algorithm implementing the search framework and its improved form, CSA_{ID}, and study constraint relaxation strategies in CSA_{ID}. In our study, we first observe a log-log relationship between constraint relaxation levels and the expected times to find a feasible solution using CSA_{ID}. Based on this relationship, we have designed $CSA_{ATCR-ID}$, the anytime search that utilizes a set of improved constraint relaxation levels and that is based on the principle of iterative deepening. We then prove the optimality of our proposed anytime search algorithm. Finally, we compare the performance of CSA with and without anytime constraint relaxations.

3.1 A General Framework to Look for SP_{dn} and its Implementation

For mixed integer or discrete problems, one of the observations from existing work is that there are various approaches to look for discrete-space saddle points but lacks a general framework that unifies these mechanisms. Without such a framework, it is impossible to know whether different algorithms are actually variations of each other. Therefore, we use the following unified framework in looking for saddle points. The framework allows us to show that many leading algorithms, such as DLM [173], CSA [164], and GA search of penalty formulations [112, 109], are similar algorithms that differ only in some components of the framework.

3.1.1 General framework

Based on the first-order necessary and sufficient conditions in Theorem 1.3, Figure 3.1 depicts a general stochastic optimization procedure to look for SP_{dn}. The procedure maintains a list of candidate points to be searched. It consists of two loops: the x loop that updates the variables in x in order to perform descents of $L_d(x, \lambda)$ in the x subspace, and the λ loop that updates the variables in λ, if there are unsatisfied constraints for any candidate in the list, in order to perform ascents of $L_d(x, \lambda)$ in the λ subspace. The procedure quits when no better probes can be generated in both the x and λ subspaces.

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

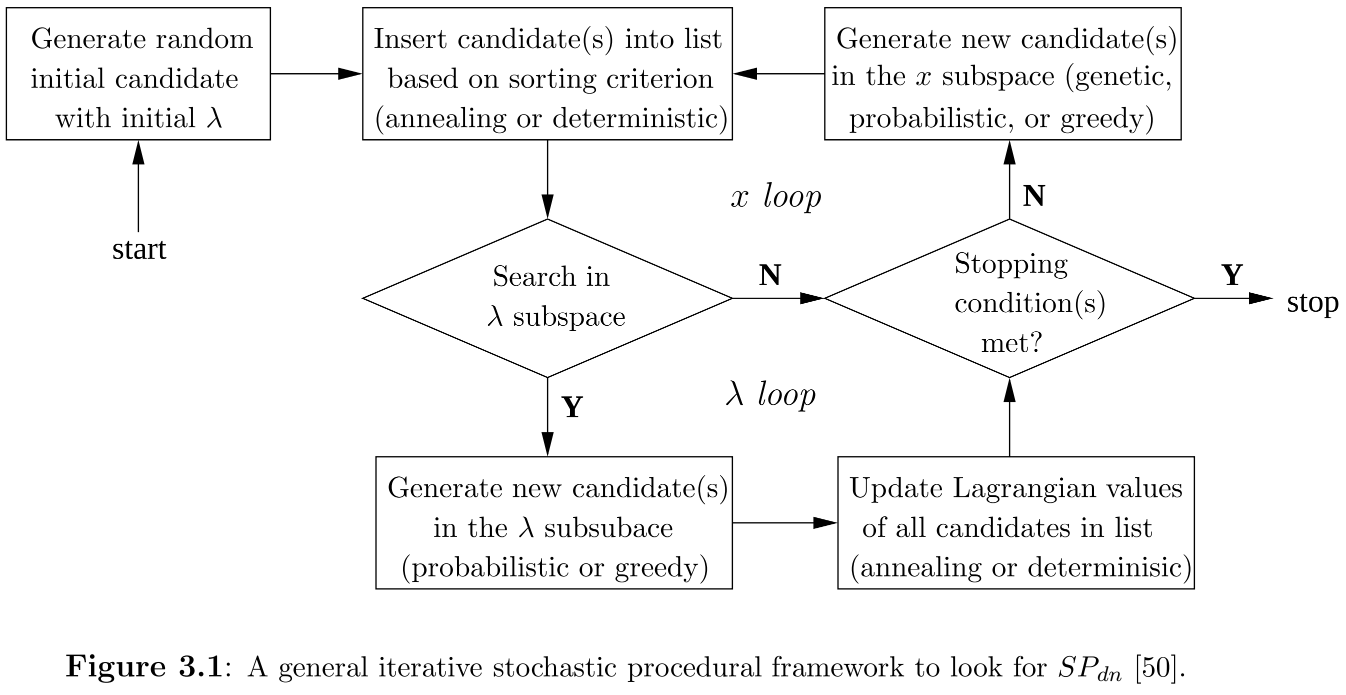

Generate random initial candidate with initial λ
Insert candidate(s) into list based on sorting criterion (annealing or deterministic)
Generate new candidate(s) in the x subspace (genetic, probabilistic, or greedy)
start
x loop
N
Search in λ subspace
N
Stopping condition(s) met?
Y
stop
Y
λ loop
Generate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

Figure 3.1: A general iterative stochastic procedural framework to look for SP_{dn} [50].

```
1.  procedure CSA
2.      set starting point $\mathbf{x} = (x, \lambda)$;
3.      set starting temperature $T = T^0$ and cooling rate $0 < \alpha < 1$;
4.      set $N_T$ (number of trials per temperature);
5.      while stopping condition is not satisfied do
6.          for $k \leftarrow 1$ to $N_T$ do
7.              generate a trial point $\mathbf{x}'$ from $\mathcal{N}(\mathbf{x})$ using $G(\mathbf{x}, \mathbf{x}')$;
8.              accept $\mathbf{x}'$ with probability $A_T(\mathbf{x}, \mathbf{x}')$
9.          end_for
10.         reduce temperature by $T \longleftarrow \alpha \times T$;
11.     end_while
12. end_procedure
```

Figure 3.2: CSA: constrained simulated annealing procedure [164].

The significance of the procedural framework in Figure 3.1 is that it provides a unified problem-independent way to implement Theorem 1.3.

3.1.2 Constrained simulated annealing (CSA)

As we have discussed, a variety of algorithms can fit into the above framework. As an example, we summarize in this section one of its implementation, constrained simulated annealing (CSA) [164].

CSA looks for discrete-space saddle points by performing both probabilistic descent of L_d in the x subspace and probabilistic ascent of L_d in the λ subspace in order to satisfy all the constraints in (1.1).

Figure 3.2 shows the basic procedure of CSA. A detailed discussion of each line in Figure 3.2 can be found in [164, 162]. The fundamental idea is explained as follows. CSA starts from an initial temperature T^0 and a starting point that is either specified or randomly generated.

In each iteration it generates a trial point (Line 7) $\mathbf{x}'$ in the neighborhood $\mathcal{N}(\mathbf{x})$ of the current point $\mathbf{x}$, using a generation probability $G(\mathbf{x}, \mathbf{x}')$.

It then decides whether to accept trial point $\mathbf{x}'$ using acceptance probability $A_T(\mathbf{x}, \mathbf{x}')$, defined as follows:

$$A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} exp\left(- \frac{(L_d(\mathbf{x}')-L_d(\mathbf{x}))^+}{T}\right) & \text{if } \mathbf{x}' = (x', \lambda) \\ exp\left(- \frac{(L_d(\mathbf{x})-L_d(\mathbf{x}'))^+}{T}\right) & \text{if } \mathbf{x}' = (x, \lambda'). \end{cases} \tag{3.1}$$

where $(a)^+ = \max(a, 0)$ for all $a \in R$.

Periodically it decreases temperature T using a geometric schedule in order for the annealing process to stop in finite time.

Asymptotic Convergence of CSA. CSA has been proved [162, 164] to converge asymptotically to a CGM_{dn} with probability one. The main result is summarized in the following theorem [162, 164].

Theorem 3.1 *Asymptotic convergence of CSA [164, 162].* The Markov chain modeling CSA converges asymptotically to CGM_{dn} $x^* \in X_{opt}$ with probability one.

In summary, CSA is a powerful algorithm in two aspects. First, it is a unified algorithm that can solve any general discrete, continuous and mixed-integer constrained optimization problems after discretizing all continuous variables. Second, it can find CGM_{dn} asymptotically, given a sufficiently slow cooling schedule. Of course, it is not practical to use a cooling schedule that is infinitely long, although, CSA can find a CGM_{dn} with a high probability by choosing a proper finite cooling schedule. Further, by restarting CSA search with a short duration, we can improve the success probability of finding a CGM_{dn}. CSA has been tested to work well on a variety of nonlinear benchmarks [162].

3.1.3 CSA with iterative deepening

One of the practical difficulties in using CSA is to determine the *cooling schedule*, or the way that temperatures are decreased in order to allow a solution with some prescribed quality to be found quickly.

Definition 3.1 An *optimal cooling schedule* is one that leads to the minimum average total number of probes of multiple runs of CSA from random starting points in order to find a solution of prescribed quality.

```
1. procedure $CSA_{ID}(Q)$
2.     set initial number of probes $N_\alpha = N_0$;
3.     set $K$ = number of CSA runs at fixed $N_\alpha$;
3.     repeat /*using iterative deepening to find Q*/
4.         for $i \leftarrow 1$ to $K$ do call CSA($N_\alpha$) end_for
5.         set $N_\alpha \leftarrow \rho \times N_\alpha$ (typically $\rho = 2$);
6.     until a feasible solution of quality Q has been found
           or $N_\alpha$ exceeds a maximum number allowed
           or (no better solution has been found in two
               successive increases of $N_\alpha$ and $N_\alpha > \rho^5 N_0$
               and a feasible solution has been found);
7. end_procedure
```

Figure 3.3: CSA_{ID}: CSA with iterative deepening, called with desired solution quality Q [160].

An *optimal cooling schedule* for a problem is hard to find because it is problem dependent. *CSA with iterative deepening* (CSA_{ID}) [160] has been developed to find the optimal cooling schedule using expected time of the same order of magnitude as that of the optimal cooling schedule. We summarize the results of CSA_{ID} as follows.

CSA_{ID} is shown in Figure 3.3. The algorithm uses a set of increasing cooling schedules (through iterative deepening) and runs CSA multiple times in each cooling schedule. By choosing an appropriate set of increasing cooling schedules and an appropriate number of runs in each, it can achieve a geometric growth in the number of probes under each cooling schedule. Hence, the total average overhead over all the schedules is dominated by that of the last schedule and is of the same order of magnitude as that of the multiple runs of the original CSA with the best cooling schedule. This result is stated formally in the following theorem [160].

Theorem 3.2 [160] Let $P_R(N_\alpha)$ be the reachability probability of one run of CSA under cooling schedule N_α. Let β be the expected total number of probes to find a solution using CSA_{IT-ID}, B_{opt} be the expected total number of probes to find a solution by the original CSA under the optimal schedule, and β be the total number of probes required by CSA_{ID}.

Then $\beta = O(B_{opt})$ if:

a) $P_R(N_\alpha)$ is monotonically non-decreasing for N_α in $(0, \infty)$;

b) $P_R(0) = 0$, and $\lim_{N_\alpha \to \infty} P_R(N_\alpha) = 1$;

c) $P''_R(0) > 0$;

d) $(1 - P_R(N_\alpha))^k \rho < 1$.

3.2 Constraint Relaxations During Annealing in CSA

For these optimization problems with many equality constraints, their feasible regions may be very small when compared to the whole search space and may be disjointed with each other. Hence, it may be hard to find a feasible region and to go from one feasible region to another.

To overcome this problem, we propose to relax equality constraints into inequality constraints first and tighten the constraints with decreasing temperatures in a single CSA run. As temperatures go to zero, relaxation levels go to zero at a faster rate than that of temperatures in order to guarantee asymptotic convergence (to be discussed). To simply the discussion, we consider in this section a discrete equality-constrained NLP:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject} \quad \text{to} \quad & h(x) = 0, \end{aligned} \tag{3.2}$$

where $x = (x_1, \ldots, x_n)$ is a vector of discrete variables and $f(x)$ and $h(x) = (h_1(x), ..., h_n(x))$ are bounded functions. Without loss of generality, inequality constraint $g(x) \leq 0$ can be transformed to equivalent equality constraint $\max(g(x), 0) = 0$.

3.2.1 Relaxation of equality constraints

Using relaxation, (3.2) is transformed into the following form:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject} \quad \text{to} \quad & |h(x)| \leq r(T), \end{aligned} \tag{3.3}$$

where $r(T) = (r_1(T), ..., r_n(T))$ is a vector deferentiable with respect to T, and $r_i(T)$ is a monotonically increasing function of T. Further we transform inequality constraint $|h_j(x)| \leq$

$r_j(T)$ into an equivalent equality constraint $\tilde{h}_i(x) = \max(0, |h_j(x)| - r_j(T)) = 0$. We define the corresponding Lagrangian function as follows:

$$L_r(x, \lambda) = f(x) + \sum_i \lambda_i \tilde{h}_i(x). \tag{3.4}$$

```
1.  **procedure RCSA**
2.      set starting point $\mathbf{x} = (x, \lambda)$ by randomly generating
            $x$ and setting $\lambda \longleftarrow 0$
3.      set starting temperature $T = T_0$ to be large enough
            and the cooling rate $0 < \alpha < 1$;
4.      set $N_T$ (number of probes per temperature);
5.      set initial relaxation for equality constraints
6.      **while** stopping conditions are not satisfied **do**
7.          **for** $n \leftarrow 1$ **to** $N_T$ **do**
8.              generate $\mathbf{x}'$ from $\mathcal{N}(\mathbf{x})$ using $G(\mathbf{x}, \mathbf{x}')$;
9.              accept $\mathbf{x}'$ with probability $A'_T(\mathbf{x}, \mathbf{x}')$
10.         **end_for**
11.         reduce temperature by $T \longleftarrow \alpha \times T$;
12.         reset the relaxation for equality constraints
13.     **end_while**
14. **end_procedure**
```

Figure 3.4: Relaxed constraint simulated annealing algorithm (RCSA)

We solve the relaxed problem using RCSA, the CSA with relaxed constraints (Figure 3.4) and the Lagrangian function (3.4).

To satisfy the requirement on asymptotic convergence, we choose $r(T) = k \times T^{\alpha}$ where $\alpha > 1$. Further we set k according to the initial violation (without relaxation). Suppose the starting point is $x^0 = (x_1^0, \ldots, x_n^0)$. we can set $k = c \times \max(h_1(x^0), \ldots, h_n(x^0))/T_0^{\alpha}$ or $k = c \times \text{median}(h_1(x^0), \ldots, h_n(x^0))/T_0^{\alpha}$. Intuitively, The second rule is preferred because k should not depend on starting points, especially when a few constraints dominate the maximum violation. Moreover, if the cooling rate is high, then we propose to reset k using the same rule after temperature drops for the first time. This will allow k to be more independent of the starting point.

In the following, we prove that RCSA can still converge to a CGM_{dn} with probability one under some conditions.

3.2.2 Theory of generalized simulated annealing

Recall some notations and results on generalized simulated annealing in [116, 154].

Set E denotes a finite search space, and q, a Markov kernel on E, is called the communication kernel. Further, q is irreducible, which means that for all distinct positions i and j in the search space, there exists a path from i to j, i.e. $\exists (i_k),\ k = 0, \ldots, n$, such that $i_0 = i, i_n = j$, and $q(i_k, i_{k+1}) > 0$ for all $k \leq n - 1$.

Consider a family of Markov kernels Q_T on E such that for all $T > 0$ and all $i, j \in E$,

$$\frac{1}{k} q(i,j) e^{-V_T(i,j)/T} \leq Q_T(i,j) \leq k q(i,j) e^{-V_T(i,j)/T},$$

where $(i,j) \in E \times E$, and $V(i,j) < +\infty$ iff $q(i,j) > 0$.

Using the notion of A-graph defined by Wentzell and Freidlin in [63], let $(i \to j)$ denotes the pair (i,j).

Definition 3.2 Let $A \subset E$. A set g of edges $(i \to j)$, in $A^c \times E$ is an *A-graph* iff: (a) For each $i \in A^c$, there is a unique $j \in E$ such that $(i \to j) \in g$. (b) For each $i \in A^c$, there is a path in g ending on a point in A.

When A contains only one point $\mathbf{x}$, the A-graph G is actually a spanning tree rooted at $\mathbf{x}$. Let $G(A)$ denote the set of A-graphs. Furthermore, for each $g \in G(A)$, we define the communication cost of g as

$$V_T(g) = \sum_{i \to j \in g} V_T(i,j).$$

Definition 3.3 For each state $x \in E$, its *virtual energy* $W(x)$ *is:*

$$W(x) = \min_{g \in G(\{x\})} V_T(g).$$

The *virtual energy* of $\mathbf{x}$ is actually the cost of the minimum spanning tree rooted at $\mathbf{x}$. Let

$$R_T(x) = \sum_{g \in G(x)} Q_T(g), \qquad Q_T(g) = \prod Q_T(y, z),$$

then whenever Q_T is homogeneous, its *invariant distribution* π_T is given by:

$$\pi_T(x) = \frac{R_T(x)}{\sum_{z \in E} R_T(z)},$$

which is the probability of hitting point x in the Markov chain.

The following two theorems are proved in [116]. Both of them have been adapted for our special formulation.

Theorem 3.3 [116] Suppose the inhomogeneous Markov kernel is $Q_T(\mathbf{x}, \mathbf{x}')$. Assume:

$$\sup_{\mathbf{x},T} |L_T(x)| < +\infty, \qquad \sup_{\mathbf{x},T} |T^2 \frac{d}{dT} L_T(\mathbf{x})| < +\infty. \tag{3.5}$$

When temperature T has a parametric form $T = K/(\log t)$ for K sufficiently large, the Markov chain converges to an invariant distribution π_T and $\lim_{T \to 0} P(X_T \in W^*) = 1$, where

$$W^* = \{\mathbf{x} \in E : W(\mathbf{x}) = \min_E W\}.$$

This theorem states that, under (3.5), generalized simulated annealing can be regarded as an optimization algorithm minimizing the virtual energy, which is implicitly defined by the communication cost function.

Theorem 3.4 [116] Let the Markov kernel be $Q'_T(x, y) = q(\mathbf{x}, \mathbf{x}')e^{-V_T(\mathbf{x},\mathbf{x}')/T}$. Suppose the assumptions of Theorem 3.3 are satisfied and for every $q(\mathbf{x}, \mathbf{x}') > 0$,

$$|V_T(\mathbf{x}, \mathbf{x}') - V(\mathbf{x}, \mathbf{x}')| = o(T), \tag{3.6}$$

as $T \to 0$. Then the Markov chain of the kernel Q'_T converges to the same invariant distribution as that of kernel Q_T, and $\lim_{T \to 0} P(X_T \in W^*) = 1$, where

$$Q_T = q(\mathbf{x}, \mathbf{x}')e^{-V(\mathbf{x},\mathbf{x}')/T}.$$

3.2.3 Asymptotic convergence of RCSA

In the original CSA algorithm, we have defined an irreducible Markov kernel $Q_T(\mathbf{x}, \mathbf{x}') = q(\mathbf{x}, \mathbf{x}')e^{-V(\mathbf{x},\mathbf{x}')/T}$ where $q(\mathbf{x}, \mathbf{x}')$ is the probability of generating $\mathbf{x}'$ from $\mathbf{x}$,

$$V(\mathbf{x}, \mathbf{x}') = \begin{cases} (L(\mathbf{x}') - L(\mathbf{x}))^+ & \text{if } \mathbf{x}' = (x', \lambda) \\ (L(\mathbf{x}) - L(\mathbf{x}'))^+ & \text{if } \mathbf{x}' = (x, \lambda') \end{cases}$$

and $L(\mathbf{x}) = f(x) + \lambda^T|h(x)|,\ \mathbf{x} = (x, \lambda) \in E$, E is a finite state space, and $(a)^+ = \max(0, a)$. Theorem 3.1 states that the corresponding Markov chain converges to CGM_{dn} $x^* \in X_{opt}$ with probability one.

In RCSA, let

$$\begin{aligned} L_T(\mathbf{x}) &= f(x) + \lambda^T(|h(x)| - r(T))^+, \\ V_T(\mathbf{x}, \mathbf{x}') &= \begin{cases} (L_T(\mathbf{x}') - L_T(\mathbf{x}))^+ & \text{if } \mathbf{x}' = (x', \lambda) \\ (L_T(\mathbf{x}) - L_T(\mathbf{x}'))^+ & \text{if } \mathbf{x}' = (x, \lambda'), \end{cases} \\ Q'_T(\mathbf{x}, \mathbf{x}') &= q(\mathbf{x}, \mathbf{x}')e^{-V_T(\mathbf{x},\mathbf{x}')/T}, \end{aligned}$$

where $r(T) = K \times T^\alpha$ is the degree of relaxation on constraints, and $K > 0$, $\alpha > 1$.

Q'_T is the Markov kernel of RCSA. In the following, we prove that the Markov chain of Q'_T also converges to a CGM_{dn} with probability one under some specified conditions. In other words, Q'_T has the same asymptotic convergence property as Q_T. We need some Lemmas first.

Lemma 3.1 $|(a)^+ - (b)^+| \leq |a - b|$.
This can be proved by simply enumerating the different cases of a and b.

Lemma 3.2 $|(a - r)^+ - a| \leq r$ if $r \geq 0$ and $a \geq 0$.
Proof. If $a > r$, then $|(a - r)^+ - a| = r$; otherwise, $|(a - r)^+ - a| = a < r$.

Theorem 3.5 The Markov chain with kernel Q'_T converges asymptotically to a CGM_{dn} with probability one if:
(a) Temperature T takes the form $T = K/\log t$ where t is time and K is sufficiently large,

(b) $|T^2 \frac{d}{dT} r(T)| < +\infty$,

(c) $r(T) = o(T)$ as $T \to 0$.

Proof. First, we show the Markov chain with kernel Q_T converges to the same invariant distribution as that with kernel Q_T. From Theorems 3.3 and 3.4 , it suffices to show that (3.5) and (3.6) are satisfied.

It is clear that $V_T(\mathbf{x}, \mathbf{x}') \to V(\mathbf{x}, \mathbf{x}')$ as $T \to 0$; hence, $|V_T(\mathbf{x}, \mathbf{x}')| < |V(\mathbf{x}, \mathbf{x}')| + C < +\infty$, Since $\frac{d}{dT} r(T) < +\infty$, it is easy to verify $|\frac{d}{dT} V_T(\mathbf{x}, \mathbf{x}')| < +\infty$. Hence, (3.5) is satisfied. For (3.6), if $\mathbf{x} = (x, \lambda)$ and $\mathbf{x}' = (x', \lambda)$,

$$\begin{aligned}
&|(V_T(\mathbf{x}, \mathbf{x}') - V(\mathbf{x}, \mathbf{x}'))| \\
&= |(f(x') - f(x) + \lambda((|h(x')| - r(T))^+ - (|h(x)| - r(T))^+))^+ \\
&\quad - (f(x') - f(x) + \lambda(|h(x')| - |h(x)|))^+| \\
&\le |(f(x') - f(x) + \lambda((|h(x')| - r(T))^+ - (|h(x)| - r(T))^+)) \\
&\quad - (f(x') - f(x) + \lambda(|h(x')| - |h(x)|))| \text{ (from Lemma 3.1)} \\
&= \lambda|(((|h(x')| - r(T))^+ - |h(x')|) - ((|h(x)| - r(T)) + -h(x)))| \\
&\le \lambda(|(|h(x')| - r(T))^+ - |h(x')| + |((|h(x)| - r(T)) + -|h(x)|)|) \\
&\le 2\lambda r(T) \quad \text{(from Lemma 3.2)}
\end{aligned} \tag{3.7}$$

Similarly, if $\mathbf{x} = (x, \lambda)$ and $\mathbf{x}' = (x, \lambda')$,

$$\begin{aligned}
&|(V_T(\mathbf{x}, \mathbf{x}') - V(\mathbf{x}, \mathbf{x}'))| \\
&= |(f(x) - f(x) + (\lambda - \lambda')(|h(x)| - r(T))^+)^+ - (f(x) - f(x) + (\lambda - \lambda')|h(x)|)^+| \\
&\le |(\lambda - \lambda')((|h(x')| - r(T))^+ - |h(x)|)| \text{ (from Lemma 3.1)} \\
&\le |\lambda - \lambda'| r(T) \quad \text{(from Lemma 3.2)}
\end{aligned} \tag{3.8}$$

Since λ and λ' are finite (given that the λ sub-space is finite), as long as $\lim_{T \to 0} r(T) = o(T)$, we will get

$$|V_T(\mathbf{x}, \mathbf{x}') - V(\mathbf{x}, \mathbf{x}')| = o(T),$$

Hence, the Markov chain with kernel Q_T and that with kernel Q'_T converge to the same invariant distribution π_T as $T \to 0$.

Finally, from Theorem 3.1, since the Markov chain modeling CSA (with kernel Q_T) converges to a CGM_{dn} with probability one, the Markov chain modeling RCSA (with kernel Q_T) also converges to a CGM_{dn} with probability one. This completes the proof.

The convergence proof of RCSA is important in two respects. First, it shows that, using our constraint relaxation strategy, CSA still preserves its asymptotic convergence property and can still find a CGM_{dn} with probability one. Second, it guides us in choosing an appropriate relaxation schedule in practice.

3.2.4 Experimental results

In this section, we apply different constraint relaxation schedules to solve nonlinear discrete equality constrained problems. To compare the experimental results for different solution quality, we use the following objective transformation:

$$F(f(x), f') = \begin{cases} f(x) & \text{if} f(x) > f', \\ f' & \text{otherwise}, \end{cases} \tag{3.9}$$

where f' is the target solution level, and a smaller f' indicates a better solution quality.

We performed experiments on four difficult Floudas and Pardalos' problems 5.2, 5.4, 7.3 and 7.4. The parameters of the these four problems are shown in Table 3.1.

Table 3.1: Parameters of four Floudas and Pardalos problems .

Problem ID	5.2	5.4	7.2	7.4
number of variables	46	32	27	38
number of equality constraints	36	26	19	23
number of inequality constraints	0	0	0	0

Table 3.2 compares the success ratio of the original CSA and $RCSA$ under different relaxation schedules. In $RCSA_1$, $r(T) = \alpha T^\gamma$, where $\alpha = \frac{\beta \times \text{median}(|h_i|)}{T_0^\gamma}$, $\beta = 4.0, 7.0$ and $\gamma = 1.01$ in our experiments. In $RCSA_2$, $r(T) = \alpha T^\gamma$, and

$$\alpha = \begin{cases} \frac{\beta_1 \times \max(|h_i|)}{T_0^\gamma} & \text{if } T = T_0 \\ \frac{\beta_2 \times \max(|h_i|)}{T_1^\gamma} & \text{otherwise.} \end{cases}$$

where T_0 is the initial temperature, $T_1 = cT_0$, and c is the cooling rate. In this case, $\beta_1 = 0.01, \beta_2 = 0.1, 0.5, \gamma = 1.1$. We ran each schedule 100 times using randomly generated starting points. Each entry of the table has two numbers. The first one is the percentage of those successful runs that found a solution satisfying the objective target f' in (3.9), whereas the second is the percentage of runs that found a feasible solution (but may not achieve target f'). The two numbers indicate the success ratios of finding a solution with target f' and that of finding a feasible solution. The results show that $RCSA_1$ and $RCSA_2$ improve the two success ratios in most cases.

We have found that the performance of RCSA depends heavily on the parameters chosen for each problem. This makes it difficult to be used across different problems.

3.3 Constraint Relaxation Across Multiple CSA Runs

In the previous section we have discussed constraint relaxation during annealing in CSA, in which relaxation levels decrease with temperatures. In this section we discuss constraint relaxation across multiple CSA runs in which the constraint relaxation level is fixed during each run of CSA and decreases from one run to another.

3.3.1 Problem formulation

It has been studied in [160] that CSA_{AT-ID} can generate solutions with gradually improving objective values after finding a feasible solution. However, for a complex problem, it may be difficult to find a feasible solution itself. In such cases, one is interested in finding a solution with a small violation within a time limit. Given more time, the algorithm can focus on decreasing the violations.

An anytime algorithm is one that can generate improved solutions as more computation time is used. The goal of this section is to design a sequence of relaxation levels that allow a search schedule with iterative deepening to generate solutions with reduced violations as more time is spent, eventually finding a feasible point. Moreover, the search schedule is optimal in the sense that the time taken to generate a feasible point is of the same order as that used by the original CSA with the optimal schedule to find a feasible point directly.

Table 3.2: Experimental results showing the success ratio to find a solution with target f' and a feasible solution using different relaxation schedules in solving Floudas and Pardalos' Problems 5.2, 5.4, 7.3 and 7.4. (Different cooling rate α for each problem have been tested.)

5.2	α =0.1			α =0.5		
f'	1.0	2.0	3.0	1.0	2.0	3.0
original CSA	0/92	9/94	89/89	0/99	17/98	97/97
$RCSA_1,\ \beta = 4$	0/99	13/99	**99/99**	0/97	15/96	97/97
$RCSA_1,\ \beta = 7$	0/99	**14/99**	97/97	0/96	**46/94**	92/92
$RCSA_2,\ \beta_2 = 0.1$	0/99	7/99	95/95	**0/100**	9/98	89/89
$RCSA_2,\ \beta_2 = 0.5$	**0/100**	10/99	97/97	**0/100**	10/99	97/97

5.4	α =0.01			α =0.1		
f'	1.0	2.5	3.0	1.0	2.5	3.0
Original CSA	0/24	13/29	29/29	0/97	22/93	86/86
$RCSA_1,\ \beta = 4$	**0/43**	22/44	39/39	0/98	31/98	92/92
$RCSA_1,\ \beta = 7$	0/40	22/39	**40/40**	**0/99**	**38/97**	**97/97**
$RCSA_2,\ \beta_2 = 0.1$	0/27	**29/36**	30/30	0/96	27/95	89/89
$RCSA_2,\ \beta_2 = 0.5$	0/20	18/34	26/26	0/96	31/96	96/96

7.3	α =0.3			$\alpha = 0.5$		
f'	1.0	1.5	2.0	1.0	1.5	2.0
Original CSA	0/7	6/9	11/11	0/17	12/20	22/22
$RCSA_1,\ \beta = 4$	**0/16**	12/16	15/15	**0/28**	12/16	20/20
$RCSA_1,\ \beta = 7$	0/11	**15/19**	**17/17**	0/22	12/20	23/23
$RCSA_2,\ \beta_2 = 0.1$	**0/16**	15/17	16/16	0/22	**17/28**	**32/32**
$RCSA_2,\ \beta_2 = 0.5$	0/8	13/15	13/13	0/27	**17/28**	29/29

7.4	α =0.3			α =0.8		
f'	1.0	1.5	2.0	1.0	1.5	2.0
no R&T	0/13	9/9	8/8	0/46	36/36	42/42
R&T method 1, $\beta = 4$	0/13	8/8	**15/15**	0/49	54/54	57/57
R&T method 1, $\beta = 7$	0/4	10/10	9/9	0/47	40/40	24/24
R&T method 2, $\beta_2 = 0.1$	0/14	**15/15**	**15/15**	0/65	**68/69**	**64/64**
R&T method 2, $\beta_2 = 0.5$	**0/17**	13/13	13/13	**0/66**	63/63	63/63

The approach we take is to first study statistically the performance of CSA. Based on the statistics collected, we propose a log-log model relating constraint-relaxation levels sought by CSA and its optimal expected execution times. This model leads to the design of $CSA_{ATCR-ID}$, the anytime constraint relaxation search schedule with iterative deepening that schedules multiple runs of CSA using a set of geometrically increasing number of probes in each run and a set of geometrically decreasing constraint relaxation levels.

Let $\mathcal{T}_{opt}(r)$ be the expected time taken by the original CSA with an optimal cooling schedule to find a feasible with relaxation r, and $E_{AT-ID}(r_n)$ be the expected total time taken by $CSA_{ATCR-ID}$ to find solutions of relaxation $r_0 > \cdots > r_n$ that are gradually improved over time. Based on the principle of iterative deepening [100], we prove the optimality of $CSA_{ATCR-ID}$ by showing in the next two sections that:

$$E_{AT-ID}(r_n) = O(\mathcal{T}_{opt}(r_n)) \tag{3.10}$$

3.3.2 Constraint relaxation

Suppose a general equality constraint $h(x)$ and inequality constraint $g(x)$, respectively, are expressed as follows:

$$\begin{aligned} h(x) &= 0 \\ g(x) &\le 0. \end{aligned} \tag{3.11}$$

A relaxation of the two kinds of constraints can be expressed by a parameter called relaxation level r:

$$\begin{aligned} |h(x)| &\le r \\ g(x) &\le r \end{aligned} \tag{3.12}$$

The main difference of the constraint relaxation between 3.12 and 3.3 is that in 3.12, the constraint relaxation r is fixed during each run, whereas that in 3.3 decreases as temperature T decreases.

Accordingly, the augmented Lagrangian function that only contains the terms related to the above two constraints after relaxation is expressed as follows:

$$L(x) = \lambda_h * (|h(x)| - r)^+ + \frac{1}{2}((|h(x)| - r)^+)^2 + \lambda_g * (g(x) - r)^+ + \frac{1}{2}((g(x) - r)^+)^2 \tag{3.13}$$

where $(y)^+ = max(0, y)$.

A simple observation is that any feasible solution to the original problem (3.11) is also a feasible solution to the relaxed problem (3.12). As the relaxed constraints lead to a larger pool of solution points, we expect the same algorithm to have a higher chance to hit one of these points during its search when constraints are relaxed, leading to reduced expected time to find a feasible solution to the relaxed problem. We exploit this property in $CSA_{ATRC-ID}$ next.

3.3.3 Performance modeling of CSAs for finding feasible solutions

The performance of CSA in finding feasible points for a given application problem from a random starting point can be measured by the expected time to find a feasible point, which, in turn, can be obtained by the average time of each run divided by the probability of finding feasible points. This can be derived as follows. Suppose the average time is $\mathcal{T}$ and the success probability is Pr. Then the expected time to find a feasible solution is:

$$\sum_{i=0}^{\infty} \mathcal{T} \times (1 - Pr)^i = \mathcal{T}/Pr \tag{3.14}$$

We focus on the relationship between relaxation levels and expected times to find a feasible point at each relaxation level.

In order to develop $CSA_{ATRC-ID}$ that dynamically controls its objective targets, we need to know the relationship between r, the relaxation level, and $\mathcal{T}_{opt}(r)$, the optimal expected time to find a feasible point to the relaxed constraints. In this section, we find this relationship by studying the statistical behavior of CSA in evaluating four constrained NLPs.

Figure 3.5 shows a 3-D graph relating the parameters in solving CUTE problem LAUNCH. Problem LAUNCH has 25 variables, 9 equality constraints and 19 inequality constraints. Some of the constraints in LAUNCH are highly nonlinear. The Pr in Figure 3.5 was obtained by running CSA from randomly selected starting points 100 times, for each relaxation level r. From Figure 3.5, we can derive the relationship between r and $\mathcal{T}_{opt}(r)$. Figure 3.6 shows example results derived from Figure 3.5.

Figures 3.6a and 3.6b illustrate the $Pr(T, r)$ curve and $\frac{T}{Pr(T,r)}$ curve at $r = 0.1$ derived from Figure 3.6. We see that there is an absolute minimum in the $\frac{T}{Pr(T,r)}$ curve. Note that $\mathcal{T}_{opt}(r)$ has been defined earlier as as the optimal expected time to find a feasible point with constraint relaxation level r. Figure 3.6c plots r versus $\mathcal{T}_{opt}(r)$, the absolute minimum of $\frac{T}{Pr(T,r)}$ over all T for each r.

Figure 3.6c shows a linear decreasing relationship between $\log r$ and $\log \mathcal{T}_{opt}(r)$, which can be formally written in the following log-log model:

$$\log \mathcal{T}_{opt}(r) = a - b \log r. \tag{3.15}$$

We performed experiments on different benchmarks and found that, for most of the benchmarks, such a log-log relationship as in Figure 3.6c exists.

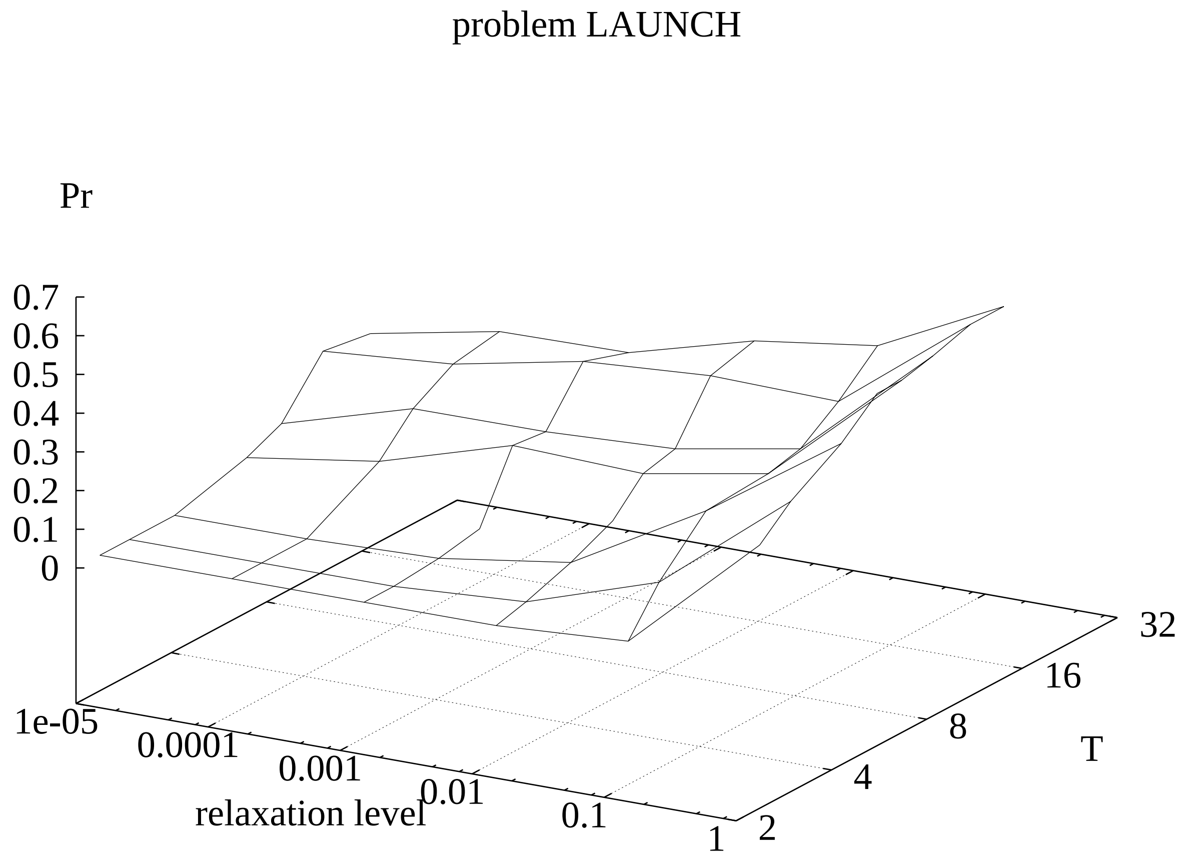
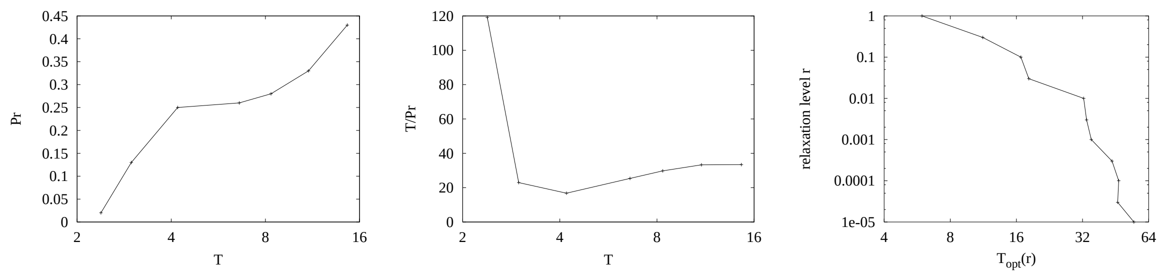


Figure 3.5: A 3-D graph showing the relationship of r, T, and $Pr(r, T)$, when CSA was applied to solve CUTE problem LAUNCH in mixed-integer version.

We performed the linear regression on some test problems in CUTE, including LAUNCH, MESH, AVION2 and BATCH, to test whether $\log \mathcal{T}_{opt}(r)$ and $\log r$ are linearly related. The test result on R^2, the coefficients of determination, indicates the degree to which this hypothesis is true. $R^2 = 1$ indicates a strict linear function. Table 3.3 summarizes the results of the R^2 test for each problem. Since R^2 is pretty close to one for all these problems, $\log \mathcal{T}_{opt}(r)$ was verified to be linear with respect to $\log r$.



a) Pr curve with $r = 0.1$ b) T/Pr curve with $r = 0.1$. c). $\mathcal{T}_{opt}(r)$ curve

Figure 3.6: An example showing the average performance model of CSA in solving LAUNCH (mixed-integer version) in CUTE. Note that there is an absolute minimum in the T/pr curve, which is defined as $\mathcal{T}_{opt}(r)$ in (c). (Each cooling schedule was run 100 times from random starting points.)

Table 3.3: The coefficients of determination R^2 on linear regression of $\log r$ and $\mathcal{T}_{opt}(r)$. The benchmarks are from the CUTE set of problems.

Problem ID	AVION2	LAUNCH	BATCH	MESH
R^2	0.96	0.87	0.95	0.93

3.3.4 Anytime schedule with iterative deepening

In this section, we design a schedule to decrease the relaxation level r in $CSA_{ATCR-ID}$ that allows it to find the minimal relaxation level using an average time of the same order of magnitude as $\mathcal{T}_{opt}(r)$.

$CSA_{ATCR-ID}$ is shown in Figure 3.7. The idea of the algorithm is to set the initial relaxation level r_0 based on the violation at the starting point or a large enough relaxation level. After finding a solution of relaxation level r_i using CSA_{ID} in Figure 3.3, Line 5 adjust r to a new relaxation level so that a better solution can be found if more time is allowed. After finding a solution, the algorithm adjusts r_{i+1} by decreasing the relaxation level by a constant factor and by using a linear regression of the two points representing the previous two relaxation levels to generate the next relaxation level in order for the predicted expected time to find a feasible solution under the new relaxation level r_{i+1} to be doubled.

```
1. procedure $CSA_{ATCR-ID}$
2.     set initial relaxation level;
3.     repeat /* over gradually improving relaxation level $r$ */
4.         call $CSA_{ID}(r)$; /* generate a solution with relaxation $r$ */
5.         reduce relaxation level $r \leftarrow r/c$;
6.     until CSA fails to solve the last relaxation level;
7. end_procedure
```

Figure 3.7: $CSA_{ATCR-ID}$: anytime CSA with iterative deepening and constraint relaxation that calls $CSA_{ID}(r)$ in Figure 3.3.

Let $E_{AT-ID}(r)$ be the expected total time taken by $CSA_{ATCR-ID}$ to find a solution with relaxation r_n, using a sequence of relaxation levels $r_0, r_1, \cdots, r_n$. The following theorem proves the relative complexities of $E_{AT-ID}(r_n)$ and $\mathcal{T}_{opt}(r_n)$.

Theorem 3.6 *$E_{AT-ID}(r_n) = O(\mathcal{T}_{opt}(r_n))$.*
Proof.

From Theorem 3.2,

$$E_{AT-ID}(r_n) = \sum_{i=0}^{n} E_{ID}(r_i) = \sum_{i=0}^{n} \mathcal{T}_{opt}(r_i) \tag{3.16}$$

Hence, from the log-log model in (3.15), we have:

$$\begin{aligned} E_{AT-ID}(r_n) &= \sum_{i=0}^{n} O(e^{a-b\log r_i}) \\ &= \sum_{i=0}^{n} O(e^{a-b\log(r_0/c^i)}) \\ &= O(e^{a-b\log(r_0/c^n)}) \\ &= O(\mathcal{T}_{opt}(r_n)) \end{aligned} \tag{3.17}$$

The theorem shows that, despite finding intermediate solutions by a sequence of improving relaxation levels during a search, the overall time spent by the search is dominated by that spent for finding a solution with the last relaxation level r_n

3.3.5 Experimental results

In this section we verify experimentally that our algorithm achieves our goal in (3.10). We applied our algorithm on the derived mixed-integer version of 4 difficult CUTE benchmarks: BATCH, LAUNCH, AVION2 and MESH. In generating mixed-integer problems, we assume that variables with odd indices are discrete variables and variables with even indices are continuous variables. In discretizing variable x_i with range (l_i, u_i), where l_i and u_i are lower and upper bounds, respectively, we force x_i to take values from the following set:

$$X_i = \begin{cases} J : J \text{ is integer and } l_i \leq J \leq u_i & \text{if } u_i - l_i \geq 10000 \\ l_i + \frac{u_i - l_i}{s} j, j = 0, 1, \cdots, s, & \text{otherwise.} \end{cases} \tag{3.18}$$

The parameters of the four problems are shown in Table 3.4

Table 3.4: Parameters of four CUTE benchmark problems in their mixed-integer version.

Problem ID	AVION2	LAUNCH	BATCH	MESH
number of variables	49	25	46	41
number of continuous variables	24	12	23	21
number of integer variables	25	13	23	20
number of equality constraints	15	9	12	24
number of inequality constraints	0	19	61	24

Figure 3.8 compare the anytime behavior of $CSA_{ITCR-ID}$ with the optimal expected times to find relaxed solutions. The solid line shows the average times to find feasible solutions for various relaxation levels r. The average time for each r was obtained by running CSA 100 times with randomly generated starting points. The three dotted lines show the performance of three runs of $CSA_{ATCR-ID}$, each from a different random starting point.

The result shows that our proposed anytime algorithm takes the same order of magnitude in execution time as the optimal expected times. Sometimes, the anytime algorithm even takes shorter time than the optimal expected time. This happens because the optimal expected time is measured by averaging the performance from randomly generated starting points, whereas the performance from one starting point may be different. The plot also shows that the time spent by $CSA_{ATCR-ID}$ under each relaxation level r_i increases exponentially, thereby achieving our goal that the total time spent for all relaxation levels is dominated by that spent for the last relaxation level.

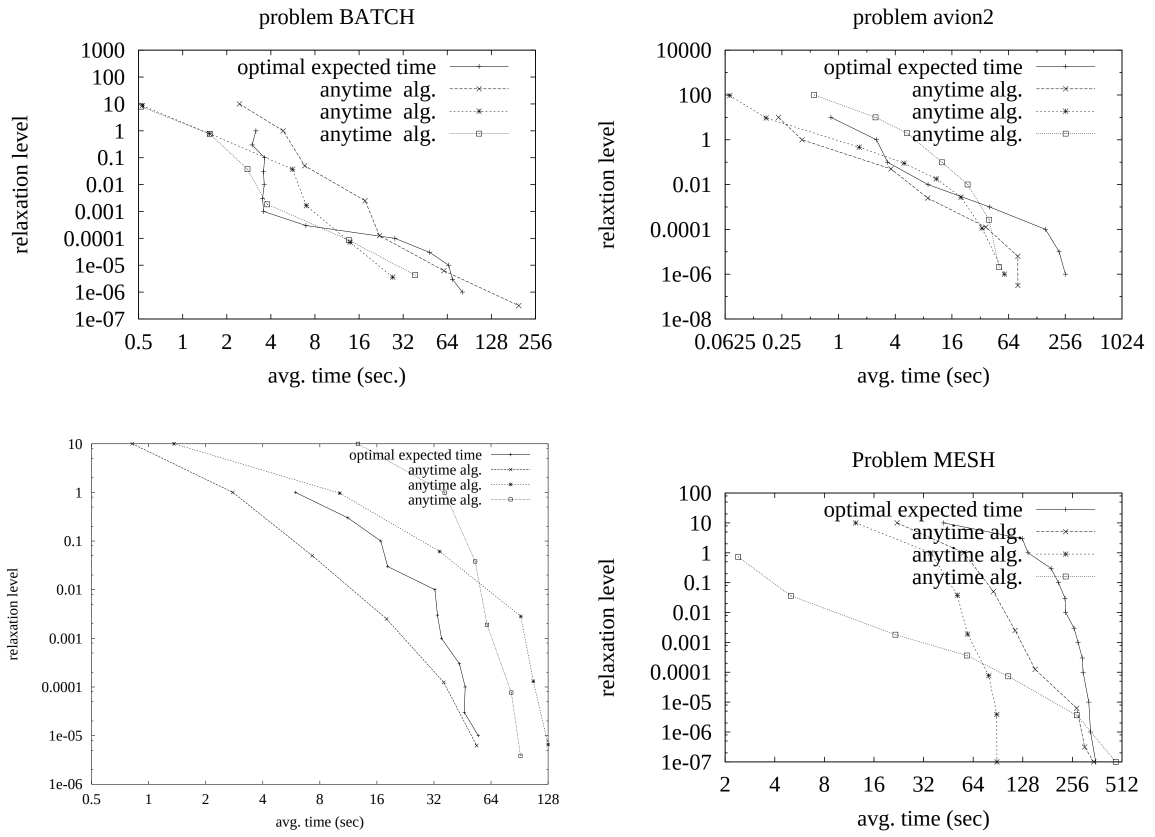


Figure 3.8: Comparison of times using $CSA_{ATCR-ID}$ with the optimal expected time to find a feasible solution under different relaxation levels. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7.

Last, we perform experiments on derived mixed-integer problems from selected problems in the CUTE benchmark.

The set of problems covers a wide range of problems, with objective functions of various types (linear, quadratic, cubic, polynomial, and nonlinear) and linear/nonlinear constraints of equalities and inequalities. 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.

For each problem, we run CSA_{ID} and $CSA_{ATCR-ID}$ to find feasible solutions and then compare their performance.

Table 3.5 compares the results for problems that are solvable by both CSA_{ID} and $CSA_{ATCR-ID}$. The first column shows the problem IDs, and the next two give the number ($n_v = n$) of variables and the number ($n_c = m + k$) of constraints. The next four columns show the number of linear equality constraints (n_{le}), the number of nonlinear equality constraints (n_{ne}), the number of linear inequality constraints (n_{li}), and the number of nonlinear inequality constraints (n_{ni}). The last two columns show the CPU times for finding feasible solutions by CSA_{ID} and $CSA_{ATCR-ID}$, respectively.

The table shows that the total time spent by $CSA_{ATCR-ID}$ is usually less than twice the time spent by CSA_{ID}. For some problems, $CSA_{ATCR-ID}$ even takes shorter time than CSA_{ID}. This happens because $CSA_{ATCR-ID}$ may take a very short time to find a solution for a large relaxation level. Starting from this solution, the algorithm will try to find a solution for a small relaxation level. If both of the solutions happen to be in the same region, then the time spent in the second step may be short, and the total time may be shorter than that spent by CSA_{ID} to directly solve the original problem.

Table 3.6 compares results for problems that cannot be solved by both CSA_{ID} and $CSA_{ATCR-ID}$. The first seven columns have the same meaning as that in Table 3.6. The last four columns show the violations when the algorithms stop and the CPU times taken. In general, $CSA_{ATCR-ID}$ is much better than CSA_{ID} in terms of the violation found, while the times spent by the two algorithms are similar.

Table 3.5: Results comparing CSA_{ID} and $CSA_{ATCR-ID}$ in finding feasible solutions for mixed-integer constrained NLPs derived from selected continuous problems in CUTE using the starting point specified in each problem. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7.

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}	$CSA_{ATCR-ID}$
			n_{le}	n_{ne}	n_{li}	n_{ni}	CPU time	CPU time
ALJAZZAF	3	1	0	1	0	0	0.04	0.14
ALLINITC	4	1	0	0	0	1	0.03	0.01
ALSOTAME	2	1	0	1	0	0	0.02	0.02
BATCH	46	73	12	0	60	1	63.45	62.03
BT11	5	3	1	2	0	0	0.06	0.15
BT12	5	3	0	3	0	0	0.38	1.36
BT8	5	2	0	2	0	0	0.02	0.22

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}	$CSA_{ATCR-ID}$
			n_{le}	n_{ne}	n_{li}	n_{ni}	CPU time	CPU time
CB2	3	3	0	0	0	3	0.01	0.05
CRESC4	6	8	0	0	0	8	0.17	0.46
DEMBO7	16	20	0	0	0	20	0.63	0.87
DIPIGRI	7	4	0	0	0	4	0.01	0.02
EXPFITA	5	22	0	0	22	0	0.04	0.05
FLETCHER	4	4	0	1	3	0	0.01	0.13
GAUSSELM	14	11	0	5	6	0	1.62	1.63
GIGOMEZ2	3	3	0	0	0	3	0.02	0.3
HATFLDG	25	25	0	25	0	0	12.34	12.85
HIMMELBI	100	12	0	0	12	0	0.35	1.65
HIMMELP2	2	1	0	0	0	1	0.03	0.02
HIMMELP6	2	5	0	0	2	3	0.01	0.04
HONG	4	1	1	0	0	0	0.01	0.04
HS100	7	4	0	0	0	4	0.03	0.04
HS101	7	5	0	0	0	5	0.07	0.43
HS102	7	5	0	0	0	5	0.08	0.43
HS103	7	5	0	0	0	5	0.06	0.44
HS104	8	5	0	0	0	5	0.15	0.42
HS108	9	13	0	0	0	13	0.05	0.20
HS111	10	3	0	3	0	0	0.27	0.61
HS114	10	11	1	2	4	4	1.7	2.3
HS117	15	5	0	0	0	5	0.03	0.04
HS119	16	8	8	0	0	0	11.6	30.6
HS12	2	1	0	0	0	1	0.01	0.01
HS18	2	2	0	0	0	2	0.02	0.04
HS19	2	2	0	0	0	2	0.01	0.03
HS20	2	3	0	0	0	3	0.01	0.01
HS23	2	5	0	0	0	5	0.01	0.04
HS27	3	1	0	1	0	0	0.02	0.03
HS29	3	1	0	0	0	1	0.01	0.01
HS30	3	1	0	0	0	1	0.02	0.02
HS32	3	2	1	0	0	1	0.03	0.06
HS33	3	2	0	0	0	2	0.02	0.01
HS34	3	2	0	0	0	2	0.03	0.02
HS36	3	1	0	0	1	0	0.03	0.01
HS37	3	2	0	0	2	0	0.01	0.04
HS39	4	2	0	2	0	0	0.07	0.22
HS40	4	3	0	3	0	0	0.07	0.51
HS41	4	1	1	0	0	0	0.01	0.02
HS42	4	2	1	1	0	0	0.03	0.06

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}	$CSA_{ATCR-ID}$
			n_{le}	n_{ne}	n_{li}	n_{ni}	CPU time	CPU time
HS43	4	3	0	0	0	3	0.02	0.03
HS46	5	2	0	2	0	0	0.08	0.22
HS54	6	1	1	0	0	0	0.08	0.12
HS57	2	1	0	0	0	1	0.02	0.02
HS59	2	3	0	0	0	1	0.01	0.03
HS60	3	1	0	1	0	0	0.05	0.05
HS61	3	2	0	2	0	0	0.03	0.05
HS62	3	1	1	0	0	0	0.02	0.03
HS64	3	1	0	0	0	1	0.04	0.04
HS68	4	2	0	2	0	0	0.04	0.09
HS69	4	2	0	2	0	0	0.06	0.55
HS7	2	1	0	1	0	0	0.05	0.03
HS71	4	2	0	1	0	1	0.03	0.04
HS73	4	3	1	0	1	1	0.06	0.26
HS78	5	3	0	3	0	0	0.13	0.26
HS83	5	3	0	0	0	3	0.01	0.07
HS84	5	3	0	0	0	3	0.02	0.03
HS99	7	2	0	2	0	0	1.19	4.58
HUBFIT	2	1	0	0	1	0	0.03	0.03
LAUNCH	25	28	6	3	12	7	49.76	51.64
LIN	4	2	2	0	0	0	0.02	0.01
LOADBAL	31	31	11	0	20	0	34.13	28.41
LOOTSMA	3	2	0	0	0	2	0.03	0.04
MADSEN	3	6	0	0	0	6	0.01	0.07
MARATOS	2	1	0	1	0	0	0.02	0.02
MATRIX2	6	2	0	0	0	2	0.03	0.02
MESH	41	48	4	20	24	0	186.5	381.7
MISTAKE	9	13	0	0	0	13	0.05	0.24
MRIBASIS	36	55	1	8	43	3	211	661
NGONE	8	8	0	0	2	6	0.03	0.05
ODFITS	10	6	6	0	0	0	0.17	0.63
OPTPRLOC	30	30	0	0	5	25	0.21	2.34
PENTAGON	6	15	0	0	15	0	0.04	0.12
POLAK1	3	2	0	0	0	2	0.04	0.05
POLAK5	3	2	0	0	0	2	0.01	0.04
QC	9	4	0	0	4	0	0.02	0.02
ROBOT	14	2	0	2	0	0	0.27	0.31
S316-322	2	1	0	1	0	0	0.02	0.05
SINROSNB	2	1	0	0	0	1	0.1	0.02
SNAKE	2	2	0	0	0	2	0.02	0.04

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}	$CSA_{ATCR-ID}$
			n_{le}	n_{ne}	n_{li}	n_{ni}	CPU time	CPU time
SPIRAL	3	2	0	0	0	2	0.01	0.02
STANCMIN	3	2	0	0	2	0	0.01	0.04
SVANBERG	10	10	0	0	0	10	0.09	0.09
SYNTHES1	6	6	0	0	4	2	0.04	0.04
SYNTHES2	11	14	1	0	10	3	0.3	0.4
SYNTHES3	17	23	2	0	17	4	0.2	0.53
WOMFLET	3	3	0	0	0	3	0.01	0.01
ZAMB2-8	138	48	0	48	0	0	567	605
ZECEVIC3	2	2	0	0	0	2	0.01	0.03
ZECEVIC4	2	2	0	0	1	1	0.01	0.02
ZY2	3	2	0	0	0	2	0.01	0.02

Table 3.6: Results comparing CSA_{ID} and $CSA_{ATCR-ID}$ in finding solutions with violations for mixed-integer constrained NLPs derived from selected continuous problems in CUTE using the starting point specified in each problem. The violations are the maximum violations across all the constraints of the solutions. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7.

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}		$CSA_{ATCR-ID}$	
			n_{le}	n_{ne}	n_{li}	n_{ni}	violation	CPU time	violation	CPU time
BT6	5	2	0	2	0	0	$\boxed{1.83}$	5.88	7.08	5.41
BT7	5	3	0	3	0	0	0.50	2.50	0.50	2.58
DNIEPER	61	24	0	24	0	0	0.022	2087	$\boxed{4.5\text{E-}4}$	2967
HS107	9	6	0	6	0	0	0.36	224	$\boxed{0.27}$	152
HS26	3	1	0	1	0	0	2	0.87	2	0.90
HS55	6	6	6	0	0	0	0.5	13.26	$\boxed{0.45}$	11.02
HS63	3	2	1	1	0	0	2.64	2.57	$\boxed{0.13}$	2.24
HS74	4	5	0	3	2	0	5.6E-3	16.23	$\boxed{3.2\text{E-}3}$	7.95
HS75	4	5	0	3	2	0	3.5E-3	22.74	$\boxed{8.5\text{E-}5}$	9.43
HS77	5	2	0	2	0	0	$\boxed{1.82}$	8.65	5.66	8.97

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}		$CSA_{ATCR-ID}$	
			n_{le}	n_{ne}	n_{li}	n_{ni}	violation	CPU time	violation	CPU time
HS79	5	3	0	3	0	0	$\boxed{2}$	2.53	26.25	2.51
HS80	5	3	0	3	0	0	0.038	22.51	$\boxed{\text{1.21E-5}}$	19.42
HS87	6	4	0	4	0	0	2.4E-3	26.28	$\boxed{\text{4.8E-4}}$	41.74
HS93	6	2	0	0	0	2	2.07	3.01	2.07	2.98
MWRIGHT	5	3	0	3	0	0	$\boxed{2}$	2.46	32.6	2.44
OPTCNTRL	32	20	10	10	0	0	6.0	338.5	$\boxed{2.0}$	406.86
READING6	102	50	0	50	0	0	223	19915	$\boxed{0.22}$	31747
RK23	17	11	4	7	0	0	0.50	26.70	0.50	30.46

3.4 Summary

Constraint relaxations are important in solving large complex constrained NLPs in which it is hard to find a feasible solution.

In this chapter we have addressed two kinds of constraint relaxation. In the first kind, we have relaxed constraints and have tightened them in the CSA algorithm by tightening the relaxation level after each iteration in order for the relaxation to go to zero when CSA finishes. We have proved the asymptotic convergence property of this new algorithm under some relaxation schedules.

In the second kind of constraint relaxations, we relax constraints between different runs of CSA. In each run, we use the iterative deepening version of CSA and fix the constraint relaxation levels. We build a log-log model between $\mathcal{T}_{opt}(r)$ and r based on experimental results. Based on the model, we decrease constraint relaxation levels between different runs of CSA_{ID} in such a way that the total time spent for all the runs of CSA_{ID} is of the same order of magnitude as the time spent for the last relaxation level using the optimal cooling schedule. We then prove the optimality of the algorithm. Experimental results verify savings in expectation time due to constraint relaxations.

Chapter 4

Constraint Relaxation in Line Search

In this chapter we present a constrained line search algorithm (CLS) to solve mixed-integer NLPs and apply our constraint relaxation strategy to such a deterministic algorithm. We solve (1.1), while assuming some variables are continuous and others are discrete. CLS is based on line search for continuous unconstrained problems and is combined with some special strategies to handle nonlinear and mixed-integer problems.

4.1 Line Search for Continuous Unconstrained Problems

In general, a line search is used for solving continuous unconstrained optimization problems. It starts by choosing a direction p and then searches along the direction using an approximate minimizer. It then repeats the above steps until some prescribed stopping conditions are satisfied. A generic line-search algorithm to minimize function $\phi(x)$ is listed in Figure 4.1.

In the general line search algorithm, direction p can be the gradient direction at the current point, or some other directions that can reduce the objective value. Moreover, β in Step 7 of Figure 4.1 need not be a constant. For example, we can use interpolations to decide the next value of η. Using the information on $\phi(0)$, $\phi'(0)$ and $\phi(\eta_0)$, we can interpolate the curve by a quadratic equation:

$$\phi_q(\eta) = \phi(0) + \phi'(0)\eta + \left(\frac{\phi(\eta_0) - \phi(0) - \eta_0\phi'(0)}{\eta_0^2}\right)\eta^2. \tag{4.1}$$

```
procedure Line_Search_for_Unconstrained
1. while stopping conditions for search are not satisfied;
2. begin
3.     set the initial step size $\eta$ ;
4.     choose a direction $p$ ;
5.     repeat
6.         evaluate $\phi(x + \eta * p)$;
7.         $\eta = \eta * \beta$;
8.     until stopping conditions for line search are satisfied.
9. end
```

Figure 4.1: A general line-search algorithm

The new trial point for η can then be one that minimizes $\phi_q(\eta)$:

$$\eta_1 = -\frac{\phi'(0)\eta_0^2}{2[\phi(\eta_0) - \phi(0) - \phi'(0)\eta_0]}. \tag{4.2}$$

After more points are searched along direction p, we can form a polynomial approximation of higher order, although it may not be easy to find minimal solutions for high-degree polynomial equations.

Note that the two stopping conditions in Lines 1 and 8 are different. The stopping conditions in Step 1 dictate when the whole search should be stopped, such as when the first-order necessary condition is satisfied or when a local minimum has been found. On the other hand, the stopping conditions in Step 8 dictate when a minimum along direction p has been found, or when a sufficient decrease of the objective function has been achieved, or when the Wolfe or Goldstein conditions are satisfied[119]. The Wolfe condition is composed of two equations:

$$\phi(x_k + \alpha_k p_k) \leq \phi(x_k) + c_1 \alpha_k \nabla \phi_k^T p_k \tag{4.3}$$

$$\nabla \phi(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla \phi_k^T p_k, \tag{4.4}$$

where $0 < c_1 < c_2 < 1$. The first condition requires a sufficient decrease of the objective function (note the $\nabla\phi_k^T p_k$ is usually negative); whereas the second requires the curvature in the new point be sufficiently different from that in the current point, indicating a sufficient distance between the new and the old points. When the new trial point is too close to the current point, the first condition is always satisfied, but the second is not.

Another important stopping condition called the Goldstein conditions are defined as follows:

$$\phi(x_k + \alpha_k p_k) \leq \phi(x_k) + c\alpha_k \nabla\phi_k^T p_k \tag{4.5}$$

$$\phi(x_k + \alpha_k p_k) \geq \phi(x_k) + (1-c)\alpha_k \nabla\phi_k^T p_k, \tag{4.6}$$

where $0 < c < 0.5$. The first condition is the same as the first Wolfe's condition, whereas the second requires a sufficient distance between the new and the current points. The advantage of the Goldenstein conditions is that it does not need to evaluate the derivative at each trial point.

4.2 Constrained Line Search Algorithms

Since our goal is to solve constrained optimization problem (1.1), we first transform the problem into an unconstrained Lagrangian function. Assuming both equality and inequality constraints, the Lagrangian function is as follows.

$$L = f(x) + \frac{1}{2}\sum_{i=1}^{k} \lambda_i h_i^2(x) + \frac{1}{2}\sum_{j=1}^{l} \lambda_{k+j} \overset{2}{\max}(g_j(x), 0). \tag{4.7}$$

We use the square of $h_i(x)$ and $max(g_i(x), 0)$ instead of their absolute values in order for the Lagrangian function to be continuous and differentiable. It can be proved easily, using this Lagrangian function, the first-order condition that a saddle point is still necessary and sufficient to be a constrained local minimum in discrete space.

After the transformation, we apply the generic search framework to look for saddle points of the Lagrangian function. In the framework, we use a line search algorithm to look for a minimum in the x-space, based on the gradient of the Lagrangian function as the search

```
procedure Constrained_Line_Search
1.  set initial value $\lambda, N_S$ ;
2.  repeat ;
3.      for k ← 1 to $N_S$ do ;
4.          compute the gradient of Lagrangian function $L$ ;
5.          use line search to find an improved Lagrangian value;
6.          if the line search ends up in a very small step size $\eta$ then
7.              Escape the local minimum using the trap escaping algorithm in Figure 4.3;
8.      end for;
9.      update $\lambda$ according to constraint violation;
10.     scale down $\lambda$ if the maximum $\lambda$ is too large ;
11. until stopping conditions are satisfied.
```

Figure 4.2: Constrained line search algorithm (CLS) for nonlinear mixed-integer optimization.

direction. The gradient of the Lagrangian function is:

$$\nabla L = f'(x) + \sum_{i=1}^{k} \lambda_i h_i(x) h'_i(x) + \sum_{j=1}^{l} \lambda_{k+j} \max(g_j(x), 0) g'_j(x). \tag{4.8}$$

We use Goldstein's conditions combined with a small step size as the stopping condition in our line search in order to avoid the evaluation of derivatives at each trial point. The complete algorithm is shown in Figure 4.2.

4.2.1 Escaping from local minimum

One issue of the original line-search algorithm is that it may have difficulty in escaping from a local minimum. At a local minimum, the gradient is zero, and going along any direction will not decrease the objective. In a constrained optimization problem, although it is possible to escape from a local minimum by adjusting the λ values for different constraints, a search may reach a point where all constraints and objective are at their own local minima. At this point, it is impossible to escape from this local minimum through changing λ. Further, even when not every constraint is at its local minimum, it may still be difficult to find appropriate λ's for the search to escape from the local minimum. For instance, in a simple special case when one constraint dominates all other constraints, and the search reaches a non-zero local minimum of this constraint, then the search will have no way to escape from this local minimum no matter how large the corresponding λ is. To address this issue, we add a rule to help escape from local minima. After each line-search step that includes a few trial steps along the search direction, we check whether the final step size is too small. A small step size suggests a local minimum in the search direction. We then look for a point from the list of previous trial points, that are sufficiently far away from the current point but whose Lagrangian value is not too large as compared to that of the current point, and traverse the search there. The algorithm is listed in Figure 4.3.

```
procedure Trap_Escaping_Algorithm($x$, $l$)
/* Escaping from a local minimum, given the list of current trial point $x_1, \cdots, x_n$
and their corresponding Lagrangian values $l_1, \cdots, l_n$. The farthest point is $x_n$,
$x_0$ and $l_0$ are, respectively, the current point and its Lagrangian value. */
1. $k = n$;
2. while ($k > 1$ ) ;
3.     if ($l_k - l_0 < \alpha \cdot abs(l_0)$) then
4.         return $x_k$ ;
5.     else
6.         $k \leftarrow k - 1$ ;
7. end while;
8. return $x_1$.
```

Figure 4.3: Trap escaping algorithm in line search

4.2.2 Strategies on discrete and continuous variables

Another important part of our algorithm is that we use different strategies on discrete and continuous variables. Part of time, the algorithm searches in the whole space, and in other time, searches in the continuous space.

An intuitive argument for a different strategy on discrete and continuous variables is that continuous variables have higher density in the search space than discrete variables. Moreover, a line-search step is computed based on the continuous situations, and the point after discretization can be far away from the search direction in the line-search step with significant violation when we always consider both discrete and continuous variables together. The situation can be demonstrated through the following example.

Assume a problem with one single constraint and no objective.

$$10000x + y = 0, \tag{4.9}$$

where x is an integer variable and y is a continuous variable. The Lagrangian function $L = \frac{1}{2}(10000x + y)^2$, and the gradient $g = (10000x + y)[10000, 1]^T = c[10000, 1]^T$, where $c = 10000x + y$ is the violation. We are looking for a step size η that minimizes $L(x', y') = \frac{1}{2}(10000x' + y')^2$, where $[x', y']^T = [x, y]^T - \eta \cdot c[10000, 1]^T$. When both x and y are continuous, it is easy to derive that $\eta = \frac{1}{c(100000001)}$ is the solution. However, after discretization, since x can only take integer value, the change to x is at least one if $x' \neq x$. Assuming $x' = x + 1$, the constraint violation at (x', y') is $c' = 10000x' + y' = (10000x + y) + 10000 + y' - y$, leading to $|c' - c| \approx 10000$. Hence when the constraint violation is already much smaller than 10000, we can assume $x' = x$ and $y' = y - \eta \cdot c$. The new violation becomes $c' = 10000x' + y' = 10000x + y + (y' - y) = c - \eta c = (1 - \eta)c \approx c - 10^{-8}$. We can see that the constraint violation is only decreased by 10^{-8}, and that will need hundreds of millions of line-search steps for the constraint to converge to 0.

Moreover, we can derive the best η in the mixed space as follows. Let $\eta' = \eta \cdot c$, we will get $[x', y']^T = [x, y]^T - \eta'[10000, 1]^T$. Without loss of generality, we can assume $c = 10000x + y$ is already relatively small. The best η' should not change the value of x (otherwise $c' = 10000 \cdot x' + y' = c + 10000 \cdot (x' - x) + (y' - y) \approx 10000(x' - x)$ when $|x' - x| \geq 1$). Hence, $10000\eta' < 1$; that is, $\eta' < 1/10000$. Thus, $x' = x$, $y' = y - \eta'$, and $c' = 10000x' + y' = 10000x + y + (y' - y) = c - \eta'$. Assume $c > 1/10000$, the best η' to reduce the violation

c' is close to but less than 1/10000 (because $\eta' < 1/10000$), and the constraint is decreased only by less than 1/10000 after each step. It will still takes about 10000 line-search steps to reduce the constraint to 0. However if we fix the discrete variable, we just need one step to decrease the constraint to 0. That is let $y' = y - c$! This example illustrates that it is very important to fix the discrete variables sometimes during a search in order to advance the continuous variables quickly.

Our strategy is to fix the discrete variables with probability α and search in the whole space with probability $(1 - \alpha)$. We have tried different α for test problems AVION2, LAUNCH, BATCH, and MESH. For AVION2, Figure 4.4a shows the relation between the resulting maximum violation and α, since no feasible solution is found. For the other three problems, Figures 4.4b, 4.4c and 4.4d show the relation between the average time to find a feasible solution and α. From these figures, we can see that the value of α is somewhat sensitive for some problems. As a heuristic, we choose $\alpha = 0.25$ that works well for most of the problems tested.

4.2.3 Experimental results and lessons learned

We have applied CLS to solve the mixed-integer problems derived from CUTE as in Chapter 3. Tables 4.1 and 4.2 compare the results using CLS and CSA_{ID}. When compared with CSA_{ID}, CLS usually uses much less time to find feasible solutions. The reason is that CLS uses gradient directions to direct its search, whereas, CSA_{ID} uses random sampling. The difference in performance is more significant for large problems, such as HYDROELL, HYDROELM, HYDROELS and ZAMB2-10. Table 4.2 also shows that, for many of the large problems, CLS uses much less time to generate some results, even though some of them still have violations.

Table 4.2 also shows that, for a few problems such as BT11, DEMBO7, CSA_{ID} can find feasible solutions while CLS cannot. The reason is that CLS uses the gradient of the Lagrangian function as its search direction. If the gradients of any constraint is 0, then this constraint will not be satisfied. Further, when constraints with zero gradient have integer variables, a gradient search may become more difficult because any value of an integer variable close to an integer value that makes the gradient zero will be discretized to that integer value. For example, given a constraint $x_2 \cdot x_1^2 = 2.8$ and x_1 is an integer variable, if

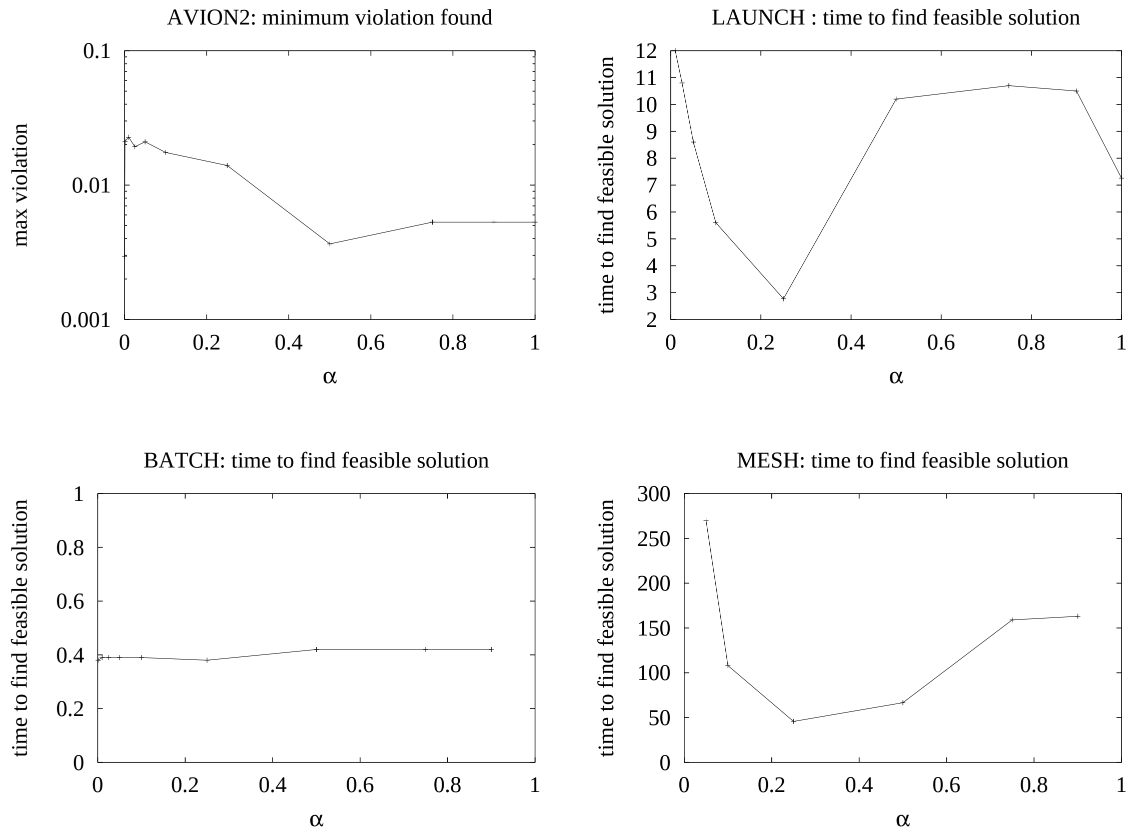


Figure 4.4: Evaluations of probability α to fix discrete variables.

x_1 falls into the region between -0.5 and 0.5, then it will be discretized to 0, and the gradient in both directions of x_1 and x_2 will be 0. This means that CLS will have no way to find a direction to reduce this violation. On the other hand, CSA_{ID} have no problem in solving such a problem because it does not rely on gradients to look for a search direction.

In short, a line search uses a linear function to approximate a Lagrangian function before solving the problem. The lesson we have learned is that a linear approximation is not enough to model a highly nonlinear function. Using quadratic approximation as in sequential quadratic programming will significantly improve the solution quality.

Table 4.1: Results comparing CSA_{ID} and CLS in finding feasible solutions to mixed-integer constrained NLPs derived from selected continuous problems in CUTE using the starting point specified in each problem. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7.

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}	CLS
			n_{le}	n_{ne}	n_{li}	n_{ni}	CPU time	CPU time
ALLINITC	4	1	0	0	0	1	0.03	$\boxed{0.01}$
ALSOTAME	2	1	0	1	0	0	$\boxed{0.02}$	0.03
BATCH	46	73	12	0	60	1	63.45	$\boxed{0.43}$
BT8	5	2	0	2	0	0	$\boxed{0.02}$	0.05
CRESC4	6	8	0	0	0	8	0.17	$\boxed{0.02}$
DIPIGRI	7	4	0	0	0	4	$\boxed{0.01}$	0.02
DIXCHLNG	10	5	0	5	0	0	0.02	$\boxed{0.01}$
EXPFITA	5	22	0	0	22	0	0.04	$\boxed{0.01}$
GAUSSELM	14	11	0	5	6	0	1.62	$\boxed{1.00}$
HATFLDG	25	25	0	25	0	0	12.34	$\boxed{1.47}$
HIMMELBI	100	12	0	0	12	0	0.35	$\boxed{0.04}$
HIMMELP2	2	1	0	0	0	1	0.03	$\boxed{0.02}$
HIMMELP6	2	5	0	0	2	3	0.01	0.01
HONG	4	1	1	0	0	0	0.01	0.01
HS100	7	4	0	0	0	4	0.03	$\boxed{0.02}$
HS101	7	5	0	0	0	5	$\boxed{0.07}$	0.57
HS102	7	5	0	0	0	5	0.08	$\boxed{0.05}$
HS103	7	5	0	0	0	5	$\boxed{0.06}$	0.62
HS104	8	5	0	0	0	5	$\boxed{0.15}$	0.37
HS108	9	13	0	0	0	13	$\boxed{0.05}$	0.15
HS111	10	3	0	3	0	0	0.27	$\boxed{0.03}$
HS114	10	11	1	2	4	4	1.7	$\boxed{0.02}$
HS117	15	5	0	0	0	5	0.03	$\boxed{0.01}$
HS119	16	8	8	0	0	0	11.6	$\boxed{0.85}$
HS12	2	1	0	0	0	1	$\boxed{0.01}$	0.03
HS18	2	2	0	0	0	2	0.02	0.02
HS20	2	3	0	0	0	3	0.01	0.01
HS23	2	5	0	0	0	5	0.01	0.01
HS24	2	3	0	0	3	0	0.02	$\boxed{0.01}$
HS27	3	1	0	1	0	0	0.02	$\boxed{0.01}$
HS29	3	1	0	0	0	1	0.01	0.01
HS30	3	1	0	0	0	1	0.02	$\boxed{0.01}$
HS32	3	2	1	0	0	1	0.03	$\boxed{0.01}$
HS33	3	2	0	0	0	2	0.02	$\boxed{0.01}$

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}	CLS
			n_{le}	n_{ne}	n_{li}	n_{ni}	CPU time	CPU time
HS34	3	2	0	0	0	2	0.03	$\boxed{0.02}$
HS36	3	1	0	0	1	0	0.03	$\boxed{0.01}$
HS37	3	2	0	0	2	0	$\boxed{0.01}$	0.02
HS41	4	1	1	0	0	0	0.01	0.01
HS42	4	2	1	1	0	0	0.03	$\boxed{0.02}$
HS43	4	3	0	0	0	3	0.02	$\boxed{0.01}$
HS46	5	2	0	2	0	0	0.08	$\boxed{0.03}$
HS54	6	1	1	0	0	0	0.08	$\boxed{0.01}$
HS57	2	1	0	0	0	1	0.02	0.02
HS59	2	3	0	0	0	1	$\boxed{0.01}$	0.02
HS60	3	1	0	1	0	0	0.05	$\boxed{0.02}$
HS61	3	2	0	2	0	0	$\boxed{0.03}$	0.05
HS62	3	1	1	0	0	0	0.02	$\boxed{0.01}$
HS64	3	1	0	0	0	1	0.04	$\boxed{0.01}$
HS68	4	2	0	2	0	0	0.04	$\boxed{0.01}$
HS69	4	2	0	2	0	0	0.06	$\boxed{0.02}$
HS71	4	2	0	1	0	1	0.03	$\boxed{0.01}$
HS83	5	3	0	0	0	3	$\boxed{0.01}$	0.03
HS84	5	3	0	0	0	3	0.02	$\boxed{0.01}$
HYDROELL	1009	1008	0	0	1008	0	10463	0.49
HYDROELM	505	504	0	0	504	0	1558	$\boxed{0.15}$
HYDROELS	169	336	0	0	336	0	66.41	$\boxed{0.04}$
HUBFIT	2	1	0	0	1	0	0.03	$\boxed{0.01}$
LAUNCH	25	28	6	3	12	7	49.76	$\boxed{10.74}$
LIN	4	2	2	0	0	0	0.02	$\boxed{0.01}$
LOADBAL	31	31	11	0	20	0	34.13	$\boxed{1.58}$
LOOTSMA	3	2	0	0	0	2	0.03	$\boxed{0.01}$
MADSEN	3	6	0	0	0	6	$\boxed{0.01}$	0.03
MARATOS	2	1	0	1	0	0	$\boxed{0.02}$	0.05
MATRIX2	6	2	0	0	0	2	0.03	$\boxed{0.02}$
MESH	41	48	4	20	24	0	186.5	$\boxed{92.58}$
MISTAKE	9	13	0	0	0	13	$\boxed{0.05}$	0.84
MRIBASIS	36	55	1	8	43	3	211	$\boxed{70.83}$
NGONE	8	8	0	0	2	6	0.03	$\boxed{0.01}$
ODFITS	10	6	6	0	0	0	$\boxed{0.17}$	1.19
OPTPRLOC	30	30	0	0	5	25	0.21	$\boxed{0.02}$
PENTAGON	6	15	0	0	15	0	0.04	$\boxed{0.01}$
POLAK1	3	2	0	0	0	2	0.04	$\boxed{0.02}$

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}	CLS
			n_{le}	n_{ne}	n_{li}	n_{ni}	CPU time	CPU time
POLAK3	12	10	0	0	0	10	$\boxed{0.03}$	0.66
POLAK5	3	2	0	0	0	2	$\boxed{0.01}$	0.37
POLAK6	5	4	0	0	0	4	0.03	0.02
QC	9	4	0	0	4	0	0.02	$\boxed{0.01}$
ROBOT	14	2	0	2	0	0	0.27	$\boxed{0.19}$
S316-322	2	1	0	1	0	0	0.02	0.02
SINROSNB	2	1	0	0	0	1	0.1	$\boxed{0.02}$
SNAKE	2	2	0	0	0	2	0.02	0.02
SPIRAL	3	2	0	0	0	2	0.01	$\boxed{0.02}$
STANCMIN	3	2	0	0	2	0	$\boxed{0.01}$	0.02
SVANBERG	10	10	0	0	0	10	0.09	$\boxed{0.01}$
SYNTHES1	6	6	0	0	4	2	0.04	$\boxed{0.01}$
SYNTHES2	11	14	1	0	10	3	0.3	$\boxed{0.03}$
SYNTHES3	17	23	2	0	17	4	0.2	$\boxed{0.02}$
TWOBARS	2	2	0	0	0	2	0.01	0.01
WOMFLET	3	3	0	0	0	3	$\boxed{0.01}$	0.02
ZAMB2-10	270	96	0	96	0	0	2245	66.72
ZAMB2-8	138	48	0	48	0	0	$\boxed{567}$	967
ZECEVIC3	2	2	0	0	0	2	$\boxed{0.01}$	0.03
ZECEVIC4	2	2	0	0	1	1	$\boxed{0.01}$	0.02
ZY2	3	2	0	0	0	2	$\boxed{0.01}$	0.03

Table 4.2: Results comparing CSA_{ID} and CLS in finding violations for mixed-integer constrained NLPs derived from selected continuous problems in CUTE using the starting point specified in each problem. The violations are the maximum violations across all the constraints of the solutions. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7. $'-'$ means that no solution could be found after 24 hours

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}		CLS	
			n_{le}	n_{ne}	n_{li}	n_{ni}	violation	CPU time	violation	CPU time
ALJAZZAF	3	1	0	1	0	0	0.0	0.04	10001	0.67
AVION2	49	15	15	0	0	0	0.35	182.8	0.0059	144.6
BT11	5	3	1	2	0	0	0	0.06	0.76	18.9
BT12	5	3	0	3	0	0	0	0.38	1	18.95
BT6	5	2	0	2	0	0	1.82	5.88	0	0.02
BT7	5	3	0	3	0	0	0.5	2.5	0.5	18.11
C-RELOAD	342	284	26	174	0	84	-	-	1	4837
CB2	3	3	0	0	0	3	0	0.01	3.0	3.12
DEMBO7	16	20	0	0	0	20	0	0.63	0.02	286.75
DNIEPER	61	24	0	24	0	0	0.022	2087	4.7E-4	153.62
ERRINBAR	18	9	0	8	1	0	336	151.1	4.61	57.75
FEEDLOC	90	259	4	15	166	74	-	-	4.95E-5	10513
FLETCHER	4	4	0	1	3	0	0	0.01	0.012	121.18
GIGOMEZ2	3	3	0	0	0	3	0	0.02	2	3.4
HS107	9	6	0	6	0	0	0.36	224	0.39	10.96
HS109	9	10	0	6	2	2	1.16	31.59	4927	27.9
HS19	2	2	0	0	0	2	0	0.01	0.40	0.56
HS26	3	1	0	1	0	0	2.0	0.87	0	0.01
HS39	4	2	0	2	0	0	0	0.07	4.0	0.68
HS40	4	3	0	3	0	0	0	0.07	0.20	3.79
HS55	6	6	6	0	0	0	0.5	13.25	0.33	4.66
HS56	7	4	0	4	0	0	2.78	39.06	1.28	3.26
HS63	3	2	1	1	0	0	2.64	2.57	3.74	2.31
HS7	2	1	0	1	0	0	0	0.05	21	9.04
HS73	4	3	1	0	1	1	0	0.06	1	0.53
HS77	5	2	0	2	0	0	1.82	0.13	0	1.27
HS78	5	3	0	3	0	0	0	0.13	2.01	2.51
HS79	5	3	0	3	0	0	2	2.53	3.76	12.98
HS80	5	3	0	3	0	0	0.39	22.51	1.3E-5	10.88
HS87	6	4	0	4	0	0	2.4E-3	26.28	1.18	10.93
HS93	6	2	0	0	0	2	2.07	3.02	0.38	5.91
LEWISPOL	6	9	0	9	0	0	0	17.97	2.5E-5	66.68
MWRIGHT	5	3	0	3	0	0	2	2.46	0	0.01
NET1	48	57	21	17	16	3	1830	2994	1	662
NET2	144	160	64	59	32	5	7.38	10451	1	2958

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}		CLS	
			n_{le}	n_{ne}	n_{li}	n_{ni}	violation	CPU time	violation	CPU time
NET3	464	521	195	199	110	17	-	-	21	15725
OPTCNTRL	32	20	10	10	0	0	6	338.5	0.83	21.88
READING6	102	50	0	50	0	0	222.9	19915	0.39	151.2
READING7	1002	500	0	500	0	0	-	-	0.46	11090
RK23	17	11	4	7	0	0	0.50	26.7	0.60	9.75
TENBARS4	18	9	0	8	1	0	1890	45.58	253.2	10.22
TWIRIMD1	1247	544	143	378	5	186	-	-	1.1E-4	48041
TWIRISM1	343	313	50	174	5	84	-	-	1.0E-4	4847
ZAMB2-11	270	96	0	96	0	0	-	-	0	347.8

4.3 Constraint Relaxation for Line Search

As we have mentioned in the first two chapters, constraint relaxation can also be applied to deterministic algorithms. In this section we study the properties of our constrained line search algorithm when constraints are relaxed.

4.3.1 Performance modeling

Based on four problems in CUTE of different sizes we study the performance of CLS under constraint relaxation. First, we apply different constraint relaxation levels r on each problem and evaluate their performance with respect to constraint relaxations. Suppose the original problem is:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \cdots, n; \\ & g_j(x) \leq 0, \quad j = 1, \cdots, n. \end{aligned} \tag{4.10}$$

After constraint relaxation with level r, the problem is transformed into

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & |h_i(x)| \leq r, \quad i = 1, \cdots, n; \\ & g_j(x) \leq r, \quad j = 1, \cdots, n. \end{aligned} \tag{4.11}$$

After constraint relaxation, we apply CLS on the four test problems and evaluate the algorithm in terms of its expected time to find a feasible solution, T_{exp}, which is defined as the average time of one run of the algorithm divided by the probability of finding a feasible solution.

Figure 4.5 plots the relationship between T_{exp} and r, which shows a log-log relationship. in some range of each plot.

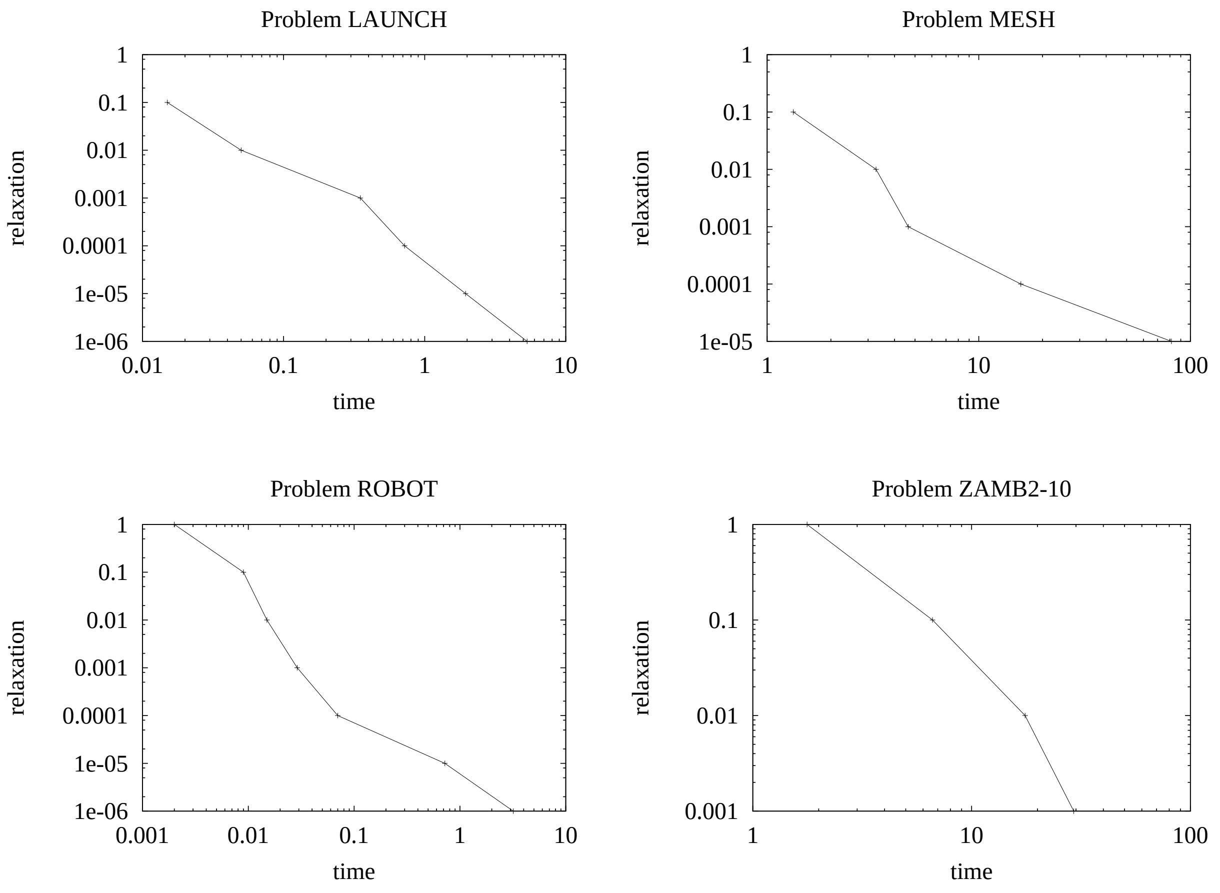


Figure 4.5: Performance models of four problems from CUTE.

Table 4.3 shows the R^2 values of hypothesis tests on whether these plots are linear curves. A value of R^2 close to 1 means that the curve is linear.

Table 4.3: Coefficients of determination R^2 on linear regression of $\log r$ and their corresponding $\log T_{exp}$. The benchmarks are from the CUTE set of problems.

Problem ID	LAUNCH	MESH	ROBOT	ZAMB2-10
R^2	0.98	0.95	0.95	0.96

Both the graphs and the statistical tests confirm that there is a log-log relationship between T_{exp} and r. This model can be formally represented as follows:

$$\log T_{exp} = A - B \log r. \tag{4.12}$$

Based on the performance model, we derive the anytime algorithm in the next section.

4.3.2 Anytime constrained line-search algorithm

Our goal in this section is to develop an anytime constrained line search algorithm, that can find a solution, possibly with some violations, in limited time. When given more time, the algorithm is expected to find better solutions. The time taken by the algorithm should be of the same order of the magnitude as the time spent in finding the last solution.

Our approach is to design a sequence of relaxation levels $r : (r_1, r_2, \cdots, r_n)$, and run CLS for each r_i, such that the total time spent for the whole sequence of r is dominated by $T_{exp}(n)$, the expected time taken to find the solution in the last run.

In the following, we derive the sequence of relaxation levels r, based on the log-log model in the last section. Suppose the expected time to find a feasible solution for r_i is $T_{exp}(i)$. Our goal is to achieve $\sum_{i=1}^{n} T_{exp}(i) \leq K \cdot T_{exp}(n)$. One simple way to achieve this is to let $T_{exp}(i)$ grows in a geometric sequence, say $T_{exp}(i+1) = C \cdot T_{exp}(i)$, where C is a constant. Applying it to (4.12), we get the following result: $r_{i+1} = r_i(C^{-1/B})$. Since the graphs in Figure 4.5 show problem dependent slopes, We will not be able to use a fixed C for every problem. Instead, we use linear regression after each run of CLS and predict the next relaxation level. The initial relaxation level is set by dividing the maximum violation by a constant. The algorithm is listed in Figure 4.6.

```
1. procedure Anytime_Constrained_Line_Search
2.     set initial target of relaxation level;
3.     repeat /* over gradually improving relaxation level $r$ */
4.         call Constrained_Line_Search; /* generate a solution with relaxation $r$ */
5.         reduce relaxation level $r$ using regression
               but bounded in a range relative to the previous $r$;
6.     until Constrained_Line_Search fails to solve the last relaxation level
           or relaxation level is 0;
7. end_procedure
```

Figure 4.6: Anytime constrained line search procedure (CLS_{AT}) that calls CLS in Figure 4.2.

4.3.3 Performance evaluation

First, we demonstrate the behavior of the anytime search by applying CLS_{AT} to the four sample problems in CUTE: LAUNCH, MESH, ROBOT, and ZAMB2-10 (ROBOT has 14 variables and 2 equality constraints; ZAMB2-10 has 270 variables and 96 equality constraints; LAUNCH and MESH are medium size problem and have tens of variables and constraints).

Figure 4.7 compares the anytime behavior of CLS_{AT} with the expected time to find a feasible solution for each relaxation level. In general, the results show that the actual time spent has a log-log relationship with the predicted relaxation level; hence, the actual time spent is geometrically increasing with respect to the number of runs of CLS. Further, the actual time spent by CLS_{AT} is of the same order of magnitude as the expected time spent by CLS in each relaxation level. It is possible that the former is even shorter than the latter. This happens because, when looking for a solution using relaxation level r_{i+1}, CLS_{AT} will start from the solution generated when using relaxation level r_i. If the two solutions happen to be in the same region, then the time spent when using relaxation level r_{i+1}, may be very short.

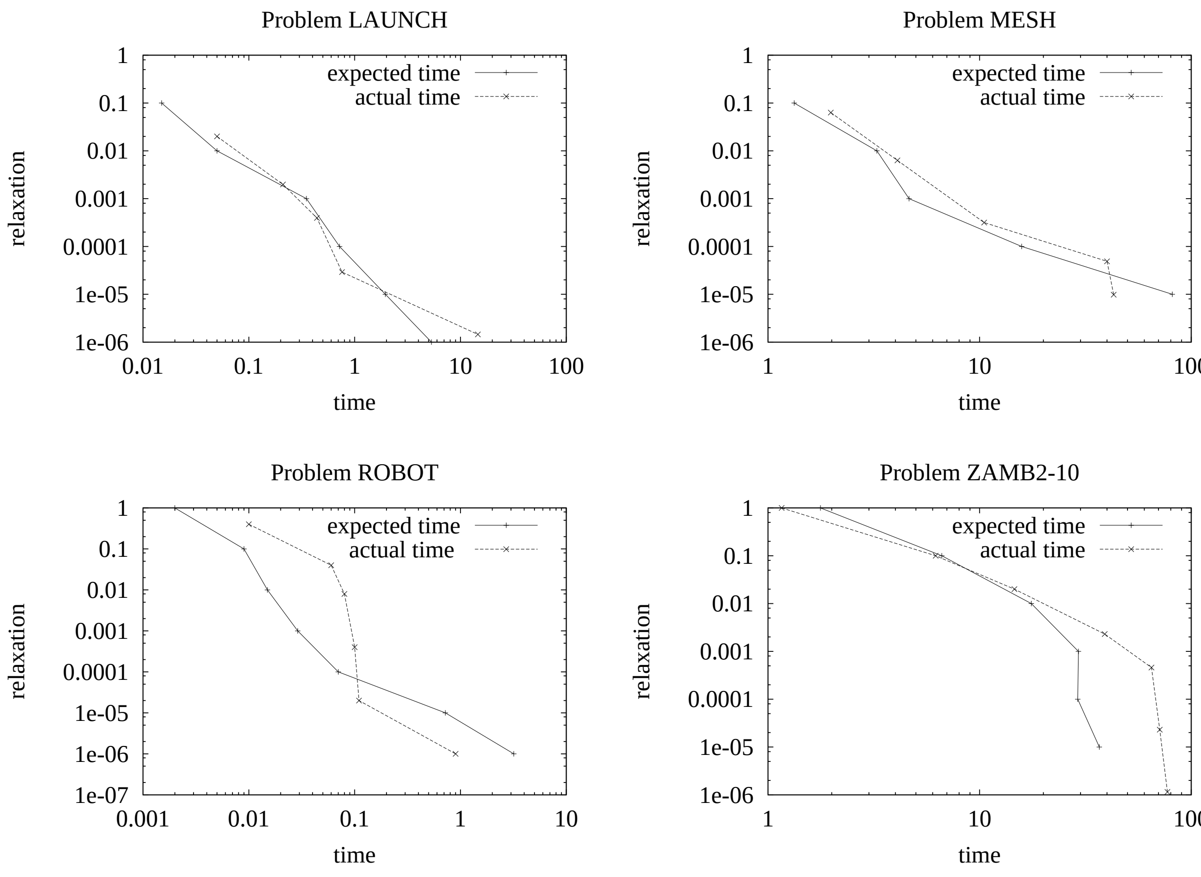


Figure 4.7: Comparison of the actual times spent to find feasible solutions by CLS_{AT} with the expected time to find a feasible solution in each relaxation level

4.3.4 Experimental results on NLP benchmarks

We have performed experiments on some derived mixed-integer constrained NLP benchmarks and have compared our experimental results with CSA_{ID} and $CSA_{ATCR-ID}$. Table 4.4 lists the problems that can be solved by CLS and CLS_{AT}, whereas Table 4.5 lists the problems that cannot be solved by either of the two algorithms and reports the violations found by each.

When compared with CSA_{ID} and $CSA_{ATCR-ID}$, the results show that CLS and CLS_{AT} use much less time to find a feasible solution for most of the problems, especially for large problems such as HYDROELL, HYDROELM, HYDROELS and ZAMB2-10. The improved efficiency is due to the use of gradients in CLS, which has been discussed earlier.

When compared with CLS, the results show that CLS_{AT} uses a similar amount of time to find feasible solutions or solutions with similar violations for most of the problems. Hence, we have achieved our goal in CLS_{AT} that incurs overhead of the same order of magnitude as that of the last run of CLS.

Note that, for a number of problems such as BT11, CSA_{ID} or $CSA_{ID-ATCR}$ can find feasible solutions but CLS and CLS_{AT} cannot. This shows the weakness of linear approximations in line search. A higher-order approximation such as that in sequential quadratic programming, should be used to achieve better solution quality .

Table 4.4: Results comparing CSA_{ID}, $CSA_{ATCR-ID}$, CLS, and CLS_{AT} in finding feasible solutions for mixed-integer constrained NLPs derived from selected continuous problems in CUTE using the starting point specified in each problem. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7.

Problem ID	n_v	n_c	$h(x)$ n_{le}	$h(x)$ n_{ne}	$g(x)$ n_{li}	$g(x)$ n_{ni}	CSA_{ID} CPU time	$CSA_{ID-ATCR}$ CPU time	CLS CPU time	CLS_{ATCR} CPU time
ALLINITC	4	1	0	0	0	1	0.03	$\boxed{0.01}$	$\boxed{0.01}$	$\boxed{0.01}$
ALSOTAME	2	1	0	1	0	0	0.02	0.02	0.03	$\boxed{0.01}$
BATCH	46	73	12	0	60	1	63.45	62.03	$\boxed{0.43}$	20.06
BT8	5	2	0	2	0	0	$\boxed{0.02}$	0.22	0.05	0.18
CRESC4	6	8	0	0	0	8	0.17	0.46	$\boxed{0.02}$	0.04
DIPIGRI	7	4	0	0	0	4	$\boxed{0.01}$	0.02	0.02	0.02
EXPFITA	5	22	0	0	22	0	0.04	0.05	$\boxed{0.01}$	0.02
GAUSSELM	14	11	0	5	6	0	1.62	1.63	$\boxed{1.00}$	2.76
HIMMELBI	100	12	0	0	12	0	0.35	1.65	$\boxed{0.04}$	0.08
HIMMELP2	2	1	0	0	0	1	0.03	$\boxed{0.02}$	$\boxed{0.02}$	$\boxed{0.02}$
HIMMELP6	2	5	0	0	2	3	$\boxed{0.01}$	0.04	$\boxed{0.01}$	0.02
HONG	4	1	1	0	0	0	$\boxed{0.01}$	0.04	$\boxed{0.01}$	0.02
HS100	7	4	0	0	0	4	0.03	0.04	$\boxed{0.02}$	$\boxed{0.02}$
HS101	7	5	0	0	0	5	$\boxed{0.07}$	0.43	0.57	1.65
HS102	7	5	0	0	0	5	0.08	0.43	$\boxed{0.05}$	2.81
HS103	7	5	0	0	0	5	$\boxed{0.06}$	0.44	0.62	0.84
HS104	8	5	0	0	0	5	$\boxed{0.15}$	0.42	0.37	1.36
HS108	9	13	0	0	0	13	$\boxed{0.05}$	0.20	0.15	0.19
HS111	10	3	0	3	0	0	0.27	0.61	$\boxed{0.03}$	$\boxed{0.03}$
HS114	10	11	1	2	4	4	1.7	2.3	$\boxed{0.02}$	$\boxed{0.02}$
HS117	15	5	0	0	0	5	0.03	0.04	$\boxed{0.01}$	0.02
HS119	16	8	8	0	0	0	11.6	30.6	0.85	$\boxed{0.63}$
HS12	2	1	0	0	0	1	$\boxed{0.01}$	$\boxed{0.01}$	0.03	0.03
HS18	2	2	0	0	0	2	$\boxed{0.02}$	0.04	$\boxed{0.02}$	$\boxed{0.02}$
HS20	2	3	0	0	0	3	$\boxed{0.01}$	$\boxed{0.01}$	$\boxed{0.01}$	$\boxed{0.01}$

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}	$CSA_{ID-ATCR}$	CLS	CLS_{ATCR}
			n_{le}	n_{ne}	n_{li}	n_{ni}	CPU time	CPU time	CPU time	CPU time
HS23	2	5	0	0	0	5	$\boxed{0.01}$	0.04	$\boxed{0.01}$	$\boxed{0.01}$
HS24	2	3	0	0	3	0	0.02	0.02	$\boxed{0.01}$	$\boxed{0.01}$
HS27	3	1	0	1	0	0	0.02	0.03	$\boxed{0.01}$	$\boxed{0.01}$
HS29	3	1	0	0	0	1	0.01	0.01	0.01	0.01
HS30	3	1	0	0	0	1	0.02	0.02	$\boxed{0.01}$	$\boxed{0.01}$
HS32	3	2	1	0	0	1	0.03	0.06	$\boxed{0.01}$	$\boxed{0.01}$
HS33	3	2	0	0	0	2	0.02	$\boxed{0.01}$	$\boxed{0.01}$	0.02
HS34	3	2	0	0	0	2	0.03	$\boxed{0.02}$	$\boxed{0.02}$	$\boxed{0.02}$
HS36	3	1	0	0	1	0	0.03	$\boxed{0.01}$	$\boxed{0.01}$	0.03
HS37	3	2	0	0	2	0	$\boxed{0.01}$	0.04	0.02	0.02
HS41	4	1	1	0	0	0	$\boxed{0.01}$	0.02	$\boxed{0.01}$	0.02
HS42	4	2	1	1	0	0	0.03	0.06	$\boxed{0.02}$	0.03
HS43	4	3	0	0	0	3	0.02	0.03	$\boxed{0.01}$	$\boxed{0.01}$
HS46	5	2	0	2	0	0	0.08	0.22	$\boxed{0.03}$	0.09
HS54	6	1	1	0	0	0	0.08	0.12	$\boxed{0.01}$	$\boxed{0.01}$
HS57	2	1	0	0	0	1	0.02	0.02	0.02	0.02
HS59	2	3	0	0	0	1	$\boxed{0.01}$	0.03	0.02	0.02
HS60	3	1	0	1	0	0	0.05	0.05	$\boxed{0.02}$	$\boxed{0.02}$
HS61	3	2	0	2	0	0	$\boxed{0.03}$	0.05	0.05	0.06
HS62	3	1	1	0	0	0	0.02	0.03	$\boxed{0.01}$	$\boxed{0.01}$
HS64	3	1	0	0	0	1	0.04	0.04	$\boxed{0.01}$	0.02
HS68	4	2	0	2	0	0	0.04	0.09	$\boxed{0.01}$	0.02
HS69	4	2	0	2	0	0	0.06	0.55	0.02	$\boxed{0.01}$
HS71	4	2	0	1	0	1	0.03	0.04	$\boxed{0.01}$	$\boxed{0.01}$
HS83	5	3	0	0	0	3	$\boxed{0.01}$	0.07	0.03	0.05
HS84	5	3	0	0	0	3	0.02	0.03	$\boxed{0.01}$	$\boxed{0.01}$
HYDROELL	1009	1008	0	0	1008	0	10463	10389	$\boxed{0.49}$	$\boxed{0.51}$
HYDROELM	505	504	0	0	504	0	1558	1589	$\boxed{0.15}$	$\boxed{0.15}$
HYDROELS	169	336	0	0	336	0	66.41	67.84	$\boxed{0.04}$	$\boxed{0.04}$
HUBFIT	2	1	0	0	1	0	0.03	0.03	$\boxed{0.01}$	$\boxed{0.01}$
LAUNCH	25	28	6	3	12	7	49.76	51.46	10.74	$\boxed{1.33}$
LIN	4	2	2	0	0	0	0.02	$\boxed{0.01}$	$\boxed{0.01}$	0.02
LOADBAL	31	31	11	0	20	0	34.13	28.41	1.58	$\boxed{0.35}$
LOOTSMA	3	2	0	0	0	2	0.03	0.04	$\boxed{0.01}$	0.02
MADSEN	3	6	0	0	0	6	$\boxed{0.01}$	0.07	0.03	$\boxed{0.01}$
MARATOS	2	1	0	1	0	0	$\boxed{0.02}$	$\boxed{0.02}$	0.05	0.14
MATRIX2	6	2	0	0	0	2	0.03	$\boxed{0.02}$	$\boxed{0.02}$	0.03
MESH	41	48	4	20	24	0	186.5	381.7	92.58	$\boxed{40.35}$
MISTAKE	9	13	0	0	0	13	$\boxed{0.05}$	0.24	0.84	2.30
MRIBASIS	36	55	1	8	43	3	211	661	$\boxed{70.83}$	277
NGONE	8	8	0	0	2	6	0.03	0.05	$\boxed{0.01}$	0.02

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}	$CSA_{ID-ATCR}$	CLS	CLS_{ATCR}
			n_{le}	n_{ne}	n_{li}	n_{ni}	CPU time	CPU time	CPU time	CPU time
ODFITS	10	6	6	0	0	0	0.17	0.63	1.19	$\boxed{0.04}$
OPTPRLOC	30	30	0	0	5	25	0.21	2.34	$\boxed{0.02}$	0.06
ORTHREGB	27	6	0	6	0	0	$\boxed{1.34}$	4.01	2.71	9.21
PENTAGON	6	15	0	0	15	0	0.04	0.13	$\boxed{0.01}$	0.02
POLAK1	3	2	0	0	0	2	0.04	0.05	0.02	$\boxed{0.01}$
POLAK3	12	10	0	0	0	10	$\boxed{0.03}$	2.27	0.66	0.91
POLAK5	3	2	0	0	0	2	$\boxed{0.01}$	0.04	0.37	0.03
POLAK6	5	4	0	0	0	4	0.03	0.03	$\boxed{0.02}$	0.03
QC	9	4	0	0	4	0	0.02	0.02	$\boxed{0.01}$	$\boxed{0.01}$
ROBOT	14	2	0	2	0	0	0.27	0.31	$\boxed{0.19}$	0.94
S316-322	2	1	0	1	0	0	0.02	0.05	0.02	$\boxed{0.01}$
SINROSNB	2	1	0	0	0	1	0.1	$\boxed{0.02}$	$\boxed{0.02}$	0.03
SNAKE	2	2	0	0	0	2	0.02	0.04	0.02	$\boxed{0.01}$
SPIRAL	3	2	0	0	0	2	$\boxed{0.01}$	0.02	0.02	0.02
STANCMIN	3	2	0	0	2	0	$\boxed{0.01}$	0.04	0.02	0.02
SVANBERG	10	10	0	0	0	10	0.09	0.09	$\boxed{0.01}$	$\boxed{0.01}$
SYNTHES1	6	6	0	0	4	2	0.04	0.04	$\boxed{0.01}$	$\boxed{0.01}$
SYNTHES2	11	14	1	0	10	3	0.3	0.4	0.03	0.03
SYNTHES3	17	23	2	0	17	4	0.2	0.53	$\boxed{0.02}$	$\boxed{0.02}$
TWOBARS	2	2	0	0	0	2	$\boxed{0.01}$	0.04	$\boxed{0.01}$	0.02
WOMFLET	3	3	0	0	0	3	$\boxed{0.01}$	$\boxed{0.01}$	0.02	$\boxed{0.01}$
ZAMB2-10	270	96	0	96	0	0	2245	8838	$\boxed{66.72}$	68.41
ZAMB2-8	138	48	0	48	0	0	567	606	967	$\boxed{311}$
ZECEVIC3	2	2	0	0	0	2	$\boxed{0.01}$	0.03	0.03	0.02
ZECEVIC4	2	2	0	0	1	1	$\boxed{0.01}$	0.02	0.02	0.03
ZY2	3	2	0	0	0	2	$\boxed{0.01}$	0.02	0.03	$\boxed{0.01}$

Table 4.5: Results comparing CSA_{ID}, $CSA_{ID-ATCR}$, CLS, and CLS_{ATCR} in terms of violations and CPU times when applied to solve mixed-integer constrained NLPs derived from selected continuous problems in CUTE using the starting point specified in each problem. The violations are the maximum violation across all the constraints of the solution. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7. $'-'$ means that no solution could be found after 24 hours.

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}		$CSA_{ID-ATCR}$		CLS		CLS_{ATCR}	
			n_{le}	n_{ne}	n_{li}	n_{ni}	violation	time	violation	time	violation	time	violation	time
ALJAZZAF	3	1	0	1	0	0	$\boxed{0.0}$	0.04	$\boxed{0.0}$	0.14	10001	0.67	10001	0.67
AVION2	49	15	15	0	0	0	0.35	182.8	0.063	186.3	$\boxed{0.0059}$	144.6	0.021	38.93
BT11	5	3	1	2	0	0	$\boxed{0}$	0.06	$\boxed{0}$	0.15	0.76	18.9	$\boxed{0}$	0.05
BT12	5	3	0	3	0	0	$\boxed{0}$	0.38	$\boxed{0}$	1.36	1	18.95	2	0.9
BT6	5	2	0	2	0	0	1.82	5.88	7.08	5.41	$\boxed{0}$	0.02	$\boxed{0}$	0.02
BT7	5	3	0	3	0	0	$\boxed{0.5}$	2.5	$\boxed{0.50}$	2.58	$\boxed{0.5}$	18.11	1.5	19.5
CB2	3	3	0	0	0	3	$\boxed{0}$	0.01	$\boxed{0}$	0.05	3.0	3.12	1.0	3.59
DEMBO7	16	20	0	0	0	20	$\boxed{0}$	0.63	$\boxed{0}$	0.87	0.02	286.7	0.02	300
DIXCHLNG	10	5	0	5	0	0	$\boxed{0}$	0.02	1	18.84	$\boxed{0}$	0.01	$\boxed{0}$	0.02
DNIEPER	61	24	0	24	0	0	0.022	2087	4.5E-4	2967	4.7E-4	153.6	$\boxed{1.3E-4}$	178
ERRINBAR	18	9	0	8	1	0	336	151.1	$\boxed{3.95}$	173	4.61	57.75	12.3	54.0
FEEDLOC	90	259	4	15	166	74	-	-	-	-	$\boxed{4.95E-5}$	10513	0.015	9948
FLETCHER	4	4	0	1	3	0	$\boxed{0}$	0.01	$\boxed{0}$	0.13	0.012	121.2	0.015	124
GIGOMEZ2	3	3	0	0	0	3	$\boxed{0}$	0.02	$\boxed{0}$	0.3	2	3.4	2	2.75
HS107	9	6	0	6	0	0	0.36	224	0.27	152	0.39	10.96	$\boxed{0.23}$	10.9
HS19	2	2	0	0	0	2	$\boxed{0}$	0.01	$\boxed{0}$	0.03	0.40	0.56	0.26	0.57
HS26	3	1	0	1	0	0	2.0	0.87	2.0	0.90	$\boxed{0}$	0.01	$\boxed{0}$	0.02
HS39	4	2	0	2	0	0	$\boxed{0}$	0.07	$\boxed{0}$	0.22	4.0	0.68	4.0	0.68
HS40	4	3	0	3	0	0	$\boxed{0}$	0.07	$\boxed{0}$	0.51	0.20	3.79	0.20	5.05
HS55	6	6	6	0	0	0	0.5	13.25	0.45	11.02	$\boxed{0.33}$	4.66	$\boxed{0.33}$	1.39
HS56	7	4	0	4	0	0	2.78	39.06	$\boxed{4.0E-3}$	37.08	1.28	3.26	1.28	3.21
HS63	3	2	1	1	0	0	2.64	2.57	$\boxed{0.13}$	2.24	3.74	2.31	5.6	2.36
HS7	2	1	0	1	0	0	$\boxed{0}$	0.05	$\boxed{0}$	0.03	21	9.04	21	6.35
HS73	4	3	1	0	1	1	$\boxed{0}$	0.06	$\boxed{0}$	0.26	1	0.53	1	0.53
HS74	4	5	0	3	2	0	0.0056	16.23	$\boxed{3.2E-3}$	7.95	271	4.63	209	4.47
HS75	4	5	0	3	2	0	0.0035	22.73	$\boxed{8.5E-5}$	9.43	264	2.34	230	2.75
HS78	5	3	0	3	0	0	$\boxed{0}$	0.13	$\boxed{0}$	0.26	2.01	2.51	2.01	2.49
HS79	5	3	0	3	0	0	2	2.53	26.25	2.51	3.76	12.98	$\boxed{1.}$	0.95
HS80	5	3	0	3	0	0	0.39	22.51	$\boxed{1.21E-5}$	19.42	1.3E-5	10.88	2.75E-5	15.06
HS87	6	4	0	4	0	0	2.4E-3	26.28	$\boxed{4.8E-4}$	41.74	1.18	10.93	0.12	81.8
HS93	6	2	0	0	0	2	2.07	3.02	2.07	2.98	0.38	5.91	$\boxed{0.037}$	9.51
LEWISPOL	6	9	0	9	0	0	$\boxed{0}$	17.97	4.88E-5	28.45	2.5E-5	66.68	3.01E-5	69.77

continued on next page

continued from previous page

Problem ID	n_v	n_c	$h(x)$		$g(x)$		CSA_{ID}		$CSA_{ID-ATCR}$		CLS		CLS_{ATCR}	
			n_{le}	n_{ne}	n_{li}	n_{ni}	violation	time	violation	time	violation	time	violation	time
MWRIGHT	5	3	0	3	0	0	2.0	2.46	32.6	2.44	$\boxed{0}$	0.01	$\boxed{0}$	0.03
NET1	48	57	21	17	16	3	1830	2994	$\boxed{0.45}$	1652	1	662	1.85	318
NET2	144	160	64	59	32	5	7.38	10451	7.38	10371	$\boxed{1}$	2958	10.5	3347
NET3	464	521	195	199	110	17	-	-	30.5	24137	21	15725	$\boxed{12}$	14795
OPTCNTRL	32	20	10	10	0	0	6	338.5	2	406.9	$\boxed{0.83}$	21.88	1.8	5.83
READING6	102	50	0	50	0	0	222.9	19915	4.57	31747	$\boxed{0.39}$	151.2	0.55	84.6
RK23	17	11	4	7	0	0	$\boxed{0.50}$	26.7	$\boxed{0.5}$	30.46	0.60	9.75	0.60	9.72
TENBARS4	18	9	0	8	1	0	1890	45.58	$\boxed{0.047}$	428	253.2	10.22	197.7	43.3
TWIRIMD1	1247	544	143	378	5	186	-	-	-	-	1.1E-4	48041	$\boxed{1.0E-4}$	102395
TWIRISM1	343	313	50	174	5	84	-	-	-	-	$\boxed{1.0E-4}$	4847	$\boxed{1.0E-4}$	13472
ZAMB2-11	270	96	0	96	0	0	52.87	31402	2.3E-3	24440	$\boxed{0}$	347.8	$\boxed{0}$	186.9

4.4 Summary

In this chapter, we have developed a new MINLP algorithm, CLS, which uses line search to look for the minimum in the x sub-space, different strategies in the continuous and discrete sub-space to handle mixed integer problem, and a trap-escaping strategy to escape from local minima. CLS is faster than CSA in solving most of the test problems.

We have also studied constraint relaxations in CLS. First, we have studied the relationship between relaxation levels and times to find feasible solutions using CLS and have built a log-log model. Based on this model, we have developed an anytime CLS algorithm that can generate a solution with reduced violations as more time is given. Finally, we have proved that the time spent using CLS_{AT} on a sequence of relaxation levels is of the same order of magnitude as the time spent by CLS solving the problem with the last relaxation level alone.

Chapter 5

Noisy Constraint Optimization

In the optimization problems studied in the last two chapters, we assume that their objectives and constraints can be computed exactly at every point, either through a closed form or through a procedure. However, in a variety of problems, their objectives and constraints can only be evaluated in a stochastic fashion. In this case, it is very difficult to find a satisfiable solution, as in traditional optimization. This chapter addresses optimization problems when objectives and constraints are stochastic.

5.1 Introduction

Noisy constraint optimizations are widely used in real applications. In some cases, noise is due to inaccuracies in real evaluations, whereas in other cases, it is due to evaluations that are made up of Monte Carlo methods or simulations.

5.1.1 Problem definition

Generally, a noisy constraint optimization can be formulated as follows:

$$\begin{aligned} \text{minimize} \quad & E(f(x)) \\ \text{subject to} \quad & E(g(x)) \leq 0 \\ & E(h(x)) = 0, \end{aligned} \tag{5.1}$$

where $E(f(x))$ represents the expectation of the objective function's value at point x, $x = (x_1, x_2, \cdots, x_n)^T$, $g = (g_1, g_2, \cdots, g_k)^T$, and $h = (h_1, h_2, \cdots, h_m)^T$. Note that at any given point x, only $f,\ g,\ h$ can be observed, and that their observations at different points may be different but follow a fixed distribution. We assume that the distributions for $f,\ g$, and h are independent normal distributions, $f \sim \mathcal{N}(E(f), \sigma_f)$, $g \sim \mathcal{N}(E(g), \sigma_g)$, $h \sim \mathcal{N}(E(h), \sigma_h)$. All the parameters, $E(f),\ E(g),\ E(h),\ \sigma_f,\ \sigma_g,\ \sigma_h$ are unknown but fixed. We further assume that $\sigma_f,\ \sigma_g$, and σ_h are uniform in the search space.

5.1.2 Constraint satisfaction

In traditional optimization problems without noise, we assume that a constraint is satisfied when its violation is less than a prescribed small value τ, saying 10^{-5}. In noisy optimizations, it is very difficult to achieve such a goal due to uncertainties in function evaluations.

Since we have no analytic way to access $E(f),\ E(g),\ E(h)$ exactly, we have to use their observed values instead. As one observation of a function may be very inaccurate, we will need to observe multiple samples and take their average as an estimation of the expected value. In the following, we use an equality constraint function $h(x)$ as an an example.

Suppose at given point x, $h(x) = E(h(x)) + \epsilon$, where ϵ is a normally distributed noise function with mean 0 and variance σ_h^2. In traditional constraint satisfaction on constraint $E(h(x)) \le \tau$, we have to estimate $E(h(x))$ accurately enough such that the standard deviation of the estimation is at least less than τ. The only way to improve the accuracy of the estimation is to observe $h(x)$ multiple times and take their average: $\bar{h}(x) = E(h(x)) + \frac{\epsilon}{\sqrt{n}}$. To estimate $E(h(x))$, we have to make the error on $\frac{\epsilon}{\sqrt{n}}$ sufficiently small, which in turn requires a substantially large number of samples. For example, suppose the original standard deviation of noise ϵ is 1, if we want the violation tolerance τ to be 10^{-5}, we have to reduce the noise level $\frac{\epsilon}{\sqrt{n}}$ to at least 10^{-5}, leading to an unacceptably large number of 10^{10} samples at each point. Even when we reduce the violation tolerance τ to 10^{-3}, we will still need 10^6 samples at each point!

The above example shows that it is not reasonable to use the traditional constraint satisfaction criteria. As a result, we propose in the following a new constraint satisfaction criteria using the concept of confidence intervals.

For equality constraint $h(x) = 0$, suppose we observe N samples $h^i(x), i = 1, \cdots, N$, at point x, where $\bar{h}$ is the sample mean and $\hat{\sigma}^2$ is the sample variance. We can construct a $(1-\alpha)100\%$ confidence interval (LL, UU) where:

$$\begin{aligned} LL &= \bar{h} - \frac{t_{1-\alpha/2,N-1}\hat{\sigma}}{\sqrt{N}} \\ UU &= \bar{h} + \frac{t_{1-\alpha/2,N-1}\hat{\sigma}}{\sqrt{N}}. \end{aligned} \tag{5.2}$$

If confidence interval (LL, UU) does not cover 0, then we will reject the hypothesis $H_0 : E(h(x)) = 0$; otherwise, we accept it and assume that constraint $h(x) = 0$ is satisfied.

Similarly, for inequality constraint $g(x) \leq 0$, suppose we observe N samples $g^i(x),\ i = 1, \cdots, N$, at point x, where $\bar{g}$ is the sample mean and $\hat{\sigma}^2$ is the sample variance. We can construct a $(1-\alpha)100\%$ confidence interval (LL, UU) where:

$$\begin{aligned} LL &= -\infty \\ UU &= \bar{g} + \frac{t_{1-\alpha,N-1}\hat{\sigma}}{\sqrt{N}}. \end{aligned} \tag{5.3}$$

If the confidence interval (LL, UU) does not cover 0, we reject the hypothesis $H_0 : E(g(x)) \leq 0$; otherwise we accept it and assume that constraint $g(x) \leq 0$ is satisfied. When confidence level α is fixed, the estimated standard deviation of the mean $\bar{h}$, $\hat{\sigma}(\bar{h}) = \hat{\sigma}/\sqrt{N}$, will decide the width of the confidence interval and the accuracy of estimating $E(h)$ using $\bar{h}$. In other words, when $\hat{\sigma}/\sqrt{N}$ is large, the confidence interval is wide, and it is very inaccurate to estimate $E(h)$ using sample average $\bar{h}$. On the other hand, when $\hat{\sigma}/\sqrt{N}$ is small, the estimated $E(h(x)$ using $\bar{h}$ is accurate, and the constraint can be regarded satisfied. When $\hat{\sigma}/\sqrt{N}$ is 0, the confidence interval for h becomes one point $\bar{h}$, and the confidence interval for g becomes $(-\infty, \bar{g})$. Hence, both $\bar{h}$ and $\bar{g}$ become the exact estimates of $E(h)$ and $E(g)$.

In theory, we hope the width of a confidence interval to go to 0 when time goes to infinity. In practice, we require $\hat{\sigma}/\sqrt{N} \leq \sigma_f$ when the algorithm finishes.

There are several ways to satisfy this requirement on the width of a confidence interval or $\hat{\sigma}/\sqrt{N}$. One simple method is to take enough samples at each point from the beginning in such a way that $\hat{\sigma}/\sqrt{N}$ is always equal to or less than σ_f. But we know that it may not be necessary to make an accurate estimate of all constraints in the beginning of a search, since most of them are not satisfied. A better approach is to take fewer samples at each point and use a larger confidence interval in the beginning of a search. Then we increase the number of samples and decrease the size of confidence interval gradually to the required final confidence interval size σ_f when the algorithm ends.

5.1.3 Performance measures

In optimization without noise, it is natural to use the number of iterations as a performance measure. However, when noise exists, each iteration may include drawing multiple samples, and the number of samples in each iteration may vary. Hence, it is not reasonable to use the number of iterations as a performance measure. As a result, we use the total number of samples as our measure instead.

Suppose in the i'th iteration, we take $N(i)$ samples and compute the average and the confidence interval for each constraint and objective. In the first alternative, we use a fixed schedule that draws N samples in each iteration, i.e., $N(1) = N(2) = \cdots = N(I)$. Consequently, $\sigma(1)/\sqrt{N} = \sigma(2)/\sqrt{N} = \cdots = \sigma(I)/\sqrt{N}$. In the second alternative, we use an increasing sampling schedule: $N(1) \leq N(2) \leq \cdots \leq N(I)$, leading to decreasing confidence intervals. The total number of samples drawn is:

$$Samples = \sum_{i=1}^{I} N(i). \tag{5.4}$$

Although the number of samples is an important characteristic of an algorithm, it is not adequate because not all algorithms can find a solution with 100% probability. Suppose Pr is the probability to find a solution with a prescribed quality. What is important is the expected total number of samples , that is, $Samples/Pr$. Suppose algorithm A takes 10,000 samples and its success probability to find a solution is 1%, and algorithm B takes 100,000 samples and its success probability is 50%. Then B is preferred because we only need to run B twice (20,000 samples) on the average in order to get a solution, assuming all runs are independent, whereas A will take 1,000,000 samples on the average to find a solution. In this chapter, we use $Samples/Pr$ as our measure to evaluate alternative algorithms.

5.1.4 Problem formulation

Since the number of samples plays an important role in performance measure $Samples/Pr$, it is critical to determine its value in each iteration appropriately. As an example, consider the application of CSA (discussed in Chapter 3) to the following problem:

$$\begin{array}{lll} \text{minimize} & E(f(x)) = E(F(10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)), 190) + \epsilon(x)) & \\ \text{subject to} & \quad E(\sum_{i=1}^{n} |x_i - 3.8| + \epsilon(x)) \leq 0.1 \quad \text{for } n = 10, & (5.5) \end{array}$$

where

$$F(y, f') = \begin{cases} f', & \text{if } y \leq f' \\ y, & \text{otherwise.} \end{cases}$$

We have tried three schedules. In the first schedule, each iteration uses the same number of samples, that is, $N(i) = N_f$ for $i = 1, \cdots, I$, where N_f is the number of samples that satisfies $\sigma/\sqrt{N_f} \leq \sigma_f$. Accordingly, $\sigma(i)/\sqrt{N}$ is constant. In the second schedule, we use a linearly increasing number of samples in each iteration; that is, $N(i) = \lfloor i/C \rfloor$, where $C = I/N_f$. Hence, $\sigma(i)/\sqrt{N} = \sigma_0/\sqrt{\lfloor i/C \rfloor}$. The third schedule increases the number of samples exponentially; i.e., $N(i) = \lfloor \exp(2i/D) \rfloor$. Hence, $\sigma(i)/\sqrt{N} = \sigma_0/\sqrt{\lfloor \exp(2i/D) \rfloor} \approx \sigma_0/\exp(i/D)$, where D is chosen such that when the algorithm finishes, $\sigma(i)/\sqrt{N} = \sigma_f$.

Figure 5.1a plots $\sigma(i)/\sqrt{N}$ for each sampling schedule with various I's. For each sampling schedule, we have tried four cooling schedules in CSA. The solid lines show sampling schedule 1, in which the number of samples drawn in each iteration is constant, and so is $\sigma(i)/\sqrt{N}$. The dashed lines show sampling schedule 2, in which the number of samples increases linearly with respect to the number of iterations and $\sigma(i)/\sqrt{N}$ decreases at a rate proportional to a square root of the number of iterations. We have shown four cooling schedules (with different I) in which the leftmost curve represents the fastest cooling schedule. Finally, the dotted lines show sampling schedule 3, in which sample numbers increase exponentially with respect to the number of iterations and $\sigma(i)/\sqrt{N}$ decreases exponentially. The rightmost dotted curve represents the slowest cooling schedule (with the largest I).

For each cooling schedule and each sampling schedule, we ran CSA 100 times and recorded its average numbers of samples and probabilities to find feasible solutions. Figure 5.1b plots $Samples/Pr$ for each cooling schedule and each sampling schedule, where each curve represents one sampling schedule and a point on the curve represents the performance of a cooling schedule for that sampling schedule. We have found that there exists an optimal cooling schedule that minimize $Samples/Pr$, for each sampling schedule. However, the minimum $Samples/Pr$ for different sampling schedules varies greatly. From the plot, we see that $Samples/Pr$ is minimized for sampling schedule 3 by one or two order of magnitude, as compared to those of the other two sampling schedules. Thus, we use sampling scheduling 3 in our experiments discussed in the rest of this chapter.

Our goal in this chapter is to study the performance of CSA when applied to solve stochastic optimization functions and to propose an anytime algorithm that can find a solution with a larger uncertainty when given shorter time and find a solution with a smaller uncertainty when given longer time.

Our approach is to first study the relationship between $Samples/Pr$ and the width of the final confidence interval σ_f. We then develop our algorithm based on the relationship found.

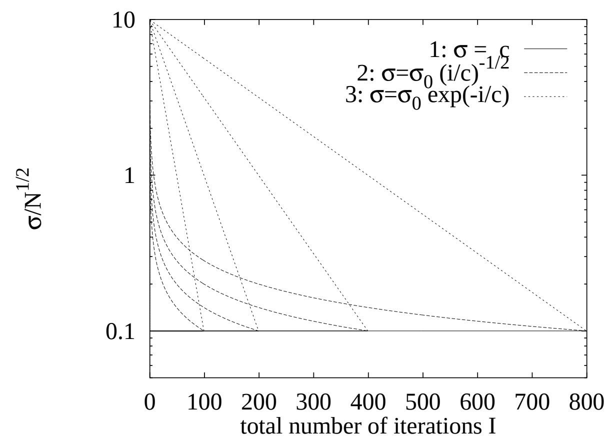
We first show some theoretical results on the asymptotic convergence of our proposed algorithm. The theoretical results show that if we choose a slow enough sampling schedule, then the modified CSA algorithm can preserve the property of asymptotic convergence to the global minimum in a noisy situation.

5.2 Theoretical Results

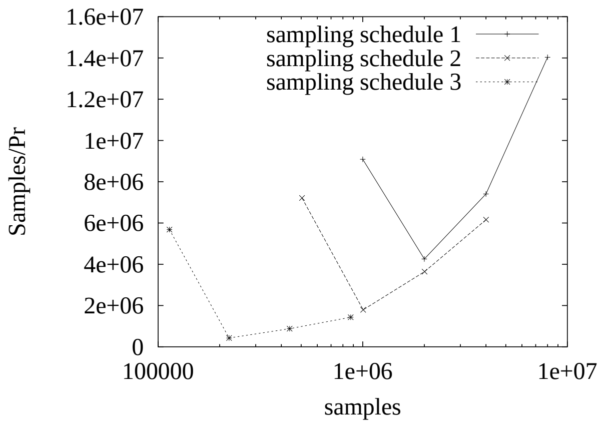
5.2.1 Modified CSA algorithm 1

We have discussed the CSA algorithm and its anytime version in the previous chapter. As an initial step, we modify CSA to solve (5.1). Since we cannot get the exact objective and constraint values at any given point x, we have to take some number of samples and use their average to estimate the values. Here, we only consider equality constraints. Figure 5.2 shows the algorithm. Accordingly, the Lagrangian function and acceptance probability are as follows:

$$L(x) = \bar{f}(x) + \lambda^T H(\bar{h}(x)) + \frac{1}{2}\bar{h}^2(x)$$



a) $\sigma(i)/\sqrt{N}$ of each sampling schedule



b) Effects of different sampling schedules on $Sample/Pr$

Figure 5.1: Performance of three different sampling schedules

$$L'(x) = \bar{f}(x') + \lambda^T H(\bar{h}(x)) + \frac{1}{2}\bar{h}^2(x)$$

$$A'(x,x') = \begin{cases} 1, & \text{if } L'(x) \le L(x) \\ \exp\left(-\frac{(L'(x)-L(x))^+}{T}\right), & \text{otherwise.} \end{cases} \tag{5.6}$$

The algorithm increases $N(i)$, the number of samples drawn at each point, as temperature is decreased.

5.2.2 Convergence theorem

We prove under certain conditions that, our modified CSA algorithm 1 will retain the asymptotic convergence property as the original CSA algorithm. In the proof, we take the Lagrangian function without the augmented term. The proof can be done similarly when the augmented term is included. In order to prove our result, we state some theorems and lemmas first.

Aarts and Korst have proved the following asymptotic convergence theorem for simulated annealing [16].

Theorem 5.1 [16] Assume that the following conditions hold in the simulated annealing algorithm:

```
1.  procedure CSA_with_Noise_1
2.      set starting point $\mathbf{x} = (x, \lambda)$;
3.      set starting temperature $T = T^0$ and cooling rate $0 < \alpha < 1$;
4.      set $N_T$ (number of trials per temperature);
5.      $I = 0$
6.      while stopping condition is not satisfied do
7.          for $j \leftarrow 1$ to $N_T$ do
8.              $I = I + 1$
9.              generate trial point $\mathbf{x}'$ from $\mathcal{N}(\mathbf{x})$ using $q(\mathbf{x}, \mathbf{x}')$;
10.             generate $N(I)$ independent observations of $f$ and $h$ on both $\mathbf{x}$ and $\mathbf{x}'$.
11.             accept $\mathbf{x}'$ with probability $A'(\mathbf{x}, \mathbf{x}')$
12.         end_for
13.         reduce temperature by $T \longleftarrow \alpha \times T$ ;
14.         update $N(I)$;
15.     end_while
16. end_procedure
```

Figure 5.2: Modified CSA algorithm 1

(i) The underlying graph representing the Markov chain is connected. The nodes of the graph are the states x of the Markov chain, and its edges are defined by the neighborhood relations in which (x, x') is an edge if and only if $x' \in \mathcal{N}(x)$.
(ii) The sequence of temperature T_m satisfies:

$$T_m \geq \frac{(L+1)\Delta}{\log(m+2)} \quad (m = 0, 1, \cdots),$$

where

$$\Delta = \max_{x \in S, x' \in \mathcal{N}(x)} (f(x') - f(x)), \text{ } S \text{ is the search space,}$$

and L is the minimum number of transitions required to reach an optimal solution from an arbitrary starting point.
Then, for an arbitrary initial distribution $q(0)$, the distribution of the current solution con-

verges to the uniform distribution on the set S_{opt} of global optimizers:

$$\lim_{k\to\infty} q(k) = q(0)P(1)\dots P(k) = q^*. \tag{5.7}$$

Gutjahr and Pflug [80] further proved the following convergence lemma for simulated annealing in a noisy situation. Assume that, in a noisy situation, the acceptance probability in simulated annealing uses $\exp\left(-\frac{(\tilde{f}_k(x')-\tilde{f}_k(x))^+}{T}\right)$, instead of $\exp\left(-\frac{(f(x')-f(x))^+}{T}\right)$, where $\tilde{f}_k(x)$ is an evaluation of f with noise, defined as $\tilde{f}_k(x) = f(x) + \epsilon_k(x)$. Further, assume that the noise variable $\epsilon_k(x)$ is independent and $\mathcal{N}(0, \sigma_k^2)$ distributed.

Lemma 5.1 [80] Assume the conditions in Theorem 5.1 are satisfied, $P(k)$ is the transition matrix of the Markov chain modeling simulated annealing, and $\tilde{P}(k)$ is the transition matrix of the Markov chain modeling simulated annealing in a noisy situation. If

$$\sum_{k=1}^{\infty} ||P(k) - \tilde{P}(k)|| < \infty, \tag{5.8}$$

where $||A|| = \max_i \sum_j |a_{ij}|$ for $A = (a_{ij})$, then:

$$\lim_{k\to\infty} q(k) = q(0)P(1)\dots P(k) = q^*.$$

In the above lemma and its proof, there is no special requirements on transition matrix $P(k)$. Since the transition matrix modeling the Markov chain of CSA satisfies the same requirements as that of simulated annealing, this lemma can also be applied to CSA.

In CSA, the transition matrix of the Markov chain modeling state transition $P(k) = (P_{ij}(k))$ is given by:

$$P_{ij}(k) = \begin{cases} q(i,j)\exp\left(-\frac{(L_j-L_i)^+}{T_k}\right), & \text{if } x_i \neq x_j,\ \lambda_i = \lambda_j \\ q(i,j)\exp\left(-\frac{(L_i-L_j)^+}{T_k}\right), & \text{if } x_i = x_j,\ \lambda_i \neq \lambda_j, \end{cases} \tag{5.9}$$

where $q(i,j)$ is the probability of generating a trial point to j from i.

In our modified CSA algorithm 1, we still use a Markov chain to model state transitions, and transition matrix $\tilde{P}(k) = (\tilde{P}_{ij}(k))$ is given by:

$$\tilde{P}_{ij}(k) = \begin{cases} q(i,j)\exp\left(-\frac{(\tilde{L}_j-\tilde{L}_i)^+}{T_k}\right), & \text{if } x_i \neq x_j,\ \lambda_i = \lambda_j \\ q(i,j)\exp\left(-\frac{(\tilde{L}_i-\tilde{L}_j)^+}{T_k}\right), & \text{if } x_i = x_j,\ \lambda_i \neq \lambda_j, \end{cases} \tag{5.10}$$

where $\tilde{L}_i$ is the Lagrangian value with noise at point i.

In the following, we show that transition matrix $\tilde{P}(k)$ can satisfy (5.8) in Lemma 5.1. The proof is done in two parts. In the first part, we only allow the objective function to be stochastic. In the second part, both the objective and constraints are stochastic.

First, suppose only the objective has noise and $\tilde{P}_{ij}(k)$ is the transition probability from state i to j. In this case, we prove the following lemma:

Lemma 5.2

$$P_{ij} - \tilde{P}_{ij} = \begin{cases} q(i,j)\left(\exp(-\frac{(L_j-L_i)^+}{T}) - \Phi(-\frac{L_j-L_i}{s})\right) - \\ q(i,j)\exp\left(\frac{s^2}{2T^2} - \frac{(L_j-L_i)}{T}\right)\left[1-\Phi(\frac{s}{T}-\frac{L_j-L_i}{s})\right], \text{ if } x_i \neq x_j \text{ and } \lambda_i = \lambda_j, \\ q(i,j)\left(\exp(-\frac{(L_i-L_j)^+}{T}) - \Phi(-\frac{(L_i-L_j)}{s})\right) - \\ q(i,j)\exp\left(\frac{s^2}{2T^2} - \frac{L_i-L_j}{T}\right)\left[1-\Phi\left(\frac{s}{T}-\frac{L_i-L_j}{s}\right)\right], \text{ if } x_i = x_j \text{ and } \lambda_i \neq \lambda_j, \end{cases} \tag{5.11}$$

where $s = 2\sigma$ and σ^2 is the variance of the noise.

Proof. From (5.9), we have:

$$P_{ij} = \begin{cases} q(i,j)\exp\left(-\frac{(L_j-L_i)^+}{T}\right), & \text{if } x_i \neq x_j, \\ q(i,j)\exp\left(-\frac{(L_i-L_j)^+}{T}\right), & \text{if } x_i = x_j. \end{cases}$$

Now we compute $\tilde{P}_{ij}$. Here $\tilde{L}(\mathbf{x},\lambda) = f(\mathbf{x}) + \epsilon + \lambda|h(\mathbf{x})|$, with independent noise ϵ_i (for L_i) and ϵ_j (for L_j)$\sim N(0,\sigma)$. Hence, the transition probability for given ϵ_i and ϵ_j is

$$\tilde{P}_{ij}^{(\epsilon_i,\epsilon_j)} = \begin{cases} q(i,j)\exp\left(-\frac{(L_j-L_i+\epsilon_j-\epsilon_i)^+}{T}\right), & \text{if } x_i \neq x_j,\ \lambda_i = \lambda_j, \\ q(i,j)\exp\left(-\frac{(L_i-L_j+\epsilon_i-\epsilon_j)^+}{T}\right), & \text{if } x_i = x_j,\ \lambda_i \neq \lambda_j, \end{cases} \tag{5.12}$$

$$= \begin{cases} q(i,j)\exp\left(-\frac{(L_j-L_i+sz)^+}{T}\right), & \text{if } x_i \neq x_j,\ \lambda_i = \lambda_j, \\ q(i,j)\exp\left(-\frac{(L_i-L_j+sz)^+}{T}\right), & \text{if } x_i = x_j,\ \lambda_i \neq \lambda_j. \end{cases} \tag{5.13}$$

where $z \sim N(0,1)$, and $s = 2\sigma$. Let $\Delta L = L_j - L_i$. If $x_i \neq x_j$, then

$$\begin{aligned} \frac{\tilde{P}_{ij}}{q(i,j)} &= \int_{-\infty}^{\infty} \exp\left(-\frac{(\Delta L + sz)^+}{T}\right)\phi(z)dz \\ &= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(\Delta L + sz)^+}{T} - \frac{z^2}{2}\right)dz \\ &= \int_{-\infty}^{-\Delta L/s} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right)dz + \int_{-\Delta L/s}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\Delta L + sz}{T} - \frac{z^2}{2}\right)dz \\ &= \Phi\left(-\frac{\Delta L}{s}\right) + \exp\left(\frac{s^2}{2T^2} - \frac{\Delta L}{T}\right)\left[1 - \Phi\left(\frac{s}{T} - \frac{\Delta L}{s}\right)\right]. \end{aligned} \tag{5.14}$$

Hence, the first equation in Lemma 5.2 is proved.

Similarly, if $x_i = x_j$ then

$$\frac{\tilde{P}_{ij}}{q(i,j)} = \Phi\left(-\frac{\Delta L}{s}\right) + \exp\left(\frac{s^2}{2T^2} - \frac{\Delta L}{T}\right)\left[1 - \Phi\left(\frac{s}{T} - \frac{\Delta L}{s}\right)\right], \tag{5.15}$$

and the second equation in Lemma 5.2 is proved. Thus, this Lemma is proved.

In the following, we use the above result to prove that P_{ij} and $\tilde{P}_{ij}$ satisfy (5.8) in Lemma 5.1.

Theorem 5.2 Assume in the inhomogeneous Markov chain modeling the modified CSA algorithm in Figure 5.2 that σ_k is $O(k^{-\gamma})$ with fixed $\gamma > 1$. Further, assume in the Markov chains modeling the modified CSA and the original CSA that $T_k = C(\log k)^{-1}$, where C is a large enough positive constant. Then

$$\sum_{k=1}^{\infty} |P_{ij}(k) - \tilde{P}_{ij}(k)| < \infty$$

for all $i, j \in S$.

Proof. We only prove the case when $x_i \neq x_j$, and the proof of the case when $x_i = x_j$ is similar and simpler. Firstly, recall that

$$s_k \;=\; 2\sigma_k = O(k^{-\gamma}), \tag{5.16}$$

Thus,

$$s_k \;\leq\; \frac{1}{C_1 k}, \tag{5.17}$$

where C_1 and C_2 are positive constants.
Case (i): $L_i = L_j$. From Lemma 5.2, we get

$$\frac{|P_{ij} - \tilde{P}_{ij}|}{q_{ij}} = \left| \exp\left(\frac{s_k^2}{2T_k^2}\right)\left(1 - \Phi\left(\frac{s_k}{T_k}\right)\right) - \frac{1}{2} \right|.$$

If we let $x = \frac{s_k}{T_k}$, we get

$$\frac{|P_{ij}(k) - \tilde{P}_{ij}(k)|}{q_{ij}} = \left| \exp\left(\frac{x^2}{2}\right)\Phi(-x) - \frac{1}{2} \right| = \exp\left(\frac{x^2}{2}\right)\Phi(-x) - \frac{1}{2}$$

(since $\exp(\frac{x^2}{2}) > 1$ and $\Phi(-x) > \frac{1}{2}$). Since $x = \frac{s_k}{T_k} \to 0$ as $k \to \infty$, $\exp\left(\frac{x^2}{2}\right) = 1 + O(x^2)$ and $\Phi(x) = \frac{1}{2} + \frac{x}{\sqrt{2\pi}} + O(x^2)$ (as $x \to 0$), we get:

$$|\tilde{P}_{ij} - P_{ij}| = q_{ij}\left(-\frac{x}{\sqrt{2\pi}} + O(x^2)\right) \quad (\text{as } x \to 0).$$

Therefore, for sufficiently large k,

$$\sum_k |P_{ij}(k) - P_{ij}(k)| \;=\; O\left(\sum_k |x|\right)$$

$$\begin{aligned} &= O\left(\sum_k \frac{s_k}{T_k}\right) \\ &= O\left(\sum_k k^{-\gamma} \cdot \log k\right) \\ &\leq O\left(\sum_k k^{-\frac{\gamma+1}{2}}\right). \end{aligned} \tag{5.18}$$

Since $\frac{\gamma+1}{2} > 1$, $\sum_k |P_{ij}(k) - \tilde{P}_{ij}(k)|$ converge to a finite value.

Case (ii): $L_i > L_j$ Let

$$\begin{aligned} D_k &= \Phi\left(-\frac{L_j - L_i}{s_k}\right) - \exp\left(-\frac{(L_j - L_i)^+}{T_k}\right), \\ E_k &= \exp\left(\frac{s_k^2}{2T_k^2} - \frac{L_j - L_i}{T_k}\right)\left[1 - \Phi\left(\frac{s_k}{T_k} - \frac{(L_j - L_i)}{s_k}\right)\right]. \end{aligned} \tag{5.19}$$

From Lemma 5.2, we get $\tilde{P}_{ij} - P_{ij} = q(i,j)(D_k + E_k)$.

First, since $L_i > L_j$, $(L_j - L_i)^+ = 0$. Further, since the search space is finite and f, g, h, and λ are finite, $L_i - L_j$ is finite. Moreover, from (5.17), we get $\frac{1}{s_k} \geq C_1 k$. Hence,

$$|D_k| = \left|\Phi\left(-\frac{L_j - L_i}{s_k}\right) - 1\right| = \Phi\left(\frac{L_j - L_i}{s_k}\right) \leq \Phi(-(L_i - L_j) \cdot C_1 k) = \Phi(-C_3 k),$$

where C_3 is a positive constant. Since

$$\Phi(-x) \leq \frac{1}{|x|\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right), \tag{5.20}$$

$|D_k| \to 0$ (as $k \to \infty$) at an exponentially decreasing rate. Consequently, $\sum_k |D_k| \leq \infty$ holds.

Now, for sufficiently large k, $\frac{s_k^2}{T_k^2} = O(k^{-2\gamma} \log^2 k) = O(1)$. Thus,

$$\begin{aligned} 0 \leq E_k &= \exp\left(\frac{s_k^2}{2T_k^2}\right) \exp\left(\frac{-(L_j - L_i)}{T_k}\right) \Phi\left(\frac{L_j - L_i}{s_k} - \frac{s_k}{T_k}\right) \\ &\leq C_4 \cdot \exp\left(-\frac{L_j - L_i}{T_k}\right) \Phi\left(\frac{L_j - L_i}{s_k}\right), \end{aligned} \tag{5.21}$$

where C_4 is a positive constant. Based on (5.17) and (5.20), we have, for sufficiently large k, that,

$$\begin{aligned}
\exp\left(-\frac{L_j - L_i}{T_k}\right)\Phi\left(\frac{L_j - L_i}{s_k}\right) &\leq \frac{s_k}{(L_i - L_j)\sqrt{2\pi}}\exp\left(\frac{-(L_j - L_i)}{T_k} - \frac{(L_j - L_i)^2}{2s_k^2}\right) \\
&\leq C_5 \cdot \exp\left(\frac{(L_i - L_j)\log k}{C} - \frac{(L_j - L_i)^2 C_1^2 k^2}{2}\right) \\
&\leq C_5 \cdot \exp\left(-\frac{(L_j - L_i)^2 C_1^2 k^2}{2}\right), \qquad (5.22) \\
& \qquad (5.23)
\end{aligned}$$

where C_5 is a positive constant and for sufficiently large k, $\frac{(L_i - L_j)\log k}{C} \leq \frac{(L_j - L_i)^2 C_1^2 k^2}{4}$. Hence $E_k \to 0$ (as $k \to \infty$) at an exponentially decreasing rate with respect to k. This proves that $\sum_k |E_k| < \infty$.

Case (iii): $L_i < L_j$. We divide $\tilde{P}_{ij}(k) - P_{ij}(k)$ into three parts. Let

$$\begin{aligned}
A_k &= \Phi\left(-\frac{L_j - L_i}{s_k}\right), \\
B_K &= \exp\left(\frac{s_k^2}{2T^2} - \frac{L_j - L_i}{T_k}\right) - \exp\left(-\frac{L_j - L_i}{T_k}\right), \\
C_k &= \exp\left(\frac{s_k^2}{2T^2} - \frac{L_j - L_i}{T_k}\right) \cdot \Phi\left(\frac{s_k}{T_k} - \frac{L_j - L_i}{s_k}\right). \qquad (5.24)
\end{aligned}$$

Now $P_{ij}(k) = q(i,j)(A_k + B_k + C_k)$. We need to prove that $\sum_k |A_k| < \infty$, $\sum_k |B_k| < \infty$, and $\sum_k |C_k| < \infty$. First, since $0 < \Phi\left(-\frac{L_j - L_i}{s_k}\right) < \Phi(-(L_j - L_i) \cdot C_1 k) \to 0$ (as $k \to \infty$) at an exponentially decreasing rate (from (5.17) and (5.20)), we get $\sum_k |A_k| < \infty$.

Second, we have:

$$0 \leq B_k = \exp\left(-\frac{L_j - L_i}{T_k}\right)\left(\exp\left(\frac{s_k^2}{2T_k^2}\right) - 1\right) \leq \exp\left(\frac{s_k^2}{2T_k^2}\right) - 1 = \exp\left(\frac{x^2}{2}\right) - 1,$$

where $x = \frac{s_k}{T_k} \to 0$ as $k \to \infty$, and $\exp\left(\frac{x^2}{2}\right) - 1 = O\left(\frac{x^2}{2}\right)$ as $x \to 0$. Since $x = \frac{s_k}{T_k} = O(k^{-\gamma}\log k)$ and $\gamma > 1$, $\sum_k \frac{x^2}{2}$ converges, and so does $\sum_k |B_k|$.

Finally, $C_k \geq 0$, and for sufficiently large k, $\frac{s_k}{T_k} \leq 1$; thus, we obtain

$$\exp\left(\frac{s_k^2}{2T_k^2} - \frac{L_j - L_i}{T_k}\right) \leq \exp\left(\frac{1}{2} - \frac{L_j - L_i}{T_k}\right) \leq \exp\left(\frac{1}{2}\right) = \sqrt{e},$$

and from (5.17),

$$\Phi\left(\frac{s_k}{T_k} - \frac{L_j - L_i}{s_k}\right) \leq \Phi\left(1 - (L_j - L_i) \cdot C_1 k\right) \to 0 \text{ (as } k \to \infty)$$

at an exponentially decreasing rate with respect to k. This yields $\sum_k |C_k| < \infty$. This proves the assertion for Case (iii).

Based on the three cases, the proof is complete.

By now we have proved that, if only the objective has noise, then Condition (5.8) is satisfied and the Markov chain modeling the modified CSA algorithm in Figure 5.2 will converge asymptotically. In the next step, we consider the situation when both the objective and constraints have noise. Let $\tilde{P}'_{ij}$ be the transition matrix in this situation. We first show that:

$$\sum_{k=1}^{\infty} |\tilde{P}'_{ij}(k) - \tilde{P}_{ij}(k)| < \infty. \tag{5.25}$$

Based on the fact that $\sum_{k=1}^{\infty} \tilde{P}_{ij}(k)$ converges and Lemma 5.1, we conclude $\sum_{k=1}^{\infty} \tilde{P}'_{ij}(k)$ converges, which is our final goal.

Theorem 5.3 Assume in the inhomogeneous Markov chain modeling the modified CSA algorithm in Figure 5.2 that σ_k is $O(k^{-\gamma})$ with fixed $\gamma > 1$. Further, assume in the Markov chains modeling the modified CSA and the original CSA that $T_k = C(\log k)^{-1}$, where C is a large enough positive constant. Then

$$\sum_{k=1}^{\infty} |\tilde{P}'_{ij}(k) - \tilde{P}_{ij}(k)| < \infty \tag{5.26}$$

for all $i, j \in s$.

Proof. We only show the case in which $x_i \neq x_j$ and $\lambda_i = \lambda_j$, and the case in which $x_i = x_j$ and $\lambda_i \neq \lambda_j$ is similar. In the first step of the derivation, since index k exists in the subscript of each variable, we do not specify k explicitly in order to simplify the notations. Note that in the two situations when only the objective has noise and when both the objective and constraints have noise, their Lagrangian functions are, respectively,

$$\begin{aligned}
\tilde{L}(x,\lambda) &= f(x)+\epsilon+\lambda|h(x)| \\
\tilde{L}'(x,\lambda) &= f(x)+\epsilon+\lambda|h(x)+\epsilon'|
\end{aligned}$$

Their corresponding transition probabilities, including noise ϵ_i, ϵ_j, ϵ'_i, and ϵ'_j (where i, j stand for the two states), is:

$$\begin{aligned}
\tilde{p}^{\epsilon}_{ij} &= q(i,j)\exp\left(\frac{-\Delta\tilde{L}^{+}}{T}\right) \\
\tilde{p}'^{\epsilon}_{ij} &= q(i,j)\exp\left(\frac{-\Delta\tilde{L}'^{+}}{T}\right),
\end{aligned}$$

where $\Delta\tilde{L}=\tilde{L}_j-\tilde{L}_i$, and $\Delta\tilde{L}'=\tilde{L}'_j-\tilde{L}'_i$. So,

$$\begin{aligned}
\frac{\tilde{p}'^{\epsilon}_{ij}}{\tilde{p}^{\epsilon}_{ij}} &= \exp\left(-\frac{\Delta\tilde{L}'^{+}-\Delta\tilde{L}^{+}}{T}\right) \\
&\le \exp\left(\frac{(\Delta\tilde{L}'-\Delta\tilde{L})^{+}}{T}\right) \\
&= \exp\left(\frac{((\tilde{L}_j-\tilde{L}'_j)-(\tilde{L}_i-\tilde{L}'_i))^{+}}{T}\right) \\
&= \exp\left(\frac{\lambda(|h(x_j)|-|h(x_j)+\epsilon'_j|-(|h(x_i)|-|h(x_i)+\epsilon'_i|))^{+}}{T}\right) \\
&\le \exp\left(\frac{2\lambda(|\epsilon'_i|+|\epsilon'_j|)}{T}\right) \\
&= \exp(A(|\epsilon'_i|+|\epsilon'_j|)), \qquad (5.27)
\end{aligned}$$

where $A=\frac{2\lambda}{T}$ is a constant, and $\epsilon'_i,\ \epsilon'_j\sim N(0,\sigma)$.

Hence, the difference of the expected transition probabilities is:

$$
\begin{aligned}
|\tilde{P}'_{ij} - \tilde{P}_{ij}| &\leq \int_{-\infty}^{\infty} (|\tilde{P}'^{\epsilon}_{ij} - \tilde{P}^{\epsilon}_{ij}|) d\epsilon'_i d\epsilon'_j && (5.28) \\
&= \int_{-\infty}^{\infty} \left| \left(\frac{\tilde{P}'^{\epsilon}_{ij}}{\tilde{P}^{\epsilon}_{ij}} - 1 \right) \tilde{P}^{\epsilon}_{ij} \right| d\epsilon'_i d\epsilon'_j && (5.29) \\
&\leq M \int_{-\infty}^{\infty} \left| \frac{\tilde{P}'^{\epsilon}_{ij}}{\tilde{P}^{\epsilon}_{ij}} - 1 \right| d\epsilon'_i d\epsilon'_j && (5.30) \\
&\leq M \int_{-\infty}^{\infty} (\exp(A(|\epsilon'_i| + |\epsilon'_j|)) - 1) d\epsilon'_i d\epsilon'_j && (5.31) \\
&= M \left(4 \int_0^{\infty} \exp(A\epsilon'_i) d\epsilon'_i \int_0^{\infty} \exp(A\epsilon'_j) d\epsilon'_j - 1 \right) && (5.32) \\
&= M(4 \exp(2A\sigma).\Phi^2(-A\sigma) - 1), && (5.33)
\end{aligned}
$$

where (5.30) is true because $\sum_{k=0}^{\infty} \tilde{P}_{ij}(k)$ converges and thus $\tilde{P}_{ij}(k)$ is less than a constant. Further, (5.31) is true because $\exp(A(|\epsilon'_i| + |\epsilon'_j|))$ is always greater than 1 and, thus, the absolute sign can be taken off. (5.33) is derived from the following integration:

$$
\int_0^{\infty} \exp(A\epsilon'_i) d\epsilon'_i = \int_0^{\infty} \exp(\sigma A z) \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2} \right) \sigma dz = \exp(A\sigma)\Phi(-A\sigma). \qquad (5.34)
$$

Now consider the effect due to k. For each k, the constraint at x_i has noise $\epsilon'_i(k)$, the constraint at x_j has noise $\epsilon'_j(k)$, and σ_k is the standard deviation of both noises. Let $t = A\sigma_k$.

$$
|\tilde{P}'_{ij}(k) - \tilde{P}_{ij}(k)| = M(4 \exp(2A\sigma_k) \cdot \Phi^2(-A\sigma_k) - 1) = M(4 \exp(2t)\Phi^2(-t) - 1). \quad (5.35)
$$

Since $\exp(2t) = 1 + 2t + O(t^2)$ (as $t \to 0$) and $\Phi(-t) = \frac{1}{2} + \frac{t}{\sqrt{2\pi}} + O(t^2)$ (as $t \to 0$), $|\tilde{P}'_{ij}(k) - \tilde{P}_{ij}(k)| = O(t)$ (as $t \to 0$). Because $t = A\sigma_k = \frac{2\lambda}{T_k} \cdot \sigma_k = O\left(\frac{k^{-\gamma}}{\log k} \right)$, $\sum_k |\tilde{p}'_{ij}(k) - \tilde{p}_{ij}(k)|$ converges to a finite value. This completes the proof of the theorem.

5.3 Fast Practical Implementations

Although we have proved that *CSA* with slow cooling and sampling schedules can converge to a CGM_{dn} asymptotically with probability one, it is desirable to develop fast algorithms to find constrained local minimum in practice.

5.3.1 Using confidence intervals to achieve prescribed constraint violations

Note that in the modified CSA algorithm 1 (Figure 5.2), we only measure the average of multiple sample values at each point as an approximate mean of f, g or h, but do not use the confidence interval to measure the quality of the sample mean. As we have said earlier, this is impractical in use because it will incur too many sample evaluations.

To add the concept of confidence intervals into our modified CSA algorithm, we first transform equality and inequality constraints as follows:

$$h''(x) = \begin{cases} 0, & \text{if } LL \leq 0 \text{ and } UL \geq 0 \\ LL, & \text{if } LL > 0 \\ UL, & \text{if } UL < 0 \end{cases} \tag{5.36}$$

$$g''(x) = \begin{cases} 0, & \text{if } UU \leq 0 \\ UU, & \text{otherwise.} \end{cases} \tag{5.37}$$

We then compute the Lagrangian function and acceptance probability according to the transformed constraints $h''(x)$ and $g''(x)$. The complete algorithm is listed in Figure 5.3. Since the confidence interval only depends on the estimated standard deviation of the sampling average, $\hat{\sigma}/\sqrt{N}$, we need to schedule the samples properly so that when the modified CSA stops its run, $\hat{\sigma}/\sqrt{N}$ should also arrive a prescribed level σ_f.

5.3.2 Behavior of the modified CSA algorithm 2

After incorporating the confidence interval into our algorithm, we now demonstrate the behavior of the modified CSA algorithm 2 under different constraint violation level σ_f.

The problem we study is the truncated 10-dimensional Rastrigin function. We add some random noise on the function to test our algorithm as follows:

$$\text{minimize} \qquad E(f(x)) = E(F(10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)), f') + \epsilon(x))$$

```
1. procedure modified_CSA_2
2.     set starting point $\mathbf{x} = (x, \lambda)$;
3.     set starting temperature $T = T^0$ and cooling rate $0 < \alpha < 1$;
4      $I = 0$
5.     set $N_T$ (number of trials per temperature);
6.     while stopping condition is not satisfied do
7.         for $j \leftarrow 1$ to $N_T$ do
8              $I = I + 1$
9.             generate a trial point $\mathbf{x}'$ from $\mathcal{N}(\mathbf{x})$ using $q(\mathbf{x}, \mathbf{x}')$;
10.            generate $N(I)$ independent observations of $f, g$ and $h$
               on both $\mathbf{x}$ and $\mathbf{x}'$ respectively.
11.            calculate $h''(\mathbf{x})$, and $h''(\mathbf{x}')$ ((5.36) and (5.37))
12.            accept $\mathbf{x}'$ with probability $A'(\mathbf{x}, \mathbf{x}')$ using $h''(\mathbf{x})$, and $g''(\mathbf{x}')$
13.        end_for
14.        reduce temperature by $T \longleftarrow \alpha \times T$ ;
15.    end_while
16. end_procedure
```

Figure 5.3: Modified CSA algorithm 2

$$\text{subject to} \qquad E(\sum_{i=1}^{n} |x_i - 3.8| + \epsilon(x)) \le 0.1 \quad \text{for } n = 10. \tag{5.38}$$

where

$$F(y, f') = \begin{cases} f', & \text{if } y \le f' \\ y, & \text{otherwise,} \end{cases}$$

$\epsilon(x)$ is a normally distributed random number with mean 0 and variance 10. We choose $f' = 180$ here in the example and $\sigma_f = 1,\ 0.5,\ 0.25,\ 0.125$ as the final standard deviation, respectively, in each graph of Figure 5.4. For each σ_f, we set the sample size at the k'th iteration to be $m_k = \lfloor C^k \rfloor$, where C is a constant. Suppose N is the number of probes in cooling rate α. Then the total number of samples in one run of the modified CSA is

$$S_\alpha = \sum_{k=1}^{N} m_k \approx \frac{C^{N+1} - 1}{C - 1}$$

We applied the modified CSA 2 on problem (5.38) using different sampling schedules, and for each sampling schedule, we ran the algorithm 100 times and recorded the average number of samples used for each run and the probability to find a solution with prescribed quality. In one sampling schedule, suppose P_r is the probability to find a solution of quality f' and $Samples$ is the average number of samples. then $Samples/Pr$ is the expected number of samples to find the solution. We used a 99-percent confidence interval to construct the confidence interval in the experiment. Figure 5.4 plots the relations between $Samples/Pr$ and $Samples$.

Definition 5.1 An *optimal sampling schedule* is one that minimizes the average total number of samples of multiple runs of the modified CSA algorithm 2 in order to find a solution of prescribed quality.

It can be easily observed that there is an optimal sampling schedule for each σ_f to find the solution from Figure 5.4.

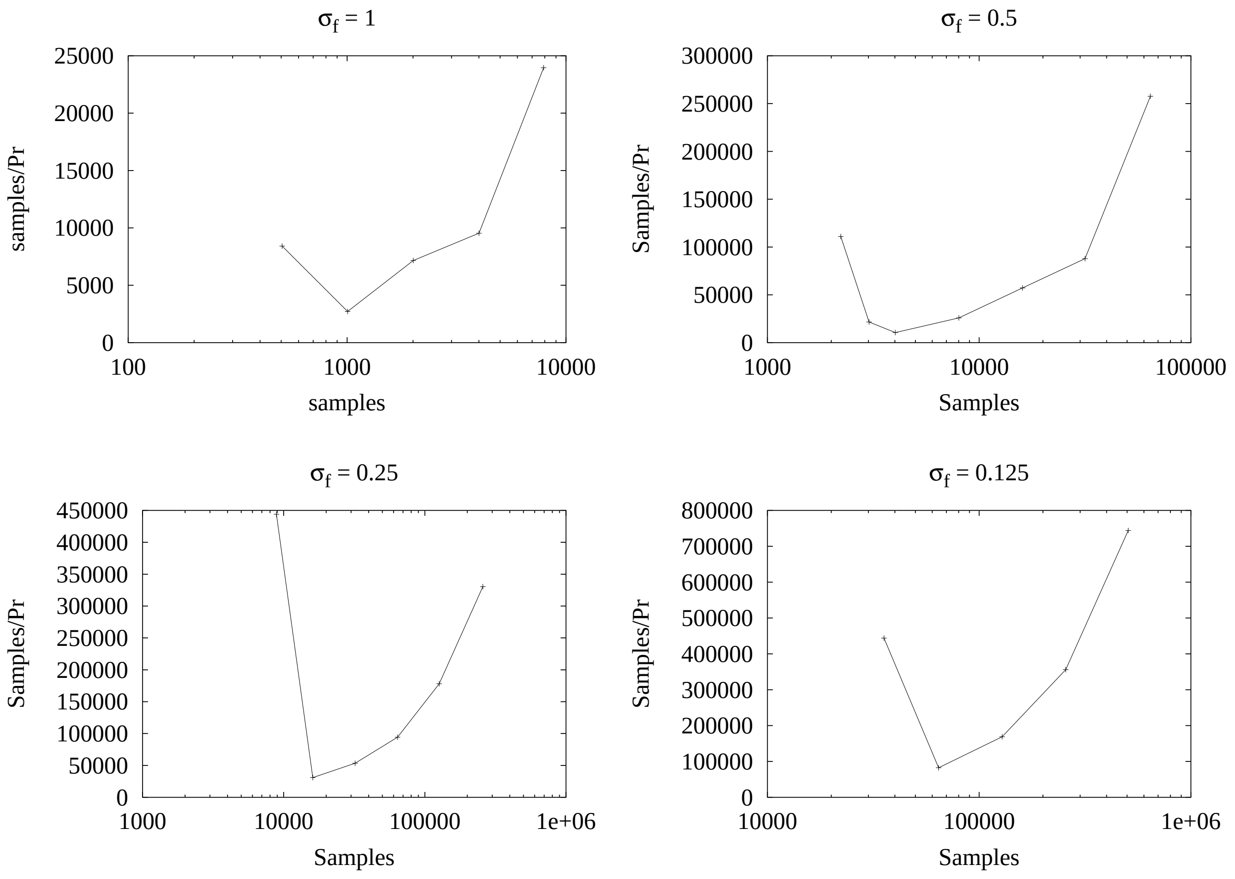
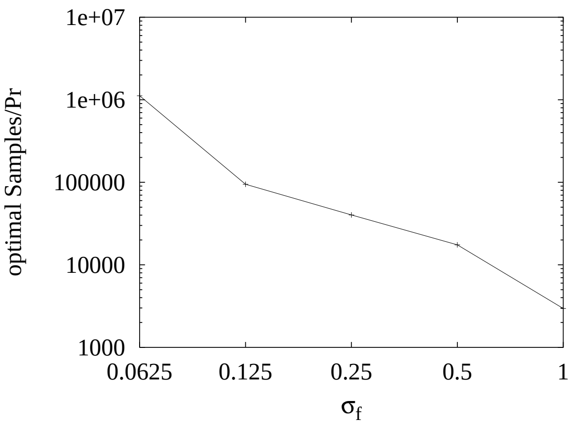
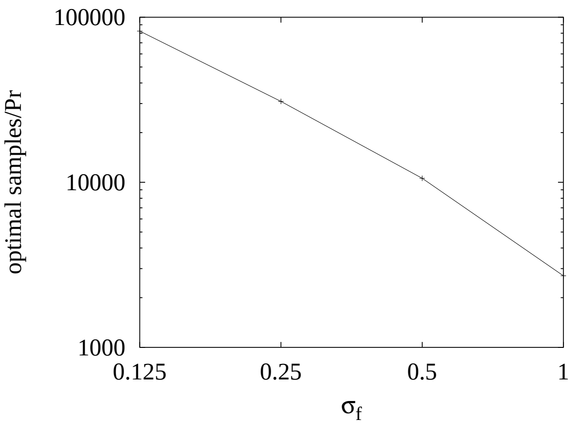


Figure 5.4: Behavior of modified CSA algorithm 2 with different sampling schedules to solve the truncated Rastrigin function with noise (5.38). Objective target $f' = 180$. Each graph was generated for a different final error standard deviation σ_f

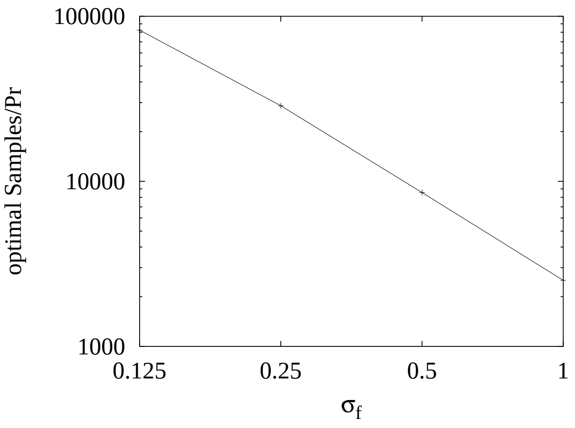
Let $S_{opt}(\sigma_f)$ represents the minimal $Sample/Pr$ for a given σ_f. Picking $S_{opt}(\sigma_f)$ for each σ_f from the graph in Figure 5.4, we can look at the relationship of $S_{opt}(\sigma_f)$ vs σ_f. We plotted their relationship for different objective target levels in Figure 5.5.



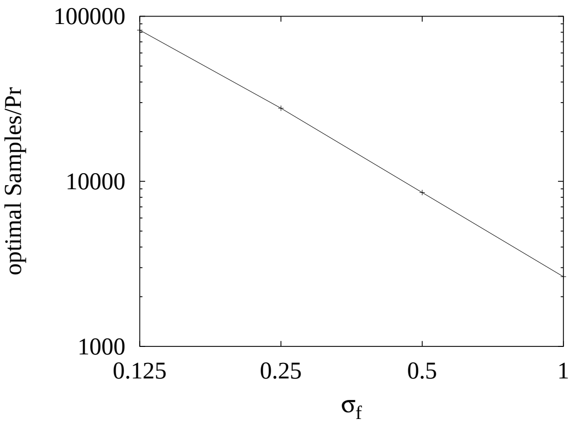
$f' = 170$



$f' = 180.$



$f' = 190$



$f' = 200.$

Figure 5.5: Exponential relationships between σ_f and $Samples/Pr$ for different objective targets f'.

We can see that there is a log-log relationship between S_{opt} and σ_f. We can model the relationship by the following equation:

$$\log S_{opt}(\sigma_f) = A - B \log \sigma_f \tag{5.39}$$

We did a linear regression for $\log S_{opt}$ and $\log \sigma_f$ for each objective level and listed the R^2 value in table 5.1. All the R^2 values are very close to 1, which indicates that this model is a very good fit for the experimental data.

Based on this exponential model, we develop the anytime algorithm in the next section.

Table 5.1: The coefficients of determination R^2 on linear regression of $\log S_{opt}$ and $\log \sigma_f$. The problems are truncated 10 dimensional Rastrigin problem (5.38).

objective target	170	180	190	200
R^2	0.958	0.994	0.999	0.999

5.3.3 Anytime CSA with noise

Corresponding to $CSA_{ATCR-ID}$ in Section 3.3.4, we also have two components in this modified anytime CSA algorithm (Figure 5.6). The first component (Lines 6-12) is to use iterative deepening to find the optimal sampling schedule. In this component (Lines 6-12) we solve the problem with a fixed violation target level σ_f. Suppose S_i is the number of samples in the ith run of the modified CSA algorithm 2. From the analysis in the previous part, we know that

$$S_i \approx \frac{C_i^{N_i+1} - 1}{C_i - 1}$$

where N_i is the total number of probes and C_i is the constant to decide the number of samples drawn in each probe (number of samples at iteration k is $M_k = \lfloor C^k \rfloor$).

To carry out the iterative deepening, we need to double the number of samples S_i between each run but also need to keep the $\sigma_f(i) = \sigma / \sqrt{M_{N_i}}$ constant. This means $S_{i+1} = 2S_i$ whereas $\sigma_f(i+1) = \sigma_f(i)$. Since $\sigma_f(i) = \frac{\sigma}{\sqrt{M_{N_i}}}$, keeping $\sigma_f(i)$ constant requires keeping $M_{N_i} = C_i^{N_i}$ constant; that is $C_i^{N_i} = C_{i+1}^{N_{i+1}}$. On the other hand, since $S_{i+1} = 2S_i$ we have:

$$\frac{C_{i+1}^{N_{i+1}+1} - 1}{C_{i+1} - 1} = 2 \times \frac{C_i^{N_i+1} - 1}{C_i - 1}.$$

Substituting $C_{i+1}^{N_{i+1}}$ by $C_i^{N_i}$, we get:

$$\frac{C_i^{N_i} \cdot C_{i+1} - 1}{C_{i+1} - 1} = 2 \times \frac{C_i^{N_i+1} - 1}{C_i - 1}.$$

After simplification, we get:

$$C_{i+1} = \frac{2C_i^{N_i+1} - C_i - 1}{C_i^{N_i+1} + C_i^{N_i} - 2}.$$

```
1.  procedure anytime-CSA-with-noise
2.  set initial target of solution quality at $\sigma_f = \sigma_{f0}$;
3.  set $nRepRT$ = number of CSA runs at fixed cooling rate
4.  repeat
5.      set initial cooling rate $\alpha = \alpha_0$ and $N_k$
6       repeat
7.          for $j \leftarrow 1$ to $nRepRT$ do
8.              evaluate CSA with transformed constraints (5.36) and (5.37);
9.              if CSA succeeds then goto 13; endif
10.         end for
11.         increase cooling rate and $C_k$ such that
            $samples_{\alpha,N_k} \leftarrow \rho \times samples_{\alpha,N_k}$(typically $\rho = 2$) but keep $\sigma_f$ invariant ;
12      until CSA succeeds or stop condition is satisfied.
13.     reduce target level $\sigma_f \leftarrow \sigma_f \cdot d$ (typically $d = 1/2$)
14. until $\sigma_f = \sigma_{ff}$ or the total number of samples exceeds a fixed number
```

Figure 5.6: Anytime CSA algorithm for solving NSPs with noise

Notice that C_i is pretty much close to 1. (If $C_i = 1.1$ and $N_i = 200$, then the number of samples at the N_i's iteration is $M_{N_i} = 1.9E8$.) In general, $C_i^{N_i} >> 1$. Hence, we can get a simplified equation:

$$C_{i+1} \approx \frac{2C_i}{1 + C_i}.$$

Choosing such a sampling schedule, we find that the total number of samples is dominated by the sample size in the last iteration ($\sum_{i=1}^{n} S_i = 2 \times S_n - S_1$).

In the second component (Lines 5-13), our goal is to reduce the violation target level σ_f gradually and find a solution for each $\sigma_f(j)$. Note that index j is for the index in the outer iteration. We use a geometric form: $\sigma_f(j+1) = \sigma_f(j) \cdot d$. Typically we set $d = 1/2$.

5.3.4 Experimental results

We test our modified anytime CSA algorithm on truncated Rastrigin functions with noise. We use four different levels of objective cutoff.

Figure 5.7 compares the behavior of the modified anytime CSA algorithm 2 and that of the optimal sampling schedule. The performance of CSA with an optimal sampling schedule is obtained by running CSA using different sampling schedules, each with 50 runs in order to find the optimal sampling schedule.

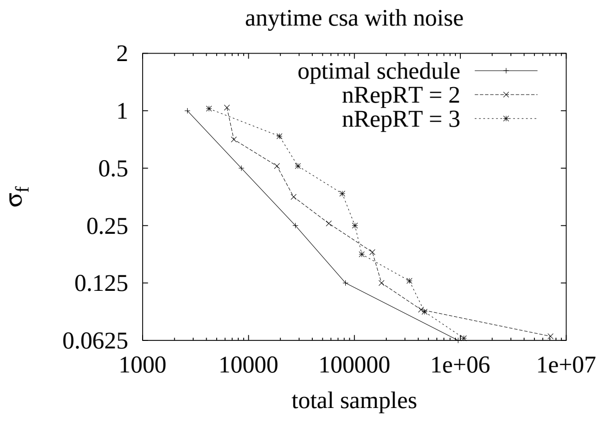
The results show that anytime CSA uses about 2-3 times of number of samples to find a solution for a given σ_f when compared to CSA with an optimal sampling schedule. With increasing time, constraint violations will be lowered. We have tried two sampling schedules in each experiment by using two different *nRepNT*'s, where *nRepNT* is the number of the CSA runs in each cooling rate. Using *nRepNT* $= 2$ could be faster in the beginning with a larger target σ_f than that of *nRepNT* $= 3$. But the anytime CSA with *nRepNT* $=3$ is more stable when σ_f is smaller. In general, we recommend using *nRepNT* $= 3$.

5.4 Summary

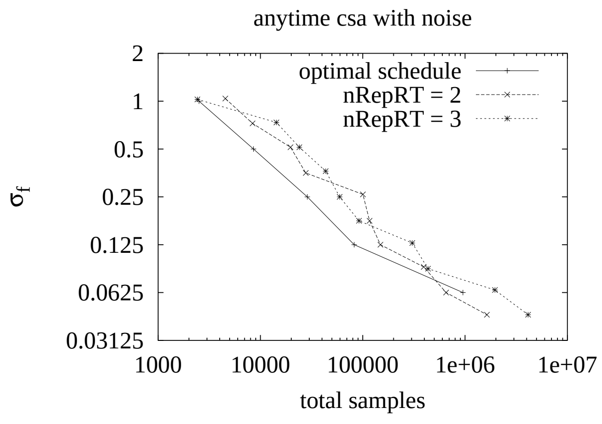
In this chapter we have studied the solution of stochastic optimization problems. In this kind of problems, we cannot evaluate the objective and constraints accurately. We first define a new constraint satisfaction criteria, since using the one in deterministic optimization is too computationally expensive. We then apply modified CSA to solve stochastic optimization problems in which the sample averages are used to estimate mean values in function evaluations. To minimize the expected number of samples, we can start with a small number of samples in each probe and increase the number of samples gradually.

In theory, we have proved that, with a slow enough cooling and sampling schedules, CSA preserves its asymptotic convergence property.

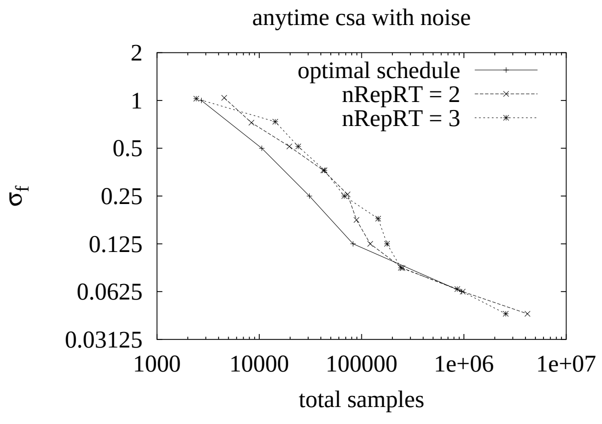
In practice, we have found that the best sampling schedule is to increase the number of samples geometrically when the final violation tolerance level σ_f is given. We have also applied the anytime CSA algorithm to solve stochastic optimization problems in such a way that, given a shorter time, the anytime algorithm can generate a solution with a larger violation tolerance level and, given longer time, it can improve the violation tolerance level gradually. The overhead for the anytime algorithm is no more than a few times over the expected number of samples when using an optimal sampling schedule.



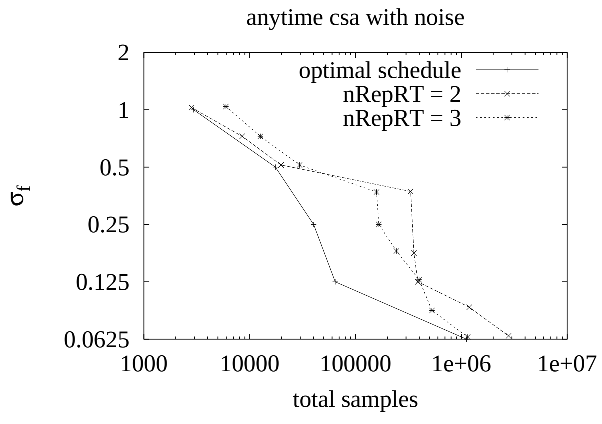
(a) obj_cutoff = 200



(b) obj_cutoff = 190



(c) obj_cutoff = 180



(d) obj_cutoff = 170

Figure 5.7: Performance comparisons of modified CSA_{AT-ID} and modified CSA algorithm 2 with an optimal sampling schedule. $nRepNT$ is the number of times CSA was run under each cooling rate.

Chapter 6

Conclusions

In this thesis, we have studied constraint relaxations for constrained NLP algorithms in order to find acceptable results in limited time. The problems we have studied include NLPs with and without uncertainty. The NLP algorithms used include stochastic and deterministic algorithms.

Using CSA as an example of stochastic algorithms and CLS as an example of deterministic algorithms, we have proposed anytime search schedules that can generate solutions with improved qualities (and smaller violations) when given more execution time. Such an algorithm is desirable when solving large complex NLPs and in real time when solutions with zero violations are hard to achieve and those with small violations are acceptable. Based on a log-log model relating constraint relaxation levels and expected search times, we have proposed algorithms that decrease constraint relaxation levels in such a way that the expected time to find a feasible solution for each relaxation level increases geometrically, leading to the result that the total search time is dominated by the time used to find solutions for the last relaxation level. Experimental results of our proposed anytime schedule on a wide range of engineering benchmarks have shown that our proposed schedule is applicable, adaptable and robust.

When solving constrained NLP problems with noise, we have proposed new constraint satisfaction criteria, leading to more natural definition of constraint relaxation. Similar to the case in NLPs without noise, we have proposed a geometric relaxation schedule to decrease constraint relaxation levels in such a way that the expected number of samples drawn for each relaxation level increases geometrically and that the number of samples spent for finding the solution with the last relaxation level dominate the overall number of samples used.

In short, our constraint relaxation schedule can be applied to most search algorithms, including stochastic and deterministic algorithms on constrained NLP problems with or without noise in order to generate solutions of improved quality as more time is given.

Bibliography

[1] ftp://ftp.mathworks.com/pub/contrib/v5/optim/bnb/.

[2] ftp://ftp.systemtechnik.tu-ilmenau.de/pub/tools/omuses/.

[3] http://archimedes.scs.uiuc.edu/baron/baron.html.

[4] http://at8.abo.fi/˜ hasku/mittlp/.

[5] http://gachinese.com/aemdesign/FSQPframe.htm.

[6] http://plato.la.asu.edu/donlp2.html.

[7] http://www-neos.mcs.anl.gov/neos/solvers/SLP:AUGMENTED.

[8] http://www.abo.fi/˜ twesterl/.

[9] http://www.coe.uncc.edu/˜ zbyszek/evol-systems.html.

[10] http://www.cse.clrc.ac.uk/Activity/LANCELOT.

[11] http://www.gams.com/solvers/solvers.htm#SBB.

[12] http://www.maths.dundee.ac.uk/˜ sleyffer.

[13] http://www.mosek.com/.

[14] http://www.orfe.princeton.edu/˜ loqo/.

[15] http://www.sbsi-sol-optimize.com/Minos.htm.

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

[17] M.A. Ahmed, T.M. Al Khamis, and H. Merza. Optimizing discrete stochastic systems using simulated annealing and simulation. *Computers and Industrial Engineering*, 32(4):823–836, Sept. 1997.

[18] M. M. Ali and C. Storey. Modified controlled random search algorithms. *Int'l Journal of Computer Mathematics*, 53:229–235, 1994.

[19] M. M. Ali and C. Storey. Modified controlled random search algorithms. *Int. Journal of Computer Mathematics*, 53:229–235, 1994.

[20] M. M. Ali and C. Storey. Aspiration based simulated annealing algorithms. *Journal of Global Optimization*, 11:181–191, 1997.

[21] M. M. Ali, A. Torn, and S. Viitanen. A direct search simulated annealing algorithms for optimization involving continuous variables. Technical report, Turku Centre for Computer Science, Abo Akademi University, Finland, 1997.

[22] M. M. Ali, A. Torn, and S. Viitanen. A numerical comparison of some modified controlled random search algorithms. *Journal of Global Optimization*, 11:377–385, 1997.

[23] Talal M. Alkhamis, Mohamed A. Ahmed, and Vu Kim Tuan. Simulated annealing for discrete optimization with estimation. *European Journal of Operational Research*, 116:530–544, 1999.

[24] E. D. Andersen and K. D. Andersen. The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In *High Performance Optimization Techniques, Proceedings of the HPOPT-II conference*, 1997.

[25] E. Anderson and M. Ferris. A direct search algorithm for optimization with noisy function evaluations. *SIAM Journal of Optimization*, 11(3):837–857, Mar. 2000.

[26] K. J. Arrow and L. Hurwicz. Gradient method for concave programming, I: Local results. In K. J. Arrow, L. Hurwica, and H. Uzawa, editors, *Studies in Linear and Nonlinear Programming*. Stanford University Press, Stanford, CA, 1958.

[27] T. Back, F. Hoffmeister, and H. P. Schwefel. A survey of evolution strategies. In *Proc. of 4th Int'l Conf. on Genetic Algorithms*, pages 2–9, 1991.

[28] T. Back and H. P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[29] E. Balas. *Minimax and Duality for Linear and Nonlinear Mixed-Integer Programming.* North-Holland, Amsterdam, Netherlands, 1970.

[30] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical report, CMU-CS-95-193, Carnegie Mellon University, Pittsburgh, PA, 1995.

[31] S. Baluja and S. Davies. Fast probabilistic modeling for combinatorial optimization. In *Proc. of 15th National Conf. on Artificial Intelligence (AAAI)*, 1998.

[32] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[33] M. S. Bazaraa and J. J. Goode. A survey of various tactics for generating Lagrangian multipliers in the context of Lagrangian duality. *European Journal of Operational Research*, 3:322–338, 1979.

[34] J. C. Bean and A. B. Hadj-Alouane. A dual genetic algorithm for bounded integer programs. *Technical Report 92-53, Department of Industrial and Operations Engineering*, 1992.

[35] J. E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94:392–404, 1996.

[36] A. Ben-Tal, G. Eiger, and V. Gershovitz. Global minimization by reducing the duality gap. *Mathematical Programming*, 63:193–212, 1994.

[37] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numer. Math*, pages 238–242, 1962.

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

[39] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, pages 1–52, 1995.

[40] J. S. De Bonet, C. L. Isbell, and J. Paul Viola. MIMIC: Finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems*, 1997.

[41] J. A. Boyan and A. W. Moore. Using prediction to improve combinatorial optimization search. In *Proc. of 6th Int'l Workshop on Artificial Intelligence and Statistics*, 1997.

[42] J. A. Boyan and A. W. Moore. Learning evaluation functions for global optimization and Boolean satisfiability. In *Proc. of 15th National Conf. on Artificial Intelligence (AAAI)*, 1998.

[43] D. W. Bulger and G. R. Wood. Hesitant adaptive search for global optimization. *Mathematical Programming*, 81:89–102, 1998.

[44] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[45] Y. J. Cao and Q. H. Wu. Mechanical design optimization by mixed-variable evolutionary programming. *Proc. 1997 IEEE Int'l Conf. on Evolutionary Computation*, pages 443–6, 1997.

[46] Claus C. Carøe and Rüdiger Schultz. Dual decomposition in stochastic integer programming. *Oper. Res. Lett.*, 24(1-2):37–45, 1999.

[47] K. Chen, C. Parmee, and C. R. Gane. Dual mutation strategies for mixed-integer optimization in power station design. *Proc. 1997 IEEE Int'l Conf. on Evolutionary Computation*, pages 385–90, 1997.

[48] Q. Chen and M. C. Ferris. FATCOP: A fault tolerant condor-pvm mixed integer program solver. *Mathematical Programming Technical Report 99-05, University of Wisconsin*, 3 1999.

[49] Q. Chen, M. C. Ferris, and J. T. Linderoth. FATCOP 2.0: Advanced features on an opportunistic mixed integer programming solver. *Data Mining Institute Technical Report 99-11, University of Wisconsin*, 12 1999.

[50] Y. X. Chen. *Optimal Anytime Search for Constrained Nonlinear Programming.* M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 2001.

[51] K. F. M. Choi, J. H. M. Lee, and P. J. Stuckey. A Lagrangian reconstruction of a class of local search methods. In *Proc. 10th Int'l Conf. on Artificial Intelligence Tools*. IEEE Computer Society, 1998.

[52] A.R. Conn, N. Gould, and Ph. L. Toint. *LANCELOT, A Fortran Package for Large-Scale Nonlinear Optimization.* Springer Verlag, 1992.

[53] A. Dekkers and E. Aarts. Global optimization and simulated annealing. *Mathematical Programming*, 50:367–393, 1991.

[54] DONLP2. Spellucci's mixed SQP/ECQP method for general continuous nonlinear programming problems. *ftp://plato.la.asu.edu/pub/donlp2*, 2000.

[55] M. A. Duran and I. E. Grossmann. A mixed-integer nonlinear programming algorithm for process systems synthesis. *Chemical Engineering J.*, pages 592–596, 1986.

[56] M. A. Duran and I. E. Grossmann. An outer approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, pages 306–307, 1986.

[57] R. W. Eglese. Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 46:271–281, 1990.

[58] L. Eshelman and J. Schaffer. Real-coded genetic algorithms and interval schemata. In L. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 187–202. Morgan Kaufmann Publishes, San Francisco, 1993.

[59] Y. G. Evtushenko, M. A. Potapov, and V. V. Korotkich. Numerical methods for global optimization. In C. A. Floudas and P. M. Pardalos, editors, *Recent Advances in Global Optimization*, pages 274–297. Princeton University Press, 1992.

[60] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization.* Topics in Chemical Engineering. Oxford University Press, 1995.

[61] C. A. Floudas and P. M. Pardalos, editors. *Recent Advances in Global Optimization.* Princeton University Press, 1992.

[62] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Trans. on Neural Networks*, 5(1):3–14, January 1994.

[63] M. I. Freidlin and A. D. Wentzell. *Random perturbations of dynamical systems.* New York : Springer, 1984, 1984.

[64] H.I Gassmann. Mslip: A computer code for multi-stage stochastic linear programming problem. *Mathematical Programming*, 47:407–423, 1990.

[65] B. Gavish. On obtaining the 'best' multilpliers for a Lagrangean relaxation for integer programming. *Comput. & Ops. Res.*, 5:55–71, 1978.

[66] S.B. Gelfand and S.K. Mitter. Simulated annealing with noisy or imprecise energy measurements. *Optimization Theory and Applications*, 62:49–62, 1989.

[67] S. Geman and D. Geman. Stochastic relaxation, Gibbs distribution and the Bayesian restoration in images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[68] A. M. Geoffrion. Generalized Benders decomposition. *J. Optim. Theory and Appl.*, pages 237–241, 1972.

[69] A. M. Geoffrion. Lagrangian relaxation for integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

[70] F. R. Giles and W. R. Pulleyblank. *Total Dual Integrality and Integer Polyhedra*, volume 25. Elsevier North Holland, Inc., 1979.

[71] P. E. Gill. User's guide for snopt 5.3: A fortran package for large-scale nonlinear programming. *Dept. of Maths, Univ. of Califonia, San Diego*, 1999.

[72] F. Glover. Tabu search — Part I. *ORSA J. Computing*, 1(3):190–206, 1989.

[73] F. Glover and G. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In *Proc. of Int'l Conf. on Metaheuristics for Optimization*, pages 113–133, 1995.

[74] F. Glover and M. Laguna. Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems (C. R. Reeves ed.)*, 1993.

[75] F. Glover and E. Woolsey. Further reduction of zero-one polynomial programs to zero-one linear programming. *Operations Research*, 1(21):156–161, 1973.

[76] F. Glover and E. Woolsey. Converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22:180–182, 1975.

[77] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Pub. Co., 1989.

[78] D. Granot, F. Granot, and J. Kallberg. Covering relaxation for positive 0-1 polynomial programs. *Management Science*, 3(25):264–273, 1979.

[79] D. Granot, F. Granot, and W. Vaessen. An accelerated covering relaxation algorithm for solving positive 0-1 polynomial programs. *Mathematical Programming*, 22:350–357, 1982.

[80] Walter J. Gutjahr and Georg Ch. Pflug. Simulated annealing for noisy cost functions. *Journal of Global Optimization*, 8:1–13, 1996.

[81] J. Haddock and J. Mittenthal. Simulation optimization using simualted annealing. *Computers and Industrial engineering*, 20(4):387–395, 1992.

[82] A. B. Hadj-Alouane and J. C. Bean. Genetic algorithm for the multiple-choice integer program. *Technical Report 92-50, Department of Industrial and Operations Engineering*, 1992.

[83] P. L. Hammer, I. Rosenberg, and S. Rudeanu, editors. *Boolean Methods in Operations Research and Related Areas.* Springer, New York, 1968.

[84] E. R. Hansen. *Global optimization using interval analysis.* M. Dekker, New York, 1992.

[85] P. Hansen, B. Jaumard, and V. Mathon. Constrained nonlinear 0-1 programming. *ORSA Journal on Computing*, 5(2):97–119, 1993.

[86] W. E. Hart. A theoretical comparison of evolutionary algorithms and simulated annealing. In *Proc. of 5th Annual Conf. on Evolutionary Programming (EP96)*, pages 147–154, 1996.

[87] L He and E. Polak. Multistart method with estimation scheme for global satisfying problems. *Journal of Global Optimization*, 3:139–156, 1993.

[88] J. H. Holland. *Adaption in Natural and Adaptive Systems.* University of Michigan Press, Ann Arbor, 1975.

[89] K. Holmberg. On the convergence of the cross decomposition. *Mathematical Programming*, pages 269–316, 1990.

[90] K. Holmberg. Generalized cross decomposition applied to nonlinear integer programming problems. *Optimization J.*, pages 341–364, 1992.

[91] A. Homaifar, S. H. Y. Lai, and X. Qi. Constrained optimization via genetic algorithms. *Simulation*, 62:242–254, 1994.

[92] R. Horst and P. M. Pardalos. *Handbook of Global Optimization.* Kluwer Academic Publishers, 1995.

[93] R. Horst and H. Tuy. *Global optimization: Deterministic approaches.* Springer-Verlag, Berlin, 1993.

[94] M. E. Hribar. Large scale constrained optimization. *Ph.D. Disertation, Northeasten University*, 1996.

[95] L. Ingber. *Adaptive Simulated Annealing (ASA).* Lester Ingber Research, 1995.

[96] 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.

[97] A. E. W. Jones and G. W. Forbes. An adaptive simulated annealing algorithm for global optimization over continuous variables. *Journal of Optimization Theory and Applications*, 6:1–37, 1995.

[98] J. Kim and H. Myung. Evolutionary programming techniques for constrained optimization problems. *IEEE Trans. on Evolutionary Computation*, 1(2):129–140, 1997.

[99] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[100] R. Korf. Heuristics as invariants and its application to learning. *Machine Learning*, pages 100–103, 1985.

[101] V. Kvasnicka and J. Pospichal. A hybrid of simplex method and simulated annealing. *Chemometrics and Intelligent Laboratory Systems*, 39:161–173, 1997.

[102] LANCELOT. http://www.dci.clrc.ac.uk/activity/lancelot.

[103] G. Laporte, F.V. Louveaux, and L. van Hamme. Exact solution of a stochastic location problem by an integer L-shaped algorithm. *Transportation Science*, 28(2):95–103, 1994.

[104] E. L. Lawler and M. D. Bell. A method for solving discrete optimization problems. *Operations Research*, 14:1098–1112, 1966.

[105] Y. X. Li and M. Gen. Nonlinear mixed integer programming problems using genetic algorithm and penalty function. *IEEE Int'l Conf. on Systems, Man and Cybernetics, Information Intelligence and Systems*, 4:2677–82, 1996.

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

[107] C. Y. Maa and M. A. Shanblatt. A two-phase optimization neural network. *IEEE Trans. on Neural Networks*, 3(6):1003–1009, 1992.

[108] Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs.* Springer-Verlag, Berlin, 1994.

[109] Z. Michalewicz, D. Dasgupta, R. G. LeRiche, and M. Schoenauer. Evolutionary algorithms for constrained engineering problems. *Computers and Industrial Engineering Journal*, 30(2):851–870, 1996.

[110] Z. Michalewicz and C. Z. Janikow. Handling constraints in genetic algorithms. In *Proc. of 4th Int'l Conf. on Genetic Algorithms*, pages 151–157, 1991.

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

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

[113] J. Mockus. *Bayesian Approach to Global Optimization*. Kluwer Academic Publishers, Dordrecht-London-Boston, 1989.

[114] J. Mockus. Application of Bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4:347–365, 1994.

[115] R. Moore and E. Hansen amd A. Leclerc. Rigorous methods for global optimization. In C. A. Floudas and P. M. Pardalos, editors, *Recent Advances in Global Optimization*, pages 321–342. Princeton University Press, 1992.

[116] P. Del Moral and L. Miclo. On the convergence and applications of generalized simulated annealing. *Society for Industrial and Applied Mathematics*, 37(4):1222–1250, 1999.

[117] P. Morris. The breakout method for escaping from local minima. In *Proc. of 11th National Conf. on Artificial Intelligence*, pages 40–45, Washington, DC, 1993.

[118] A. E. Nix and M. D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Math. and Artificial Intel.*, 5:79–88, 1992.

[119] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag New York, Inc., 1999.

[120] D. Orvosh and L. Davis. Shall we repair? genetic algorithms, combinatorial optimization, and feasibility constraints. In *Proc. of 5th Int'l Conf. on Genetic Algorithms*, 1993.

[121] P. M. Pardalos and J. B. Rosen. *Constrained Global Optimization: Algorithms and Applications*, volume 268 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1987.

[122] J. Paredis. Co-evolutionary constraint satisfaction. In *Proc. of 3rd Conf. on Parallel Problem Solving from Nature*, pages 46–55, 1994.

[123] K. Park and B. Carter. On the effectiveness of genetic search in combinatorial optimization. In *Proc. of 10th ACM Symposium on Applied Computing, Genetic Algorithms and Optimization Track*, pages 329–336, 1995.

[124] N. R. Patel, R. L. Smith, and Z. B. Zabinsky. Pure adaptive search in Monte Carlo optimization. *Mathematical Programming*, 43:317–328, 1988.

[125] V. Petridis, S. Kazarlis, and A. Bakirtzis. Varying fitness functions in genetic algorithm constarined optimization: The cutting stock and unit commitment problems. *IEEE Trans. on System, Man, and Cybern. - Part B: Cybernetics*, 28(5):629–640, 1998.

[126] D. Powell and M. M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proc. of 5th Int'l Conf. on Genetic Algorithms*, pages 424–431, 1993.

[127] M. J. D. Powell. Direct search algorithm for optimization calculations. *Acta Numerica*, 7, 1998.

[128] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in Fortran 77.* Cambridge University Press, 1992.

[129] W. L. Price. A controlled random search procedure for global optimization. In L. C. Dixon and G. P. Szego, editors, *Towards Global Optimization 2*, pages 71–84. North-Holland, Amsterdam, Holland, 1978.

[130] R. L. Rardin. *Optimization in Operations Research.* Upper Saddle River, N.J. : Prentice Hall, 1998.

[131] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In *Proc. of 3rd Int'l Conf. on Genetic Algorithms*, pages 191–197, 1989.

[132] N. Roenko. Simulated annealing under uncertainty. Technical report, Inst. of Operations Research , University Zurich, 1990.

[133] H. E. Romeijn and R. L. Smith. Simulated annealing for constrained global optimization. *Journal of Global Optimization*, 5(2):101–126, September 1994.

[134] I. Rosenberg. Minimization of pseudo-Boolean functions by binary development. *Discrete Mathematics*, 7:151–165, 1974.

[135] T. J. Van Roy. Cross decomposition for mixed integer programming. *Mathematical Programming*, pages 25–46, 1983.

[136] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Trans. on Neural Networks*, 5(1):96–101, 1994.

[137] N. V. Sahinidis. BARON: User's manual version 4.0. *Dept. of Chemical Engineering, Univ. of Illinois at Urbana Champaign*, 2000.

[138] E. Sandgren. Nonlinear integer and discrete programming in mechanical design optimization. *J. of Mechanical Design*, pages 223–229, 1990.

[139] M. S. Sarma. On the convergence of the Baba and Dorea random optimization methods. *Journal of Optimization Theory and Applications*, 66:337–343, 1990.

[140] F. Schoen. Stochastic techniques for global optimization: A survey on recent advances. *Journal of Global Optimization*, 1(3):207–228, 1991.

[141] M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In *Proc. of 4th Parallel Problem Solving from Nature*, 1996.

[142] M. Schoenauer and S. Xanthakis. Constrained GA optimization. In *Proc. of 5th Int'l Conf. on Genetic Algorithms*, 1993.

[143] R. Schultz, L. Stougie, and M.H. van der Vlerk. Two-stage stochastic integer programming: a survey. *Statist. Neerlandica*, 50(3):404–416, 1996.

[144] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proc. of 13'th Int'l Joint Conf. on Artificial Intelligence*, pages 290–295, 1993.

[145] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Proc. of 2nd DIMACS Challenge Workshop on Cliques, Coloring, and Satisfiability, Rutgers University*, pages 290–295, oct 1993.

[146] J. F. Shapiro. Generalized Lagrange multipliers in integer programming. *Operations Research*, 19:68–76, 1971.

[147] D. G. So and K. A. Dowsland. Simulation annealing: Application to simulation optimization. *OR 35 Conference , University of York*, Sept. 1993.

[148] F. J. Solis and R. J-B. Wets. Minimization by random search techniques. *Math. Operations Res.*, 6:19–30, 1981.

[149] R. Spaans and R. Luus. Importance of search-domain reduction in random optimization. *Journal of Optimization Theory and Applications*, 75(3):635–638, December 1992.

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

[151] Inc. Stanford Businees Software. Using AMPL/MINOS.

[152] J. Tind and L. A. Wolsey. An elementary survey of general duality theory in mathematical programming. *Mathematical Programming*, pages 241–261, 1981.

[153] A. Törn and A. Žilinskas. *Global Optimization.* Springer-Verlag, Berlin, 1989.

[154] Alain Trouve. Cycle decompositions and simulated annealing. *Mathematics Subject Classification*, 1991.

[155] R. J. Vanderbei. LOQO user's manual, version 3.10. *Technical Report No. SOR-97-08, Princeton University*, 1997.

[156] T. L. Vincent, B. S. Goh, and K. L. Teo. Trajectory-following algorithms for min-max optimization problems. *Journal of Optimization Theory and Applications*, 75(3):501–519, December 1992.

[157] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[158] A. Žilinskas. A review of statistical models for global optimization. *Journal of Global Optimization*, 2:145–153, 1992.

[159] B. W. Wah and Y.-J. Chang. Trace-based methods for solving nonlinear global optimization problems. *J. of Global Optimization*, 10(2):107–141, March 1997.

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

[161] B. W. Wah and Y. Shang. A discrete Lagrangian-based global-search method for solving satisfiability problems. In Ding-Zhu Du, Jun Gu, and Panos Pardalos, editors, *Satisfiability Problem: Theory and Applications*, pages 365–392. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1997.

[162] B. W. Wah and T. Wang. Constrained simulated annealing with applications in nonlinear constrained global optimization. In *Proc. Int'l Conf. on Tools with Artificial Intelligence*, pages 381–388. IEEE, November 1999.

[163] B. W. Wah and T. Wang. Efficient and adaptive Lagrange-multiplier methods for nonlinear continuous global optimization. *J. of Global Optimization*, 14(1):1–25, January 1999.

[164] 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.

[165] 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.

[166] T. Wang and B. W. Wah. Adaptive Lagrange-Multiplier methods for continuous nonlinear optimization. In *Proc. Symposium on Applied Computing*, pages 361–365, Atlanta, GA, February 1998. ACM.

[167] T. Wang and B. W. Wah. Handling inequality constraints in continuous nonlinear global optimization. *J. of Integrated Design and Process Science*, 2(3):1–10, 1998.

[168] X. D. Wang. An algorithm for nonlinear 0-1 programming and its application in structural optimization. *Journal of Numerical Method and Computational Applications*, 1(9):22–31, 1988.

[169] L. J. Watters. Reduction of integer polynomial programming to zero-one linear programming problems. *Operations Research*, 15:1171–1174, 1967.

[170] T. Westerlund and K. Lundqvist. Alpha-ECP, version 4.0, an interactive MINLP-solver based on the extended cutting plane method. *Process Design Laboratory, Abo Akademi University.*

[171] M. H. Wright. Interior methods for constrained optimization. In A. Iserles, editor, *Acta Numerica 1992*, pages 341–407. Cambridge University Press, 1992.

[172] S. J. Wright. *Primal-dual interior-point methods.* Philadelphia: Society for Industrial and Applied Mathematics, 1997.

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

[174] 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.

[175] X. Yao. Simulated annealing with extended neighborhood. *Int. Journal of Computer Mathematics*, 40:169–189, 1991.

[176] T. Yokota, M. Gen, and Y.-X. Li. Genetic algorithm for nonlinear mixed integer programmig problmes and its applications. *Int'l J. of Comp. and Indust. Eng.*, 30(4), 1994.

[177] Z. B. Zabinsky. Stochastic methods for practical global optimization. *Journal of Global Optimization*, 13:433–444, 1998.

[178] Z. B. Zabinsky, *et al.* Improving hit-and-run for global optimization. *Journal of Global Optimization*, 3:171–192, 1993.

[179] C. Zhang and H. P. Wang. Mixed-discrete nonlinear optimization with simulated annealing. *Engineering Optimization*, pages 277–291, 1993.