©Copyright by
Yixin Chen
2001

OPTIMAL ANYTIME SEARCH FOR CONSTRAINED NONLINEAR PROGRAMMING

BY

YIXIN CHEN

B.S., University of Science and Technology of China, China, 1999

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 *optimal anytime* stochastic search algorithms (SSAs) for solving general constrained nonlinear programming problems (NLPs) in discrete, continuous and mixed-integer space. The algorithms are general in the sense that they do not assume differentiability or convexity of functions. Based on the search algorithms, we develop the *theory of SSAs* and propose *optimal SSAs with iterative deepening* in order to minimize their expected search time. Based on the optimal SSAs, we then develop optimal *anytime* SSAs that generate improved solutions as more search time is allowed.

Our SSAs for solving general constrained NLPs are based on the theory of discrete constrained optimization using Lagrange multipliers that shows the equivalence between the set of constrained local minima (CLM_{dn}) and the set of discrete-neighborhood saddle points (SP_{dn}). To implement this theory, we propose *a general procedural framework* for locating an SP_{dn}. By incorporating genetic algorithms in the framework, we evaluate new constrained search algorithms: *constrained genetic algorithm* (CGA) and *combined constrained simulated annealing and genetic algorithm* (CSAGA).

All these algorithms are SSAs. One of the most important and difficult issues in using SSAs is the *scheduling of SSAs* in order to optimize the average search efficiency. Our research shows that SSAs can be scheduled in such a way that minimizes their expected search time. The theory proposes to use iterative deepening to identify the optimal (up to a constant factor) schedules in such a way that minimizes the expected search time when

compared to that of the same SSA run under an optimal schedule. We also extend the theory to identify optimal schedules in parallel processing of SSAs, and show that the search of an optimal parallel schedule is NP-complete. We propose to use iterative deepening to find a suboptimal parallel schedule and derive performance bounds between our suboptimal schedule and the optimal parallel schedule. The theory is general enough for SSAs and has been applied to CSA, CGA and CSAGA to develop optimal schedules for these algorithms.

Based on the optimal schedules, we propose optimal *anytime* SSAs that generate solutions of improved quality as more time is used. We propose new schedules to improve the quality levels in such a way that the total time of solving all quality levels is of the same order of magnitude as that of finding a constrained global optimum.

Finally, we apply our optimal anytime SSAs to solve a collection of engineering application benchmarks. Much better results have been reported in comparison with other existing methods.

To Junhong, and my parents

Acknowledgments

I am greatly indebted to my research advisor, Benjamin W. Wah, for his continuous support and encouragement. He taught me what doing research is all about, and how to think independently and creatively in research. His enthusiasm for understanding problems and the highest standards for scientific research were always guiding me to be a real scientist. He gave me full freedom in pursuing my own ideas. Most of all, his open mind and confidence inspired me all the time.

I would like to thank all the previous and current members in our research group for providing fruitful cooperations and a pleasant atmosphere. The unforgettable weekly group seminars and group meetings helped me a lot in the work. Special thanks go to Dr. Zhe Wu for his research in the theory of Lagrange multipliers for discrete optimization, and to Dr. Tao Wang for his development of CSA.

I deeply thank my parents for their everlasting love, patience and support.

Finally and the most specially, I wish to thank my fiancee, Junhong, for her continuous understanding, care and support. She made my graduate study life colorful and full of love.

This research was supported by National Aeronautics and Space Administration Contract NAS2-1230.

Table of Contents

Chapter

1 Introduction 1

2 Previous Work 15

3 Theory of Stochastic Search Algorithms 33

4 Optimal Schedules of SSAs for Constrained Optimization 55

5 Optimal Anytime Schedules of SSAs for Constrained Optimization . . . 75

6 Conclusions 113

Bibliography 116

Vita 131

List of Tables

List of Figures

Chapter 1

Introduction

Many engineering applications can be formulated as constrained *nonlinear programming problems* (NLPs). Examples include production planning, computer integrated manufacturing, chemical control processing, and structure optimization [65, 121, 148].

1.1 Problem Definition

Constrained NLPs can be solved by existing methods if they are specified in well-defined formulae that are differentiable and continuous. However, only special cases can be solved when they do not satisfy the required assumptions. For instance, sequential quadratic programming [46] cannot handle problems whose objective and constraint functions are not differentiable or whose variables are discrete or mixed. Since many applications involving optimization may be formulated by non-differentiable functions with discrete or mixed-integer variables, it is important to develop new methods for handling these optimization problems.

We divide in Table 1.1 constrained NLPs into 9 classes, according to their variable and function types. The variables of an NLP can be continuous, discrete or mixed-integer; and its objective and constraint functions can be continuous and differentiable, continuous but

Table 1.1: Definition of the 9 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

non-differentiable, or discontinuous. Different combinations of variable and function types, thus, lead to a total of 9 classes, C1-C9, as listed in Table 1.1.

We study in this thesis constrained NLPs in classes C2-C9. We *do not* consider in this research constrained NLPs in C1, which have continuous variables and continuous and differentiable objective and constraint functions. Such problems can be solved efficiently by existing methods, such as sequential quadratic programming (SQP) [46].

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

continuous NLPs has been studied elsewhere [170]. Of course, the accuracy of solutions cannot be scaled beyond the precision representable by floating-point numbers.

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

$$\begin{aligned} &\text{minimize} \quad f(x) \quad \text{where } x = [x_1, x_2, \ldots, x_n]^T \text{ is a vector of discrete variables,} \\ &\text{subject to} \quad g(x) \leq 0 \text{ and } h(x) = 0. \end{aligned} \tag{1.1}$$

Here, $f(x)$ is a lower-bounded objective function; $h(x) = [h_1(x), \cdots, h_m(x)]^T$ is a vector of m equality constraints; $g(x) = [g_1(x), \cdots, g_k(x)]^T$ is a vector of k inequality constraints; and all discrete variables in x are finite. Functions $f(x)$, $g(x)$, and $h(x)$ are not necessarily differentiable and can be either linear or nonlinear, continuous or discrete, and analytic or procedural. The search space X is the Cartesian product of discrete domains of all variables in x. Without loss of generality, we consider only minimization problems, knowing that maximization problems can be transformed into minimization problems by negating their objective functions.

Solutions to (1.1) cannot be characterized in ways similar to those of problems with differentiable functions and continuous variables. In the latter class of problems, solutions are defined with respect to neighborhoods of open spheres with radius approaching zero asymptotically. Such a concept does not exist in problems with discrete variables. To characterize solutions sought in discrete space, we define the following concepts on neighborhoods and constrained solutions in discrete space [163, 169]:

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

Definition 1.1 $\mathcal{N}_{dn}(x)$, the *discrete neighborhood* [23] of point $x \in X$, is a *finite* user-defined set of points $\{x' \in X\}$ such that $x' \in \mathcal{N}_{dn}(x) \Longleftrightarrow x \in \mathcal{N}_{dn}(x')$, and that for any $y^1, y^k \in X$, it is possible to find a finite sequence of points in X, $y^1, \cdots, y^k$, such that $y^{i+1} \in \mathcal{N}_{dn}(y^i)$ for $i = 1, \cdots k-1$.

Definition 1.2 Point $x \in X$ is called a *discrete-neighborhood constrained local minimum* (CLM_{dn}) if it satisfies two conditions: a) x is a feasible point, implying that x satisfies all the constraints $g(x) \leq 0$ and $h(x) = 0$, and b) $f(x) \leq f(x')$ for all $x' \in \mathcal{N}_{dn}(x)$, where x' is feasible. A special case in which x is a CLM_{dn} is when x is feasible and all its neighboring points in $\mathcal{N}_{dn}(x)$ are infeasible.

Definition 1.3 Point $x \in X$ is called a *discrete-neighborhood constrained global minimum* (CGM_{dn}) iff a) x is a feasible point, and b) for every feasible point $x' \in X$, $f(x') \geq f(x)$. The set of all CGM_{dn} is X_{opt}. According to our definitions, a CGM_{dn} must also be a CLM_{dn}.

1.2 Theory of Lagrange Multipliers for Solving Discrete Constrained NLPs

In this section, we briefly overview a new theory of discrete constrained optimization using Lagrange multipliers [163, 169] developed in our research group. In contrast to Lagrangian methods that work only for continuous constrained NLPs [45, 109], our new theory was derived for solving discrete constrained NLPs and can be extended to solve both continuous and mixed-integer NLPs. More importantly, its first-order necessary and sufficient condition on CLM_{dn} provides a strong theoretic foundation for developing global optimization methods for solving constrained NLPs.

The theory is based on solving discrete equality-constrained NLPs similar to that in (1.1) [163, 170]:

$$\begin{aligned} &\text{minimize} \quad f(x) \ \text{ where } x = [x_1, x_2, \ldots, x_n]^T \text{ is a vector of discrete variables,} \\ &\text{subject to} \quad h(x) = 0. \end{aligned} \tag{1.2}$$

A *generalized discrete augmented Lagrangian function* of (1.2) is defined as follows:

$$L_d(x, \lambda) = f(x) + \lambda^T H\left(h(x)\right) + \frac{1}{2}||h(x)||^2, \quad \lambda \in R^m \tag{1.3}$$

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

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

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*) \tag{1.4}$$

for all $x \in \mathcal{N}_{dn}(x^*)$ and all $\lambda \in R^m$.

The concept of SP_{dn} is very important in discrete problems because, starting from them, we can derive a first-order necessary and sufficient condition for CLM_{dn} that leads to local and global minimization procedures. This is stated formally in the following theorem [163]:

Theorem 1.1 *First-order necessary and sufficient condition for* CLM_{dn}. A point in the discrete search space of (1.2) is a CLM_{dn} iff it satisfies condition (1.4) for any $\lambda \geq \lambda^*$, where $\lambda \geq \lambda^*$ means that each element of λ is not less than the corresponding element of λ^*.

Requiring H in (1.3) to be non-negative is easy to achieve. Three such examples are $H(h(x)) = [|h_1(x)|, \cdots, |h_m(x)|]^T$, $H(h(x)) = [\max(h_1(x), 0), \cdots, \max(h_m(x), 0)]^T$, and $H(h(x)) = [|h_1^2(x)|, \cdots, |h_m^2(x)|]^T$. Note that these transformation functions are not used in Lagrange-multiplier methods in continuous space because they are not differentiable at

$H(h(x)) = 0$. However, they do not pose problems here because we do not require their differentiability.

Transformation H provides an easy way to handle inequality constraints because we can transform an inequality constraint, say $g_j(x) \leq 0$, into an equivalent equality constraint $\max(g_j(x), 0) = 0$. For this reason, we only consider problems with equality constraints in the rest of this chapter.

Theorem 1.1 is of great importance because it implies that, for discrete-space optimization, it is sufficient to search for SP_{dn} in order to find CLM_{dn}. Furthermore, global optimization, aiming to find a CGM_{dn}, amounts to finding SP_{dn} with the minimum objective value. In contrast, there is no similar results in continuous-space optimization [109, 170].

1.3 A General Framework to Look for SP_{dn}

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 propose in this thesis a framework [159] for solving constrained NLPs that unifies simulated annealing (SA), genetic algorithms (GA), and greedy searches in looking for saddle points. The framework allows us to show that many leading algorithms, such as DLM [169], CSA [162], and GA search of penalty formulations [117, 114] are similar algorithms that differ only in some components of the framework.

Based on the first-order necessary and sufficient conditions in Theorem 1.1, Figure 1.1 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

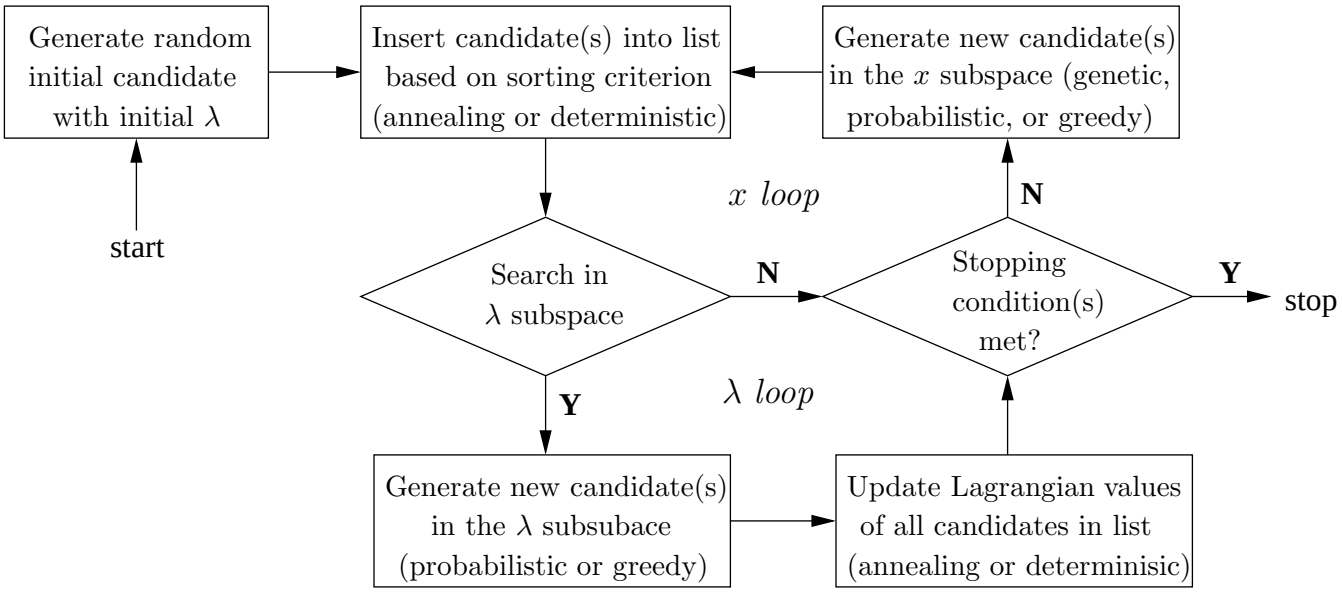


Figure 1.1: A general iterative stochastic procedural framework to look for SP_{dn}.

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 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 minima in the x subspace of $L_d(x, \lambda)$ and a local maxima in the λ subspace.

The significance of the procedural framework in Figure 1.1 is that it provides a unified problem-independent way to implement Theorem 1.1. By designing the four components of the framework, we can design new constrained optimization algorithms to look for CLM_{dn}. Further, by realizing that existing algorithms are special cases of the general framework, we can improve these algorithms by tuning one or more of their components in the framework.

Our survey later shows that existing algorithms looking for SP_{dn}, such as DLM and CSA, fit into this framework. In Chapter 4, we study various mechanisms and their combinations for performing ascents in the x subspace and descents in the λ subspace.

1.4 Optimization Algorithms with Stochastic Behavior

In this thesis, we are interested in studying algorithms with random components or random starting points. These *stochastic search algorithms* (SSA) do not always lead to the same solution (or solutions of similar quality) every time the algorithm is run. They may utilize the differentiability of functions or may rely on sampling to generate new probes. Examples of such algorithms include local search with random starting points, global search with random components to escape from local traps, and stochastic global optimization algorithms, such as SA and GA.

Among the parameters that control the behavior of an SSA, two of the most important ones are N (number of probes allowed for one run of the algorithm), and Q (the quality of solutions desired). In general, the longer an algorithm is allowed to run, the better the solutions it can generate.

Assuming that an SSA iterates by generating probes in a search space and that each probe is independent of the incumbent already found, the performance of one run of such an algorithm can be characterized by the number of probes made (or CPU time taken) and the convergence and reachability probabilities defined as follows:

Definition 1.4 The *convergence probability* $P_C(N, Q)$ of an SSA is the probability that it will find a solution of quality Q in its N^{th} probe.

Among various convergence conditions, the strongest one is asymptotic convergence to a solution of desired quality with probability one. In this case, the search stops at a solution of

desired quality in the last iteration with probability one, as the number of probes approaches infinity. This concept is of theoretical interest only but is not of practical importance, because an algorithm with asymptotic convergence does not imply that it will find better solutions with higher probabilities when terminated in finite time.

Definition 1.5 The *reachability probability* $P_R(N, Q)$ of an SSA is the probability that it will find a solution of quality Q in any of the N probes it has made.

$P_R(N, Q)$ measures the probability of finding an incumbent of desired quality Q after the SSA has made N probes. In general, $P_R(N, Q)$ is a non-decreasing function of N and converges to 1 as $N \to \infty$.[2] The relationship between $P_C(N, Q)$ and $P_R(N, Q)$ can be characterized by the following equation:

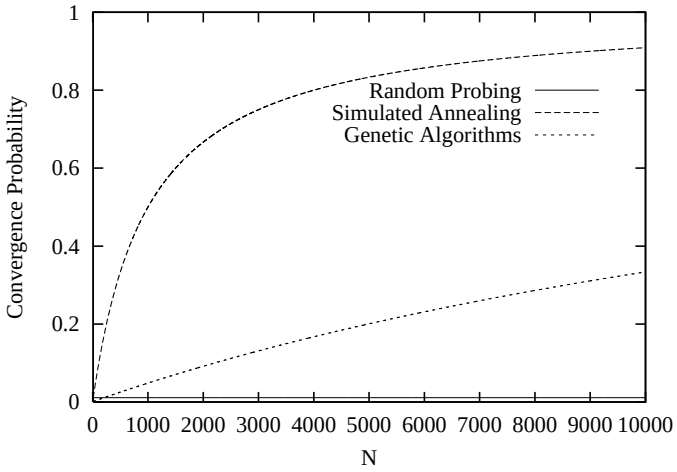
$$P_R(N, Q) = 1 - \prod_{j=1}^{N} (1 - P_C(j, Q)), \tag{1.5}$$

assuming that all probes are independent (a simplifying assumption). Reachability can be maintained by keeping the incumbent at any time and by reporting it when the algorithm stops.

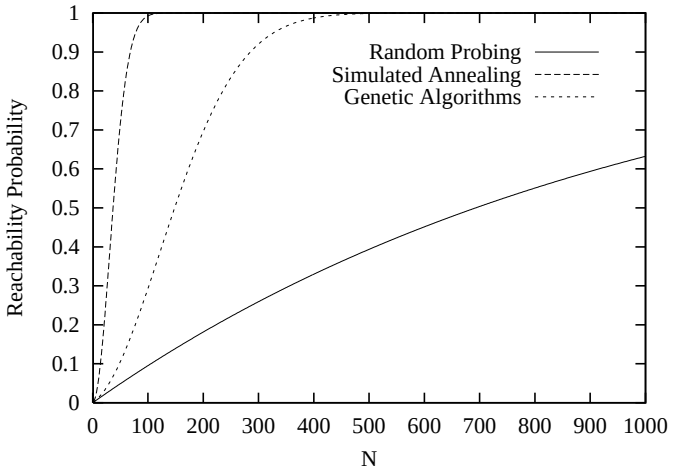
From (1.5), we see that $P_R(N, Q)$ is less than 1 unless $P_C(N, Q)$ reaches 1. For an SSA with asymptotic convergence, $P_C(N, Q) = 1$ only when N approaches infinity. Hence, all SSAs will have $P_R(N, Q)$ *less than* 1 with finite N.

Figure 1.2 shows examples of $P_C(N, Q)$ and $P_R(N, Q)$ of three SSAs. Figures 1.2a and 1.2b show the convergence and reachability probabilities of random probing ($P_C(N, Q) = 1/R$, where R is the number of states in the search space). Other SSAs, such as SA and GA, have $P_C(N, Q)$ increasing with respect to N, as shown in Figure 1.2a.

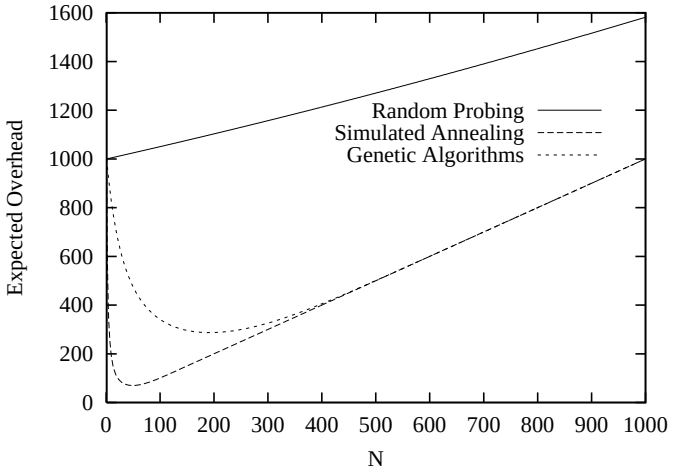
[2]For simplicity of notations, we use $P_C(N, Q)$ and $P_R(N, Q)$ to denote, respectively, the convergence and reachability probabilities when we focus on a fixed solution quality level Q.



a) $P_C(N, Q)$



b)$P_R(N, Q)$



c) $E(B_s(S^{ICS(N)}, Q)) = \frac{N}{P_R(N,Q)}$

Figure 1.2: Examples showing $P_C(N, Q)$, $P_R(N, Q)$, and $E(B_s(S^{ICS(N)}, Q))$ for random probing, SA and GA.

Although $P_R(N,Q)$ is less than 1 for finite N and its exact value is hard to estimate and control for a given SSA, we can always improve the chance of finding a solution of quality Q by running the same algorithm multiple times from random starting points, each examining the search space by N probes. Let $E(B_s(S^{ICS(N)},Q))$[3] be the expected total number of probes using a schedule $S^{ICS(N)}$ of multiple independent runs of an SSA (each with N probes) in order to find a solution of quality Q, we have,

$$E(B_s(S^{ICS(N)},Q)) = \sum_{j=1}^{\infty} P_R(N,Q)(1-P_R(N,Q))^{j-1} \times N \times j = \frac{N}{P_R(N,Q)} \tag{1.6}$$

Figure 1.2c shows examples of $E(B_s(S^{ICS(N)},Q))$ for three SSAs. We see that $E(B_s(S^{ICS(N)},Q))$ is convex for SA and GA, each with an absolute minimum with respect to N in $(0,\infty)$ that minimizes $E(B_s(S^{ICS(N)},Q))$. In other words, there is an optimal number of probes that minimizes the expected overhead of finding a solution. Hence, if the algorithm is run multiple times using a suitable number of probes each, then the expected time to find a solution of quality Q can be minimized.

This observation generally holds for all SSAs, including SSAs for constrained nonlinear optimization. As an example, Table 1.2 illustrates such an trade-offs between N_α, the number of probes spent in one run, and $\frac{N_\alpha}{P_R(N_\alpha)}$, the expected overhead, in using CSA [162] (a constrained SSA using simulated annealing) to solve a constrained NLP with a 10-dimensional Rastrigin function as its objective:

$$\begin{aligned} &\text{minimize} \quad f(x) = F\left(10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i)), 200\right) \\ &\text{subject to} \quad |(x_i - 4.2)(x_i + 3.2)| \leq 0.1 \qquad \text{for } n = 10, \end{aligned} \tag{1.7}$$

[3]This notation is used in our theory of SSAs. See Chapter 3 for details.

Table 1.2: An example illustrating trade-offs between the expected total number of probes in multiple runs of CSA to find a CGM_{dn}, the number of probes used in each run, and the probability of success in each run. The optimal N_α at $N_\alpha = 2960$ leads to the minimum average total number of probes to find a CGM_{dn}. Note that the probability of success is not the highest in one run using the optimal N_α. (The problem solved is defined in (1.7). Each cooling schedule is run 200 times using $f' = 200$.)

N_α	cooling schedule length	998	1480	2075	**2960**	4345	7980	24140	69635
T_α	avg. CPU time of one run	0.026	0.036	0.050	**0.074**	0.11	0.18	0.54	1.58
$P_R(N_\alpha)$	succ. prob. of one run	1%	10%	25%	**40**%	55%	70%	85%	95%
$\frac{1}{P_R(N_\alpha)}$	avg. runs to find sol'n	100	10	4	**2.5**	1.82	1.43	1.18	1.05
$\frac{N_\alpha}{P_R(N_\alpha)}$	avg. probes to find sol'n	99800	14800	8300	**7400**	7900	11400	28400	73300
$\frac{T_\alpha}{P_R(N_\alpha)}$	avg. time to find sol'n	2.6	0.36	0.20	**0.19**	0.20	0.25	0.64	1.7

where F is the transformation function defined later in (5.2). A run of CSA is successful if it finds a feasible point with objective value less than or equal to 200 in this run, and the probability to hit a CGM_{dn} is calculated by the percentage of successful runs over 200 independent runs.

Table 1.2 shows that $P_R(N_\alpha)$ increases towards one when N_α is increased. An N_α too large is generally undesirable because the expected number of probes in (1.6) is large, even though the success probability in one run of CSA approaches one. On the other hand, if N_α is too small, then the success probability in one run of CSA is low, leading to a large expected number of probes in (1.6). An optimal N_α is one in which CSA is run multiple times and the expected total number of probes in (1.6) is the smallest. Table 1.2 also shows that the number of probes expended is closely related to the execution time.

Table 1.2 shows that $\frac{N_\alpha}{P_R(N_\alpha)}$ is a convex function with a minimum at $N_\alpha = 2960$. That is, the average total number of probes of multiple runs of CSA to find a CGM_{dn} first decreases

and then increases, leading to an optimal N_α of 2960 and an average of 2.5 runs of CSA to find a CGM_{dn}.

Another important parameter that controls the behavior of an SSA is the solution quality desired: a more relaxed Q will lead to a smaller minimum point in $E(B_s(S^{ICS(N)}, Q))$. We exploit this trade-off in designing strategies in order to refine Q after relaxed solutions have been found.

1.5 Research Goals and Contributions

There are three major research goals of this thesis:

a) *Design of efficient constrained optimization algorithms.* We develop new efficient algorithms for solving constrained NLPs whose functions are not necessarily differentiable and whose variables may be discrete, mixed-integer, or continuous. Our algorithms, based on the theory of Lagrange multipliers for discrete constrained optimization, look for discrete-neighborhood saddle points SP_{dn} in Lagrangian formulations of NLPs, since there is a one-to-one correspondence between $SP_{dn}$and CLM_{dn}. The framework in Figure 1.1 provides a general approach to locate SP_{dn}. Existing algorithms, such as DLM and CSA, are shown to be special cases of this framework. To generate better probes, we include genetic search in the original-variable space and suitable combinations of mechanisms in order to lead to efficient searches.

b) *Optimal sequential and parallel search schedules.* We develop a new theory of *stochastic search algorithms* (SSAs) which shows that running a sequence of SSAs according to a prescribed schedule of quality desired Q and time allowed T will minimize, up to a constant factor, the expected search time in (1.6) when compared to that of the same SSA run under optimal Q and T. We develop the schedule using *iterative deepening* and give the sufficient

conditions for the schedule to be optimal. We then extend the theory when an SSA is run on multiple processors and develop optimal schedules for running SSAs in parallel. Based on the search algorithms derived from the framework in (a), we develop optimal schedules for these algorithms.

c) *Optimal anytime search.* Based on SSAs scheduled optimally, we propose new *anytime* algorithms that generate solutions of improved quality Q as more time is allowed, eventually finding a CGM_{dn}. We develop new schedules with improved quality targets Q in such a way that the total time of solving all quality levels is of the same order of magnitude as that of finding a CGM_{dn}.

1.6 Outline of This Thesis

This thesis is organized as follows. In Chapter 2, we survey existing work in solving the two classes of constrained NLPs studied in this research. Chapter 3 presents the mathematical theory of SSAs. Chapter 4 presents constrained genetic algorithms by incorporating genetic search in the general framework and the theory of SSAs for finding their optimal schedules. Based on the optimal schedules, Chapter 5 presents optimal anytime SSAs that always generate improved solutions as more search time is spent and shows experimental results on applying our anytime algorithms to solve three sets of standard engineering benchmarks. Finally, Chapter 6 summarizes our research work presented in this thesis.

Chapter 2

Previous Work

In this chapter, we summarize previous work in the literature on the two classes of problems formulated in Chapter 1. We classify these approaches based on whether they utilize the derivatives of functions. We also overview two existing algorithms that implement the general search framework for finding SP_{dn}.

2.1 Derivative-Free Methods

We first summarize existing approaches that can be applied to solve constrained NLPs defined in Chapter 1 and that do not require the differentiability of functions. 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: transformations into constrained 0-1 NLP problems, direct solution methods, Lagrangian relaxation, and transformations into unconstrained problems using penalty formulations.

2.1.1 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 [88].

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 [167, 79, 80]. However, linearization only works for problems with a few simple nonlinear terms, because it introduces many new variables and constraints, leading to much larger problems. Second, algebraic methods [86, 130] express an objective function as a polynomial function of its variables and their complements. These only work for cases in which all the constraints can be removed. Last, cutting-plane methods [82, 83] reduce a constrained nonlinear 0-1 problem into a generalized covering problem. In these methods, the objective is assumed to be linear or is linearized. However, they are limited because not all nonlinear 0-1 problems can be transformed this way.

For problems with highly nonlinear objective and constraint functions, transformations into nonlinear constrained 0-1 problems are not helpful because existing techniques for solving these problems are very limited. On the other hand, if a nonlinear problem with continuous variables can be linearized, then existing linear programming methods generally have well-defined algorithmic steps and stopping conditions for locating CLM_{dn}.

2.1.2 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 approaches. One major approach is based on rejecting, discarding [32, 33, 126] or repairing [98] methods that try to avoid infeasible points. This approach, however, has difficulty in handling nonlinear

constraints whose feasible regions may be very hard to locate, leading to the generation of mostly infeasible points that are rejected.

The other approach is based on enumeration or randomized search techniques [95]. Enumerative branch-and-bound algorithms [108, 165] 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 fixed a priori ordering, and functions may be approximated by piecewise linear and nonlinear functions. When constraints are nonlinear and lower bounds are hard to estimate, these methods do not perform well because lower bounds found using linearized constraints may be inaccurate when constraints are highly nonlinear, and inaccurate bounds may lead to incorrect pruning and infeasible solutions.

2.1.3 Lagrangian relaxation

There is a class of algorithms called *Lagrangian relaxation* [72, 74, 68, 142, 41] proposed in the literature that should not be confused with our proposed discrete constrained optimization method using Lagrange multipliers. Lagrangian relaxation reformulates a *linear* integer minimization problem:

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

into the following form:

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

Obviously, the new relaxed problem is simple and can be solved efficiently for any given vector λ. The method is based on Lagrangian Duality theory [151] upon which a general

relationship between the solution to the original minimization problem and the solution to the relaxed problem can be deduced. There was some research [34] addressing nonlinear optimization problems. However, as pointed out in [151], Lagrangian relaxation aims to find an optimal primal solution given an optimal dual solution, or vice versa. This approach is simple in the case of linear functions but does not work well for nonlinear functions.

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. A typical penalty formulation is as follows:

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

where $f(x)$ is the objective function, and w_i is the i^{th} weight coefficient to be determined.

A simple solution is to use a *static-penalty formulation* [45, 109] that sets w_i to be a large static positive value. This way, a local minimum of $eval(x)$ is a CLM_{dn}, and a global minimum of $eval(x)$ is a CGM_{dn}. However, if the w_i's are too large, they will cause the search space to be very rugged. Consequently, feasible solutions are difficult to be located by local-search methods, because it is hard for these methods to escape from deep local minima after getting there and to move 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 $eval(x)$ may not even be feasible solutions to the original constrained problem.

In general, hard-to-satisfy constraints should carry larger penalties than easy-to-satisfy ones. However, the degree of difficulty in satisfying a constraint may depend on other con-

straints in a problem. Without the ability to vary penalties dynamically, search techniques for unconstrained problems will likely get stuck in infeasible local optima.

Dynamic-penalty methods address the difficulties in static-penalty methods by gradually increasing penalties. By transforming (1.1) into a sequence of unconstrained subproblems with increasing penalties, dynamic-penalty methods employ 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 is solved optimally [45, 109]. Optimality in each subproblem is, however, difficult to achieve in practice, given only finite amount of time to solve each subproblem. This leads to suboptimal solutions when the result in one subproblem is not optimal. Moreover, the solutions to intermediate subproblems may not be related to the final goal of finding CLM_{dn} or CGM_{dn} when penalties are not large enough. Approximations to the process that sacrifice global optimality of solutions have been developed [102, 111].

Various constraint handling techniques have been developed based on dynamic-penalty formulations in [94, 97, 114, 125, 115, 77, 32, 138, 122, 137, 42]. Besides requiring domain-specific knowledge, most of these heuristics have difficulties either in finding feasible regions or in maintaining feasibility for nonlinear constraints, and get stuck easily in local minima [117, 114]. Some typical constraint-handling techniques are explained next. Note that these techniques are all heuristic in nature.

In general, methods based on dynamic-penalty formulations can at best, but have no guarantee to, find CLM_{dn}. Consider a simple problem with only one constraint function $h_1(x)$ and an objective $f(x)$. If a dynamic penalty-based algorithm starts from x^* and $|h_1(x^*)| = min_{x \in \mathcal{N}_{dn}(x^*) \cup \{x^*\}}\{|h_1(x)|\} > 0$ and $f(x^*) = min_{x \in \mathcal{N}_{dn}(x^*) \cup \{x^*\}}\{f(x)\}$, then regardless of 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 optimization.

Local Search Based on Penalty Formulations. Local search methods based on penalty formulations rely on local probes or perturbations to generate candidate trial points and advance their search trajectories. Typical methods include greedy search and hill climbing [31, 109]. These methods usually have well-defined stopping conditions that are closely related to their algorithmic steps. They, however, have difficulties in finding feasible solutions and get stuck easily in infeasible local minima when the weight coefficients in (2.3) are not chosen properly. For this reason, local search methods are often combined with various global search techniques to find high-quality solutions.

Global Search Based on Penalty Formulations. Global search methods based on penalty formulations normally involve techniques to escape from the attraction of local minima or valleys in a search space. Typical methods include tabu search [76, 78, 40], multi-start [136, 134, 90, 145], heuristic repair methods [53], break-out strategies [119], guided local search (GLS) [156], random walk [140, 139], learning-based approaches [49, 35, 36, 47], and Bayesian methods [118, 157, 152].

In general, global search methods based on penalty formulations can at best find constrained local minima (CLM_{dn}), given large penalties on constraints. However, as mentioned before, selecting suitable penalties often proves to be difficult, and most current methods use heuristics to select penalties. To address this issue, we describe in the following a systematic way to locate CGM_{dn}.

Stochastic Global Optimization Based on Penalty Formulations. Given sufficiently large penalties on constraints and time approaching infinity, stochastic global optimization

methods, based on penalty transformations for solving discrete constrained problems, are able to find CGM_{dn}. In practice, it is difficult to achieve global optimality 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) [103] and genetic algorithms (GA) [91] are two well-known stochastic global optimization algorithms for solving unconstrained NLPs.

SA is a typical global optimization algorithm with asymptotic convergence. It has been applied successfully to solve unconstrained optimization problems [128, 106, 57, 26]. Its basic idea is to not only accept trial points with improved objective values, but also accept trial points with worse objective values probabilistically . For a new trial point x' generated from the current point x, a typical acceptance probability is the Metropolis probability [73] defined as follows:

$$A_T(x, x') = \exp\left(-\frac{(f(x') - f(x))^+}{T}\right) \tag{2.4}$$

where

$$a^+ = \begin{cases} a, & a > 0 \\ 0, & a \leq 0. \end{cases} \tag{2.5}$$

and T is a control parameter called temperature that is set initially to be a large value and then reduced slowly.

One 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 slow enough logarithmic cooling schedule to decrease temperature T [70]. This important property makes SA promising in practice and has led to many efficient variations of SA, like fast simulated annealing (FSA) [150], simulated annealing with extended neighborhood [173], adaptive

simulated annealing [96, 166], and a combination of simulated annealing with local search heuristics [112, 37].

The advantage of SA lies in its global convergence property. However, when applied to solve constrained NLPs using penalty formulations, the global convergence of SA only ensures that a search will converge to an optimal solution of the penalty function (that may be an infeasible point, a CLM_{dn}, or a CGM_{dn} of the original constrained problem). 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) [117, 113, 81, 66, 114, 132, 120, 123, 89, 63, 43] is a stochastic global optimization algorithm with reachability. It generally maintains a population of individuals. In each generation, it uses some genetic operators, such as crossovers and mutations, to generate new points. All the old and new points are then ranked based on a fitness function, which is the objective function in case of unconstrained optimization problems. It then starts a new generation using the top-fit individuals. After iterating for a sufficient number of generations, it is expected to converge to the best individual.

Similar to SA, GA requires a good choice of penalties in a penalty formulation in order for the search to converge to a CGM_{dn} of the original constrained problem. Otherwise, the search may end up finding only CLM_{dn} or even infeasible solutions.

Some variants of penalty formulations have been used in GAs to handle constraint. These include methods with multi-level static penalties [94], generation-based dynamic penalties [97], annealing penalties [114], and adaptive penalties [42, 84, 125]. Although they differ in their ways of modifying the penalties, all of them adjust penalties at the end of each generation, instead of when the unconstrained problem at previous penalty levels has been

minimized. Accordingly, these methods cannot achieve asymptotic convergence. Also, they have difficulties in choosing suitable penalties for different constrained NLPs.

Besides SA and GA, there are some other stochastic global optimization strategies, although not as popular, that can be applied to solve discrete constrained NLPs using penalty formulations. These include random search [174, 144], adaptive search [124, 50], controlled random search (CRS) [27, 25] and improved hit-and-run (IHR) [175, 174] methods.

Many of these random search methods do not work well because: a) they depend heavily on a good choice of penalties in order to find feasible solutions; and b) the trial point generator, normally based on some random distributions, has no bearing to the final goal of finding CGM_{dn} or CLM_{dn}.

Limitations of penalty methods The major difficulty in penalty formulations is that a local/global minimum of an unconstrained penalty formulation is only *a necessary but not a sufficient condition* for a constrained local/global minimum of the original NLP. Only when penalties are chosen correctly that a local minimum of a penalty formulation is a constrained local minimum of the original NLP. In general, there is no single procedure to choose penalties for a general NLP in such a way that a local minimum of an unconstrained penalty formulation is a constrained local minimum of the original NLP. It follows that existing algorithms do not necessarily solve the original NLP by looking for local/global minima in an unconstrained penalty formulation.

2.2 Methods Requiring the Differentiability of Functions

In this section, we summarize previous methods for solving constrained NLPs that utilize the differentiability of functions. We classify them into two categories: methods for solving continuous constrained NLPs and those for solving discrete constrained NLPs and mixed-integer constrained NLPs (MINLPs).

2.2.1 Continuous constrained NLPs

As discussed in Chapter 1, we are interested in this research on constrained NLPs whose variables are continuous and whose functions are not differentiable. Hence, methods that require differentiability *cannot* be applied. Such methods include interval methods requiring derivatives (such as interval-Newton methods) [101, 100, 87], gradient-descent methods, Newton's method [109, 121, 129], trajectory methods [158, 155, 58, 135, 30, 143, 149, 28, 152], covering methods requiring derivatives [38, 39], penalty methods requiring derivatives [52, 69, 152], and Lagrange-multiplier methods [109]. The last class of methods is the basis on which sequential quadratic programming [46] is based.

2.2.2 Discrete and mixed-integer constrained NLPs

Penalty formulations of discrete and mixed-integer NLPs with differentiable functions can be solved by unconstrained algorithms that require differentiability, such as Newton's method and conjugate-gradient methods. These methods suffer the same limitations of derivative-free penalty-based methods discussed in Section 2.1.4.

Methods exploiting the convexity of functions This approach generally formulates an MINLP in a Lagrangian formulation and then decomposes it into subproblems in such

a way that, after fixing a subset of the variables, the resulting subproblem is convex and can be solved easily. Although these methods may not require the derivatives of continuous subproblems, differentiability can help improve solution time and quality. For this reason, we classify these methods under this class. There are three types of these methods.

a) *Generalized Benders decomposition* (GBD) [64, 71, 44] computes at 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, whereas master problem is derived through nonlinear duality theory. Its major disadvantage is that it is only applicable to a class of MINLPs with restrictions on their variable space, such as a nonempty and convex continuous subspace with convex objective and constraint functions.

b) *Outer approximation* (OA) [62, 61] solves MINLPs by a sequence of approximations in which each approximated subproblem contains the original feasible region. It is similar to GBD except that the master problem is formulated based on primal information and outer linearization. This method requires the continuous subspace to be a nonempty, compact, and convex set, and the objective and constraint functions to be convex.

c) *Generalized cross decomposition* (GCD) [64, 92, 93, 131] iteratively alternates 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 proper convex functions.

The major limitation of these convexity-based methods is that they are only applicable to specific classes of MINLPs whose convex subproblems can be constructed and solved. Accordingly, their application to solve general MINLPs is very restricted. Even for cases in

which convex subproblems can be constructed, they are prohibitively expensive because the number of convex subproblems may be too large to be enumerated.

2.3 Existing Algorithms Implementing Theorem 1.1

The first-order necessary and sufficient condition in Theorem 1.1 implies that a strategy looking for SP_{dn} with the minimum objective value will result in a CGM_{dn}, because there is a one-to-one correspondence between CGM_{dn} and SP_{dn}. Among the many ways to look for SP_{dn}, we review two such methods in this section. Both methods fit into the problem-independent framework for finding SP_{dn} in Figure 1.1.

2.3.1 Discrete Lagrangian method

The first algorithm is the *discrete Lagrangian method* (DLM) [163, 141, 169, 172, 171]. It is an iterative local search that looks for SP_{dn} by greedy descents in the x subspace, while occasionally performing ascents in the λ subspace in order to increase the penalties of violated constraints.

Given a starting point (x^0, λ^0), DLM [163] iterates the following steps:

$$x \textit{ loop:} \quad x^{k+1} = x^k \oplus \Delta_x L_d(x^k, \lambda^k) \tag{2.6}$$

$$\lambda \textit{ loop:} \quad \lambda^{k+1} = \lambda^k + c_1 \cdot H\left(h(x^k)\right). \tag{2.7}$$

The *x loop* performs neighborhood descents of L_d in the x subspace, where L_d is the generalized discrete augmented Lagrangian function defined in (1.3). $\Delta_x L_d(x^k, \lambda^k)$ points in a direction where L_d is decreasing, which can be obtained by probing all the points in $\mathcal{N}_{dn}(x)$. Last, $\oplus$ is the vector-addition operator, where $x \oplus y = (x_1 + y_1, x_2 + y_2, \ldots x_n + y_n)$.

The λ *loop* performs ascents in the Lagrange-multiplier subspace in order to suppress constraint violations, if they exist; H is the transformation function in (1.3); and c_1 is a constant controlling the speed of changing the Lagrange multipliers.

It can be shown that the point where DLM stops is a CLM_{dn} when the number of neighborhood points is small enough to be enumerated in each descent in the x subspace [170]. However, when the number of neighborhood points is too large to be enumerated, then the point where DLM stops is a feasible point but not necessarily a SP_{dn}. Also, as a local search method, DLM may get stuck at local minimal points instead of a CGM. Global search strategies have been incorporated to help DLM escape from local suboptimal traps [141, 169, 172, 171].

2.3.2 Constrained simulated annealing

The second algorithm is constrained simulated annealing (CSA) [162, 161, 164], an application of Theorem 1.1 to look for CGM_{dn} with asymptotic convergence. CSA looks for discrete-space saddle points by performing both probabilistic descents of L_d in the original-variable subspace and probabilistic ascents of L_d in the Lagrange-multiplier subspace in order to satisfy all the constraints in (1.1).

Figure 2.1 shows the basic procedure of CSA. Line 2-3 starts CSA from an initial temperature T^0 and a randomly generated starting point. T^0 is decided empirically by sampling a certain number of points in the search space: it is large if the samples differ a lot in their objective values and constraint violations; otherwise, T^0 is set to be a smaller value.

Line 4 sets N_T, the number of trials at each temperature. In [162, 161], N_T was set to be $\zeta(20n + m)$, where $\zeta = 10(n + m)$, n is the number of variables, and m is the number of equality constraints.

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 2.1: CSA: constrained simulated annealing procedure [162].

Line 5-11 implement the annealing process. Line 7 generates a trial point $\mathbf{x}'$ in the neighborhood $\mathcal{N}(\mathbf{x})$ of the current point $\mathbf{x}$, using a generation probability $G(\mathbf{x}, \mathbf{x}')$, where $\mathcal{N}(\mathbf{x})$ is defined as follows:

$$\mathcal{N}(\mathbf{x}) = \{(x', \lambda) \text{ where } x' \in \mathcal{N}_{dn}(x)\} \cup \{(x, \lambda') \text{ where } \lambda' \in \mathcal{N}_{cn}(\lambda)\} \tag{2.8}$$

$$\mathcal{N}_{cn}(\lambda) = \{\mu \in \Lambda \mid \mu < \lambda, \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\} \bigcup \{\mu \in \Lambda \mid \mu > \lambda, \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\},$$

where $\lambda > \mu$ means that every element of λ is larger than or equal to the corresponding element of μ, and that at least one element of λ is strictly larger than the corresponding element of μ. Hence, point $\mathbf{x} = (x, \lambda)$ has two sets of neighbors: (x', λ) and (x, λ'). Trial point (x', λ) is a neighbor to (x, λ) if x' is a neighbor to x in discrete subspace X, and (x, λ') is a neighbor to (x, λ) if λ' is a neighbor to λ in continuous subspace Λ and $h(x) \neq 0$. Neighborhood $\mathcal{N}_{cn}(\lambda)$ prevents λ_i from being changed when the corresponding constraint is satisfied, *i.e.*, $h_i(x) = 0$.

Line 8 decides whether to accept trial point x' using acceptance probability $A_T(\mathbf{x}, \mathbf{x}')$. defined as follows:

$$A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} exp\left(-\frac{(L_d(\mathbf{x}')-L_d(\mathbf{x}))^+}{T}\right) & \text{if } \mathbf{x}' = (x', \lambda) \\ exp\left(-\frac{(L_d(\mathbf{x})-L_d(\mathbf{x}'))^+}{T}\right) & \text{if } \mathbf{x}' = (x, \lambda') \end{cases} \tag{2.9}$$

where $(a)^+ = a$ if $a > 0,$ and $(a)^+ = 0$ otherwise for all $a \in R$.

$A_T(\mathbf{x}, \mathbf{x}')$ allows a search to find points with smaller Lagrangian value in a stochastic fashion. Stochastic descents of L_d in the x subspace try to enforce a smaller Lagrangian value, implying either a smaller objective-function value or a smaller total constraint violation, both of which are desirable. In contrast, probabilistic ascents of L_d, which are absent from traditional SA, try to increase the Lagrange multipliers on violated constraints stochastically in order to force them into satisfaction.

Line 10 decreases temperature T using a geometric schedule in order for the annealing process to work.

Asymptotic Convergence of CSA. CSA has been proved [161, 162] to converge asymptotically to a CGM_{dn}. The proof is done by modeling the annealing process by an inhomogeneous Markov chain, showing that the Markov chain is strongly ergodic, proving that the Markov chain minimizes an implicit virtual energy based on the framework of generalized SA (GSA) [153], and showing that the virtual energy is at its minimum at any CGM_{dn}. Details of the proofs can be found in [164].

The main result of CSA [161, 162] is summarized in the following theorem.

Theorem 2.1 *Asymptotic convergence of CSA* [162, 161]. The Markov chain modeling CSA converges asymptotically to CGM_{dn} $x^* \in X_{opt}$ with probability one.

Table 2.1: Both DLM and CSA fit into the general iterative search framework for finding SP_{dn}. They differs in their implementations of the four components of the framework.

Framework Components	DLM	CSA
Generation of x probes	greedy	probabilistic
Generation of λ probes	greedy	probabilistic
Insertion of x probes	deterministic	annealing
Insertion of λ probes	deterministic	annealing

To summarize, CSA is a powerful method in two aspects. First, it is able to solve general discrete, continuous and mixed-integer constrained optimization problems in a unified fashion. 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. However, by choosing an appropriate finite cooling schedule, CSA can find CGM_{dn} with a high probability. By making multiple runs of CSA with finite schedules, we can improve the success probability of finding a CGM_{dn}. CSA has been tested to work well for solving nonlinear benchmark problems [161].

The major deficiency of CSA is that, as a sampling method, it usually takes a very long time to converge to a CGM and is, therefore, not efficient for solving large complex constrained NLPs. Also, it is very difficult to determine suitable cooling schedules for CSA to solve different NLPs.

Both DLM and CSA fit into the general search framework in Figure 1.1 for finding SP_{dn}, each maintaining a list of one candidate at any time. Table 2.1 summarizes how DLM and CSA implements the four major components of the general search framework. DLM entails greedy generations in the x and λ subspaces, deterministic insertions into the list of candidates, deterministic acceptance of candidates, and stopping the update of λ when all

constraints are satisfied. On the other hand, CSA generates new probes randomly along one of the x or λ variables, accepts them based on the Metropolis probability if L_d increases along the x dimension and decreases along the λ dimension, and stops updating λ when all constraints are satisfied.

2.4 Summary

We have surveyed in this chapter existing work for solving discrete, continuous, and mixed-integer constrained NLPs. Our survey shows that direct-solution methods generally have difficulties in solving a constrained NLP because they involve strategies that try to satisfy multiple nonlinear constraints simultaneously. A more viable approach is to transform a constrained NLP into an unconstrained penalty function and solve it by existing unconstrained search algorithms. Penalty formulations are also general enough to cover the two classes of problems defined in Chapter 1, although other formulations may lead to more efficient algorithms but are more restricted in their scope. Among the methods for solving penalty formulations, local-search and global-search methods can at best find local minima of a penalty function but have no guarantee of finding any global optimum. Stochastic search algorithms (SSAs), such as SA and GA, are general global optimization algorithms that can find global optimal points of unconstrained problems. For these reasons, we focus only on penalty formulations and SSAs in this thesis.

Existing search algorithms solving an unconstrained penalty formulation look for unconstrained local/global minima to the penalty formulation. These minima are only necessary but not sufficient for them to be constrained local minima in the original constrained NLP. When penalties are not chosen properly, constraints in the original NLP at local minima of

the penalty function may not be satisfied. Hence, existing penalty-based algorithms are not guaranteed to solve the original constrained NLP.

The newly developed theory of discrete constrained optimization using Lagrange multipliers characterizes constrained local minima in a discrete constrained NLP by a necessary and sufficient condition on points in the corresponding unconstrained penalty function. Such a one-to-one correspondence between SP_{dn} and CLM_{dn} enables the search for points satisfying the necessary and sufficient condition in the unconstrained penalty function in order to locate CLM_{dn} of the original NLP. Although these conditions are based on a discrete variable space, they can be extended to work in continuous and mixed-integer space after discretizing continuous variables finely and after evaluating errors due to discretization. Two existing algorithms implementing the necessary and sufficient condition, DLM and CSA, are able to solve general discrete, continuous and mixed-integer constrained NLPs in a unified fashion. CSA can further achieve asymptotic convergence to $CGM_{dn}$with probability one. Both DLM and CSA fit into our proposed general search framework for finding SP_{dn}.

In the rest of this thesis, based on the necessary and sufficient conditions, we extend the mechanisms of the general search framework for finding SP_{dn} by including genetic operators. We also study the theory of SSAs for determining the optimal schedules of SSAs in order to minimize the expected search time and its applications in existing stochastic constrained optimization algorithms.

Chapter 3

Theory of Stochastic Search Algorithms

In this chapter, we present the theory of SSAs for minimizing the expected search time of sequential as well as parallel SSAs.

Based on the observation that a large class of SSAs have a convex relationship between the expected overhead to find a solution and the duration of each run, we propose to use iterative deepening to find the optimal duration and prove the optimality of the proposed schedule. Next, we extend our theory to parallel processing of SSAs and show that the problem of scheduling of SSAs on multiple processors in order to minimize the expected completion time is NP-hard. We then analyze the performance limit of parallel SSAs and propose parallel SSAs with iterative deepening. Finally we derive performance bounds between the optimal and our proposed parallel schedules.

For clarity, we list in Table 3.1 major notations used in our theory of SSAs. The notations will be used in the rest of this thesis.

Table 3.1: List of major notations and their meaning in the theory of SSAs. We aligned corresponding notations for sequential and parallel SSAs in the same row of the table. $n \geq 2$ is the number of processors for running SSAs in parallel. Since our study is focused on a fixed quality level Q in Chapter 3 and 4, we drop the argument Q in in all the notations in these two chapters for simplicity. We investigate the effect of Q in Chapter 5.

Sequential SSAs		Parallel SSAs	
Symbol	Meaning	Symbol	Meaning
S	a schedule of running sequentially an SSA multiple times	$PS(n)$	a schedule of running an SSA multiple times in parallel on n processors
no counterpart		$PS^{async}(n)$	an asynchronous parallel schedule
no counterpart		$PS^{sync}(n)$	a synchronous parallel schedule
S^{ACS}	an ACS sequential schedule	$PS^{ACS}(n)$	a synchronous-ACS parallel schedule
$S^{ICS(N)}$	an ICS sequential schedule with all components being N	$PS^{ICS(N)}(n)$	a synchronous-ICS parallel schedule with all components being N
$B_s(S, Q)$	a random variable denoting the length of one run of S	$B_p(PS(n), Q)$	a random variable denoting the length of one run of $PS(n)$
$\mathcal{T}_{opt}(1, Q)$	the optimal $E(B_s(S, Q))$ over all sequential schedules	$\mathcal{T}_{opt}(n, Q)$	the optimal $E(B_p(PS(n), Q))$ over all parallel schedules on n processors
$N_{opt}^{ICS}(1, Q)$	the optimal N that minimizes $E(B_s(S^{ICS(N)}, Q))$	$N_{opt}^{ICS}(n, Q)$	the optimal N that minimizes $E(B_p(PS^{ICS(N)}(n), Q))$
$\mathcal{T}_{opt}^{ICS}(1, Q)$	the minimum $E(B_s(S^{ICS(N)}, Q))$ over all ICS sequential schedules	$\mathcal{T}_{opt}^{ICS}(n, Q)$	the minimum $E(B_p(PS^{ICS(N)}(n), Q))$ over all sychronous-ICS parallel schedules
no counterpart		$\mathcal{T}_{opt}^{sync}(n, Q)$	the minimum $E(B_p(PS^{sync}(n), Q))$ over all sychronous parallel schedules
$E_{ID}(1, Q)$	the expected length of SSA_{ID}	$E_{ID}(n, Q)$	the expected length of parallel SSA_{ID}

a) ICS: $N, N, N, \cdots$ b) ACS : $N_1, N_2, N_3, N_4, \cdots$

Figure 3.1: ICS and ACS sequential schedules of SSAs.

3.1 Existence of Optimal Sequential Schedules in SSAs

Let $\mathcal{N}$ be the set of possible numbers of probes in one run of an SSA, and a *sequential schedule* S of running an SSA be an infinite sequence $S = \{N_1, N_2, N_3, \cdots\}, N_i \in \mathcal{N}, i = 1, 2, \cdots$. We call N_i the i^{th} *component* of S that represents the i^{th} run of the SSA, when none of the previous runs were successful. All runs are assumed to be independent and non-preemptive; that is, after an SSA has been assigned to a processor, it cannot be interrupted until it has completed.

An execution of a sequential schedule entails running an SSA on a single processor multiple times, in which the i^{th} run will make N_i probes to the search space, until a solution of desired quality is found.

A major difference between scheduling SSAs and traditional job scheduling [99] is that the completion time of an SSA schedule is *not* deterministic, and the number of times an SSA has to be run before success may be unbounded.

Let $B_s(S)$ be a random variable denoting the total number of probes in one execution of schedule S. We have that $B_s(S) = \sum_{i=1}^{k} N_i$, assuming the SSA first succeeds in the k^{th} run. Our goal is to determine schedule S in order to minimize $E(B_s(S))$, the expected number of probes of schedule S.

We classify sequential schedules of SSAs into two types, as shown in Figure 3.1. The schedule in Figure 3.1a let an SSA run multiple times, each with a constant number of probes (i.e. $N_i = N$). We call such a schedule ICS (Identical-Component Schedule) and denote it

by $S^{ICS(N)}$. Another type of schedules, as shown in Figure 3.1b, run an SSA multiple times, each with a different number of probes. We call such a schedule ACS (Arbitrary-Component Schedule) and denote it by S^{ACS}. In the following, we first find the optimal schedule in ICS. Then we show that the optimal schedule in ICS remains optimal when compared to schedules in ACS.

From (1.6), we know that for $S^{ICS(N)}$, its expected total number of probes is:

$$E(B_s(S^{ICS(N)})) = \frac{N}{P_R(N)}. \tag{3.1}$$

From Figure 1.2c, we see that the $E(B_s(S^{ICS(N)}))$ curves of SA and GA are convex and have an absolute minimum of N in $(0, \infty)$. We also see that $E(B_s(S^{ICS(N)}))$ of random probing is monotonically increasing and has an absolute minimum at $N = 1$. In general, there exists an optimal N that minimizes (3.1) because $P_R(0) = 0$, $\lim_{N\to\infty} P_R(N) = 1$, $\frac{N}{P_R(N)}$ is bounded below by zero, and $\frac{N}{P_R(N)} \to \infty$ as $N \to \infty$. Therefore, for all SSAs, each has an optimal N that minimizes the expected search overhead of its ICS schedule in (3.1). We denote this optimal N by $N_{opt}^{ICS}(1)$.

Intuitively, for any SSA whose $E(B_s(S^{ICS(N)}))$ is convex, such as SA and GA, if each component N of S is set to be too small, then the corresponding $P_R(N)$ will be too low, and many runs will have to be made before success; if N is set to be too large, then each run will have a high $P_R(N)$ but an unnecessarily high cost, leading to a high cost before success. Such trade-off, thus, leads to the existence of $N_{opt}^{ICS}(1)$.

Next, we show that the optimal ICS schedule, if exists, is still optimal across all ACS schedules. Let $\mathcal{T}_{opt}^{ICS}(1)$ be the minimum $E(B_s(S^{ICS(N)}))$ of all ICS schedules, and $\mathcal{T}_{opt}(1)$ be the minimum $E(B_s(S))$ over all possible sequential schedules, including ICS and ACS. We have:

Theorem 3.1 *Minimum expected number of probes of any sequential schedule of SSAs.*

$$\mathcal{T}_{opt}(1) = \mathcal{T}_{opt}^{ICS}(1) = \frac{N_{opt}^{ICS}(1)}{P_R(N_{opt}^{ICS}(1))}.$$

Proof. It suffices to prove that the expected number of probes $E(B_s(S^{ACS}))$ of any ACS schedule S^{ACS} is always no less than $\mathcal{T}_{opt}^{ICS}(1)$. In fact, for any ACS whose components are $N_i, i = 1, 2, \cdots$, we have

$$E(B_s(S^{ACS})) = \sum_{i=1}^{\infty}\left[\left(\prod_{j=1}^{i-1}(1 - P_R(N_j))\right)P_R(N_i)\sum_{j=1}^{i} N_j\right], \tag{3.2}$$

which may be rearranged by exchanging the order of the two summations as

$$E(B_s(S^{ACS})) = \sum_{i=1}^{\infty}\left\{\left(\prod_{j=1}^{i-1}(1 - P_R(N_j))\right)\left[\sum_{k=i}^{\infty}\left(\prod_{l=i}^{k-1}(1 - P_R(N_l))\right)P_R(N_k)\right]N_i\right\}. \tag{3.3}$$

For any infinite sequence $P_1, P_2, P_3, \cdots$, where $P_i \in [0, 1]$ for $i = 1, 2, 3, \cdots$, we can trivially prove by induction that:

$$\sum_{i=1}^{\infty}\left[\left(\prod_{j=1}^{i-1}(1 - P_j)\right)P_i\right] = 1, \tag{3.4}$$

Hence, (3.3) can be reduced to:

$$E(B_s(S^{ACS})) = \sum_{i=1}^{\infty}\left[\left(\prod_{j=1}^{i-1}(1 - P_R(N_j))\right)N_i\right] \tag{3.5}$$

From definition of $N_{opt}^{ICS}(1)$, we know:

$$\frac{N_i}{P_R(N_i)} \geq \frac{N_{opt}^{ICS}(1)}{P_R(N_{opt}^{ICS}(1))} \tag{3.6}$$

$$\text{or,} \qquad N_i \geq P_R(N_i)\frac{N_{opt}^{ICS}(1)}{P_R(N_{opt}^{ICS}(1))} \qquad i = 1, 2, \cdots \tag{3.7}$$

Therefore, from (3.5) and (3.7),

$$E(B_s(S^{ACS})) \geq \left[\sum_{i=1}^{\infty}\left(\prod_{j=1}^{i-1}(1 - P_R(N_j))\right)P_R(N_i)\right]\frac{N_{opt}^{ICS}(1)}{P_R(N_{opt}^{ICS}(1))} \tag{3.8}$$

Use (3.4) again, we have:

$$E(B_s(S^{ACS})) \geq \frac{N_{opt}^{ICS}(1)}{P_R(N_{opt}^{ICS}(1))} = \mathcal{T}_{opt}^{ICS}(1) \tag{3.9}$$

■

The significance of Theorem 3.1 is that, it shows that the optimal expected number of probes $\mathcal{T}_{opt}^{ICS}(1)$ derived for ICS, is actually the lower bound of that for ACS. Therefore, Theorem 3.1 implies that $S^{ICS(N_{opt}^{ICS}(1))}$ is the optimal sequential schedule of SSAs.

3.2 Optimal Sequential SSA Schedules with Iterative Deepening

Our goal in this section is to design a practical strategy of scheduling multiple runs of SSAs in order to minimize the expected overhead of finding a solution. Further, we prove that the average overhead of the new strategy is optimal, i.e., of the same order of magnitude as $\mathcal{T}_{opt}(1)$.

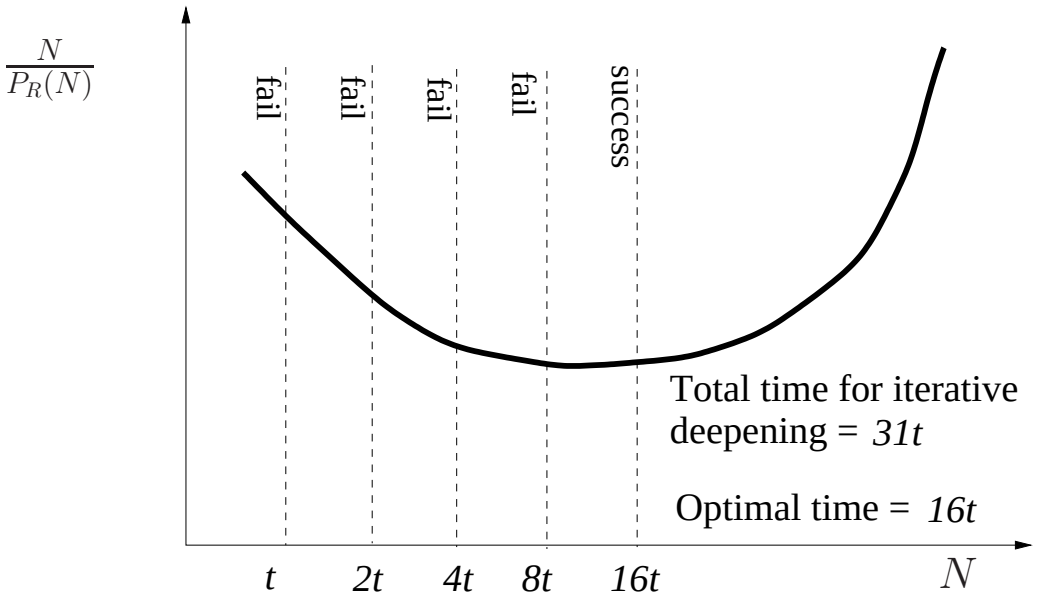


Figure 3.2: The principle of iterative deepening: the last run dominates the overhead.

The optimal schedule is generally problem dependent and hard to be determined. We propose to use *iterative deepening* to estimate it. The basic idea of iterative deepening is shown in Figure 3.2. If we make a series of runs with geometrically increasing durations, namely t, $2t$, $4t$, $\cdots$, the total running time will be dominated by the last run that succeeds in finding a solution.

Our approach in Figure 3.3 starts from a small number of probes $N = N_0$ and then uses a set of geometrically increasing numbers of probes in the SSA schedule:

$$N_i = \rho^i N_0, \qquad i = 0, 1, \ldots \tag{3.10}$$

Under N_i, an SSA is run multiple times for a maximum of K times but stops immediately when a solution is found. For iterative deepening to work, $\rho > 1$.

Let $E_{ID}(1)$ be the expected total number of probes taken by SSA_{ID} in Figure 3.3 to find a solution. The following theorem shows the relative complexities of $E_{ID}(1)$ and $\mathcal{T}_{opt}(1)$.

```
1.  procedure SSA_ID(Q)
2.      set initial maximum number of probes N = N_0;
3.      set K = number of runs at fixed N;
4.      repeat
5.          for i ← 1 to K do
6.              evaluate SSA with N probes;
7.              if success then goto 11; end_if
8.          end_for
9.          increase maximum number of probes N ← ρ × N (typically ρ = 2);
10.     until stopping condition satisfied or solution of quality Q is found
11. end_procedure
```

Figure 3.3: SSA_{ID}: A sequential SSA schedule with iterative deepening.

Theorem 3.2 *Optimality of sequential SSA schedules with iterative deepening.* [160]

$E_{ID}(1) = O(\mathcal{T}_{opt}(1))$ if

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

b) $(1 - P_R(N_{opt}^{ICS}(1)))^K \rho < 1$.

Proof. Based on (3.10) (the schedule used by SSA_{ID}), define q such that

$$N_{q-1} < N_{opt}^{ICS}(1) \leq N_q \quad \text{where } N_q = \rho N_{q-1} \tag{3.11}$$

If $P_R(x)$ is monotonically non-decreasing in $(0, +\infty)$, we have,

$$P_R(N_{q-1}) \leq P_R(N_{opt}^{ICS}(1)) \leq P_R(N_q). \tag{3.12}$$

Let $P_{NR}(N)$ be the probability that SSA does not succeed in K runs with N probes in each run. Since all runs are independent, we have:

$$P_{NR}(N) = (1 - P_R(N))^K \tag{3.13}$$

From (3.12) and (3.13),

$$P_{NR}(N_{opt}^{ICS}(1)) \geq P_{NR}(N_q) \geq P_{NR}(N_{q+1}) > \cdots \tag{3.14}$$

Let E_{B_i} be the expected total number of probes spent by Lines 4-10 in Figure 3.3, if SSA is run for a maximum of K times at N_i probes each, but stop immediately if it succeeds in any one of these runs, then:

$$E_{B_i} = \sum_{j=0}^{K-1} [1 - P_R(N_i)]^j \cdot N_i \leq K \cdot N_i = O(N_i) \quad i = 0, 1, 2, ... \tag{3.15}$$

There are two possibilities for Lines 6-12 in Figure 3.3 to find a target solution f': succeed in iterations running SSA with $N_0, \ldots, N_q$, or succeed afterwards. Since the success probability of all iterations running SSA with $N_0, \ldots, N_q$ probes is larger than the success probability of the iteration running the SSA with N_q probes alone, we have:

$$E_{ID}(1) \leq [1 - P_{NR}(N_q)]E_{B_q} + P_{NR}(N_q) \sum_{i=q+1}^{\infty} E_{B_i} \prod_{j=q+1}^{i-1} P_{NR}(N_j) \tag{3.16}$$

From (3.14), (3.16) can be reduced to:

$$E_{ID}(1) \leq [1 - P_{NR}(N_{opt}^{ICS}(1))]E_{B_q} + P_{NR}(N_{opt}^{ICS}(1)) \sum_{i=q+1}^{\infty} E_{B_i} \prod_{j=q+1}^{i-1} P_{NR}(N_j) \tag{3.17}$$

which can be further reduced to:

$$E_{ID}(1) \leq E_{B_q} + P_{NR}(N_{opt}^{ICS}(1)) \sum_{i=q+1}^{\infty} E_{B_i} \prod_{j=q+1}^{i-1} P_{NR}(N_j) \tag{3.18}$$

From (3.14), (3.15) and (3.18), we have:

$$\begin{aligned} E_{ID}(1) &\leq \sum_{i=0}^{\infty}(P_{NR}(N_{opt}^{ICS}(1)))^i E_{B_{q+i}} = \sum_{i=0}^{\infty}(P_{NR}(N_{opt}^{ICS}(1)))^i O(N_{q+i}) \\ &= O(\rho^q N_0)\sum_{i=0}^{\infty}(P_{NR}(N_{opt}^{ICS}(1))\rho)^i. \end{aligned} \tag{3.19}$$

For the search to converge in finite average time,

$$P_{NR}(N_{opt}^{ICS}(1))\rho = (1 - P_R(N_{opt}^{ICS}(1)))^K \rho < 1. \tag{3.20}$$

Hence, we have proved the result in the theorem:

$$E_{ID}(1) = O(\rho^q N_0) = O(\rho N_{opt}^{ICS}(1)) = O(\mathcal{T}_{opt}(1)). \tag{3.21}$$

■

We thus achieved our goal by showing that $E_{ID}(1)$ is up to a constant factor over $\mathcal{T}_{opt}(1)$ under certain conditions. The second condition in Theorem 3.2 requires K to be large enough because iterative deepening may not succeed before $N_{opt}^{ICS}(1)$ and may overshoot then. A large enough K ensures that the expected total overhead will not go unbounded in case overshoot happens. We discuss in Chapter 4 that it is generally sufficient to set $K = 3$.

3.3 Parallel SSAs

We discuss in this section parallel processing of SSAs. The goal of parallel processing is to minimize the expected search time of parallelizing multiple runs of an SSA on n processors. Here we have a basic assumption that each SSA run is non-preemptive. In other words, once an SSA run has been assigned to a processor, it will run in that single processor until

completion without further interruption or partitioning. We have made this assumption because most SSAs, such as SA and GA, are iterative algorithms that have little parallelism to be exploited in one run. We are interested in scheduling multiple runs of an SSA on multiple processors in order to minimize the expected execution time of the search.

In the rest of this section, we first prove that the scheduling of multiple runs of an SSA in parallel in order to minimize the expected completion time is NP-hard. Then we find the optimal schedules under certain restricted conditions, and study performance limitations of scheduling multiple runs of SSAs in parallel. Finally, we extend iterative deepening to the scheduling of SSAs in parallel and prove its optimality.

3.3.1 Complexity of scheduling multiple runs of SSAs in parallel

Given n identical processors, a *parallel schedule* $PS(n)$ of running an SSA on n processors consists of n sequential schedules:

$$\begin{aligned} PS(n) &= \Big\{ S_1, S_2, \cdots, S_n \Big\} \\ &= \Big\{ \{N_{1,1}, N_{1,2}, N_{1,3}, \cdots\}, \{N_{2,1}, N_{2,2}, N_{2,3}, \cdots\}, \cdots, \{N_{n,1}, N_{n,2}, N_{n,3}, \cdots\} \Big\}. \end{aligned}$$

$N_{i,j}$ is the j^{th} component of sequential schedule S_i, for $1 \leq i \leq n$ and $j = 1, 2, \cdots$. All components are assumed to be integers.

In one run of a parallel schedule, processor i will run an SSA with sequential schedule S_i, The parallel search will terminate once an SSA (in any processor) succeeds in finding a solution. Let random variable $B_p(PS(n))$ be the length of one run of a parallel schedule $PS(n) = \{S_1, S_2, \cdots, S_n\}$. We define:

$$B_p(PS(n)) = \min\Big\{ B_s(S_1), B_s(S_2), \cdots, B_s(S_n) \Big\} \tag{3.22}$$

Our goal of parallel processing of multiple runs of SSAs is minimize $E(B_p(PS(n)))$, the expected length of a parallel schedule.

This optimization problem does not belong to class NP [55, 67] if we consider its corresponding decision problem: given a positive number C, is there a parallel schedule $PS(n)$ such that its $E(B_p(PS(n))) \leq C$? Any instance of this problem involves n infinite sequences and will take infinite time to compute $E(B_p(PS(n)))$ in general. Therefore, any solution instance cannot be verified in polynomial time. It follows that this problem does not belong to class NP. More formally, this problem cannot be solved by a nondeterministic one-tape Turning machine (NDTM) [24, 67] in polynomial time, since any guessed instance of NDTM should have a finite size and can be examined in polynomially bounded time.

The problem of parallel scheduling of multiple runs of SSAs is not in NP if its instances have infinite sizes. In practice, however, we can only consider scheduling of finite components, because any algorithm has to terminate in finite time and an infinite schedule is never used. Further, we can choose a success probability as close to that needed using only finite number of runs. We show in the following that, even if we restrict a parallel schedule to have only finite number of runs, the parallel scheduling problem is NP-hard, which implies its intractability.

If we only have a finite number of SSAs to be scheduled, then a parallel schedule will consist of n sequential schedules with finite durations. In one run of the parallel schedule, each processor will execute its assigned schedule and terminate when either an SSA running on any processor succeeds in finding a solution or all the SSAs in the parallel schedule are run. Hence, for a parallel schedule $PS(n)$ with a finite number of SSA runs, we can define $B_p(PS(n))$ (the length of one run) and $E(B_p(PS(n)))$ (the expected length) in a similar way.

Formally, we define the problem of PARALLEL SCHEDULING OF SSAs as follows:

INSTANCE:

- A finite multiset of k components $M = \{N_1, N_2, \cdots, N_k\}$, where N_i (positive integer) is the number of probes in one run of an SSA, for $1 \leq i \leq k$.
- n processors, where $n \geq 2$.
- A total function P_R: $M \longrightarrow [0, 1]$.
- A positive real number C.

QUESTION: Is there a parallel schedule $PS(n)$ assigning all the k components in M to n processors such that

$$E(B_p(PS(n))) \leq C \quad ?$$

Theorem 3.3 PARALLEL SCHEDULING OF SSAs is NP-complete.

Proof. PARALLEL SCHEDULING OF SSAs can be solved by a nondeterministic polynomial algorithm: one that guesses parallel schedule $PS(n)$, computes $E(B_p(PS(n)))$, and compares that with C; hence PARALLEL SCHEDULING OF SSAs is in NP.

Next we show that a known NP-complete problem, PARTITION [67], can be reduced to PARALLEL SCHEDULING OF SSAs. The PARTITION problem asks whether a given multiset $A = \{a_1, a_2, \cdots, a_m\}$ of m positive integers has a subset A' such that $\sum_{a \in A'} a = \sum_{a \in A - A'} a$.

We first prove that PARTITION can be reduced to PARALLEL SCHEDULING OF SSAs for $n = 2$, and the extension to the case of $n > 2$ is trivial. Given any instance $A = \{a_1, a_2, \cdots, a_m\}$ of the PARTITION problem, define an instance of PARALLEL SCHEDULING OF SSAs as follows: $k = m$, $M = \{a_1, a_2, \cdots, a_m\}$, $n = 2$, $P_R(a_i) = 0$, for $1 \leq i \leq m$, and $C = \frac{1}{2} \sum_{a \in A} a$. Such a transformation can obviously be done in

polynomial time. For any parallel schedule $PS(n)$ on this instance, since $P_R(a_i) = 0$ for $1 \leq i \leq m$, any run of $PS(n)$ can only terminate after all SSA runs have been finished. Therefore, $E(B_p(PS(n))) = \max\{L_1, L_2\}$, where L_1 and L_2 are the total numbers of probes of the SSAs assigned to processors 1 and 2, respectively. There is a schedule $PS(n)$ whose $E(B_p(PS(n))) \leq C$ if and only if there is a partition for A, in which case $L_1 = L_2 = \frac{1}{2}\sum_{a \in A} a = C$, leading to $E(B_p(PS(n))) = C$. ■

Given the $P_R(N)$ curve of an SSA on a problem and n processors, define the optimal expected completion time $\mathcal{T}_{opt}(n)$ to be the minimum $E(B_p(PS(n)))$ over all possible parallel schedules. In general, $\mathcal{T}_{opt}(n)$ is difficult to be determined since the parallel scheduling problem is NP-hard (when the number of SSA runs to be scheduled is restricted to be finite), and $\mathcal{T}_{opt}(n)$ is not likely to be found in polynomial time. In the rest of this section, we look for efficient heuristic schedules. Section 3.3.2 finds $\mathcal{T}_{opt}^{sync}(n)$, the minimum expected completion time of an important class of parallel schedules, and derives two lower bounds of $\mathcal{T}_{opt}(n)$. Section 3.3.3 proposes parallel SSAs with iterative deepening, whose expected completion time is of the same order of magnitude as $\mathcal{T}_{opt}^{sync}(n)$.

3.3.2 Limitations of parallel processing of SSAs

In this subsection, we first establish a classification of parallel schedules. Then, we find $\mathcal{T}_{opt}^{sync}(n)$, the minimum expected completion time of a class of parallel schedules in the classification. Finally, we give two lower bounds of $\mathcal{T}_{opt}(n)$.

3.3.2.1 A classification of parallel schedules of SSAs

Figure 3.4 shows a classification of parallel schedules of SSAs. Figure 3.5 depicts each class of schedules schematically. We divide parallel schedules into synchronous and asynchronous

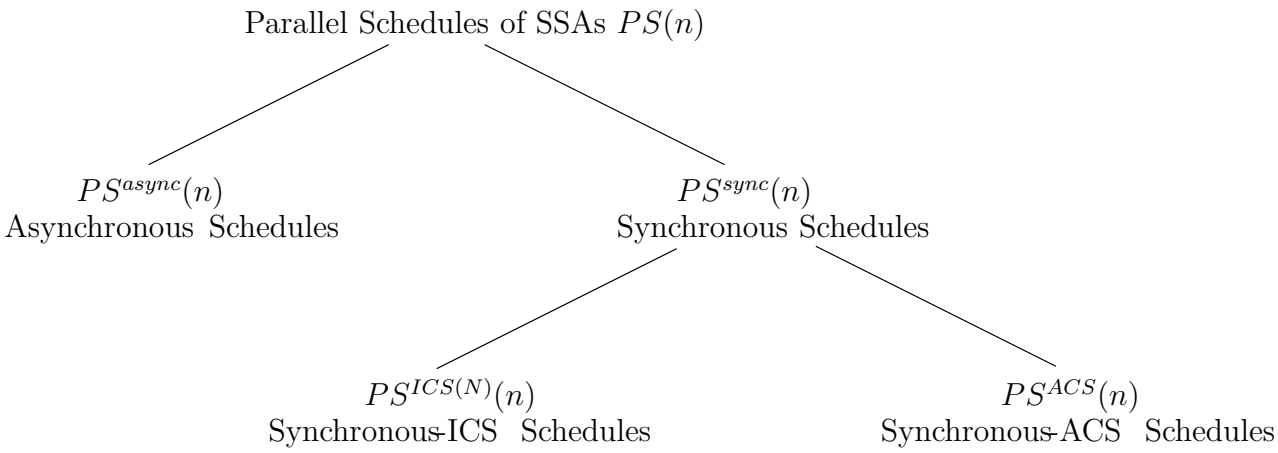
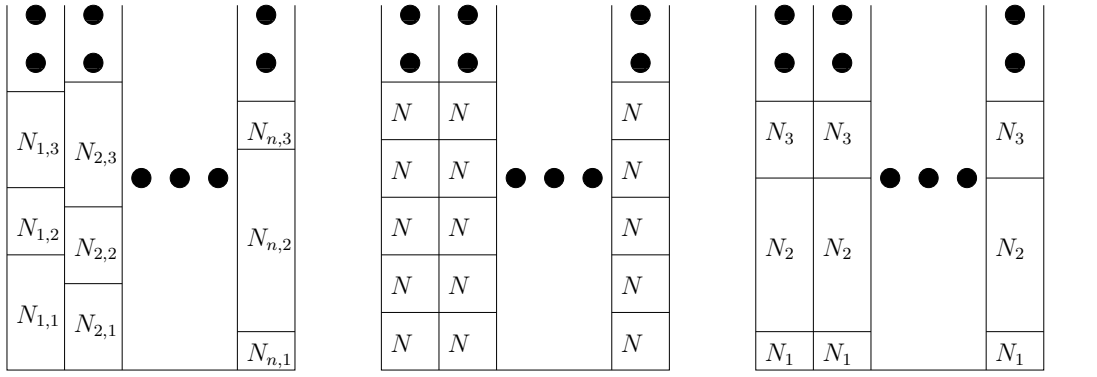


Figure 3.4: A classification of parallel schedules of SSAs.



a) Asynchronous Schedules b) Synchronous-ICS Schedules c) Synchronous-ACS Schedules

Figure 3.5: Parallel SSAs: Asynchronous, Synchronous-ICS, and Synchronous-ACS parallel schedules.

ones, denoted by $PS^{sync}(n)$ and $PS^{async}(n)$, respectively. For synchronous schedules, all processors run SSAs of the same duration at any time. As in the sequential case, synchronous schedules can be further divided to ICS and ACS schedules. For a synchronous-ICS schedule $PS^{ICS(N)}(n)$ (Figure 3.5b), all processors run an SSA with N probes repeatedly until success. For a synchronous-ACS schedule $PS^{ACS}(n)$ (Figure 3.5c), all processors run an SSA with the same number of probes at any time, although the number of probes run at a different time may change.

3.3.2.2 Minimum expected completion time of synchronous parallel schedules

In this subsection, we derive the minimum expected completion time of all possible synchronous parallel schedules, including both ICS and ACS. Similar to the sequential case, we first consider synchronous-ICS, and then extend the result to synchronous-ACS. For a synchronous-ICS schedule $PS^{ICS(N)}(n)$, define the *parallel reachability probability* $\Gamma_R(n, N)$ to be the probability that a solution of desired quality is found by at least one of the n processors, each making runs of SSA with N probes. We have:

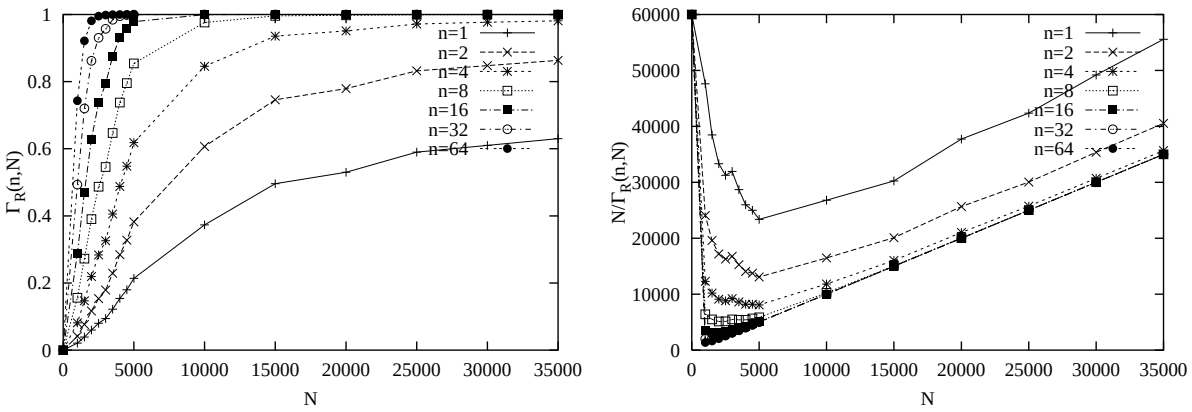
$$\Gamma_R(n, N) = 1 - (1 - P_R(N))^n \tag{3.23}$$

The expected completion time of $PS^{ICS(N)}(n)$ is:

$$\begin{aligned} E(B_p(PS^{ICS(N)}(n))) &= \sum_{i=1}^{\infty} N \cdot i \cdot [1 - \Gamma_R(n, N)]^{i-1} \cdot \Gamma_R(n, N) \\ &= \frac{N}{\Gamma_R(n, N)} \end{aligned} \tag{3.24}$$

Figure 3.6 shows the parallel reachability probability $\Gamma_R(n, N)$ and the expected completion time $\frac{N}{\Gamma_R(n,N)}$ of CSA run under synchronous-ICS schedules on a 10-D Rashtrigin problem (defined in (1.7)). We see that for synchronous-ICS schedules, there are optimal N's that can minimize $E(B_p(PS^{ICS(N)}(n)))$. Note that $\Gamma_R(n, N)$ plays the same role in (3.24) as $P_R(N)$ for sequential schedules in (3.1)

Let $N_{opt}^{ICS}(n)$ be the optimal N that minimizes $E(B_p(PS^{ICS(N)}(n)))$ in (3.24), and $\mathcal{T}_{opt}^{ICS}(n)$ be the minimum $E(B_p(PS^{ICS(N)}(n)))$ of synchronous-ICS schedules. We prove in the following that, the minimum $E(B_p(PS^{ICS(N)}(n)))$ for synchronous-ICS is actually the minimum $E(B_p(PS^{sync}(n)))$ for all synchronous schedules, including ICS and ACS. Let $\mathcal{T}_{opt}^{sync}(n)$ be the minimum $E(B_p(PS^{sync}(n)))$ of all synchronous parallel schedules. We have:



a) $\Gamma_R(n, N)$ b) $\frac{N}{\Gamma_R(n,N)}$

Figure 3.6: The parallel reachability probability $\Gamma_R(n, N)$ and the expected completion time $\frac{N}{\Gamma_R(n,N)}$ of CSA run under synchronous-ICS parallel schedules on a 10-D Rashtrigin problem. Note the existence of $N_{opt}^{ICS}(n)$.

Lemma 3.1 *Minimum expected completion time of synchronous parallel schedules of SSAs.*

$$\mathcal{T}_{opt}^{sync}(n) = \mathcal{T}_{opt}^{ICS}(n) = \frac{N_{opt}^{ICS}(n)}{\Gamma_R(n, N_{opt}^{ICS}(n))} \quad . \tag{3.25}$$

Proof. The proof follows that in Theorem 3.1 by substituting $P_R(N)$ by $\Gamma_R(n, N)$. ■

Lemma 3.1 is important because it gives the optimal completion time for a restricted class of the parallel scheduling problem of SSAs. Note that the general scheduling problem is NP-hard.

3.3.2.3 Lower bounds of $\mathcal{T}_{opt}(n)$

$\mathcal{T}_{opt}(n)$ is difficult to be determined and is not likely to be expressed in a simple form like $\mathcal{T}_{opt}^{sync}(n)$ in (3.25). Currently, we know two theoretical lower bounds of $\mathcal{T}_{opt}(n)$.

Proposition 3.1

$$\mathcal{T}_{opt}(n) \geq \frac{\mathcal{T}_{opt}(1)}{n} \tag{3.26}$$

Proof. For any parallel schedule $PS(n)$ of SSAs, suppose:

$$\begin{aligned} PS(n) &= \Big\{S_1, S_2, \cdots, S_n\Big\} \\ &= \Big\{\{N_{1,1}, N_{1,2}, N_{1,3}, \cdots\}, \{N_{2,1}, N_{2,2}, N_{2,3}, \cdots\}, \cdots, \{N_{n,1}, N_{n,2}, N_{n,3}, \cdots\}\Big\}. \end{aligned}$$

Let the j^{th} SSA run in the i^{th} processor be the first SSA run that succeeds in finding a solution of desired quality. Hence, the completion time of $PS(n)$ is:

$$F_{i,j} = \sum_{k=1}^{j} N_{i,k} \tag{3.27}$$

Let Z^+ be the set of positive integers. Define a one-to-one function $g : Z^+ \rightarrow Z^+ \times Z^+$, such that for $\forall i, j \in N, i < j$, if $g(i) = (i_1, i_2)$ and $g(j) = (j_1, j_2)$, then $F_{i_1,i_2} < F_{j_1,j_2}$ or, $F_{i_1,i_2} = F_{j_1,j_2}$ and $i_1 < j_1$.

Further, for $\forall i \in Z^+$, define :

$$\eta_i = \frac{\sum_{k=1}^{i} N_{g(k)}}{F_{g(i)}} \tag{3.28}$$

We have:

$$E(B_p(PS(n))) = \sum_{i=1}^{\infty} F_{g(i)} P_R(N_{g(i)}) \prod_{k=1}^{i-1} (1 - P_R(N_{g(k)})) \tag{3.29}$$

$$= \sum_{i=1}^{\infty} \left[\left(\frac{\sum_{k=1}^{i} N_{g(k)}}{\eta_i} \right) P_R(N_{g(i)}) \prod_{k=1}^{i-1} (1 - P_R(N_{g(k)})) \right] \tag{3.30}$$

It is obvious that for $\forall i, j \in N, i \neq n$, $\eta_{g^{-1}(i,j)} < n$. Therefore,

$$\begin{aligned} E(B_p(PS(n))) &> \frac{\sum_{i=1}^{\infty}[(\sum_{k=1}^{i} N_{g(k)}) \cdot P_R(N_{g(i)}) \prod_{k=1}^{i-1}(1 - P_R(N_{g(k)}))]}{n} \\ &\geq \frac{\mathcal{T}_{opt}(1)}{n} \end{aligned} \tag{3.31}$$

The last inequality was obtained from Theorem 3.1. ■

Proposition 3.2

$$\mathcal{T}_{opt}(n) \geq \frac{1 - \Gamma_R(n, N_{lnopt})}{\Gamma_R(n, N_{lnopt})} N_{lnopt}, \tag{3.32}$$

where $N_{lnopt} = \text{argmin}_N \left(\frac{N}{-\ln(1-P_R(N))} \right)$.

Proof. Following the definitions in the last proof, define

$$P_\sigma(i) = 1 - \prod_{j=1}^{i}(1 - P_R(N_{g(i)})) \tag{3.33}$$

$E(B_p(PS(n)))$ in (3.29) can be rewritten as:

$$E(B_p(PS(n))) = F_{g(1)} + \sum_{i=2}^{\infty}\Big[(F_{g(i)} - F_{g(i-1)})(1 - P_\sigma(i))\Big] \tag{3.34}$$

It is easy to verify that N_{lnopt} is so chosen that, for $\forall i, k \in Z^+$, $F_{g(i)} < kN_{lnopt}$, the following inequality holds:

$$P_\sigma(i) \leq 1 - (1 - \Gamma(n, N_{lnopt}))^k \tag{3.35}$$

1. **procedure** *Parallel* $SSA_{ID}(Q)$
2. set initial number of probes $N = N_0$;
3. set K = number of runs at fixed N;
4. **repeat**
5. **for** $i \leftarrow 1$ **to** K **do**
6. Execute SSA with N probes on each processor;
7. **if** success on any processor **then goto** 11; **end_if**
8. **end_for**
9. increase number of probes $N \leftarrow \rho \times N$ (typically $\rho = 2$);
10. **until** stopping condition is satisfied or solution of quality Q is found
11. **end_procedure**

Figure 3.7: Parallel SSA_{ID}: Parallel scheduling of SSAs with iterative deepening.

From (3.35) and (3.34), we get:

$$\begin{aligned} E(B_p(PS(n))) &\geq \sum_{k=1}^{\infty} \left[N_{lnopt} (1 - \Gamma_R(n, N_{lnopt}))^k \right] \\ &= \frac{1 - \Gamma_R(n, N_{lnopt})}{\Gamma_R(n, N_{lnopt})} N_{lnopt} \end{aligned} \tag{3.36}$$

■

3.3.3 Parallel scheduling of SSAs with iterative deepening

In this subsection, we design a parallel schedule of SSAs with iterative deepening in such a way that its expected completion time is up to a constant factor over $\mathcal{T}_{opt}^{sync}(n)$. Figure 3.7 shows our algorithm to schedule multiple runs of SSAs in parallel while using iterative deepening to increase the number of probes in successive runs of SSAs. All the processors execute a series of SSAs with geometrically increasing number of probes synchronously un-

til a solution of desired quality is found. Parallel SSA_{ID}, therefore, can be classified as a synchronous-ACS schedule. For each SSA scheduled to run on, it has a reachability probability $\Gamma_R(n, N)$ to succeed, and it will double N if it does not succeed. Hence, its expected completion time is equivalent to that of a sequential schedule of SSA with iterative deepening, with $\Gamma_R(n, N)$ as its reachability probability making N probes. Therefore, let $E_{ID}(n)$ be the expected completion time of parallel SSA_{ID} on n processors. From Theorem 3.2, we have,

Theorem 3.4 *Optimality of parallel SSA_{ID}.* $E_{ID}(n) = O(\mathcal{T}_{opt}^{sync}(n))$ if

a) $\Gamma_R(n, 0) = 0$; $\Gamma_R(n, N)$ is monotonically non-decreasing for $N \in (0, \infty)$; $\lim_{N\to\infty} \Gamma_R(n, N) \leq 1$;

b) $(1 - \Gamma_R(n, N_{opt}^{ICS}(n)))^K \rho < 1$.

The expected completion time of the parallel SSA_{ID} is of the same order of magnitude as that of the optimal synchronous parallel schedules. That is, although the parallel SSAs scheduling problem is NP-hard in general, we have developed a sub-optimal parallel schedule using iterative deepening that is optimal (up to a constant factor) for the special class of synchronous parallel schedules.

3.4 Summary

In this chapter, we first show that SSAs have an optimal sequential schedule that can minimize its expected overhead of finding a solution of desired quality by multiple runs of an SSA. We have then proposed to use iterative deepening to develop the optimal schedule. We have proved that multiple runs of SSAs with iterative deepening are optimal in the sense that their expected overhead is of the same order of magnitude as the optimal expected overhead.

We have extended our theory to the scheduling of multiple runs of SSAs in parallel, proved the NP-completeness of parallel scheduling of SSAs under the assumption that the number of SSAs to be scheduled is finite, and studied the performance limitations of scheduling multiple runs of SSAs in parallel. We have then proposed to use iterative deepening to determine the number of probes in the scheduling of multiple runs of SSAs in parallel and proved its optimality.

Although there are many past studies aimed at designing schedules that allow one run of an SSA to have a higher chance of success, there was no prior studies that examine trade-offs between multiple runs using different durations of one run and the improved probabilities of getting a solution. Our approach based on iterative deepening is unique because it exploits the convex relationship between the expected overhead and the duration of one run of an SSA and the fact that the total overhead is dominated by the last run. The theory is general enough and can be applied to many existing SSAs, as shown in the following chapters.

Chapter 4

Optimal Schedules of SSAs for Constrained Optimization

In this chapter we develop new stochastic constrained optimization algorithms and their optimal schedules. The constrained algorithms are derived from the general search framework for finding SP_{dn}, and their optimal versions are developed by using iterative deepening to estimate their optimal schedules. This chapter consists of three parts. First, in Section 4.1, we develop *Constrained Genetic Algorithm* (CGA) and its combination with CSA (CSAGA) by incorporating genetic operators to search the original-variable subspace in the general framework. Then, in Section 4.2, we apply the theory of SSAs to CSA, CGA, and CSAGA and develop their optimal versions. They are optimal in the sense that they generate feasible solutions of certain prescribed quality using an average time of the same order of magnitude as that spent by the original SSA with an optimal schedule in generating a solution of similar quality. Finally, Section 4.3 evaluates 24 possible combinations of the search framework and identifies the optimal combinations.

4.1 Constrained SSAs with Genetic Search

In Figure 1.1 of chapter 1, we have proposed a general problem-independent framework that unifies various search mechanisms for solving constrained nonlinear programming (NLP) problems whose functions are not necessarily differentiable and continuous. The framework is based on the first-order necessary and sufficient conditions for constrained local minimization in discrete space that show the equivalence between discrete-space saddle points and constrained local minima. It implements the search for discrete-space saddle points by performing ascents in the original-variable subspace and descents in the Lagrange-multiplier subspace in order to reach equilibrium at saddle points.

From our survey, existing algorithms looking for SP_{dn}, such as DLM and CSA , fit into the search framework, each maintaining a list of one candidate. DLM entails greedy generations in the x and λ subspaces, deterministic insertions into the list of candidates and deterministic acceptance of candidates. CSA generates new probes randomly along one of the x or the λ variables, accepts them based on the Metropolis probability if L_d increases along the x dimension and decreases along the λ dimension. Both methods stop updating λ when all the constraints are satisfied. In this section, we extend the mechanisms in the framework to include genetic operators and present in Section 4.1.1 CGA and in Section 4.1.2 the combined CSAGA.

4.1.1 Constrained genetic algorithm (CGA)

CGA was developed based on the general framework in Figure 1.1 that looks for discrete-space saddle points. Similar to traditional GA, it organizes a search into a number of generations, each involving a population of candidate points in the search space. However, it searches in the Lagrangian space, using genetic operators to generate new probes in the

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

Figure 4.1: CGA: Constrained genetic algorithm. P is the population size and N_g is the number of generations.

original-variable subspace, either greedy or probabilistic generations in the λ subspace, and deterministic organization of candidates according to their Lagrangian values. Figure 4.1 outlines the pseudo code of the algorithm that is called with P, the population size, and N_g, the number of generations. Table 4.1 shows how CGA fits into the general framework to look for discrete-space saddle points.

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

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

Table 4.1: Both CGA and CSAGA fit into the general iterative search framework for finding discrete-space saddle points.

Framework Components	CGA	CSAGA
Generation of x probes	GA	SA & GA
Generation of λ probes	probabilistic	probabilistic
Insertion of x probes	annealing	annealing
Insertion of λ probes	annealing	annealing

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

Lines 6-9 explore the original-variable subspace using genetic search by selecting from $\mathcal{P}(t)$ individuals to reproduce using genetic operators and by inserting the new individuals generated in $\mathcal{P}(t)$ according to their Lagrangian (fitness) values.

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

$$\mathcal{H}_i(h, \mathcal{P}(t)) = max_{x \in \mathcal{P}(t)} H(h_i(x)) \qquad i = 1, 2, \cdots, m, \tag{4.1}$$

where $h_i(x)$ is the i^{th} constraint function, H is the non-negative transformation in (1.3), and c is a positive step-wise parameter controlling how fast the Lagrange multipliers change.

As shown in the framework, there are two options in implementing operator $\oplus$. The first is to generate a new λ in a probabilistic or greedy fashion. In a probabilistic way, we generate a new λ from a uniform distribution on $\{-c \times \mathcal{H}/2, c \times \mathcal{H}/2\}$. In contrast, in a greedy way, we generate new λ from a uniform distribution on $\{0, c \times \mathcal{H}\}$, in order for only ascents in the Lagrangian function in the λ subspace.

The second option in searching the λ subspace is to accept the probes deterministically or by annealing rules. In a deterministic way, we only accept candidates that improve the Lagrangian function. In contrast, using an annealing rule, we accept candidates λ' that reduce the Lagrangian function with the Metropolis probability: $exp\left\{-\frac{L(\lambda)-L(\lambda')}{T}\right\}$ where T is a control parameter that decreases over time. In any case, a Lagrange multiplier will not be changed if its corresponding constraint is satisfied.

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

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

One difficulty encountered in applying CGA is in choosing a suitable population size. For the benchmark problems tested, the optimal population size ranges between 20 and 50. Although we have developed a dynamic procedure to select a suitable population size at run time, we do not present the procedure here because CGA does not perform as well as CSAGA presented in the next subsection, which uses a small and fixed population size.

Another difficulty is in determining the proper number of generations to use in each run. It is not necessary to run CGA once for an exceedingly long duration in order for it to find a CGM_{dn}. Similar to CSA, CGA is another example of the SSAs studied in Chapter 3 in which multiple runs of CGA can be scheduled in order to minimize the expected overhead of

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

Figure 4.2: *CSAGA*: Combined *CSA* and *CGA* called with population size P and number of generations N_g.

finding a solution of desired quality. In Section 4.2, we use iterative deepening to determine to optimal number of generations.

4.1.2 Combined constrained SA and GA (CSAGA)

Based on the general framework in Figure 1.1, we design CSAGA by integrating CSA in Figure 2.1 and CGA in Figure 4.1 into a combined procedure. Table 4.1 shows how CSAGA fits into the general framework for finding SP_{dn}.

Figure 4.2 shows the pseudo code of the combined algorithm. The new algorithm uses both SA and GA to generate new probes in the original-variable subspace. Again, the procedure is called with P, the population size, and N_g, the number of generations.

Line 2 initializes population $\mathcal{P}(0)$. Unlike CGA, any individual $\mathbf{x}$ in the population in CSAGA is defined in the joint original-variable and Lagrange-multiplier subspaces, *i.e.*, $\mathbf{x} = (x, \lambda)$. Initially, x can be either user-provided or randomly generated, and λ is initialized to zero.

Lines 4-10 perform CSA using q probes on every individual in the population, generating probabilistically a point in the x and λ subspaces, and accepting the point based on annealing and the Metropolis probability distribution. The control parameter q is set to be $\frac{N_g}{6}$ after experimental evaluations.

Lines 11-14 start a GA search after the SA part is completed. The algorithm searches in the original-variable subspace using GA and evaluates the augmented Lagrangian value of each individual in order to select the individuals with the best fitness. In ordering the individuals, since each individual has its own vector of Lagrange multipliers, we first get the average value of each Lagrange multiplier over the population and then calculate the Lagrangian value for each individual using the averaged Lagrange multipliers.

Note that we have difficulties with CSAGA similar to those with CGA in deciding on a proper number of candidates to use in the population and in the duration of each run. We address these issues in the CSAGA with iterative deepening in the next section.

4.1.3 Implementation details

In theory, algorithms derived from the framework, such as CSA, CGA, and $CSAGA$, will look for SP_{dn}. In practice, however, it is important to choose appropriate neighborhoods

and generate proper trial points in the x and λ subspaces in order to solve constrained NLPs efficiently.

An important component of these methods is the frequency at which λ is updated. Like in CSA [162], we have set experimentally in CGA and $CSAGA$ the ratio of generating trial points in the x and λ subspaces from the current point to be $10n$ to m, where n is the number of variables and m is the number of constraints. This ratio means that x is updated more often than λ.

In generating trial points in the x subspace, we have used a dynamically controlled neighborhood size in the SA part [162] based on the 1:1 ratio rule [56], whereas in the GA part, we have used the seven operators in Genocop III [116] and L_d as our fitness function. In implementing CGA and CSAGA, we have used the default parameters of CSA [162] in the SA part and those of Genocop III [116] in the GA part.

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

$$\lambda'_j = \lambda_j + r_1 \, \phi_j \qquad \text{where } j = 1, \cdots, m. \tag{4.2}$$

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

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

$$\phi = w \otimes \mathcal{H}(x) = [w_1 \mathcal{H}_1(x), w_2 \mathcal{H}_2(x), \cdots, w_m \mathcal{H}_m(x)], \tag{4.3}$$

where $\otimes$ represents vector product, and $\mathcal{H}$ is the vector of maximum violations defined in (4.1). When $\mathcal{H}_i(x)$ is satisfied, λ_i does not need to be updated; hence, $\phi_i = 0$. In contrast, when a constraint is not satisfied, we adjust ϕ_i by modifying w_i according to how fast $\mathcal{H}_i(x)$ is changing:

$$w_i = \begin{cases} \eta_0 \, w_i & \text{if } \mathcal{H}_i(x) > \tau_0 T \\ \eta_1 \, w_i & \text{if } \mathcal{H}_i(x) < \tau_1 T \end{cases} \tag{4.4}$$

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

4.2 SSAs with Iterative Deepening

One of the difficulties in using SSAs, such as CSA, CGA and CSAGA, is to determine N, the number of probes in one run of the algorithm. As we have shown earlier, all SSAs have $P_R(N) < 100\%$ with a finite N. In previous applications of SSAs, users generally have to experiment by trial and error with different N's until a solution with desired quality can be found. Such tuning is obviously not practical, especially in solving large complex problems. In this section, we are interested in running a single version of an SSA that can adjust its schedule adaptively in order to find a schedule close to the optimal one.

Our theory of SSAs in Chapter 3 has shown that for SSAs, there is generally an optimal schedule of multiple runs of SSAs that can minimize the expected overhead of finding a solution. In practice, we have also observed that when using an ICS schedule with N probes per run of the SSA, setting N to be either too large or too small is inefficient, and there is an optimal N that will result in the least average search time.

Unfortunately, it is very difficult to determine the optimal N that depends on the problem-dependent $P_R(N)$ curve of the SSA. We have proposed in Chapter 3 to use *iterative deepening* in order to estimate the optimal duration of a run. In the rest of this section, we

apply iterative deepening to CSA, CGA, and CSAGA, resulting in CSA_{ID}, CGA_{ID}, and $CSAGA_{ID}$. Based on the theory of SSAs, we show the optimality of the iterative-deepening versions that do not require specific problem-dependent tuning of search schedules.

4.2.1 Optimal schedules of SSAs

In order to apply iterative deepening, it is important to identify the parameters of an SSA that control its number of probes in one run of the search framework in Figure 1.1.

In general, for algorithms fitting the search framework in Figure 1.1, let $\mathcal{L}$ be the length of the candidate list, and $\mathcal{S}$ be the total number of iterations in one run of the search framework, the total number of probes N in one run is:

$$N = \mathcal{L} \times \mathcal{S} \tag{4.5}$$

We first consider CSA in which case $\mathcal{L} = 1$, then CGA and CSAGA with $\mathcal{L} > 1$.

CSA maintains a list of one candidate during the search, and like conventional *simulated annealing* (SA) [103], the *cooling schedule* determines the total number of probes to be generated. Initially, the asymptotic convergence of CSA to a $CGM_{dn}$with probability one was proved with respect to a cooling schedule in which temperatures are decreased in a logarithmic fashion [162], based on the original necessary and sufficient condition developed by Hajek for SA [85]. It requires an infinitely long cooling schedule in order to approach a $CGM_{dn}$with probability one.

In practice, asymptotic convergence can never be exploited since any algorithm must terminate in finite time. There are two ways to complete CSA in finite time. The first approach uses an infinitely long logarithmically decreasing cooling schedule but terminates CSA in finite time. This is not desirable because CSA will most likely not have converged to any feasible solution when terminated at high temperatures.

The second approach is to design a cooling schedule that can complete in prescribed finite time. In this thesis we use the following *geometric cooling schedule* with *cooling rate* α [162]:

$$T_{j+1} = \alpha \times T_j, \qquad j = 0, \cdots, N_\alpha - 1, \tag{4.6}$$

where $\alpha < 1$, j measures the number of probes in CSA (assuming one probe is made at each temperature and all probes are independent), and N_α is the total number of probes of the cooling schedule. A probe here is a neighboring point examined by CSA, independent of whether CSA accepts it or not. Given $T_0 > T_{N_\alpha} > 0$ and α, we can determine N_α, the *length of a cooling schedule*, as:

$$N_\alpha = log_\alpha \frac{T_{N_\alpha}}{T_0}. \tag{4.7}$$

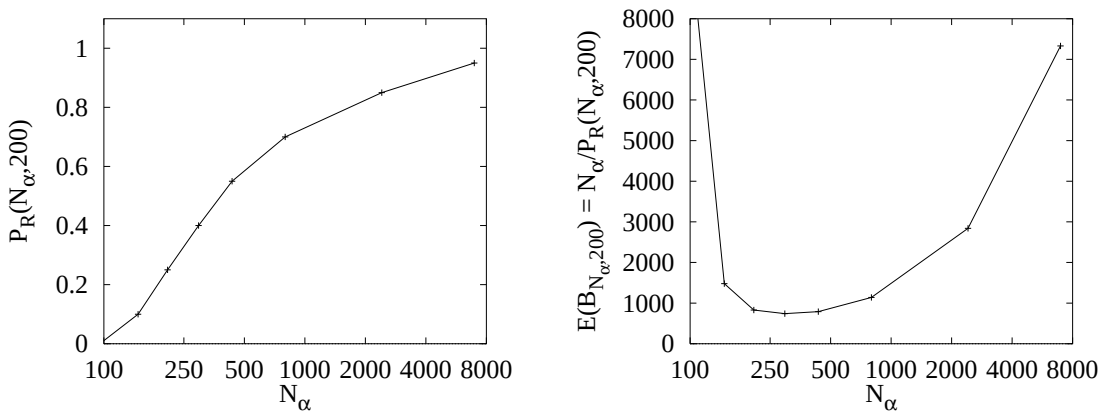
Thus, we can control the total number of iterations in CSA by first setting N_α and then setting α according to (4.7):

$$\alpha = (\frac{T_{N_\alpha}}{T_0})^{\frac{1}{N_\alpha}}. \tag{4.8}$$

The expected overhead of multiple runs of CSA in order to find a solution of desired quality, with N_α probes in one run, is:

$$E(B_s(S^{ICS(N_\alpha)})) = \frac{N_\alpha}{P_R(N_\alpha)} \tag{4.9}$$

In order to verify the existence of an optimal N_α that minimizes (4.9), we have collected statistics on $P_R(N_\alpha)$ and $\frac{N_\alpha}{P_R(N_\alpha)}$ by using CSA to solve test problems G1-G10 [117] and problem 2.7.1, 5.2, 5.4 in the Floudas and Pardalos' problem set [65]. The results indicate that, for all the test problems, $\frac{N_\alpha}{P_R(N_\alpha)}$ has an absolute minimum in (0,∞). In other words, CSA has an optimal N_α that minimizes $E(B_s(S^{ICS(N_\alpha)}))$ for each of these problems.



a) $P_R(N_\alpha)$ b) Absolute minimum in $\frac{N_\alpha}{P_R(N_\alpha)}$

Figure 4.3: An example showing the existence of a minimum in $\frac{N_\alpha}{P_R(N_\alpha)}$ when CSA was applied to solve (1.7). Each cooling schedule is run 200 times.

.

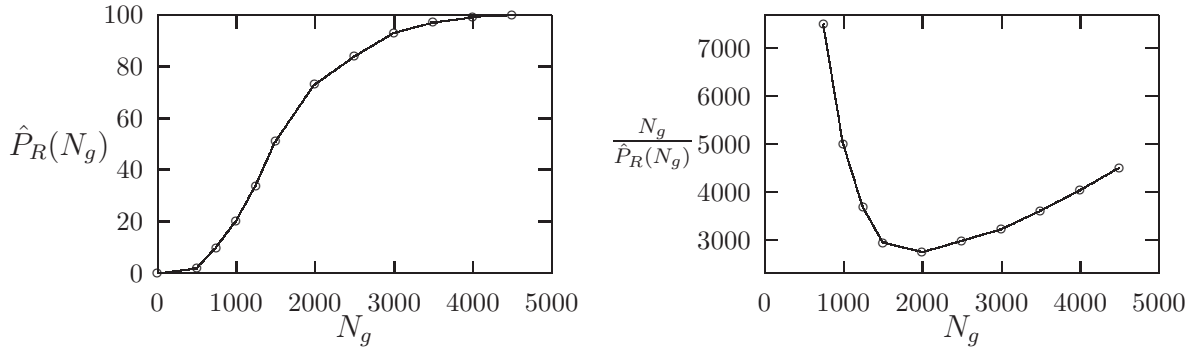
Figure 4.3 illustrates the existence of the optimal N_α in applying CSA to solve (1.7). We see that the $\frac{N_\alpha}{P_R(N_\alpha)}$ curve is convex and has an optimal N_α.

CGA and CSAGA maintain a list of more than one candidates. The number of probes expended in CGA and CSAGA is $N = P \times N_g$, where P is the population size and N_g is the number of generations. Therefore, for CSA and CSAGA,

$$E(B_s(S^{ICS(N)})) = \frac{N}{P_R(N)} = \frac{P \times N_g}{P_R(P \times N_g)} \tag{4.10}$$

As a result, there are two parameters controlling N and $E(B_s(S^{ICS(N)}))$ of an SSA. For these cases, it is generally difficult to use iterative deepening to adjust the multiple control parameters simultaneously in order to lead to schedules that minimize $E(B_s(S^{ICS(N)}))$.

Our research has found that the performance of CGA and CSAGA is more sensitive to N_g than to P. The performance of CGA and CSAGA does not differ much in certain range of P, while varying N_g will have a significant impact to the search efficiency. For



$\hat{P}_R(N_g)$ b) Absolute minimum in $\frac{N_g}{\hat{P}_R(N_g)}$

Figure 4.4: An example showing the existence of a minimum in $\frac{N_g}{\hat{P}_R(N_g)}$ when CSAGA with a population size $P = 3$ was applied to solve G1 [117]. The experiments were run 200 times at each N_g.

different problems with different size and complexity, the best N_g can differ by several orders of magnitude, whereas the best P remains within a relatively small range. For this reason, in developing optimal schedules of CGA and CSAGA, we apply iterative deepening to N_g with a fixed P.

For any fixed P, let $\hat{P}_R(N_g)$ be the reachability probability with N_g generations; i.e., $\hat{P}_R(N_g) = P_R(P \times N_g)$. The expected number of probes using multiple runs of CGA is:

$$\frac{N}{P_R(N)} = \frac{P \times N_g}{P_R(P \times N_g)} = P\frac{N_g}{\hat{P}_R(N_g)} \tag{4.11}$$

Therefore, N_g determines the expected overhead of CGA with a fixed population size P. Hence, (4.11) has the same form as (1.6) except for constant P, and there is generally an optimal N_g that minimizes $\frac{N_g}{\hat{P}_R(N_g)}$.

We have also collected statistics on $\frac{N_g}{\hat{P}_R(N_g)}$ at various P by using CGA and CSAGA to solve the set of problems solved by CSA. The results indicate that, for both CGA and CSAGA, $\frac{N_g}{\hat{P}_R(N_g)}$ has an absolute minimum in (0,∞). In other words, each of these problems

has an optimal number of generations that minimizes the expected overhead to find a solution by multiple runs of CGA or CSAGA.

Figure 4.4 shows the existence of an optimal N_g in applying CSAGA to solve problem G1 with $P = 3$. We see that $\frac{N_g}{\hat{P}_R(N_g)}$ is convex with an optimal N_g.

4.2.2 Iterative deepening applied to CSA, CGA and CSAGA

Figure 4.5 shows CSA_{ID}, the scheduling of multiple runs of CSA with iterative deepening. Derived from the generic algorithm in Figure 3.3, CSA_{ID} uses a set of geometrically increasing cooling schedules:

$$N_{\alpha_i} = \rho^i N_0, \qquad i = 0, 1, \ldots, \tag{4.12}$$

where N_0 is the (small) initial cooling schedule. Under each cooling schedule, CSA is run for a maximum of K times but stops immediately when a solution is found. For iterative deepening to work, $\rho > 1$.

In a similar way, we apply iterative deepening to derive the schedule of running CGA multiple times. CGA_{ID} in Figure 4.6 uses a set of geometrically increasing N_g to find a solution of quality Q:

$$N_{g_i} = \rho^i N_0, \qquad i = 0, 1, \ldots \tag{4.13}$$

where N_0 is the (small) initial number of generations used. CSAGA with iterative deepening ($CSAGA_{ID}$) can be obtained by substituting CGA with CSAGA in Figure 4.6.

According to Theorem 3.2, we have $E_{ID}(1) = O(\mathcal{T}_{opt}(1))$ if the sufficient conditions in the theorem are satisfied. Typically, $\rho = 2$, and in all the benchmarks tested, both $P_R(N_{opt}^{ICS}(1))$ and $\hat{P}_R(N_{opt}^{ICS}(1))$ are no less than 0.25. Substituting these values into the second condition in Theorem 3.2 yields $K > 2.4$. In our experiments, we have used $K = 3$. In addition, we

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_α;
3. **repeat** /*using iterative deepening to find Q*/
4. **for** $i \leftarrow 1$ **to** K **do** call CSA(N_α) **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_α exceeds a maximum number allowed
or (no better solution has been found in two
successive increases of N_α **and** $N_\alpha > \rho^5 N_0$
and a feasible solution has been found);
7. **end_procedure**

Figure 4.5: CSA_{ID}: CSA with iterative deepening, called with desired solution quality Q.

have set $N_0 = 10 \cdot n_v$, and $N_{max} = 1.0 \times 10^8 n_v$ in our experiments, where n_v is the number of variables, and N_0 and N_{max} are, respectively, the initial and maximum number of probes.

Line 6 in Figures 4.5 and 4.6 imposes a stopping condition in order to let the algorithm terminate. The algorithm will stop when a solution of target quality Q has been found, or when the total number of probes spent in solving a target level Q exceeds a maximum threshold, or when at least five deepenings have been performed, and no better solution has been found in two successive geometric increases of the schedule after at least one feasible solution has been found. These stopping conditions are to ensure that iterative deepening has been applied a sufficient number of times.

In practice, $CSA_{ID}(Q)$, $CGA_{ID}(Q)$, $CSAGA_{ID}(Q)$ may be called as a module by the anytime search algorithms developed in the next chapter. In that case, the SSAs scheduled in an iterative deepening fashion will be used as a module in the anytime search that will call it with different quality targets and will not stop until the allowed search resource (CPU

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

Figure 4.6: CGA_{ID}: CGA with iterative deepening, called with desired solution quality Q.

time) has been used up. The solution quality depends on how much search time the user can afford. The longer the search time, the better the solution quality will be delivered.

The only remaining issue left in the design of CGA_{ID} and $CSAGA_{ID}$ is in choosing a suitable population size P in each generation. For $CSAGA_{ID}$, we have set $P = 3$ in our experiments. Our experimental results in the next section show that, although the optimal P may be slightly different from 3, the corresponding expected overhead to find a solution differs very little from that when a constant P is used. For CGA_{ID}, the optimal population size ranges from 4 to 40 and is difficult to determine a priori. Although it is possible to choose a suitable population size dynamically, we do not present the algorithm here because it performs worse than $CSAGA_{ID}$.

Table 4.2: Results on CGA_{ID} and $CSAGA_{ID}$ with $P = 3$ in evaluating the 24 combinations of strategies in the framework in Figure 1.1 on problem G2. All CPU times in seconds were averaged over 10 runs and were collected on a Pentinum III 500-MHz computer with Solaris 7. '−' means that no solution with desired quality can be found.

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

4.3 Evaluations of Combinations of Strategies in the Search Framework

The purpose of this section is to evaluate the 24 possible combinations of strategies in the general framework in Figure 1.1, and determine the best combination for generating probes and for organizing candidates.

Table 4.2 shows the evaluation results on problem G2 [117, 105] using each of the 24 combinations of strategies in Figure 1.1 when SA, GA, or combined SA and GA was used to generate probes in the x subspace. We show the average time of 10 runs for each combination of strategies in order to reach two solution quality levels (1% or 10% worse than CGM_{dn}). We have also gotten similar evaluation results on other benchmark problems.

Our results show that using pure GA to generate probes in the x subspace is usually less efficient than using SA or combined SA and GA. When using CSA_{ID} or $CSAGA_{ID}$, we found better performance when probes generated in the x subspace were accepted by annealing rather than by deterministic rules. The former prevents a search from getting stuck in local minima or infeasible points. On the other hand, we found little performance difference when new probes in the λ subspace were generated by probabilistic or by greedy rules and when new candidates were inserted according to annealing or deterministic rules. In short, generating probes in the λ subspace probabilistically and inserting candidates in both the x and λ subspaces by annealing rules leads to good and stable performance. For this reason, we use this combination of strategies in our following experiments.

Next, we show experimental results of evaluating our proposed algorithms on ten constrained NLPs G1-G10 [117, 105]. These problems have objective functions of various types (linear, quadratic, cubic, polynomial, and nonlinear) and constraints of linear inequalities, nonlinear equalities, and nonlinear inequalities. The number of variables is up to 20, and that of constraints, including simple bounds, is up to 42. The ratio of feasible space with respect to the whole search space varies from 0% to almost 100%, and the topologies of feasible regions are quite different.

Table 4.3 compares the performance of CSA_{ID}, CGA_{ID}, and $CSAGA_{ID}$ with respect to $\bar{T}_{ID}(f^*)$, the expected total CPU time of multiple runs until a solution of value f^* is found.

The first two columns show the problem IDs and the corresponding known f^*. Since CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ can all find a CGM_{dn} in all 10 runs, we compare their performance with respect to $\bar{T}_{ID}(f^*)$, the average total overhead of multiple runs until a CGM_{dn} is found. The third and fourth columns show, respectively, the average time and number of $L_d(x, \lambda)$ function evaluations CSA_{ID} takes to find f^*.

Table 4.3: Results on CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ in finding the best-known solution f^* for 10 discretized constrained NLPs. $\bar{T}_{ID}(f^*)$, the CPU time in seconds to find the best-known solution f^*, was averaged over 10 runs and was collected on a Pentinum III 500-MHz computer with Solaris 7. $\#L_d$ represents the number of $L_d(x, \lambda)$-function evaluations. The best $\bar{T}_{ID}(f^*)$ for each problem is boxed.)

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

The next two columns show the performance of CGA_{ID} with respect to P_{opt}, the optimal population size found by enumeration, and the average time to find f^*. These results show that CGA_{ID} is not competitive as compared to CSA_{ID}, even when P_{opt} is used.

Finally, the last five columns show the performance of $CSAGA_{ID}$. The first three present the average times and number of $L_d(x, \lambda)$ evaluations using a constant P, whereas the last two show the average times using P_{opt} found by enumeration. These results show little improvements in using P_{opt}. Further, $CSAGA_{ID}$ has between 9% and 38% in improvement in $\bar{T}_{ID}(f^*)$, when compared to that of CSA_{ID}, for the 10 problems except for G4 and G10.

4.4 Summary

In this chapter, we have extended the general framework that unifies various mechanisms to look for SP_{dn}, that has a one-to-one correspondence to CLM_{dn}. New constrained optimization algorithms, CGA and CSAGA, have been derived from the framework by including genetic search as one of the strategies.

Being SSAs, CSA, CGA and CSAGA all have similar difficulties in determining a suitable duration for each run in order to minimize the expected time to find a solution of desired quality in multiple runs. In this chapter, we have applied the theory of SSAs to these algorithms and have developed their schedules in which successive runs are increased in duration by iterative deepening.

We have evaluated the 24 possible combinations of strategies in the framework experimentally and have identified a good combination that works well in practice. We have demonstrated improvement in performance of CGA and CSAGA when compared to that of CSA. Further, we have found that CGA does not perform well and that CSAGA outperforms CSA and CGA in terms of average search times to find a CGM_{dn}.

Chapter 5

Optimal Anytime Schedules of SSAs for Constrained Optimization

In this chapter, based on the optimal schedule of multiple runs of SSAs proposed in Chapter 4, we design an anytime version of scheduling multiple runs of SSAs that generates gradually improved feasible solutions as more time is spent, eventually finding a CGM_{dn}. In our study, we have observed an exponential relationship between the objective target that SSA_{ID} is looking for and the average completion time. Based on this exponential relationship, we have designed SSA_{AT-ID}, the anytime search that utilizes a set of improved objective quality targets based on the principle of *iterative deepening* . We then prove the optimality of our proposed anytime search algorithm. Finally, we test our proposed algorithm on a set of standard benchmarks and compare its performance with other existing methods.

5.1 Problem Formulation

We have developed in Chapter 4 CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ that are able to determine the schedules of multiple runs of SSAs, in order to minimize the expected search time. However, even the optimal search time may be too long when solving some large complex

constrained NLPs or in real-time applications. For such cases, one is interested in an optimization algorithm that will find a suboptimal solution within a time limit.

An anytime algorithm is an algorithm that can generate improved solutions as more computation time is used. The goal of this chapter is to design a set of objective targets that allow a search schedule with iterative deepening to generate improved solutions as more time is spent, eventually finding a CGM_{dn}. Moreover, the search schedule is optimal in the sense that the time it takes in generating a CGM_{dn} is of the same order as that used by the original SSA with the optimal schedule to find a CGM_{dn}.

The approach we take is to first study statistically the performance of SSAs. Based on the statistics collected, we propose an exponential model relating the value of objective targets sought by SSA and the optimal average execution time. This model leads to the design of SSA_{AT-ID}, the anytime search schedule with iterative deepening, that schedules multiple runs of SSA using a set of geometrically increasing number of probes in each run and a set of linearly improving objective targets.

Let $\mathcal{T}_{opt}(1, f^*)$ be the expected number of probes taken by the original SSA with an optimal sequential schedule to find a CGM_{dn}, and $E_{AT-ID}(1, f^*)$ be the expected total number of probes taken by SSA_{AT-ID} to find solutions of values $Q_0 > \cdots > f^*$ that are gradually improved with time. Based on the principle of iterative deepening [104], we prove the optimality of SSA_{AT-ID} by showing:

$$E_{AT-ID}(1, f^*) = O(\mathcal{T}_{opt}(1, f^*)) \tag{5.1}$$

5.2 Performance Modeling of SSAs for Constrained Optimization

The performance of an SSA procedure to solve a given application problem from a random starting point can be measured by the probability that it will find a solution of a prescribed quality when it stops and the average time it takes to find the solution. We have studied in previous chapters the relationship between the search time and the success probability of one run. In this section, we focus on the relationship between objective targets and the average time of finding a desirable solution using the optimal schedules.

5.2.1 Relaxation of objective target

One way to improve the chance of finding a solution by an SSA is to look for CLM_{dn} instead of CGM_{dn} even though its original goal is to look for CGM_{dn}. An approach to achieve this is to stop the SSA whenever it finds a CLM_{dn} of a prescribed quality. This approach is not desirable in general because SSA may only find a CLM_{dn} when it has used up most of its allocated time, leading to little difference in times between finding CLM_{dn} and CGM_{dn}. Further, it is necessary to prove the asymptotic convergence of the relaxed SSA procedure.

A second approach that we adopt in our study is to modify the constrained NLP in such a way that a CLM of value smaller than Q in the original NLP is considered a CGM_{dn} in the relaxed NLP. Since the SSA procedure is unchanged, its asymptotic convergence behavior remains the same. The relaxed NLP is obtained by transforming the *objective target* of the original NLP:

$$F(f(x), Q) = \begin{cases} Q & \text{if } f(x) \leq Q \\ f(x) & \text{if } f(x) > Q \ . \end{cases} \tag{5.2}$$

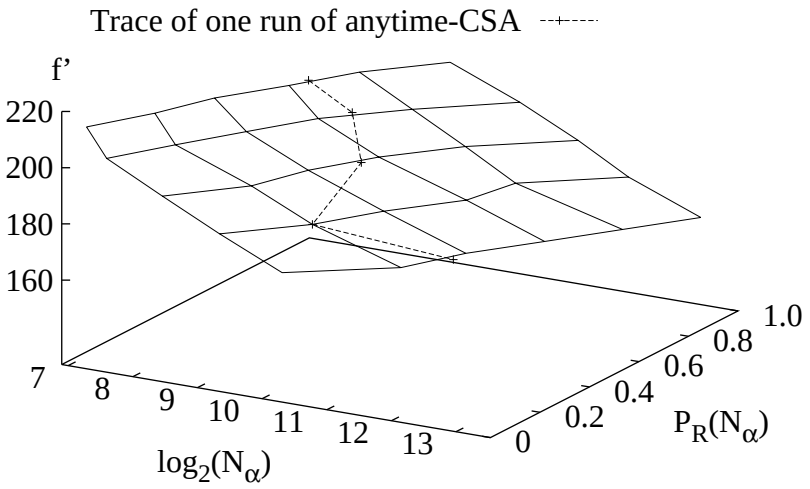


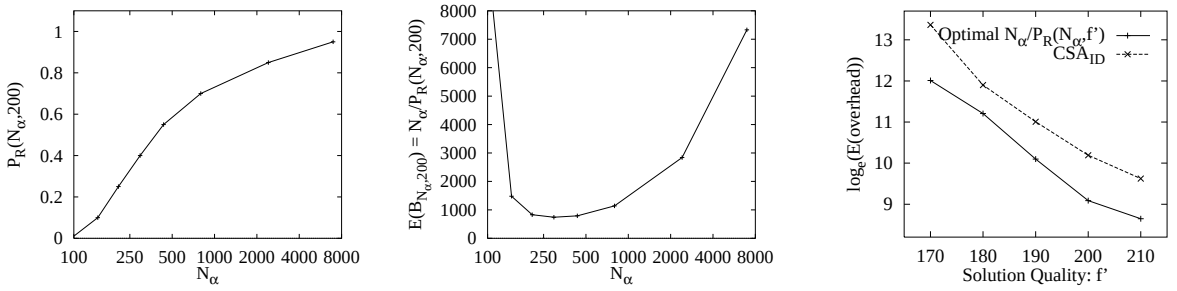
Figure 5.1: A 3-D graph showing the statistics relating Q, N_α, and $P_R(N_\alpha, Q)$, when CSA is applied to solve (1.7). The dotted line shows the trace taken in a run of CSA_{AT-ID}.

Assuming that f^* is the value of the CGM_{dn} in the original NLP, it follows that the value of the CGM_{dn} of the relaxed NLP is f^* if $Q \leq f^*$ and is Q if $Q > f^*$. Moreover, since the relaxed problem is a valid NLP solvable by CSA, CSA will converge asymptotically to a CGM_{dn} of the relaxed NLP with probability one.

As a relaxed objective function leads to a possibly larger pool of solution points, we expect SSAs to have a higher chance of hitting one of these points during its search. This property will be exploited in SSA_{AT-ID} in the next section.

5.2.2 Exponential model relating Q and $\mathcal{T}_{opt}(1, Q)$

In order to develop SSA_{AT-ID} that dynamically controls its objective targets, we need to know the relationship among Q, the degree of objective relaxation, N, the number of probes in one run of SSA, and $P_R(N, Q)$, the reachability probability. In this section we find this relationship by studying the statistical behavior of SSAs in evaluating five constrained NLPs.



a) $P_R(N_\alpha, Q = 200)$ b)$E(B_s(S^{ICS(N_\alpha)}, Q = 200))$ c) $\mathcal{T}_{opt}(1, Q) = O(e^{-Q})$

Figure 5.2: An example showing the performance model of CSA. Note that there is an absolute minimum in the $\frac{N_\alpha}{P_R(N_\alpha,Q)}$ curve, and the optimal expected overhead increases exponentially with an improved solution quality level. (The problem solved is defined in (1.7). Each cooling schedule is run 200 times). a) and b) show the results with $Q = 200$.

Figure 5.1 shows a 3-D graph relating the parameters in solving (1.7), in which $P_R(N_\alpha, Q)$ was obtained by running CSA 200 times for each combination of N_α and Q. From Figure 5.1, we can further derive the relationship between Q and the optimal $\frac{N_\alpha}{P_R(N_\alpha,Q)}$. Figure 5.2 shows example results derived from Figure 5.1.

Figures 5.2a and 5.2b illustrate the $P_R(N_\alpha, Q)$ curve and $\frac{N_\alpha}{P_R(N_\alpha,Q)}$ curve at $Q = 200$ derived from Figure 5.1. We see that there is an absolute minimum in the $\frac{N_\alpha}{P_R(N_\alpha,Q)}$ curve. We can get similar curves at other objective levels. Hence, for each Q, there exists an optimal N_α that minimizes $\frac{N_\alpha}{P_R(N_\alpha,Q)}$. Figure 5.2c shows the optimal expected overhead $\mathcal{T}_{opt}(1, Q)$ under different solution quality level Q.

From the model, we see an exponentially decreasing relationship between Q and $\mathcal{T}_{opt}(1, Q)$. The observation leads to the following exponential model:

$$\mathcal{T}_{opt}(1, Q) = ke^{-aQ} \qquad \text{for positive real constants } a \text{ and } k. \tag{5.3}$$

Problem ID	Problem G1	Problem G2	Rastrigin	Problem 5.2	Problem 5.4
R^2	0.9623	0.9475	0.9842	0.9508	0.9931

Table 5.1: The coefficients of determination R^2 on linear fits of Q and $log_2(\mathcal{T}_{opt}(1, Q))$. The benchmarks evaluated are G1, G2 [117], Rastrigin, Floudas and Pardalos' Problem 5.2 and 5.4 [65].

To verify statistically our proposed model, we performed experiments on several benchmarks of different complexities: G1, G2 [117], Rastrigin (1.7), and Floudas and Pardalos' Problem 5.2 and 5.4 [65]. For each problem, we collected statistics on Q and $\mathcal{T}_{opt}(1, Q)$, regressed a linear function on Q and $log_2(\mathcal{T}_{opt}(1, Q))$ to find a best fit, and calculated the coefficient of determination R^2 of the fit. Table 5.1 summarizes R^2 of the linear fit for each test problem, where R^2 very close to 1 shows a good fit. Since R^2 are very close to one for all problems, $\mathcal{T}_{opt}(1, Q)$ is verified to be exponential with respect to Q.

5.3 Anytime Schedule with Iterative Deepening

We design in this section a schedule to decrease objective target Q in SSA_{AT-ID} that allows it to find f^* using an average time of the same order of magnitude as $\mathcal{T}_{opt}(1, f^*)$.

SSA_{AT-ID} in Figure 5.3 first finds low-quality feasible solutions in relatively small amounts of time. It then tightens its requirement gradually, tries to find a solution at each quality level, and outputs the best solution when it stops.

It is important to point out that SSA_{AT-ID} does not use regression at run time in order to find the values of parameters of (5.3). One reason is that the problem-dependent parameters of the model are hard to estimate. Rather, SSA_{AT-ID} exploits the exponential relationship between Q and $\mathcal{T}_{opt}(1, Q)$ in order to derive a set of SSA runs with different parameters. The

1. **procedure** SSA_{AT-ID}
2. set initial target of solution quality at $Q = \infty$;
3. **repeat** /* over gradually improving solution quality target Q */
4. **call** $SSA_{ID}(Q)$; /* generate a solution of quality Q */
5. **if** $(Q == \infty)$ **then** $Q = Q_0$; **end_if** /* Q_0 is the first fesasible solution found */
6. reduce target level $Q \leftarrow Q - c$;
7. **until** the SSA fails to solve the last target level;
8. **end_procedure**

Figure 5.3: SSA_{AT-ID}: Anytime SSA procedure with iterative deepening that calls $SSA_{ID}(Q)$ in Figure 3.3. The only problem-dependent run-time information used is Q_0.

only run-time information used in SSA_{AT-ID} is Q_0, the value of the first feasible solution found with an initial objective target of $Q = \infty$.

Recall that in $SSA_{ID}(Q)$ (Figure 3.3), we have imposed a stopping condition that will terminate the search if it fails after sufficient deepenings. Therefore, in SSA_{AT-ID}, we terminate the anytime search if $SSA_{ID}(Q)$ fails to solve the last solution quality target level Q. In practice, we can eliminate the stopping conditions in $SSA_{ID}(Q)$, and use SSA_{AT-ID} in an anytime fashion. The anytime procedure will not stop until the allowed search resource (CPU time) has been used up. The solution quality depends on how much search time the user can afford. The longer the search time, the better the solution quality will be delivered.

5.3.1 Anytime search using decreasing objective targets

After finding a solution of quality Q using SSA_{ID} in Figure 3.3, Line 6 adjusts Q to a new objective target so that better solutions will be found if more time is allowed. (If this were the first time that a feasible solution was found, then Line 5 updates Q to Q_0, the value of first feasible solution with an initial objective target of $Q = \infty$.) Based on the exponential

model in (5.3) and the principle in iterative deepening [104], the average number of probes to find a solution of value Q grows geometrically if Q is decreased using the following *linear schedule*:

$$Q_{j+1} = Q_j - c, \text{ where } c \text{ is a positive constant.} \tag{5.4}$$

In our experiments, we have set $c = \max\{10\%Q_0, 0.1\}$.

Let $E_{AT-ID}(1, Q_n)$ be the expected total number of probes SSA_{AT-ID} takes to find Q_n, starting at objective target levels $Q_0, Q_1, \ldots, Q_n$. The following theorem proves the relative complexities of $E_{AT-ID}(1, Q_n)$ and $\mathcal{T}_{opt}(1, Q_n)$.

Theorem 5.1 $E_{AT-ID}(1, Q_n) = O(\mathcal{T}_{opt}(1, Q_n))$.

Proof.

From Theorem 3.2,

$$E_{AT-ID}(1, Q_n) = \sum_{i=0}^{n} E_{ID}(1, Q_i) = \sum_{i=0}^{n} O(\mathcal{T}_{opt}(1, Q_i)) \tag{5.5}$$

Hence, from the exponential model in (5.3), we have:

$$\begin{aligned} E_{AT-ID}(1, Q_n) &= \sum_{i=0}^{n} O(e^{-aQ_i}) = \sum_{i=0}^{n} O(e^{-a(Q_0 - ic)}) \\ &= O(e^{-a(Q_0 - nc)}) = O(\mathcal{T}_{opt}(1, Q_n)) \end{aligned} \tag{5.6}$$

■

The theorem shows that, despite finding solutions of intermediate quality by a linear sequence of improving objective targets during the search, the overall complexity is dominated by that in finding solutions to the last objective target Q_n. In particular, we have established (5.1) by showing that $E_{AT-ID}(1, f^*) = O(\mathcal{T}_{opt}(1, f^*))$.

Our anytime optimal schedules can be extended without any difficulty to scheduling of multiple runs of SSAs in parallel on multiple processors. We shall use the parallel $SSA_{ID}(Q)$ (Figure 3.7) to find a solution of quality Q, verify statistically the exponential relationship between $\mathcal{T}_{opt}^{sync}(n, Q)$, the expected overhead of parallel $SSA_{ID}(Q)$, and Q, the target quality level, and use the linear schedule to reduce Q so that the total overhead is dominated by the overhead for finding the solution of the best Q. Hence, we can obtain *parallel* SSA_{AT-ID} by substituting $SSA_{ID}(Q)$ in Figure 5.3 with parallel $SSA_{ID}(Q)$ (Figure 3.7).

Let $E_{AT-ID}(n, Q_n)$ be the expected total number of probes parallel SSA_{AT-ID} takes to find Q_n, starting at objective target levels $Q_0, Q_1, \ldots, Q_n$. Assuming the exponential relationship between $\mathcal{T}_{opt}^{sync}(n, Q)$ and Q, we have:

Theorem 5.2 $E_{AT-ID}(n, Q_n) = O(\mathcal{T}_{opt}^{sync}(n, Q_n))$.

Proof. The proof follows that in Theorem 5.1 by substituting $\mathcal{T}_{opt}(1, Q_n)$ by $\mathcal{T}_{opt}^{sync}(n, Q_n)$. ■

5.3.2 Behavior of anytime search

We demonstrate the behavior of the anytime search by applying CSA_{AT-ID} on four constrained NLPs of different sizes and degrees of difficulty. G2 [117] and Rastrigin (1.7) are relatively easy NLPs with multiple feasible regions. In particular, (1.7) is characterized by a large number of deep infeasible local minima in the objective function. Floudas and Pardalos' Problems 5.2 and 7.3 [65] are large and difficult NLPs with many equality constraints.

Figure 5.4 compares the anytime behavior of CSA_{AT-ID} and the original CSA in terms of solution quality and execution time. The anytime performance of the original CSA was found by running CSA using the same cooling schedule multiple times until a CGM_{dn} was

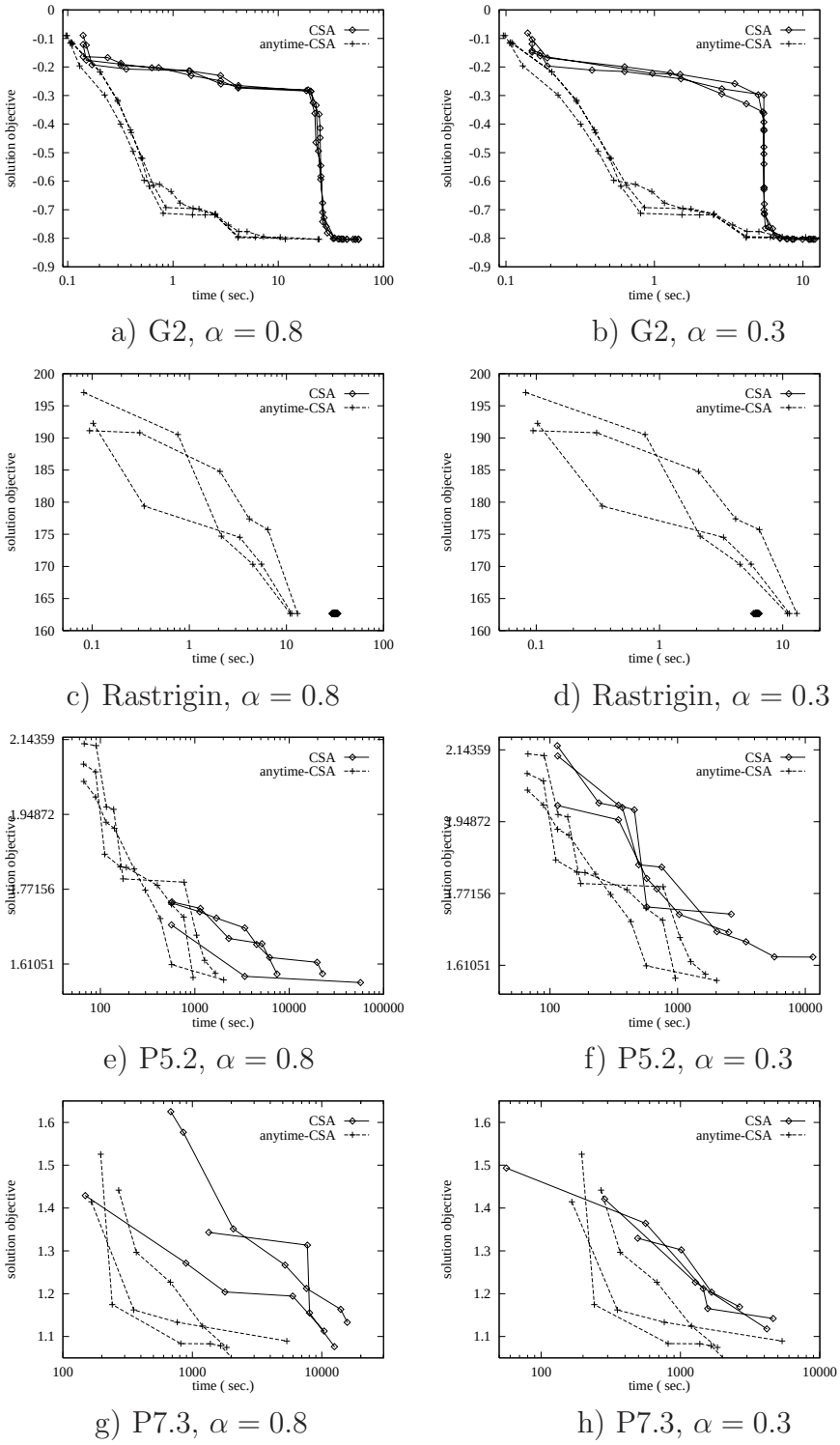


Figure 5.4: A comparison of the anytime performance of CSA_{AT-ID} and the original CSA in solving four constrained minimization NLPs.

found. Without knowing its optimal schedule, we tried two geometric schedules with $\alpha = 0.3$ and $\alpha = 0.8$, respectively. CSA and CSA_{AT-ID} were each ran from three random starting points.

In general, the results show that CSA_{AT-ID} performs substantially better than the original CSA as an anytime algorithm. When compared against a given amount of time, CSA_{AT-ID} found much better suboptimal solutions than CSA. When compared against solutions of the same quality, CSA_{AT-ID} took between one to two orders less time than CSA.

5.4 Experimental results on constrained NLP benchmarks

In this section, we first survey existing constrained NLP benchmarks and solvers in Section 5.4.1. Then, in Section 5.4.2, we report experimental results of comparing CSA_{AT-ID} and $CSAGA_{AT-ID}$ with SQP methods to solve three sets of discrete and mixed-integer optimization benchmarks, derived from continuous benchmark problem sets. Last, in Section 5.4.3, we compare our methods with branch and bound methods for solving constrained MINLPs.

5.4.1 Existing constrained NLP solvers and benchmarks

We summarize existing constrained NLP benchmarks in Table 5.2. The first column lists the names of the benchmarks. Columns 2-10, if checked, indicate that the benchmark contains the corresponding class of constrained NLPs. The next two columns indicate, respectively, the number of variables and the number of constraints (other than fixed variables and bounds) of the NLPs in that benchmark suite. The last column shows the available for-

mats of the benchmark. We see that G1-G10 [117], Floudas and Pardalos' problem set [65], CUTE [48, 12], COPS [59, 60, 7], the Sparse Optimization Problem Set [110, 11], and the Testenvironment for DONLP2 [146] mainly contains continuous constrained NLPs with continuous and differentiable functions. MacMINLP [17], SIF MINLP Test Problems [18], GAMS model library [14], AMPL model library [22], and MIPLIB [9] contains continuous, discrete, and mixed-integer constrained NLPs whose functions are not necessarily differentiable.

We summarize existing constrained NLP solvers in Table 5.3. The first column lists the names of the solvers. Columns 2-10, if checked, indicate that the solver can be applied to solve the corresponding class of constrained NLPs. Column 11 indicates if the solver requires the convexity of functions; Column 12 shows the main method used by the solver; Column 13 shows the input formats acceptable by the solver; and, column 14 shows any additional requirements needed by the solver. We see that DONLP2 [146, 6], LANCELOT [54, 13, 107]. LOQO [154, 20], MINOS [147, 21], KNITRO [51], SNOPT [75], FSQP [5], HQP/OMUSES [2], and MOSEK [29, 19] mainly deal with continuous differentiable constrained NLPs. LOQO [154, 20] and MOSEK [29, 19], in addition, require the convexity of functions. Genocop [116, 10] and COBYLA2 [127] can solve continuous constrained NLPs whose functions are not differentiable or continuous. BARON [3, 133], BNB [1], MINLP_BB [16], SBB [15], mittlp [4], and AlphaEcp [168, 8] can solve continuous, discrete and mixed-integer constrained NLPs whose functions are continuous and differentiable.

5.4.2 Comparison results with SQP methods

SQP is a leading algorithm for solving constrained NLPs in C1. Existing SQP packages, such as DONLP2 [146] and LANCELOT [54, 107], can solve large-scale constrained NLPs in C1

Table 5.2: Summary of existing constrained NLP benchmarks.

Constrained NLP Benchmark	Coverage									Number of Variables	Number of Constraints	Available Formats
	C1	C2	C3	C4	C5	C6	C7	C8	C9			
G1-G10	√									up to 20	up to 9	Fortran
Floudas and Pardalos Problem Set	√									up to 46	up to 36	GAMS, Fortran
CUTE	√	√								up to 14000	up to 14000	SIF, AMPL
COPS	√									parameterized	parameterized	AMPL
Sparse Optimization Problem Set	√									parameterized	parameterized	Fortran
Testenvironment for DONLP2	√									up to 183	up to 1539	Fortran
MacMINLP				√			√			up to 12906	up to 3338	AMPL
SIF MINLP Test Problems				√			√			up to 621	up to 531	SIF
GAMS Model Library	√			√			√			parameterized	parameterized	GAMS
AMPL Model Library	√	√		√			√			parameterized	parameterized	AMPL
MIPLIB				√			√			up to 87482	up to 6803	MPS

Table 5.3: Summary of existing constrained NLP solves.

Constrained NLP Solver	Applicability									Convexity Required	Principal Constraints	Input Formats	Special Requirement
	C1	C2	C3	C4	C5	C6	C7	C8	C9				
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	primal-dual interior-point	AMPL	
SNOPT	√									No	sequential quadratic programming	GAMS, Fortran ,AMPL	modest free variables
CONOPT	√									No	feasible path method	GAMS	
FSQP	√									No	sequential quadratic programming	AMPL	
HQP/OMUSES	√									No	interior-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	f,g,h be 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	f,g,h be pseudo-convex

very efficiently. They can also be used to solve constrained NLPs in C4 and C7, by applying them to solve the corresponding problems in C1 (by regarding variables as continuous ones) and discretizing the solutions found.

5.4.2.1 Continuous benchmark problems and their derived discrete and mixed-integer versions

To compare our methods with SQP methods in solving constrained NLPs in C4 and C7, we derive discrete and mixed-integer benchmarks from the following three sets of continuous benchmarks [117, 105, 65, 48]: 1)G1-G10 [117, 105], 2)Floudas and Pardalos' problems [65], and 3) selected problems from CUTE [48], a constrained and unconstrained testing environment.

In generating a constrained DNLP, we assume that all variables are discrete. In generating a constrained MINLP, we assume that variables with odd indices are continuous and those with even indices are discrete. In discretizing continuous variable x_i in range $[l_i, u_i]$, where l_i and u_i are lower and upper bounds of x_i, respectively, we force x_i to take values from the set:

$$A_i = \begin{cases} \left\{a_i + \frac{b_i - a_i}{s} j,\ j = 0, 1, \cdots, s\right\} & \text{if } b_i - a_i < 1 \\ \left\{a_i + \frac{1}{s} j,\ j = 0, 1, \cdots, \lfloor (b_i - a_i)s \rfloor\right\} & \text{if } b_i - a_i \geq 1, \end{cases} \tag{5.7}$$

where $s = 10000$.

These problems have 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.

Problems G1-G10 [117, 105] were originally developed for testing and tuning various constraint handling techniques in evolutionary algorithms (EAs). The number of variables in this test set is up to 20, and that of constraints, including simple bounds, is up to about 50. The second set of benchmarks [65] were collected by Floudas and Pardalos and were derived from practical applications. The number of variables in this test set is up to about 50, and that of constraints, including simple bounds, is up to about 100. The last test problem set was selected from CUTE [48] based on the criterion that at least the objective or one of the constraints is nonlinear. Both the number of variables and the number of constraints in CUTE can be as large as several thousand.

5.4.2.2 Experimental results on G1-G10 and Floudas and Pardalos' problems

In this section, we report experimental results of CSA_{AT-ID} and $CSAGA_{AT-ID}$ on ten constrained NLPs G1-G10 [117, 105] and all of Floudas and Pardalos' benchmarks [65]. As a comparison, we also solved these problems using DONLP2 [146], a popular SQP package. SQP is an efficient local-search method widely used for solving continuous constrained NLPs. For both discrete and mixed-integer problems, SQP discretizes its corresponding continuous solutions as a result.

For every problem, all the three algorithms run 100 times with the same sequence of random starting points. For comparison, we measure the performance of DONLP2 using T_r/P_r, where T_r is the average time to finish one run and P_r is the probability of finding a feasible solution with prescribed quality of solution. Therefore, T_r/P_r gives the average time to find a feasible solution with prescribed quality.

Table 5.4 shows experimental results on derived *discrete* problems G1-G10. The first two columns show the problem IDs and the best-known solutions. The next six columns list, respectively, T_r/P_r to find feasible solutions that differ within 0%, 1%, 5%, and 20% from

Table 5.4: Performance comparison of DONLP2 (SQP), CSA_{AT-ID} and $CSAGA_{AT-ID}$ in solving derived discrete constrained NLPs G1-G10. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7. '-' stands for no feasible solution found for the specified solution quality within 100 runs. All algorithms use the same sequence of starting points.

Problem	Best-known	DONLP2 for Discrete NLPs				CSA for Discrete NLPs				CSAGA for Discrete NLPs			
ID	Solutions	0%	1%	5%	20%	0%	1%	5%	20%	0%	1%	5%	20%
G1 (min)	-15	1.13	1.13	1.03	0.163	1.61	1.61	1.61	1.10	2.08	2.03	2.03	1.93
G2 (max)	0.8036	-	-	-	-	11.73	9.03	3.09	0.51	9.09	8.04	2.90	0.70
G3 (max)	1.0	0.405	0.405	0.405	0.405	1.38	1.38	1.38	1.38	1.69	1.69	1.69	1.69
G4 (min)	-30665.5	1.17	1.17	1.17	1.17	0.499	0.344	0.0051	0.0031	0.687	0.655	0.0044	0.0016
G5 (min)	4221.9	-	0.105	0.105	0.105	0.84	0.84	0.84	0.84	0.76	0.76	0.76	0.76
G6 (min)	-6961.81	-	-	-	-	11.26	6.22	0.74	0.074	8.33	8.042	0.88	0.067
G7 (min)	24.3062	-	-	-	-	3.08	2.42	2.39	1.46	3.04	2.70	2.70	2.21
G8 (max)	0.095825	0.056	0.056	0.056	0.056	0.074	0.051	0.028	0.007	0.097	0.068	0.024	0.010
G9 (min)	680.63	0.0425	0.0425	0.0425	0.0425	0.40	0.026	0.004	0.002	0.48	0.038	0.017	0.011
G10 (min)	7049.33	-	-	-	-	2.93	2.93	2.13	1.08	3.29	2.84	2.40	1.79

the best-known solutions. The following eight columns show the results for CSA_{AT-ID} and $CSAGA_{AT-ID}$. The timing results are the average time that CSA_{AT-ID} and $CSAGA_{AT-ID}$ find solutions that differ within 0%, 1%, 5%, and 20% from the best-known solutions during an anytime search run. Table 5.5 reports the results on *discrete* problems derived from Floudas and and Pardalos' continuous benchmarks [65].

Table 5.6 and Table 5.7 show the comparison results of DONLP2, CSA_{AT-ID} and $CSAGA_{AT-ID}$ on solving derived MINLPs.

Obviously, CSA_{AT-ID} and $CSAGA_{AT-ID}$ perform much better than DONLP2 in solving both *discrete* and *mixed-integer* constrained NLPs. CSA_{AT-ID} and $CSAGA_{AT-ID}$ are able to find best-known solutions to all the discrete and mixed-integer problems (except 7.3 and 7.4), whereas SQP even fails to find feasible solutions to many problems, such as G2, G6, 5.2 and 5.4. This means that discretization of continuous solutions found by SQP sometimes may not lead to feasible discrete and mixed-integer solutions. Besides, SQP finds very poor

Table 5.5: Performance comparison of DONLP2 (SQP), CSA_{AT-ID} and $CSAGA_{AT-ID}$ in solving derived discrete constrained NLPs from Floudas and Pardalos' continuous constrained benchmarks [65]. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7. '-' stands for no feasible solution found for the specified solution quality within 100 runs. All algorithms use the same sequence of starting points.

Problem	Best-known	DONLP2 for Discrete NLPs				CSA for Discrete NLPs				CSAGA for Discrete NLPs			
ID	Solutions	0%	1%	5%	20%	0%	1%	5%	20%	0%	1%	5%	20%
2.1 (min)	-17	0.85	0.85	0.213	0.0708	0.197	0.151	0.038	0.007	0.594	0.580	0.075	0.026
2.2 (min)	-213	0.022	0.022	0.022	0.022	0.178	0.169	0.033	0.004	0.163	0.161	0.044	0.003
2.3 (min)	-15	1.13	1.13	1.03	0.163	1.610	1.610	1.610	1.108	2.084	2.041	2.041	1.935
2.4 (min)	-11	0.134	0.103	0.0848	0.0629	0.365	0.250	0.074	0.017	0.851	0.735	0.242	0.032
2.5 (min)	-268	-	-	-	-	1.930	1.729	1.729	1.575	4.753	2.688	2.445	2.181
2.6 (min)	-39	9.50	2.38	2.38	1.36	0.436	0.261	0.145	0.026	0.955	0.806	0.516	0.187
2.7.1 (min)	-394.75	-	-	-	14.2	15.16	12.90	12.44	9.469	11.65	10.31	10.31	9.908
2.7.2 (min)	-884.75	-	-	-	-	8.407	8.407	8.344	6.462	9.756	9.559	9.559	9.417
2.7.3 (min)	-8695.0	-	-	-	-	9.24	9.24	9.24	4.93	7.65	6.31	6.31	5.90
2.7.4 (min)	-754.75	-	-	-	71.6	12.53	12.44	11.40	0.215	10.16	9.917	9.917	1.456
2.7.5 (min)	-4150.4	-	-	-	-	31.81	11.60	6.45	4.22	16.06	10.89	5.10	3.81
2.8 (min)	15639.0	-	-	-	-	8.33	6.75	1.84	1.84	9.37	7.906	2.72	1.99
3.1 (min)	7049.33	-	-	-	-	2.93	2.93	2.13	1.08	3.29	2.84	2.40	1.79
3.2 (min)	-30665.5	1.17	1.17	1.17	1.17	0.499	0.344	0.0051	0.0031	0.687	0.655	0.0044	0.0016
3.3 (min)	-310.0	-	-	-	-	0.227	0.216	0.211	0.171	0.652	0.521	0.424	0.348
3.4 (min)	-4.0	0.091	0.084	0.084	0.084	0.147	0.140	0.052	0.002	0.170	0.156	0.088	0.002
4.3 (min)	-4.51	0.18	0.18	0.18	0.15	0.193	0.193	0.193	0.189	0.221	0.199	0.199	0.199
4.4 (min)	-2.217	0.52	0.371	0.217	0.186	0.305	0.183	0.179	0.175	0.193	0.187	0.185	0.178
4.5 (min)	-13.40	-	-	-	3.10	0.411	0.400	0.397	0.397	0.482	0.482	0.482	0.482
4.6 (min)	-5.51	-	-	-	-	0.126	0.095	0.007	0.002	0.148	0.107	0.013	0.003
4.7 (min)	-16.74	0.0706	0.0461	0.0461	0.0461	0.063	0.060	0.058	0.057	0.042	0.040	0.037	0.035
5.2 (min)	1.60	-	-	-	-	4578.06	3708.74	795.65	200.52	1504.78	1342.29	561.9	188.0
5.4 (min)	1.86	-	-	-	-	2265.43	2265.43	1298.77	53.36	706.34	604.42	598.12	206.01
6.2 (max)	400.0	-	-	-	-	6.215	5.000	3.442	0.819	4.572	4.436	3.764	1.434
6.3 (max)	600.0	-	-	-	-	38.22	30.43	22.74	10.26	33.47	31.47	23.25	11.89
6.4 (max)	750.0	-	-	-	-	1.140	1.077	1.014	0.750	0.908	0.802	0.789	0.779
7.2 (min)	1.05	-	-	-	-	848.15	445.84	219.60	6.88	574.54	419.29	377.46	20.87
7.3 (min)	1.51	-	-	-	-	-	-	-	-	-	-	-	-
7.4 (min)	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 5.6: Performance comparison of DONLP2 (SQP), CSA_{AT-ID} and $CSAGA_{AT-ID}$ in solving derived mixed-integer constrained NLPs G1-G10. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7. '-' stands for no feasible solution found for the specified solution quality within 100 runs. All algorithms use the same sequence of starting points.

Problem	Best-known	DONLP2 for Mixed-Integer NLPs				CSA for Mixed-Integer NLPs				CSAGA for Mixed-Integer NLPs			
ID	Solutions	0%	1%	5%	20%	0%	1%	5%	20%	0%	1%	5%	20%
G1 (min)	-15	3.10	3.10	2.48	0.365	1.60	1.60	1.60	1.20	1.43	1.41	1.41	1.24
G2 (max)	0.8036	-	-	-	-	12.75	9.13	2.96	0.58	8.83	6.85	2.66	0.67
G3 (max)	1.0	0.405	0.405	0.405	0.405	1.33	1.33	1.33	1.33	1.07	1.07	1.07	1.07
G4 (min)	-30665.5	1.17	1.17	1.17	1.17	0.45	0.36	0.007	0.004	0.70	0.56	0.01	0.0006
G5 (min)	4221.9	0.35	0.0309	0.0309	0.0309	0.66	0.66	0.66	0.66	0.98	0.98	0.98	0.98
G6 (min)	-6961.81	-	-	-	-	12.76	9.44	0.85	0.06	10.04	7.48	0.85	0.07
G7 (min)	24.3062	-	-	-	-	4.17	3.65	2.65	2.17	2.95	2.01	2.01	1.57
G8 (max)	0.095825	0.056	0.056	0.056	0.056	0.07	0.04	0.02	0.006	0.11	0.06	0.05	0.02
G9 (min)	680.63	-	-	-	-	0.37	0.021	0.004	0.003	0.56	0.061	0.013	0.007
G10 (min)	7049.33	-	-	-	-	3.87	3.25	1.98	1.32	3.76	3.76	2.48	1.10

feasible solutions for some problems such as mixed-integer 2.7.1, where DONLP2 can only find feasible solutions within 20% worse than the best-known solutions.

However, for those problems in which SQP are able to find feasible solutions, the average CPU time of SQP is generally shorter than that of CSA_{AT-ID} and $CSAGA_{AT-ID}$, because each run of SQP is much quicker although the success ratio P_r is lower. The limitation of SQP is that it requires the differentiability of the objective and constraint functions and that it will not be able to solve NLPs whose derivatives are hard to calculate or are unavailable (such as discrete and mixed-integer NLPs).

Comparing CSA_{AT-ID} with $CSAGA_{AT-ID}$, we found that their performance does not differ much for small problems that require little solution time. However, for large and complex problems such as G2, 2.7.5, 5.2, and 5.4, the search time used by $CSAGA_{AT-ID}$

Table 5.7: Performance comparison of DONLP2 (SQP), CSA_{AT-ID} and $CSAGA_{AT-ID}$ in solving derived mixed-integer constrained NLPs from Floudas and Pardalos' continuous constrained benchmarks [65]. All times are in seconds on a Pentium-III 600-MHz computer running Solaris 7. '-' stands for no feasible solution found for the specified solution quality within 100 runs. All algorithms use the same sequence of starting points.

Problem	Best-known	DONLP2 for Mixed-Integer NLPs				CSA for Mixed-Integer NLPs				CSAGA for Mixed-Integer NLPs			
ID	Solutions	0%	1%	5%	20%	0%	1%	5%	20%	0%	1%	5%	20%
2.1 (min)	-17	0.85	0.85	0.213	0.0708	0.194	0.114	0.036	0.008	0.594	0.580	0.075	0.026
2.2 (min)	-213	0.022	0.022	0.022	0.022	0.174	0.170	0.035	0.005	0.163	0.161	0.044	0.003
2.3 (min)	-15	3.10	3.10	2.48	0.365	1.602	1.602	1.602	1.198	2.084	2.041	2.041	1.935
2.4 (min)	-11	0.134	0.103	0.0848	0.083	0.257	0.214	0.066	0.011	0.851	0.735	0.242	0.032
2.5 (min)	-268	-	-	-	-	1.913	1.653	1.653	1.635	4.753	2.688	2.445	2.181
2.6 (min)	-39	9.50	2.38	2.38	1.19	0.420	0.240	0.136	0.028	0.955	0.806	0.516	0.187
2.7.1 (min)	-394.75	-	-	-	11.7	13.41	11.75	11.37	8.097	11.65	10.31	10.31	9.908
2.7.2 (min)	-884.75	-	-	-	37.2	7.804	7.099	7.003	4.813	9.756	9.559	9.559	9.417
2.7.3 (min)	-8695.0	-	-	-	-	9.36	5.25	5.25	4.67	10.94	7.15	7.15	6.89
2.7.4 (min)	-754.75	-	-	-	71.6	12.63	12.33	11.59	0.258	10.16	9.917	9.917	1.456
2.7.5 (min)	-4150.4	-	-	9.66	9.66	37.06	18.38	5.59	4.88	19.91	11.90	6.82	5.99
2.8 (min)	15639.0	-	-	-	-	7.38	6.27	1.48	1.48	7.74	7.01	2.12	2.12
3.1 (min)	7049.33	-	-	-	-	3.87	3.25	1.98	1.32	3.76	3.76	2.48	1.10
3.2 (min)	-30665.5	1.17	1.17	1.17	1.17	0.45	0.36	0.007	0.004	0.70	0.56	0.01	0.0006
3.3 (min)	-310.0	1.30	1.30	0.229	0.0812	0.217	0.202	0.201	0.167	0.652	0.521	0.424	0.348
3.4 (min)	-4.0	0.091	0.084	0.084	0.075	0.138	0.126	0.048	0.001	0.170	0.156	0.088	0.002
4.3 (min)	-4.51	0.18	0.18	0.18	0.15	0.141	0.141	0.141	0.141	0.221	0.199	0.199	0.199
4.4 (min)	-2.217	0.52	0.371	0.289	0.236	0.142	0.142	0.141	0.141	0.193	0.187	0.185	0.178
4.5 (min)	-13.40	-	-	-	3.10	0.371	0.371	0.366	0.366	0.482	0.482	0.482	0.482
4.6 (min)	-5.51	-	-	-	-	0.117	0.086	0.009	0.001	0.148	0.107	0.013	0.003
4.7 (min)	-16.74	0.0667	0.0545	0.0545	0.0545	0.047	0.046	0.045	0.043	0.042	0.040	0.037	0.035
5.2 (min)	1.60	-	-	-	-	6238.5	3686.8	831.1	309.56	4576.91	3428.2	1190.34	476.45
5.4 (min)	1.86	-	-	-	-	2589.8	1539.1	499.1	84.66	1255.51	1255.51	473.52	219.86
6.2 (max)	400.0	-	-	-	-	5.773	4.926	3.653	0.794	4.572	4.436	3.764	1.434
6.3 (max)	600.0	-	-	-	-	39.80	34.75	25.65	10.50	33.47	31.47	23.25	11.89
6.4 (max)	750.0	-	-	-	-	1.033	0.919	0.827	0.720	0.908	0.802	0.789	0.779
7.3 (min)	1.09	-	-	-	-	-	-	-	16034	-	-	-	9766
7.4 (min)	1.08	-	-	-	-	4888.4	4888.4	3492.9	3118.1	3427.9	3427.9	3368.1	2789.0

is usually 1.3 to 3 times shorter than that taken by CSA_{AT-ID} to find a solution of similar quality, for both discrete and mixed-integer versions.

5.4.2.3 Experimental results on CUTE benchmarks

Tables 5.8 and 5.9, report respectively comparison results in solving *discrete* and *mixed-integer* constrained NLPs derived from selected CUTE benchmarks [48] using the given starting point in each problem. 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 five columns show the type of the objective function (linear, quadratic, or nonlinear), 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 next six columns show the solutions and CPU times that we obtain by using LANCELOT [54, 107], CSA_{AT-ID} and $CSAGA_{AT-ID}$, respectively.

CSA_{AT-ID} and $CSAGA_{AT-ID}$ are much better than LANCELOT in terms of their ability to solve discrete and mixed-integer NLPs. LANCELOT are unable to find feasible solutions for about half of the problems. For many problems, discretization of continuous solutions found by LANCELOT does not lead to feasible discrete and mixed-integer solutions.

For the problems that LANCELOT can solve, we see that CSA_{AT-ID} and $CSAGA_{AT-ID}$ take similar or even shorter time as those of LANCELOT in solving small NLPs, but the running times of CSA_{AT-ID} and $CSAGA_{AT-ID}$ are not competitive with those of LANCELOT for solving large continuous constrained NLPs with many variables and constraints. This happens because CSA_{AT-ID} and $CSAGA_{AT-ID}$ are sampling based, whereas LANCELOT uses information on derivatives. CSA_{AT-ID} and $CSAGA_{AT-ID}$, however, are much better in terms of solution quality. LANCELOT could not find feasible solutions for many problems, but CSA_{AT-ID} and $CSAGA_{AT-ID}$ were able to solve all the problems and obtained

better solutions than LANCELOT. For example, CSA_{AT-ID} found a solution of 188.0 and $CSAGA_{AT-ID}$ found a solution of 186.1 for discrete Problem DEMBO7, but LANCELOT could not find a feasible solution. For discrete Problem HS20, LANCELOT found a solution of 40.2, but CSA_{AT-ID} and $CSAGA_{AT-ID}$ found the best solution of 38.2.

Figure 5.5 depicts the normalized solution quality and normalized CPU time [164] of CSA_{AT-ID} and $CSAGA_{AT-ID}$ with respect to LANCELOT for those derived *discrete* CUTE benchmarks that are solvable by all three algorithms. Figure 5.6 depicts the results for *mixed-integer* benchmarks. In these two figures, the smaller is the value, the shorter will be the CPU time or better solution quality. The CPU times of CSA_{AT-ID} and $CSAGA_{AT-ID}$ are generally longer than the corresponding CPU time of LANCELOT, but their overall solution qualities are better.

Tables 5.10 and 5.11 list the discrete and mixed-integer NLPs, respectively, derived from selected CUTE problems that can be solved by LANCELOT but cannot by solved by CSA_{AT-ID} and $CSAGA_{AT-ID}$ at this time. These problems either are too large or have nonlinear constraints that make them very difficult for sample-based CSA_{AT-ID} and $CSAGA_{AT-ID}$ to find feasible points.

Table 5.8: Results comparing LANCELOT, CSA_{AT-ID} and $CSAGA_{AT-ID}$ in solving discrete 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. $'-'$ means that no feasible solution can be found by both the public version (01/05/2000) and the commercial version of LANCELOT (by submitting problems through the Internet, http://www-neos.mcs.anl.gov/neos/solvers/NCO:LANCELOT/). Boxed numbers represent the best solutions among the three methods if they have different solutions.

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		CSA_{AT-ID}		$CSAGA_{AT-ID}$	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
ALLINITA	3	1	quadratic	0	1	0	0	75.0	0.46	75.0	0.21	75.0	0.52
ALLINITC	4	1	nonlinear	0	0	0	1	-	-	30.44	1.29	30.44	1.02
ALSOTAME	2	1	nonlinear	0	1	0	0	0.082	0.57	0.082	0.109	0.082	0.132
BATCH	46	73	nonlinear	12	0	60	1	-	-	307252	4634	278626	8757
BT11	5	3	nonlinear	1	2	0	0	0.825	0.62	0.825	2.019	0.825	1.134
BT12	5	3	quadratic	0	3	0	0	6.188	0.47	6.188	2.47	6.188	2.21
BT6	5	2	nonlinear	0	2	0	0	0.277	0.56	0.277	3.72	0.277	2.90
BT7	5	3	nonlinear	0	3	0	0	306.5	0.51	306.5	1.21	306.5	1.43
BT8	5	2	quadratic	0	2	0	0	1.0	0.57	1.0	0.439	1.0	0.559
CB2	3	3	linear	0	0	0	3	1.952	0.60	1.952	0.800	1.952	0.449
CRESC4	6	8	nonlinear	0	0	0	8	-	-	1.884	1.399	1.037	2.130
CSFI1	5	4	linear	0	2	0	2	-49.07	0.63	-49.07	12.17	-49.07	4.48
DEMBO7	16	20	quadratic	0	0	0	20	-	-	188.0	9.950	186.1	12.87
DIPIGRI	7	4	nonlinear	0	0	0	4	680.6	0.68	680.6	0.980	680.6	1.62
DIXCHLNG	10	5	nonlinear	0	5	0	0	0.0	1.12	0.0	14.01	0.0	9.76
ERRINBAR	18	9	linear	0	8	1	0	-	-	30784	184.3	61317	4438.
EXPFITA	5	22	nonlinear	0	0	22	0	1.13×10^{-3}	0.65	1.13×10^{-3}	6.15	1.13×10^{-3}	2.15
FLETCHER	4	4	quadratic	0	1	3	0	19.53	0.57	11.65	7.659	11.65	12.65
GAUSSELM	14	11	linear	0	5	6	0	-2.25	0.55	-2.006	0.019	-2.20	205.5
GIGOMEZ2	3	3	linear	0	0	0	3	1.952	0.59	1.952	0.975	1.952	0.879
HIMMELBI	100	12	nonlinear	0	0	12	0	-	-	-1735	432.1	-1735	216.7
HIMMELP2	2	1	nonlinear	0	0	0	1	-62.05	0.63	-62.05	0.009	-62.05	0.009
HIMMELP6	2	5	nonlinear	0	0	2	3	-59.01	0.69	-59.01	0.019	-59.01	0.009
HONG	4	1	nonlinear	1	0	0	0	22.57	0.50	22.57	0.49	22.57	0.70
HS100	7	4	nonlinear	0	0	0	4	680.6	0.72	680.6	1.74	680.6	1.61
HS101	7	5	nonlinear	0	0	0	5	-	-	1809	407.5	1809	96.88
HS102	7	5	nonlinear	0	0	0	5	-	-	911.8	29.54	911.8	89.93
HS103	7	5	nonlinear	0	0	0	5	-	-	543.6	356.2	543.6	216.4
HS104	8	5	nonlinear	0	0	0	5	-	-	3.951	3.819	3.951	4.71

continued on next page

continued from previous page

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		CSA_{AT-ID}		$CSAGA_{AT-ID}$	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
IDs													
HS107	9	6	nonlinear	0	6	0	0	5055	0.59	5055	690.3	5055	263.4
HS108	9	13	quadratic	0	0	0	13	-	-	-0.86	9.63	-0.86	2.96
HS111	10	3	nonlinear	0	3	0	0	-47.7	0.83	-47.7	22.11	-47.7	4.32
HS114	10	11	quadratic	1	2	4	4	-	-	-1768	523.4	-1768	208.8
HS117	15	5	nonlinear	0	0	0	5	-	-	32.35	47.63	32.35	47.51
HS119	16	8	nonlinear	8	0	0	0	244.9	0.54	244.9	143.75	244.9	118.90
HS12	2	1	quadratic	0	0	0	1	-30.0	0.46	-30.0	0.019	-30.0	0.05
HS18	2	2	nonlinear	0	0	0	2	-	-	5.000	0.060	5.000	0.029
HS19	2	2	nonlinear	0	0	0	2	-	-	-6938	0.139	-6938	0.55
HS20	2	3	nonlinear	0	0	0	3	40.2	0.52	38.2	0.460	38.2	0.05
HS23	2	5	quadratic	0	0	0	5	-	-	1.999	0.399	1.999	0.439
HS24	2	3	nonlinear	0	0	3	0	-	-	-0.99	0.249	-0.99	0.270
HS26	3	1	nonlinear	0	1	0	0	0.0	0.65	0.0	0.159	0.0	0.159
HS27	3	1	nonlinear	0	1	0	0	0.04	0.49	0.04	0.219	0.04	0.409
HS29	3	1	nonlinear	0	0	0	1	-22.6	0.53	-22.6	0.18	-22.6	0.17
HS30	3	1	quadratic	0	0	0	1	1.0	0.52	1.0	0.009	1.0	0.019
HS32	3	2	nonlinear	1	0	0	1	1.0	0.54	1.0	0.17	1.000	0.27
HS33	3	2	nonlinear	0	0	0	2	-4.0	0.55	-4.58	0.15	-4.58	0.20
HS34	3	2	linear	0	0	0	2	-	-	-0.83	0.18	-0.83	0.22
HS36	3	1	nonlinear	0	0	1	0	-	-	-3299	0.079	-3299	0.140
HS37	3	2	nonlinear	0	0	2	0	-	-	-3455	0.119	-3455	0.199
HS39	4	2	linear	0	2	0	0	-1.0	0.52	-1.00	2.599	-1.00	0.879
HS40	4	3	nonlinear	0	3	0	0	-0.25	0.58	-0.25	68.77	-0.25	84.11
HS41	4	1	nonlinear	1	0	0	0	1.926	0.52	1.925	0.129	1.925	0.100
HS42	4	2	nonlinear	1	1	0	0	13.86	0.56	13.86	0.350	13.86	1.139
HS43	4	3	quadratic	0	0	0	3	-	-	-44.0	0.469	-44.0	0.740
HS46	5	2	nonlinear	0	2	0	0	0.0	0.54	0.0	2.970	0.0	1.029
HS54	6	1	nonlinear	1	0	0	0	-	-	-0.908	38.96	-0.908	11.68
HS55	6	6	nonlinear	6	0	0	0	6.667	0.49	6.333	6.679	6.333	6.780
HS56	7	4	nonlinear	0	4	0	0	-3.456	0.55	-1.0	0.001	-1.0	0.019
HS57	2	1	nonlinear	0	0	0	1	-	-	-7.80	0.079	-7.80	0.249
HS60	3	1	nonlinear	0	1	0	0	0.0326	0.62	0.0326	0.389	0.0326	0.289
HS61	3	2	quadratic	0	2	0	0	-143.65	0.57	-143.65	17.11	-143.65	1.38
HS62	3	1	nonlinear	1	0	0	0	-26273	0.61	-26273	0.32	-26273	0.53
HS63	3	2	quadratic	1	1	0	0	961.72	0.55	961.72	5.319	961.72	3.230
HS64	3	1	nonlinear	0	0	0	1	-	-	6299.8	0.060	6299.8	0.340
HS68	4	2	nonlinear	0	2	0	0	-0.92	0.72	-0.92	7.98	-0.92	9.09
HS69	4	2	nonlinear	0	2	0	0	-956.7	0.80	-956.7	15.63	-956.7	8.94
HS7	2	1	nonlinear	0	1	0	0	-1.73	0.56	-1.73	0.170	-1.73	0.159
HS71	4	2	nonlinear	0	1	0	1	17.01	0.62	17.01	2.519	17.01	9.309
HS73	4	3	linear	1	0	1	1	-	-	29.9	1.75	29.9	1.46

continued on next page

continued from previous page

Problem IDs	n_v	n_c	$f(x)$	$h(x)$ n_{le}	$h(x)$ n_{ne}	$g(x)$ n_{li}	$g(x)$ n_{ni}	LANCELOT solution	LANCELOT CPU time	CSA_{AT-ID} solution	CSA_{AT-ID} CPU time	$CSAGA_{AT-ID}$ solution	$CSAGA_{AT-ID}$ CPU time
HS77	5	2	nonlinear	0	2	0	0	0.241	0.56	0.241	2.410	0.241	10.02
HS78	5	3	nonlinear	0	3	0	0	-2.92	0.58	-2.92	71.87	-2.92	1.460
HS79	5	3	nonlinear	0	3	0	0	0.0788	0.57	0.0788	40.34	0.0788	46.06
HS80	5	3	nonlinear	0	3	0	0	0.054	0.58	0.054	430.5	0.054	222.0
HS83	5	3	quadratic	0	0	0	3	-	-	-3066	1.549	-3066	2.420
HS84	5	3	quadratic	0	0	0	3	-	-	-5280	38.15	-5280	1.149
HS87	6	4	nonlinear	0	4	0	0	-	-	8951	95.41	8946	127.24
HS93	6	2	nonlinear	0	0	0	2	-	-	135.0	1.360	135.0	2.0
HUBFIT	2	1	nonlinear	0	0	1	0	0.0169	0.46	0.0169	0.009	0.0169	0.009
LIN	4	2	nonlinear	2	0	0	0	-0.02	0.70	-0.02	1.38	-0.02	1.25
LOADBAL	31	31	nonlinear	11	0	20	0	0.453	0.69	1.546	0.159	1.546	0.489
LOOTSMA	3	2	nonlinear	0	0	0	2	-	-	1.414	0.289	1.414	0.420
MADSEN	3	6	linear	0	0	0	6	-	-	0.616	0.629	0.616	0.990
MARATOS	2	1	quadratic	0	1	0	0	-1.0	0.40	-1.00	0.211	-1.00	0.170
MATRIX2	6	2	quadratic	0	0	0	2	0.0	0.52	0.0	0.340	0.0	0.420
MESH	41	48	nonlinear	4	20	24	0	-	-	0.0	1.56	0.0	0.709
MISTAKE	9	13	quadratic	0	0	0	13	-	-	-0.99	6.579	-0.99	9.889
MWRIGHT	5	3	nonlinear	0	3	0	0	24.97	0.56	1.288	67.79	1.288	44.25
NGONE	8	8	quadratic	0	0	2	6	-0.5	0.51	-0.5	6.680	-0.5	6.719
ODFITS	10	6	nonlinear	6	0	0	0	-2380	0.50	-2380	4.110	-2380	2.150
OPTCNTRL	32	20	quadratic	10	10	0	0	550	0.51	0.0	0.049	0.0	0.015
OPTPRLOC	30	30	quadratic	0	0	5	25	-	-	-16.4	40.30	-16.4	58.97
PENTAGON	6	15	nonlinear	0	0	15	0	-	-	0.014	0.050	0.014	0.029
POLAK1	3	2	linear	0	0	0	2	-	-	2.718	0.290	2.718	0.219
POLAK3	12	10	linear	0	0	0	10	-	-	5.933	27.36	5.933	19.27
POLAK5	3	2	linear	0	0	0	2	-	-	50.00	0.330	50.00	0.379
POLAK6	5	4	linear	0	0	0	4	-	-	-43.9	3.45	-43.9	4.59
QC	9	4	nonlinear	0	0	4	0	-956.5	0.58	-956.5	0.328	-956.5	0.279
RK23	17	11	linear	4	7	0	0	0.0833	0.75	0.467	60.75	0.344	82.02
ROBOT	14	2	quadratic	0	2	0	0	5.463	0.55	5.463	6.649	5.463	5.76
S316-322	2	1	quadratic	0	1	0	0	334.3	0.48	334.3	0.129	334.3	0.179
SINROSNB	2	1	nonlinear	0	0	0	1	0.0	0.56	0.0	0.419	0.0	0.259
SNAKE	2	2	linear	0	0	0	2	-	-	0.0	0.009	0.0	0.019
SPIRAL	3	2	linear	0	0	0	2	0.0	0.71	0.0	1.563.	0.0	1.370
STANCMIN	3	2	nonlinear	0	0	2	0	4.25	0.58	4.25	0.21	4.25	0.17
SVANBERG	10	10	nonlinear	0	0	0	10	-	-	15.73	15.69	15.73	20.80
SYNTHES1	6	6	nonlinear	0	0	4	2	-	-	0.759	2.940	0.759	7.25
SYNTHES2	11	14	nonlinear	1	0	10	3	-	-	-0.55	52.49	-0.55	7.21
SYNTHES3	17	23	nonlinear	2	0	17	4	-	-	15.08	140.4	15.08	127.5
TENBARS4	18	9	linear	0	8	1	0	-	-	196226	93.38	35883.9	650.00
TWOBARS	2	2	nonlinear	0	0	0	2	1.51	0.53	1.508	0.019	1.508	0.180

continued on next page

continued from previous page

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		CSA_{AT-ID}		$CSAGA_{AT-ID}$	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
WOMFLET	3	3	linear	0	0	0	3	-	-	0.0	0.180	0.0	0.529
ZAMB2-8	138	48	nonlinear	0	48	0	0	-0.153	1.20	-0.153	18403	-0.153	5466
ZECEVIC3	2	2	quadratic	0	0	0	2	97.31	0.54	97.31	0.090	97.31	0.249
ZECEVIC4	2	2	quadratic	0	0	1	1	7.558	0.59	7.558	0.090	7.558	0.129
ZY2	3	2	nonlinear	0	0	0	2	2.0	0.46	2.0	0.300	2.0	0.200

Table 5.9: Results comparing LANCELOT, CSA_{AT-ID} and $CSAGA_{AT-ID}$ in solving 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. $'-'$ means that no feasible solution can be found by both the public version (01/05/2000) and the commercial version of LANCELOT (by submitting problems through the Internet, http://www-neos.mcs.anl.gov/neos/solvers/NCO:LANCELOT/). Boxed numbers represent the best solutions among the three methods if they have different solutions.

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		CSA_{AT-ID}		$CSAGA_{AT-ID}$	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
HAIFAM	99	150	linear	0	1	0	149	-45.0	0.63	-45.0	0.230	-45.0	0.340
HAIFAS	13	9	quadratic	0	1	0	9	-0.455	0.46	-0.455	0.230	-0.455	0.340
HAIFAZ	121	164	quadratic	0	1	0	163	75.0	0.46	75.00	0.230	75.00	0.340
ALLINITC	4	1	nonlinear	0	0	0	1	-	-	30.41	1.289	30.41	2.0
ALSOTAME	2	1	nonlinear	0	1	0	0	0.082	0.57	0.082	0.140	0.082	0.220
AVION2	49	15	nonlinear	15	0	0	0	-	-	9.47×10^7	1176	9.47×10^7	1097
BATCH	46	73	nonlinear	12	0	60	1	-	-	25990	5079	25990	4023
BT11	5	3	nonlinear	1	2	0	0	0.825	0.62	0.824	1.060	0.824	1.779
BT12	5	3	quadratic	0	3	0	0	-	-	6.188	10.81	6.188	5.039
BT6	5	2	nonlinear	0	2	0	0	0.277	0.56	0.277	0.899	0.277	1.299
BT7	5	3	nonlinear	0	3	0	0	306.5	0.51	306.5	4.730	306.5	3.190
BT8	5	2	quadratic	0	2	0	0	1.0	0.57	1.0	0.560	1.0	0.910
CB2	3	3	linear	0	0	0	3	1.952	0.60	1.952	0.610	1.952	0.460
CRESC4	6	8	nonlinear	0	0	0	8	-	- 1.910	1.429	1.232	1.810	
CSFI1	5	4	linear	0	2	0	2	-49.07	0.63	-49.0	9.689	-49.0	3.980
DEMBO7	16	20	quadratic	0	0	0	20	-	-	174.8	154.9	174.8	12.83
DIPIGRI	7	4	nonlinear	0	0	0	4	680.6	0.68	680.6	0.980	680.6	0.75
DIXCHLNG	10	5	nonlinear	0	5	0	0	0.0	1.12	0.0	129.6	0.0	17.63
DNIEPER	61	24	nonlinear	0	24	0	0	1.87×10^4	0.83	22911	4316	20035	2669
ERRINBAR	18	9	linear	0	8	1	0	-	-	10527	13.76	28.05	9273
EXPFITA	5	22	nonlinear	0	0	22	0	0.001	0.65	0.001	2.009	0.001	1.740
FLETCHER	4	4	quadratic	0	1	3	0	19.53	0.57	11.65	21.05	ovalbox11.65	13.02
GAUSSELM	14	11	linear	0	5	6	0	-2.25	0.55	0.0	0.019	-2.25	111.0
GIGOMEZ2	3	3	linear	0	0	0	3	1.952	0.59	1.952	0.360	1.952	0.460
HIMMELBI	100	12	nonlinear	0	0	12	0	-	-	-1735	67.28	-1735	89.1
HIMMELBJ	45	14	nonlinear	14	0	0	0	-	-	-1910	2207	-1910	1359
HIMMELP2	2	1	nonlinear	0	0	0	1	-62.05	0.63	-62.05	0.009	-62.05	0.001
HIMMELP6	2	5	nonlinear	0	0	2	3	-59.01	0.69	-59.01	0.019	-59.01	0.009
HONG	4	1	nonlinear	1	0	0	0	22.57	0.50	22.57	0.529	22.57	0.345
HS100	7	4	nonlinear	0	0	0	4	680.6	0.72	680.6	0.970	680.6	0.730
HS101	7	5	nonlinear	0	0	0	5	-	-	1809	188.3	1809	198.3

continued on next page

continued from previous page

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		CSA_{AT-ID}		$CSAGA_{AT-ID}$	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
IDHS102	7	5	nonlinear	0	0	0	5	911.9	*	911.9	123.5	911.9	97.37
HS103	7	5	nonlinear	0	0	0	5	-	-	544.4	19.81	543.6	100.3
HS104	8	5	nonlinear	0	0	0	5	-	-	3.951	12.34	3.951	10.01
HS107	9	6	nonlinear	0	6	0	0	5055	0.59	5055	82.11	5055	55.7
HS108	9	13	quadratic	0	0	0	13	-	-	-0.86	8.409	-0.86	1.320
HS111	10	3	nonlinear	0	3	0	0	-47.7	0.83	-47.7	171.5	-47.7	38.40
HS114	10	11	quadratic	1	2	4	4	-	-	-1768	244.8	-1768	167.6
HS117	15	5	nonlinear	0	0	0	5	-	-	32.35	20.19	32.35	21.79
HS119	16	8	nonlinear	8	0	0	0	244.9	0.54	244.9	451.4	244.9	115.3
HS12	2	1	quadratic	0	0	0	1	-30.0	0.46	-30.0	0.019	-30.0	0.029
HS18	2	2	nonlinear	0	0	0	2	4.999	0.65	4.999	0.060	4.999	0.029
HS19	2	2	nonlinear	0	0	0	2	-	-	-6907	0.039	-6954	0.109
HS20	2	3	nonlinear	0	0	0	3	40.2	0.52	38.19	0.579	38.19	0.699
HS23	2	5	quadratic	0	0	0	5	-	-	1.999	0.330	1.999	0.819
HS24	2	3	nonlinear	0	0	3	0	-1.0	0.55	-1.0	0.189	-1.0	0.280
HS26	3	1	nonlinear	0	1	0	0	0.0	0.65	0.0	0.139	0.0	0.200
HS27	3	1	nonlinear	0	1	0	0	0.04	0.49	0.04	0.190	0.04	0.409
HS29	3	1	nonlinear	0	0	0	1	-22.6	0.53	-22.6	0.189	-22.6	0.159
HS30	3	1	quadratic	0	0	0	1	1.0	0.52	1.0	0.009	1.0	0.009
HS32	3	2	nonlinear	1	0	0	1	1.0	0.54	1.0	0.290	1.0	0.200
HS33	3	2	nonlinear	0	0	0	2	-4.0	0.55	-4.58	0.140	-4.58	0.189
HS34	3	2	linear	0	0	0	2	-0.834	0.38	-0.834	0.280	-0.834	0.209
HS36	3	1	nonlinear	0	0	1	0	-	-	-3299	0.100	-3299	0.150
HS37	3	2	nonlinear	0	0	2	0	-	-	-3455	0.109	-3455	0.179
HS39	4	2	linear	0	2	0	0	-1.0	0.52	-1.0	0.409	-1.0	0.280
HS40	4	3	nonlinear	0	3	0	0	-0.25	0.58	-0.25	3.960	-0.25	1.710
HS41	4	1	nonlinear	1	0	0	0	1.926	0.52	1.926	0.119	1.926	0.100
HS42	4	2	nonlinear	1	1	0	0	13.86	0.56	13.86	0.219	13.86	0.310
HS43	4	3	quadratic	0	0	0	3	-	-	-44.0	0.469	-44.0	0.740
HS46	5	2	nonlinear	0	2	0	0	0.0	0.54	0.0	0.829	0.0	0.63
HS54	6	1	nonlinear	1	0	0	0	-	-	-0.90	9.969	-0.90	1.240
HS55	6	6	nonlinear	6	0	0	0	6.667	0.49	6.333	8.300	6.333	11.40
HS56	7	4	nonlinear	0	4	0	0	-3.456	0.55	-3.377	34.27	-2.17	15.55
HS57	2	1	nonlinear	0	0	0	1	0.03065	0.57	0.028	0.009	0.028	0.050
HS59	2	3	nonlinear	0	0	0	1	-	-	-7.80	0.070	-7.80	0.249
HS60	3	1	nonlinear	0	1	0	0	0.032	0.62	0.032	0.249	0.032	0.519
HS61	3	2	quadratic	0	2	0	0	-143.6	0.57	-143.6	2.059	-143.6	3.030
HS62	3	1	nonlinear	1	0	0	0	-26273	0.61	-26273	0.310	-26273	0.469
HS63	3	2	quadratic	1	1	0	0	961.7	0.55	961.7	32.97	961.7	3.880
HS64	3	1	nonlinear	0	0	0	1	-	-	6299	0.060	6299	0.340
HS68	4	2	nonlinear	0	2	0	0	-0.92	0.72	-0.92	1.97	-0.92	2.97

continued on next page

continued from previous page

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		CSA_{AT-ID}		$CSAGA_{AT-ID}$	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
IDs:HS69	4	2	nonlinear	0	2	0	0	-956.7	0.80	-956.7	1.18	-956.7	1.34
HS7	2	1	nonlinear	0	1	0	0	-1.73	0.56	-1.73	0.060	-1.73	0.209
HS71	4	2	nonlinear	0	1	0	1	17.01	0.62	17.01	0.479	17.01	1.029
HS73	4	3	linear	1	0	1	1	-	-	29.89	0.860	29.89	41.51
HS74	4	5	nonlinear	0	3	2	0	-	-	5126	1.340	5126	6.300
HS75	4	5	nonlinear	0	3	2	0	-	-	5407	14.73	5284	2636
HS77	5	2	nonlinear	0	2	0	0	0.241	0.56	0.241	0.699	0.241	1.029
HS78	5	3	nonlinear	0	3	0	0	-2.9	0.58	-2.9	83.12	-2.9	27.15
HS79	5	3	nonlinear	0	3	0	0	0.0788	0.57	0.116	5.010	0.0788	2.140
HS80	5	3	nonlinear	0	3	0	0	0.054	0.58	0.054	21.19	0.054	13.52
HS83	5	3	quadratic	0	0	0	3	-	-	-3066	0.790	-3066	2.049
HS84	5	3	quadratic	0	0	0	3	-	-	-5280	63.02	-5280	2.490
HS87	6	4	nonlinear	0	4	0	0	-	-	8926	14.97	8926	7.350
HS93	6	2	nonlinear	0	0	0	2	-	-	135.0	1.300	135.0	1.009
HS99	7	2	nonlinear	0	2	0	0	-	-	-8.24×10^8	53.55	-8.31×10^8	6.61
HUBFIT	2	1	nonlinear	0	0	1	0	0.0169	0.46	0.0169	0.009	0.0169	0.009
LAUNCH	25	28	nonlinear	6	3	12	7	-	-	10.57	1670	10.92	2681
LIN	4	2	nonlinear	2	0	0	0	-0.02	0.70	-0.02	1.450	-0.02	2.210
LOADBAL	31	31	nonlinear	11	0	20	0	0.453	0.69	0.647	4765	0.527	725.2
LOOTSMA	3	2	nonlinear	0	0	0	2	-	-	1.414	0.180	1.414	0.419
MADSEN	3	6	linear	0	0	0	6	-	-	0.616	0.629	0.616	0.970
MARATOS	2	1	quadratic	0	1	0	0	-1.0	0.40	-1.0	0.059	-1.0	0.079
MATRIX2	6	2	quadratic	0	0	0	2	0.0	0.52	0.0	4.159	0.0	2.810
MESH	41	48	nonlinear	4	20	24	0	-	-	0.000	0.230	0.000	0.679
MISTAKE	9	13	quadratic	0	0	0	13	-	-	-1.00	6.570	-1.00	4.530
MRIBASIS	36	55	linear	1	8	43	3	-	-	18.21	2468	21.17	2226
MWRIGHT	5	3	nonlinear	0	3	0	0	24.97	0.56	1.301	0.819	1.289	13.79
NGONE	8	8	quadratic	0	0	2	6	-0.5	0.51	-0.5	4.099	-0.5	6.5
ODFITS	10	6	nonlinear	6	0	0	0	-2380	0.50	-2380	4.599	-2380	3.809
OPTCNTRL	32	20	quadratic	10	10	0	0	550	0.51	0.0	0.050	0.0	0.149
OPTPRLOC	30	30	quadratic	0	0	5	25	-	-	-16.4	44.66	-16.4	113.5
ORTHREGB	27	6	quadratic	0	6	0	0	-	-	0.0	94.37	0.0	242.0
PENTAGON	6	15	nonlinear	0	0	15	0	1.509×10^{-4}	0.56	1.365×10^{-4}	8.159	1.365×10^{-4}	7.019
POLAK1	3	2	linear	0	0	0	2	-	-	2.718	0.289	2.718	0.409
POLAK3	12	10	linear	0	0	0	10	-	-	5.933	27.40	5.933	21.25
POLAK5	3	2	linear	0	0	0	2	-	-	50.00	0.409	50.00	0.350
POLAK6	5	4	linear	0	0	0	4	-	-	-43.9	1.570	-43.9	2.029
QC	9	4	nonlinear	0	0	4	0	-956.5	0.58	-1076	5.509	-956.5	0.620
READING6	102	50	nonlinear	0	50	0	0	-	-	-101.18	252.17	-125.38	676.0
RK23	17	11	linear	4	7	0	0	0.0833	0.75	2.883	43.61	1.555	23.25
ROBOT	14	2	quadratic	0	2	0	0	5.463	0.55	5.463	71.79	5.463	8.609

continued on next page

continued from previous page

Problem	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT		CSA_{AT-ID}		$CSAGA_{AT-ID}$	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time	solution	CPU time	solution	CPU time
IS316-322	2	1	quadratic	0	1	0	0	334.3	0.48	334.3	0.070	334.3	0.179
SINROSNB	2	1	nonlinear	0	0	0	1	0.0	0.56	0.0	1.23	0.0	1.15
SNAKE	2	2	linear	0	0	0	2	-	-	0.0	0.002	0.0	0.019
SPIRAL	3	2	linear	0	0	0	2	0.0	0.71	0.0	2.039	0.0	2.876
STANCMIN	3	2	nonlinear	0	0	2	0	4.25	0.58	4.25	0.140	4.25	0.170
SVANBERG	10	10	nonlinear	0	0	0	10	-	-	15.73	12.51	15.73	9.229
SYNTHES1	6	6	nonlinear	0	0	4	2	-	-	0.759	2.269	0.759	3.509
SYNTHES2	11	14	nonlinear	1	0	10	3	-	-	-0.55	22.71	-0.55	40.39
SYNTHES3	17	23	nonlinear	2	0	17	4	-	-	15.08	53.15	15.08	31.5
TENBARS4	18	9	linear	0	8	1	0	-	-	368.4	251.3	368.4	171.5
TWOBARS	2	2	nonlinear	0	0	0	2	1.51	0.53	1.51	0.100	1.51	0.179
WOMFLET	3	3	linear	0	0	0	3	-	-	0.0	0.170	0.0	0.169
ZAMB2-8	138	48	nonlinear	0	48	0	0	-0.153	1.20	1.279	5633	1.132	7631
ZECEVIC3	2	2	quadratic	0	0	0	2	97.31	0.54	97.30	0.079	97.30	0.119
ZECEVIC4	2	2	quadratic	0	0	1	1	7.558	0.59	7.558	0.079	7.558	0.120
ZY2	3	2	nonlinear	0	0	0	2	2.0	0.46	2.0	0.280	2.0	0.409

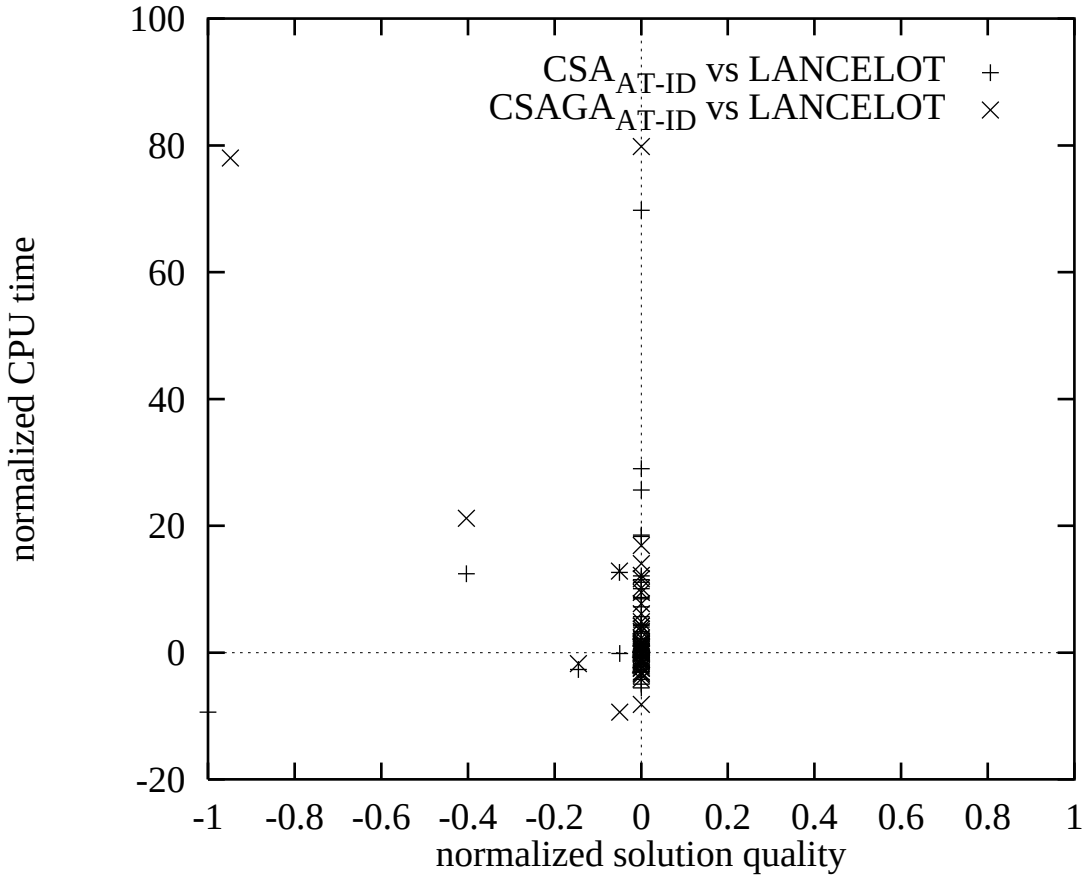


Figure 5.5: Normalized solution qualities and normalized CPU times of CSA_{AT-ID} and $CSAGA_{AT-ID}$ with respect to LANCELOT in solving derived *discrete* CUTE benchmarks that are solvable by all CSA_{AT-ID} and $CSAGA_{AT-ID}$ and LANCELOT.

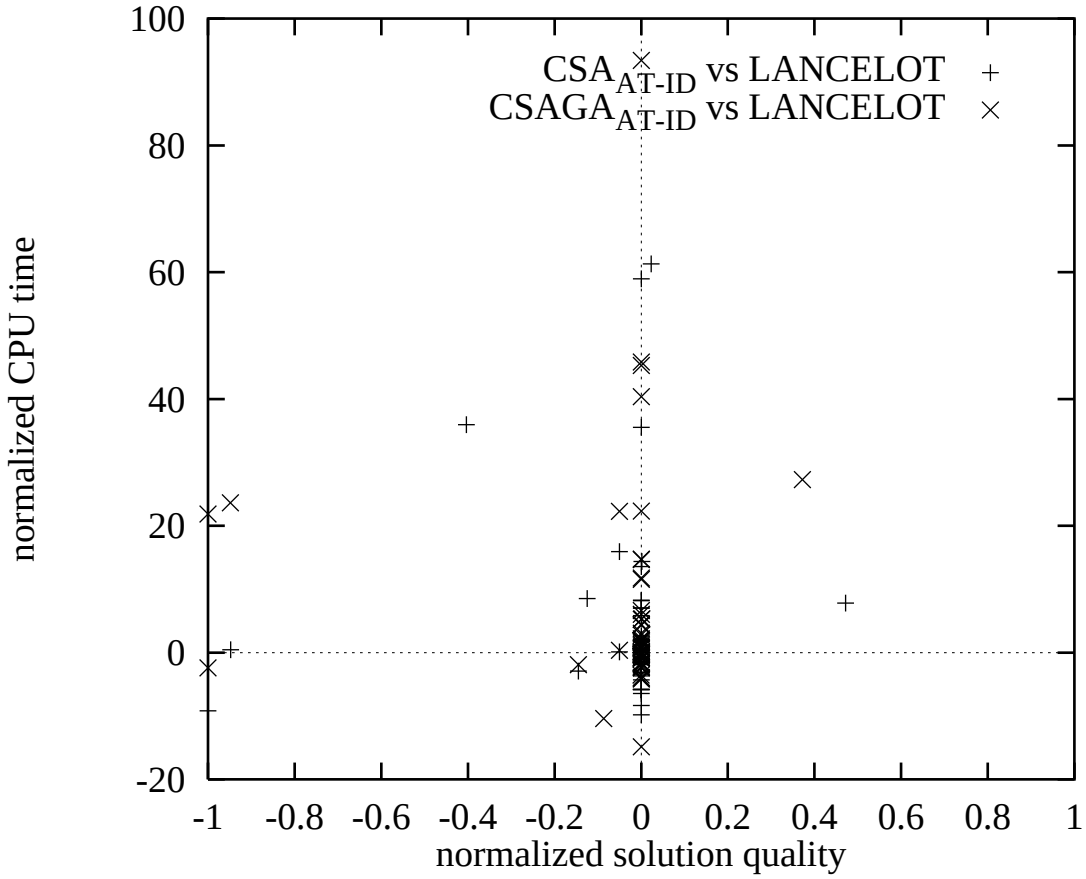


Figure 5.6: Normalized solution qualities and normalized CPU times of CSA_{AT-ID} and $CSAGA_{AT-ID}$ with respect to LANCELOT in solving derived *mixed-integer* CUTE benchmarks that are solvable by all CSA_{AT-ID} and $CSAGA_{AT-ID}$ and LANCELOT.

Table 5.10: Discrete constrained NLPs derived from selected CUTE problems that were solvable by LANCELOT but not by CSA_{AT-ID} and $CSAGA_{AT-ID}$ at this time. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7.

Problem IDs	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time
BRAINPC0	6907	6900	nonlinear	0	6900	0	0	1.5E-3	55.5
BRAINPC1	6907	6900	nonlinear	0	6900	0	0	0.0	84.8
BRAINPC2	13807	13800	nonlinear	0	13800	0	0	4.1E-8	93.2
BRAINPC3	6907	6900	nonlinear	0	6900	0	0	1.687E-4	89.4
BRAINPC4	6907	6900	nonlinear	0	6900	0	0	1.288E-3	79.1
BRAINPC5	6907	6900	nonlinear	0	6900	0	0	1.362E-3	143.7
BRAINPC6	6907	6900	nonlinear	0	6900	0	0	5.931E-5	85.2
BRAINPC7	6907	6900	nonlinear	0	6900	0	0	3.82E-5	109.4
BRAINPC8	6907	6900	nonlinear	0	6900	0	0	1.652E-4	112.8
BRAINPC9	6907	6900	nonlinear	0	6900	0	0	8.27E-4	68.2
BRITGAS	450	360	nonlinear	0	360	0	0	0.0	8.3
C-RELOAD	342	284	linear	26	174	0	84	-1.027	51.1
HYDROELL	1009	1008	nonlinear	0	0	1008	0	-3.586E6	70.5
LEAKNET	156	153	linear	73	80	0	0	8.0	25.7
SARO	4754	4015	linear	0	4015	0	0	252.3	3739.0
SAROMM	5120	5110	linear	365	4015	730	0	57.35	9147.5
TWIRISM1	343	313	nonlinear	50	174	5	84	-1.01	136.1
TWIRIMD1	1247	544	nonlinear	143	378	5	186	-1.034	10158
ZAMB2-10	270	96	nonlinear	0	96	0	0	-1.58	2.99
ZAMB2-11	270	96	nonlinear	0	96	0	0	-1.116	1.83

Table 5.11: Mixed-integer constrained NLPs derived from selected CUTE problems that were solvable by LANCELOT but not by CSA_{AT-ID} and $CSAGA_{AT-ID}$ at this time. All times are in seconds on a Pentium-III 500-MHz computer running Solaris 7.

Problem IDs	n_v	n_c	$f(x)$	$h(x)$		$g(x)$		LANCELOT	
				n_{le}	n_{ne}	n_{li}	n_{ni}	solution	CPU time
BRAINPC0	6907	6900	nonlinear	0	6900	0	0	1.5E-3	55.5
BRAINPC1	6907	6900	nonlinear	0	6900	0	0	0.0	84.8
BRAINPC2	13807	13800	nonlinear	0	13800	0	0	4.1E-8	93.2
BRAINPC3	6907	6900	nonlinear	0	6900	0	0	1.687E-4	89.4
BRAINPC4	6907	6900	nonlinear	0	6900	0	0	1.288E-3	79.1
BRAINPC5	6907	6900	nonlinear	0	6900	0	0	1.362E-3	143.7
BRAINPC6	6907	6900	nonlinear	0	6900	0	0	5.931E-5	85.2
BRAINPC7	6907	6900	nonlinear	0	6900	0	0	3.82E-5	109.4
BRAINPC8	6907	6900	nonlinear	0	6900	0	0	1.652E-4	112.8
BRAINPC9	6907	6900	nonlinear	0	6900	0	0	8.27E-4	68.2
BRITGAS	450	360	nonlinear	0	360	0	0	0.0	8.3
C-RELOAD	342	284	linear	26	174	0	84	-1.027	51.1
HYDROELL	1009	1008	nonlinear	0	0	1008	0	-3.586E6	70.5
LEAKNET	156	153	linear	73	80	0	0	8.0	25.7
SARO	4754	4015	linear	0	4015	0	0	252.3	3739.0
SAROMM	5120	5110	linear	365	4015	730	0	57.35	9147.5
TWIRISM1	343	313	nonlinear	50	174	5	84	-1.01	136.1
TWIRIMD1	1247	544	nonlinear	143	378	5	186	-1.034	10158
ZAMB2-10	270	96	nonlinear	0	96	0	0	-1.58	2.99
ZAMB2-11	270	96	nonlinear	0	96	0	0	-1.116	1.83

5.4.3 Comparison results with branch and bound methods

We compare our methods with MINLP_BB [16], an MINLP solver using the branch and bound method, in solving selected problems in the MINLP benchmark suite MacMINLP [22].

We summarize the experimental results in Table 5.12. The first column lists the problem IDs. We select all the problems with no more than 100 variables in MacMINLP [22]. The next five columns show, respectively, the type of objective function (linear, quadratic, or nonlinear), the number (n_v) of variables, the number (n_i) of integer variables, the type of constraint functions (linear, quadratic, or nonlinear), and the number (n_c) of constraints. The next six columns show the solutions and CPU times that we obtained by using MINLP_BB (by submitting problems to the NEOS server at http://www-neos.mcs.anl.gov/neos/solvers/IP:MINLP-AMPL/), CSA_{AT-ID} and $CSAGA_{AT-ID}$, respectively.

For the problems that CSA_{AT-ID} and $CSAGA_{AT-ID}$ can solve, our methods are much better than MINLP_BB in terms of solution quality. For example, CSA_{AT-ID} found a solution of objective value 1.856 and $CSAGA_{AT-ID}$ found a solution of objective value 1.877 for problem SPRING, but MINLP_BB failed to find a feasible solution. For problem TRIMLOSS2, MINLP_BB found a feasible solution of objective value 5.3, whereas CSA_{AT-ID} and $CSAGA_{AT-ID}$ found a solution of objective value 2.33.

However, the running time of CSA_{AT-ID} and $CSAGA_{AT-ID}$ are not competitive with those of MINLP_BB for solving large problems. This happens because CSA_{AT-ID} and $CSAGA_{AT-ID}$ are sampling based, whereas MINLP_BB utilizes information on derivatives of functions during search.

Finally, we should point out that branch and bound methods are not able to solve constrained NLPs in classes C5, C6, C8 and C9, since they require differentiability of functions.

Our methods, CSA_{AT-ID} and $CSAGA_{AT-ID}$, are general, derivative-free methods that are able to solve constrained NLPs in classes C2-C9.

Table 5.12: Results comparing CSA_{AT-ID}, $CSAGA_{AT-ID}$ and MINLP_BB [16] in solving selected MINLPs from MacMINLP benchmark suite [17]. All times for CSA_{AT-ID} and $CSAGA_{AT-ID}$ are in seconds on a Pentium-III 500-MHz computer running Solaris 7. All times for MINLP_BB are in seconds on the NEOS server at http://www-neos.mcs.anl.gov/neos/. '-' means that no feasible solution can be found. Boxed numbers represent the best solutions among the three methods if they have different solutions.

Problem ID	objective	n_v	n_i	constraints	n_c	MINLP_BB		CSA_{AT-ID}		$CSAGA_{AT-ID}$	
						solution	T	solution	T	solution	T
BATCH	nonlinear	46	24	nonlinear	73	285507	0.58	9.16308	96.78	9.16308	41.82
C-SCHED1	nonlinear	73	60	linear	16	-30639.3	0.42	-	-	-	-
FEEDLOC	linear	90	37	nonlinear	259	-	-	-	-	-	-
MITTELMAN	nonlinear	16	16	nonlinear	7	16	0.26	2.65625	0.01	2.65625	0.01
OPTPRLOC	quadratic	30	25	quadratic	29	-8.06414	0.78	-2.502	38.59	-2.502	21.95
SPRING	nonlinear	17	11	nonlinear	10	-	-	1.856	29.34	1.877	14.16
SYNTHES1	nonlinear	6	3	nonlinear	6	6.00976	0.01	2.3756	0.76	2.3756	0.60
SYNTHES2	nonlinear	11	5	nonlinear	14	73.0353	0.04	3.285	18.90	3.285	12.58
SYNTHES3	nonlinear	17	8	nonlinear	19	68.0097	0.10	3.2659	204.53	3.2657	370.67
TRIMLON2	linear	8	8	nonlinear	12	5.3	0.11	2.3312	52.37	2.3312	34.52
TRIMLON4	linear	24	24	nonlinear	26	11.1	6.70	2.528	30.01	2.553	105.10
TRIMLON5	linear	35	35	nonlinear	33	12.5	10.60	2.674	418.42	2.674	1385.49
TRIMLON6	linear	48	48	nonlinear	41	27	13.71	-	-	-	-
TRIMLON7	linear	63	63	nonlinear	42	-	-	-	-	-	-
TRIMLOSS2	linear	37	31	nonlinear	24	5.3	2.59	2.33	27.73	2.33	22.27
WIND-FAC	linear	15	3	nonlinear	14	0.254487	0.04	1.627	259.86	1.627	186.92

5.5 Summary

Anytime search algorithms are important in solving large complex constrained NLPs in which the time to find a CGM_{dn} is limited, and suboptimal CLM_{dn} are acceptable if they can be found within the time limit.

In searching for a CLM_{dn} of certain quality level, SSA_{AT-ID} controls the scheduling and duration of multiple runs of SSAs by iterative deepening in order for the overhead in the last SSA to dominate the total overhead of all SSA runs in the schedule.

In designing an anytime search that finds gradually improving CLM_{dn} with increased time, based on a statistically exponential model, SSA_{AT-ID} controls objective targets by iterative deepening in order for the overhead in the last SSA run to dominate the total overheads of all previous runs. The last objective target is the CGM_{dn} of the problem.

For both CSA_{AT-ID} and $CSAGA_{AT-ID}$, we have proved their optimality and show that they have the same order of magnitude in terms of completion time as that of the original SSA with an optimal schedule.

Experimental results on discrete and mixed-integer benchmarks show the improvements of CSA_{AT-ID} and $CSAGA_{AT-ID}$ over SQP. Even though SQP can quickly solve continuous benchmarks, discretization of its continuous solutions may lead to either infeasible discrete solutions or worse solutions. Further, SQP cannot be used to solve problems whose derivatives are difficult to evaluate or are unavailable.

Experimental results on a standard MINLP benchmark suite show that CSA_{AT-ID} and $CSAGA_{AT-ID}$ can improve solution qualities over branch and bound methods. But branch and bound methods are better than CSA_{AT-ID} and $CSAGA_{AT-ID}$ in terms of solution times. Similar to SQP, branch and bound methods also cannot be apply to solve problems whose functions are not differentiable.

Chapter 6

Conclusions

In this thesis, we have studied *optimal schedules* for controlling multiple runs of SSAs in order to solve general constrained NLPs in discrete, continuous and mixed-integer space. The NLPs we study are general in the sense that their functions may not be differentiable or convex.

We have proposed a general search framework for solving constrained NLPs. The framework is based on the theory of discrete constrained optimization using Lagrange multipliers. The main result of the theory gives a first-order necessary and sufficient condition for constrained local minimization in discrete space that shows a one-to-one correspondence between CLM_{dn} and SP_{dn}. To implement this theory, we propose a general search framework that performs descents of L_d in the original-variable subspace and ascents of L_d in the Lagrange-multiplier subspace in order to reach equilibrium at SP_{dn}. The general framework unifies existing methods, such as DLM and CSA. New constrained algorithms, CGA and CSAGA, were derived from the framework by incorporating genetic algorithms in its search mechanisms.

All these constrained algorithms are stochastic search algorithms (SSAs) that succeed in finding a solution of desired quality with a reachability probability less than one in one run

when the number of probes is finite. To enhance the probability of finding a solution, we make multiple runs of an SSA in order to minimize its expected time in finding a solution of desired quality. The trade-offs between the success probability and the number of probes spent raises an important issue in using SSAs; namely, the scheduling of multiple runs of an SSA. In this thesis, we have proposed the theory of SSAs that studies the optimal scheduling of SSAs. The theory proposes to use iterative deepening in order to derive the optimal schedules dynamically. We then prove the optimality of the schedules. We have also extended the theory to the scheduling of multiple runs of SSAs in parallel. We have proved the NP-completeness of the parallel scheduling problem, proved the performance limitations of different classes of parallel schedules, and proposed approximate schedules with iterative deepening that are optimal for the class of synchronous parallel schedules. Our theory is general and has been applied to CSA, CGA and CSAGA in this thesis.

Based on the optimal schedules of SSAs, we have proposed anytime search schedules that generate solutions of improved quality as more time is used. Such an algorithm is desirable when solving large complex NLPs and in real-time applications in which suboptimal solutions are acceptable if they can be found within reasonable time. Based on an exponential model relating objective quality targets and expected search times, we have proposed a linear schedule of successively improving quality levels in such a way that the time used to find a solution with the best quality dominates the overall search time. Our proposed anytime schedule is optimal in the sense that the time it takes to find feasible solutions of all quality levels is of the same order of magnitude as that taken by the original SSA with an optimal schedule to find a CGM_{dn}. Experimental results of our proposed anytime schedule on standard engineering benchmarks have shown the wide applicability, adaptability in choosing suitable schedules, and robustness in generating anytime solutions with improved quality.

In short, our general search framework provides a unified way for solving nonlinear constrained nonlinear optimization problems.

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-unix.mcs.anl.gov/~ more/cops/.

[8] http://www.abo.fi/~ twesterl/.

[9] http://www.caam.rice.edu/~ bixby/miplib/miplib.html.

[10] http://www.coe.uncc.edu/~ zbyszek/evol-systems.html.

[11] http://www.cs.cas.cz/vvvvedci/luksan/test.html.

[12] http://www.cse.clrc.ac.uk/Activity/CUTE/.

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

[14] http://www.gams.com/modlib/.

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

[16] http://www.maths.dundee.ac.uk/~ sleyffer.

[17] http://www.maths.dundee.ac.uk/~ sleyffer/MINLP_TP/AMPL/index.html.

[18] http://www.maths.dundee.ac.uk/~ sleyffer/MINLP_TP/SIF/index.html.

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

[20] http://www.orfe.princeton.edu/~ loqo/.

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

[22] http://www.sor.princeton.edu/~ rvdb/ampl/nlmodels/.

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

[24] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA., 1974.

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

[26] 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.

[27] 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.

[28] F. Aluffi-Pentini, V. Parisi, and F. Zirilli. Global optimization and stochastic differential equations. *Journal of Optimization Theory and Applications*, 47(1):1–16, September 1985.

[29] 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.

[30] N. Anderson and G. Walsh. A graphical method for a class of Branin trajectories. *Journal of Optimization Theory and Applications*, 49(3):367–374, June 1986.

[31] 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.

[32] 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.

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

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

[35] 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.

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

[37] S. Baluja and W. T. Scherer. Local optimization using simulated annealing. *IEEE Systems, Man, and Cybernetics Conference Proceedings*, pages 583–588, 10 1992.

[38] W. Baritompa. Accelerations for a variety of global optimization methods. *Journal of Global Optimization*, 4:37–45, 1994.

[39] W. Baritompa and A. Cutler. Accelerations for global optimization covering methods using second derivatives. *Journal of Global Optimization*, 4:329–341, 1994.

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

[41] 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.

[42] 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.

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

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

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

[46] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995.

[47] 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.

[48] I. Bongartz, A. R. Conn, N. Gould, and Ph. L. Toint. Cute: Constrained and unconstrained testing environment. *ACM Trans. on Mathematical Software*, 21(1):123–160, 1995.

[49] 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.

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

[51] 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.

[52] B. C. Cetin, J. Barben, and J. W. Burdick. Terminal repeller unconstrained subenergy tunneling (TRUST) for fast global optimization. *Journal of Optimization Theory and Applications*, 77, April 1993.

[53] 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.

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

[55] S. A. Cook. The complexity of theorem-proving procedures. *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158, 1971.

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

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

[58] I. Diener and R. Schaback. An extended continuous Newton method. *Journal of Optimization Theory and Applications*, 67(1):57–77, October 1990.

[59] Elizabeth D. Dolan and Jorge J. More. Benchmarking optimization software with COPS. *Technical Report ANL/MCS-246, Mathematics and Computer Science Division, Argonne National Laboratory*, 2000.

[60] Elizabeth D. Dolan and Jorge J. More. Benchmarking optimization software with performance profiles. *Preprint ANL/MCS-P861-1200, Mathematics and Computer Science Division, Argonne National Laboratory*, 2001.

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

[62] 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.

[63] 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.

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

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

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

[67] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

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

[69] R. P. Ge and Y. F. Qin. A class of filled functions for finding global minimizers of a function of several variables. *Journal of Optimization Theory and Applications*, 54(2):241–252, 1987.

[70] 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.

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

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

[73] B. Gidas. Non-stationary Markov chains and convergence of the annealing algorithm. *J. Statist. Phys.*, 39:73–131, 1985.

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

[75] 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.

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

[77] 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.

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

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

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

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

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

[83] 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.

[84] 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.

[85] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

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

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

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

[89] 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.

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

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

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

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

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

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

[96] I. K. Jeong and J. J. Lee. Adaptive simulated annealing genetic algorithm for system identification. *Eng. Applic. Artificial Intell.*, 9(5):523–532, 1996.

[97] 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.

[98] 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.

[99] A. H. G. Rinnooy Kan. *Machine Scheduling Problems: Classification, Complexity and Computations*. Nijhoff, The Hague, 1976.

[100] R. B. Kearfott. On proving existence of feasible points in equality constrained optimization problems. *Mathematical Programming*, 83:89–100, 1998.

[101] R. R. Kearfott. A review of techniques in the verified solution of constrained global optimization problems. In *Applications of Interval Computations, R. B. Kearfott and V. Kreinovich (Eds.), Kluwer Academic Publishers, Dordrecht, The Netherlands*, pages 23–60, 1996.

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

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

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

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

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

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

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

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

[110] L. Luksan and J. Vlcek. Sparse and partially separable test problems for unconstrained and equality constrained optimization. *Technical Report No. 767, Institute of Computer Science, Academy of Science of the Czech Republic*, 1999.

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

[112] O. C. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. *Technical Report CS/E 94-016, Oregon Graduate Institute*, 1994.

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

[114] 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.

[115] 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.

[116] 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.

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

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

[119] 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.

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

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

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

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

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

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

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

[135] S. Schäffler and H. Warsitz. A trajectory-following method for unconstrained optimization. *Journal of Optimization Theory and Applications*, 67(1):133–140, October 1990.

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

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

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

[139] 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.

[140] 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.

[141] Y. Shang and B. W. Wah. A discrete Lagrangian based global search method for solving satisfiability problems. *J. of Global Optimization*, 12(1):61–99, January 1998.

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

[143] J. A. Snyman and L. P. Fatti. A multi-start global minimization algorithm with dynamic search trajectories. *Journal of Optimization Theory and Applications*, 54(1):121–141, July 1987.

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

[145] 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.

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

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

[148] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application.* Krieger Publishing Company, 1989.

[149] E. G. Sturua and S. K. Zavriev. A trajectory algorithm based on the gradient method I. the search on the quasioptimal trajectories. *Journal of Global Optimization*, 1991(4):375–388, 1991.

[150] H. Szu and R. Hartley. Fast simulated annealing. *Phys. Lett. A*, 122(3-4):157–162, 1987.

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

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

[153] A. Trouve. Cycle decomposition and simulated annealing. *SIAM Journal on Control and Optimization*, 34(3):966–986, 1996.

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

[155] 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.

[156] 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.

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

[158] 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.

[159] B. W. Wah and Y. X. Chen. Constrained genetic algorithms and their applications in nonlinear constrained optimization. In *Proc. Int'l Conf. on Tools with Artificial Intelligence*, pages 286–293. IEEE, November 2000.

[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 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.

[162] 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.

[163] 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.

[164] T. Wang. *Global Optimization for Constrained Nonlinear Programming*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, December 2000.

[165] 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.

[166] Y. Wang, W. Yan, and G. Zhang. Adaptive simulated annealing for optimal design of electromagnetic devices. *IEEE Trans. on Magnetics*, 32(3):1214–1217, 1996.

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

[168] 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*.

[169] 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.

[170] 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.

[171] Z. Wu and B. W. Wah. Solving hard satisfiability problems: A unified algorithm based on discrete Lagrange multipliers. In *Proc. Int'l Conf. on Tools with Artificial Intelligence*, pages 210–217. IEEE, November 1999.

[172] Z. Wu and B. W. Wah. Trap escaping strategies in discrete Lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *Proc. 1999 National Conf. on Artificial Intelligence*, pages 673–678. AAAI, July 1999.

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

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

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

Vita

Yixin Chen received his B.S. degree from the Department of Computer Science, University of Science and Technology of China in 1999. He is expecting an M.S. degree in Computer Science from the University of Illinois at Urbana-Champaign in May, 2001.

His interests include nonlinear optimization, stochastic system modeling and analysis, neural networks, computer networks, and algorithm design and analysis, especially the excursions into parallel computations.