THE THEORY AND APPLICATIONS OF DISCRETE CONSTRAINED OPTIMIZATION USING LAGRANGE MULTIPLIERS

BY

ZHE WU

B.E., University of Science & Technology of China, 1996
M.S., University of Illinois at Urbana-Champaign, 1998

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2000

Urbana, Illinois

Abstract

In this thesis, we present a new theory of *discrete constrained optimization using Lagrange multipliers* and an associated *first-order search procedure* (DLM) to solve general constrained optimization problems in discrete, continuous and mixed-integer space. The constrained problems are general in the sense that they do not assume the differentiability or convexity of functions. Our proposed theory and methods are targeted at discrete problems and can be extended to continuous and mixed-integer problems by coding continuous variables using a floating-point representation (discretization). We have characterized the errors incurred due to such discretization and have proved that there exists upper bounds on the errors. Hence, continuous and mixed-integer constrained problems, as well as discrete ones, can be handled by DLM in a unified way with bounded errors.

Starting from new definitions on *discrete neighborhoods, constrained local minima* in discrete space, and new *generalized* augmented Lagrangian function, we have developed new *discrete-space first-order necessary and sufficient conditions* that are able to characterize all constrained local minima in discrete space. Our proposed first-order conditions show a one-to-one correspondence between a discrete-space constrained local minimum, a discrete-space saddle point, and a solution to the first-order conditions. They are important because they allow us to transform the difficult problem of looking for constrained local minima in the original-variable space to the easier problem of looking for saddle points in the discrete Lagrangian space. They provide a solid foundation for DLM to solve general constrained problems that cannot be achieved in the conventional theory of Lagrange-multipliers for solving continuous constrained nonlinear programming problems.

Finally, we demonstrate the efficiency and effectiveness of our proposed theory and methods. DLM is able to solve systematically general discrete, continuous and mixed-integer constrained benchmarks, which is a task not achieved by previous methods. DLM has found

better multiplierless filter-bank designs that improve over all of Johnston's benchmark designs using a maximum of three to six ONE bits in each filter coefficient instead of using floating-point representations. Finally, DLM has found efficiently new solutions for satisfiability problems that were not possible by existing local- and global search techniques.

To Dong Lin, as always my loving wife,
my parents Fuliang Wu, and Huiqin Zhao
and my brother Yi Wu.

Acknowledgments

First of all, I thank my advisor, Professor Benjamin W. Wah, for his large amount of patience and guidance. I had much to learn when I started my graduate study, and I am grateful to him for pointing me the way of doing research, namely, working hard and keeping a clear mind. After four years of research and study, I still have much more to learn; however, thanks to Professor Wah I now have the necessary tools to attack any new problem.

I also would like to thank all the members in our research group, Yi Shang, Jeffrey Monks, Tao Wang, Dong Lin, Xiao Su, Minglun Qian, Peter Wahle, Pohao Chang and Chienwei Li, for providing crucial comments on the work and for providing a congenial environment for me to work in.

I would like to acknowledge the support of National Science Foundation Grant MIP 96-32316 and National Aeronautics and Space Administration Grant NAG 1-613, without which this work would not have been possible.

Last, but not the least, I thank my wife for loving me, keeping my life colorful, and discussing various difficult problems with me. I thank my parents and my brother for giving me a confident and persistent personality and their ever-lasting confidence in me. I also want to thank all my friends who have supported me in this endeavor.

Table of Contents

Chapter

List of Tables

List of Figures

Chapter 1

Introduction

Many applications in engineering, decision science and operations research can be formulated as nonlinear, nonconvex, optimization problems, whose objective and constraint functions are nonlinear. Typical applications include signal processing, structural optimization, neural-network design, VLSI design, database design and processing, nuclear power-plant design and operation, mechanical engineering, physical sciences, and chemical-process control [144, 193, 17, 47, 56].

A general goal in solving nonlinear constrained optimization problems is to find feasible solutions that satisfy all the constraints. This is not an easy task because nonlinear constrained optimization problems are normally NP-hard [63]. In practice, the difficulties in solving a nonlinear constrained optimization problem arise from the challenge of searching a huge variable space in order to locate feasible points with desirable solution quality.

To achieve this goal, we propose new theories and search methods in this thesis. In this chapter, we start with the mathematical characterization of the types of optimization problems addressed in this thesis. We then introduce some fundamental concepts that are important not only to our proposed theory but also to general constrained optimization.

1.1 Problem Formulation

Conceptually, a generic optimization problem consists of three components [33]:

- an objective function to be minimized (or maximized),
- a set of unknowns or variables that affect the value of the objective function and,
- a set of constraints that allow the unknowns to take on certain values but exclude others.

Solving an optimization problem, therefore, amounts to *finding values of variables that minimize or maximize the objective function while satisfying the constraints.*

Consider, for example, a filter-bank design problem in signal processing. The main objective in designing a filter bank is to minimize its reconstruction error, while considering other metrics like transition bandwidth, energies and ripples. The variables of such a design problem are the coefficients of a filter-bank, and solving it involves the search of a space consisting of these coefficients in order to minimize the reconstruction error while at the same time satisfy constraints on bandwidth, energies and ripples.

Mathematically, the general form of a constrained nonlinear programming problem (constrained NLP) can be expressed as:

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g(x) \leq 0 \qquad x = (x_1, x_2, \ldots, x_n) \\ &&& h(x) = 0, \end{aligned} \tag{1.1}$$

where $f(x)$ is the objective function, and $h(x)$ and $g(x)$ are, respectively, equality and inequality constraint functions. Without loss of generality, we consider only minimization problems here, knowing that maximization problems can be converted to minimization ones

by negating their objectives. Note that objective function $f(x)$ in (1.1) should be lower bounded in order to make the minimization meaningful. Moreover, in their general form, $f(x)$, $g(x)$ and $h(x)$ are nonlinear functions that are either continuous or discrete, convex or non-convex, and analytic or procedural (evaluated through simulations). These assumptions imply that the functions of a constrained optimization problem are not necessarily differentiable or even continuous in this thesis.

Depending on the values x in (1.1) takes, this research addresses three classes of problems:

- discrete constrained NLPs whose x takes discrete values;
- continuous constrained NLPs whose x is continuous;
- mixed-integer continuous NLPs in which some variables are continuous and others are discrete.

To solve problems in the first class, we develop a new theory of discrete constrained optimization using Lagrange multipliers and an associated first-order search method (DLM). The major results are summarized in the discrete-space first-order necessary and sufficient conditions that are able to characterize all solutions to (1.1). By studying errors of using a floating-point representation to code continuous variables in continuous or mixed-integer constrained NLPs, we are then able to apply the theory and search method proposed to solve problems in continuous and mixed-integer constrained optimization.

The proposed theory of discrete constrained optimization using Lagrange multipliers is inspired by existing continuous Lagrange-multiplier theory and methods. However, they are derived from new foundations that differ significantly in terms of theoretical results. In order to clarify the fundamental differences between our proposed theory and the existing Lagrangian theory for solving *continuous* constrained NLPs, we compare them in a step-by-step fashion in our development of the theory in Chapter 3.

In the following section, we introduce some basic concepts.

1.2 Basic Definitions

For a discrete constrained NLP formulated in (1.1), its possible solutions are local minima satisfying all the constraints. To formally characterize the solutions to be found, we state the concepts of feasible points, neighborhoods and constrained local minima in discrete space.

Definition 1.1 Point $x \in X$ is a *feasible point* if and only if $h(x) = 0$ and $g(x) \leq 0$.

Definition 1.2 $\mathcal{N}_{dn}(x)$ [1], the *neighborhood* of point x in discrete space, is a finite set of user-defined points $\{x' \in X\}$ such that x' is reachable from x in one step and that $x' \in \mathcal{N}_{dn}(x) \Longleftrightarrow x \in \mathcal{N}_{dn}(x')$.

For example, in $\{0,1\}^n$ space in which variables are represented by a vector of binary elements, each in the set $\{0,1\}$, the neighborhood of x can be defined as points whose Hamming distance between x and y is less than 2. In modulo-integer space in which variables are vectors of integer elements, each in the set $\{0,1,\ldots,k-1\}$, x and y can be defined as neighbors if

$$\mathrm{mod}(y_1 - x_1, k) + \ldots + \mathrm{mod}(y_n - x_n, k) \leq j \tag{1.2}$$

holds, where j is a positive integer.

In contrast, in continuous space the neighborhood of point x, $\mathcal{N}_{cn}(x)$, is defined to be all points x' satisfying

$$\|x - x'\| < \epsilon \tag{1.3}$$

where ϵ is a positive real number that approaches zero asymptotically. One can view ϵ as the radius of a high-dimensional sphere. A fundamental difference between continuous and discrete neighborhoods is that a discrete neighborhood has a *finite* number of elements,

whereas a continuous neighborhood has an *infinite* number of points for any positive ϵ. The finite nature of neighborhood points in $\mathcal{N}_{dn}$ is critical in proving the theory of discrete constrained optimization using Lagrange multipliers.

Definition 1.3 Point x_l is a *discrete constrained local minimum* (CLM) if and only if it satisfies the following two properties: a) x_l is a feasible point, implying that x_l satisfies all the constraints $h(x_l) = 0$ and $g(x_l) \leq 0$, and b) $f(x_l) \leq f(x')$ for all $x' \in \mathcal{N}_{dn}(x_l)$, where x' is feasible. A special case in which x_l is a CLM is when x_l is feasible and all neighboring points of x_l are infeasible.

Based on these definitions, there are two distinct features of CLM in discrete space. First, the set of CLM in discrete space for a problem is not unique because it depends on the definition of $\mathcal{N}_{dn}(x)$; that is, point x_l may be a CLM to one $\mathcal{N}_{dn}(x)$ but may not be for another. The choice of $\mathcal{N}_{dn}(x)$, however, does not affect the validity of a search as long as a consistent definition is used throughout, although it may affect the time to find a CLM. In general, a good choice may include nearby discrete points for doing local search as well as "far-away" points for exploring larger regions in a search space. Second, the verification of a point in discrete space to be a CLM can be done by comparing its objective value against the objective value of a *finite* number of its neighbors. This feature allows the search of a descent direction to be done by enumeration rather than by differentiation.

In contrast, point x is a *constrained local minimum* [128] in continuous space if and only if, for any feasible $x' \in \mathcal{N}_{cn}(x)$, $f(x') \geq f(x)$ holds true. Unlike the definition of $\mathcal{N}_{dn}(x)$ in discrete space, $\mathcal{N}_{cn}(x)$ is well defined and unique. Thus, in continuous space points that are *constrained local minima* are well defined, although it is impossible to verify whether a given point is a CLM by enumeration since $\mathcal{N}_{cn}(x)$ is infinite in size.

To avoid confusion of CLM in discrete and continuous spaces, we denote, respectively, CLM_{dn} to be a CLM in discrete space and CLM_{cn} to be a CLM in continuous space. The difference lies in their neighborhoods, and this convention will be used throughout the thesis.

Definition 1.4 Point x is a *discrete constrained global minimum* (CGM_{dn}) if and only if: a) x is a feasible point, implying that x satisfies all the constraints, and b) for any other feasible x', $f(x') \geq f(x)$. The set of all CGM_{dn} is denoted by X_{opt}. Unlike CLM_{dn}, whether a point is a CGM_{dn} is neighborhood independent and well defined.

To characterize different algorithms for solving constrained optimization problems, some frequently used and closely related terms are described and compared next:

- *Local-search, global-search and global optimization.* *Local-search* methods rely on information from local probes to generate candidate trial points and advance their search trajectories [7, 128, 103]. Obviously, local-search methods may be trapped and confined in a small local region in their search space. *Global-search* methods, on the other hand, have techniques for escaping from the attraction of local minima or constrained local minima in their search space [77, 15, 168, 165, 94, 191, 173, 201], thereby having a better chance to find high-quality solutions. Last, *global-optimization* methods can find global optima for unconstrained problems or constrained global optima for constrained problems when the methods stop [118, 132, 19, 147, 30, 232, 213, 214].

- *Deterministic versus stochastic search methods.* An iterative search procedure is deterministic [106, 235] if each probe is generated deterministically by the procedure. Otherwise, it is called a probabilistic or stochastic procedure.

- *Complete versus incomplete search methods.* Complete methods have mechanisms to explore exhaustively the whole search space. Consequently, given a finite search space,

complete methods are able to find globally optimal solutions (or constrained global optima) in finite time. Moreover, if a constrained problem has no feasible solution, complete methods can prove infeasibility. On the other hand, incomplete methods have no guarantee to find optimal solutions and cannot prove infeasibility. In finite time, incomplete methods may be able to find a feasible solution if one exists.

- *Reachability versus asymptotic convergence.* An iterative search procedure is said to have reachability [6, 235] if the probability of its search trajectory hitting a global optimum (or a constrained global optimum for constrained problems) converges to one as time goes to infinity. In contrast, an iterative search procedure is said to have asymptotic convergence [6, 235] if the probability for its last probe hitting a global optimum (or a constrained global optimum for constrained problems) converges to one as time goes to infinity. Reachability is weaker than asymptotic convergence as it only requires hitting a constrained global minimum sometime during a search. A pure random search is an example of global optimization with reachability. In contrast, asymptotic convergence requires the current probe to converge to a constrained global minimum with probability one. Consequently, the probability of hitting a global solution increases as a search progresses, making it more likely to find the globally optimal solution (or constrained global optimum) than an algorithm with reachability alone. Note that algorithms with either reachability or asymptotic convergence are stochastic search methods and are incomplete when given finite time because global solutions are not guaranteed to be found in finite time.

Example 1.1 To illustrate the concepts of discrete neighborhoods and constrained local minima, consider the following one-dimensional nonlinear discrete constrained optimization

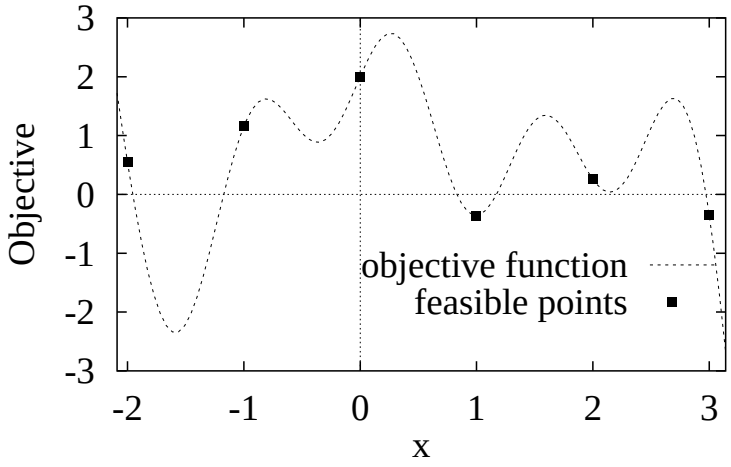


Figure 1.1: The objective function and feasible solution points defined in (1.4)

problem with constraints satisfied at integer points between -2 and 3.

$$\begin{aligned} \text{minimize} \quad & f(x) = 2 - 0.4x - 2.0x^2 + 0.75x^3 + 0.4x^4 - 0.15x^5 + \sin(5x) \\ \text{subject to} \quad & h(x) = 0 \\ \text{where} \quad & h(x) = \begin{cases} \sin(\pi x) & \text{if } -2 \le x \le 3 \\ 1 & \text{otherwise} \end{cases} \end{aligned} \tag{1.4}$$

and x is a discrete variable. Figure 1.1 plots the objective function.

By choosing the neighborhood of x to be $\{x-1, x+1\}$, $x = -2, 1, 3$ are all constrained local minima in (1.4), since $x = -3$ is infeasible, $f(-1) > f(-2)$, $f(0) > f(1)$, $f(2) > f(1)$, $f(2) > f(3)$, and $x = 4$ is infeasible. Out of the six points in the feasible region, the global minimum is at $x = 1$ where $f(1) = -0.359$. In this simple example, its CLM_{dn} and CGM_{dn} can be found by exhaustive enumeration. ■

1.3 Research Goals

The overall goal of this thesis is to solve a general class of constrained NLPs whose functions are not necessarily differentiable or convex. This goal is decomposed into four sub-goals.

Our *first sub-goal* entails the solution of discrete constrained NLPs. We develop the theory of discrete constrained optimization using Lagrange multipliers and an associated efficient discrete-space local-search method (DLM). Our proposed theory is able to characterize all CLM_{dn} of a discrete constrained NLP without differentiability or convexity assumptions.

Our *second sub-goal* is to extend our proposed theory and DLM to continuous and mixed-integer constrained NLPs. We explore floating-point representations (or discretization) of continuous variables in continuous and mixed-integer constrained NLPs in order to transform them into discrete constrained NLPs. Such transformations allow constrained NLPs in continuous and mixed-integer space to be handled in a unified way as discrete constrained NLPs by DLM.

Our *third sub-goal* is to investigate the errors introduced due to floating-point representations of continuous variables and determine analytically the upper bounds on the worst-case errors introduced by discretization.

Finally, we apply DLM to solve some constrained NLP benchmarks, designs of multipliers QMF banks, and hard satisfiability problems. We like to demonstrate that DLM can find new solutions and designs that were not possible by other existing procedures.

We have focused in this research on finding constrained local minima to (1.1) and have developed conditions to characterize such solutions. However, the theory is general and can be extended to characterize constrained global minima. Such extensions have been carried out in the development of constrained simulated annealing [213, 211].

1.4 Outline of This Thesis

In Chapter 2, we survey existing work on three classes of constrained NLPs: a) discrete constrained NLPs, b) continuous constrained NLPs, and c) mixed-integer constrained NLPs. A floating point representation, which is closely related to the implementation of numerical algorithms for solving constrained NLPs with continuous variables, is explored carefully in the beginning of Chapter 3. The analysis reveals that, by coding continuous variables in a continuous or mixed-integer constrained NLP using a floating-point representation, the original problem is transformed into a discrete constrained NLP. We prove a theorem on the upper bound between the CGM_{cn} of the original constrained NLP and the CGM_{dn} of the corresponding discrete constrained NLP. Next, we present the theory of discrete constrained optimization using Lagrange multipliers. We begin by defining a set of new concepts, like direction of maximum potential drop (DMPD) and discrete-space saddle points. We then introduce a generalized discrete Lagrangian function, propose the theory of discrete constrained optimization using Lagrange multipliers, and prove the discrete-space first-order necessary and sufficient conditions. Based on these conditions, we propose an efficient first-order search method in Chapter 4 for solving discrete constrained NLPs. We also discuss in detail related issues on neighborhood search, global search and dynamic weight adaptation. Next, we demonstrate the efficiency and effectiveness of our proposed first-order search method on two real-world applications: the design of multiplierless filter banks in Chapter 5 and the solution of satisfiability problems in Chapter 6. Finally, we conclude this thesis and point out some possible directions of future work in Chapter 7.

1.5 Significance of This Research

We list in this section the main contributions of this thesis.

- *A complete theory of discrete constrained optimization using Lagrange multipliers.* Compared to previous work [207, 178, 176], we have developed a complete theory on discrete constrained optimization using Lagrange multipliers and its associated first-order search procedure for locating CLM_{dn}. Our proposed discrete-space first-order necessary and sufficient conditions can characterize all CLM_{dn} in discrete space. In contrast, traditional first-order necessary conditions in continuous Lagrangian space are only able to characterize a subset of CLM_{cn}. Our proposed theory and methods provide a systematic way to solve general discrete constrained NLPs without convexity and differentiability assumptions.

  We have shown in this research, an upper bound on the error between a CLM_{cn} for a continuous or mixed-integer constrained NLP and the closest CLM_{dn} when continuous variables are discretized in a floating-point representation. Such a characterization is important because it allows discrete, continuous, and mixed-integer NLPs to be solved in a unified way as discrete NLPs with bounded errors.

- *Designs of better multiplierless filter banks.* We have formulated the design of multiplierless QMF (quadrature mirror filter) filter banks as nonlinear discrete constrained NLPs and have applied DLM to find new designs that were not possible using existing methods. Our designs improve over all of Johnston's benchmark designs using a maximum of three to six ONE bits in each filter coefficient, instead of using floating-point representations. Moreover, our approach is general and can be applied to design other types of multiplierless filter banks [216].

- *Solving hard satisfiability problems.* We have applied DLM with global-search heuristics to find new solutions for satisfiable benchmarks that were not possible by existing local- and global search techniques. Our experimental results on the DIMACS and SATLIB

benchmarks demonstrate that DLM is robust as well as effective in solving hard SAT problems.

Chapter 2

Previous Work

In this chapter, we summarize previous work in the literature on the three classes of problems formulated in Chapter 1. Methods for solving these three classes of problems are surveyed in the first three sections of this chapter. These methods are evaluated based on four criteria: C_1) the quality of solutions these methods can achieve, their solution time and convergence behavior; C_2) the properties of these algorithms, whether they have well-defined stopping conditions and whether their algorithmic steps are related to the stopping conditions; C_3) the type of objective and constraints that these methods can handle, linear or nonlinear; and C_4) their requirements on continuity, convexity or differentiability of the objective and constraint functions. Criterion C_1 is further decomposed into three parts: *i) solution* stands for the final solution quality achieved when an algorithm stops, where solution quality can be constrained local minimum, constrained global minimum or '*heuristic*' (implying no feasibility and optimality guarantee on the final solution); *ii) convergence* stands for the convergence behavior, namely, asymptotic convergence, reachability to a global optimal solution, or '*none*' (meaning no convergence to any globally optimal solution); and *iii) time* is the solution time required to achieve the final solution, where '*finite*' means that an algorithm will locate its

final solution in finite time when given a finite space, and '*bounded*' means that an algorithm will terminate in bounded time.

By reviewing the many existing methods for solving the three classes of problems formulated in Chapter 1, we wish to find out: a) whether there are necessary and sufficient conditions to characterize constrained local minima for the three classes of constrained NLPs; b) whether there exists a unified method for solving them; and c) whether existing methods are efficient in addressing them.

2.1 Prior Methods for Solving *Discrete* Constrained NLPs

The general formulation of a discrete constrained NLP is defined in (1.1) with respect to a vector x of discrete variables. In general, a discrete constrained NLP has a finite search space, although the number of points in it is normally very huge because a search space increases exponentially with respect to the number of variables in the problem. For instance, a very small problem with 10 variables, each taking 1000 different discrete values, has as many as 10^{30} points in its whole search space. As a result, it is impossible for any algorithm to enumerate all possibilities of even small problems.

In the past, there have been four major approaches on efficient techniques to solve discrete constrained NLPs defined in (1.1): transformations into constrained 0-1 NLP problems, transformations into unconstrained problems using penalty formulations, direct solution methods, and Lagrangian relaxation. Figure 2.1 shows a classification of these methods according to the four approaches.

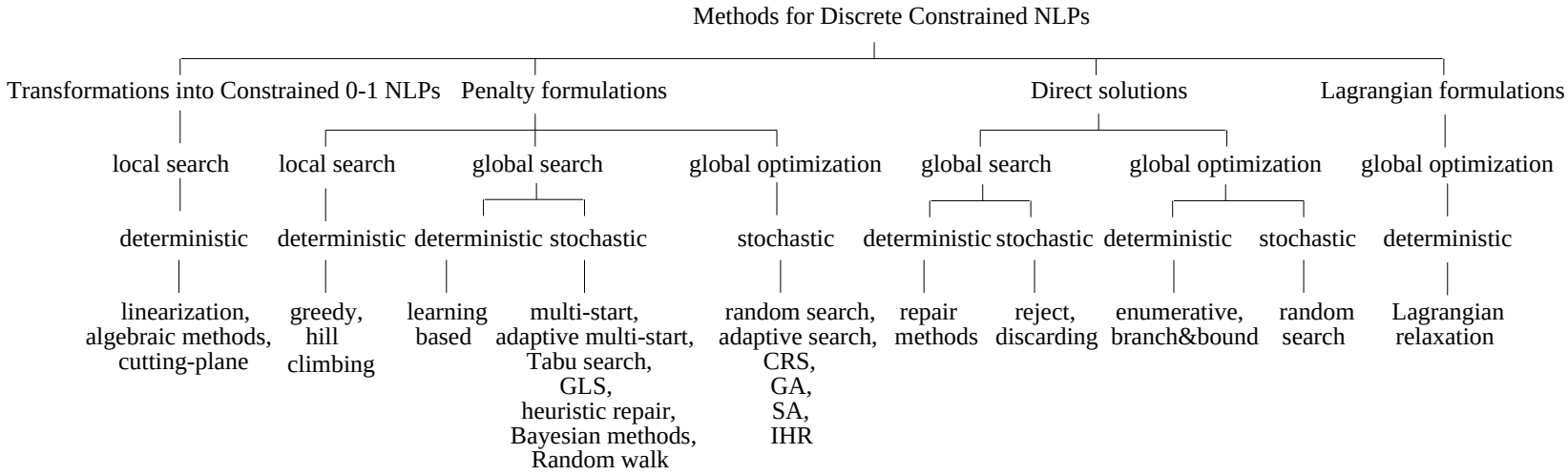


Figure 2.1: Classification of methods for solving discrete constrained NLPs

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 [92].

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 [223, 78, 79]. However, linearization often leads to large linear programs due to the many new variables and constraints introduced. In general, linearization methods will only work for problems with a few simple nonlinear terms.

Second, algebraic methods [90, 159] 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.

Third, 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.

Transformations into nonlinear constrained 0-1 problems are not helpful because existing techniques for solving these problems are very limited and all have difficulties in handling highly nonlinear objective and constraint functions. However, if a nonlinear problem can be linearized, existing linear programming methods generally have well-defined algorithmic steps and stopping conditions for locating CLM_{dn}.

2.1.2 Penalty Formulations and Methods

Penalty Formulations. A second approach transforms (1.1) into an unconstrained problem consisting of a sum of the objective and the constraints weighted by penalties before solving it by incomplete methods. A typical penalty formulation is as follows:

$$eval(x) = f(x) + p(x), \tag{2.1}$$

where $f(x)$ is the objective function and $p(x)$ is the penalty term. A widely used penalty term is:

$$p(x) = \sum_{i=1}^{n} w_i |h_i(x)|, \tag{2.2}$$

where w_i are weight coefficients to be determined.

A simple solution is to use a *static-penalty formulation* [22, 128] that sets w_i to be static large positive values. This way, a local minimum of $eval(x)$ is a constrained local minimum (CLM_{dn}), and a global minimum of $eval(x)$ is a constrained global minimum (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 constraints 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 in 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 [22, 128]. Optimality in each subproblem is, however, difficult to achieve in practice, given only finite amount of time to solve each subproblem, leading 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 the global optimality of solutions have been developed [117, 129].

Various constraint handling techniques have been developed based on dynamic-penalty formulations in [99, 113, 133, 148, 134, 76, 8, 170, 145, 169]. Besides requiring domain-specific knowledge, most of these heuristics have difficulties in finding feasible regions or in maintaining feasibility for nonlinear constraints and get stuck easily in local minima [135, 133]. Some typical constraint-handling techniques are explained next. Note that these techniques are all heuristic and were developed in an ad hoc fashion.

- One technique is to apply dynamically updated coefficients on a penalty formulation. The following formula reveals this idea:

$$eval(x) = f(x) + (cT)^{\alpha} \sum_{i}^{n} h_i^{\beta}(x), \tag{2.3}$$

  where c, α, β are constants. A typical choice of these parameters are: $c = 0.5$, and $\alpha = \beta = 2$ [113]. The critical part of this method lies in the exponentially increasing weight coefficient, $(cT)^{\alpha}$, where T is a measure of progress or time. Obviously, the weight

coefficient will be large after some time. Therefore, the penalty is indeed dynamic and eventually will be sufficiently large to force violated constraints into satisfaction. The drawback, however, is that the penalty is increasing at an exponential rate, which is too sensitive to α. In practice, it is very difficult to chose a suitable α for an unknown problem. A large α will cause the search terrain to be too rugged, making it hard to escape from local minima. Also, a large α will incur large penalties on constraints and will force a search to find only feasible solutions, while ignoring the objective.

- A second technique is based on adaptive penalties [18]. Similar to the first technique, it uses a dynamically changing weight coefficient in a penalty formulation:

$$eval(x) = f(x) + \lambda(T) \sum_{i}^{n} h_i^2(x). \tag{2.4}$$

  The difference, however, lies in the fact that $\lambda(T)$ is not simply a function of search progress. Rather, $\lambda(T)$ is dynamically updated according to constraint violations: it will be increased if many constraints are violated and stabilized if most are satisfied. A problem with this approach is that all constraints are summed up and treated with equal weights. This is counter-intuitive because different constraints may have different levels of difficulty to be satisfied and, therefore, should carry different weights.

In general, methods based on the above dynamic-penalty formulations can at best, but have no guarantee to, achieve constrained local minima (CLM_{dn}). Consider a simple problem with only one constraint function $h(x)$ and an objective $f(x)$. If a dynamic penalty-based algorithm starts from x^* and $|h(x^*)| = min_{x \in \mathcal{N}_{dn}(x^*) \cup \{x^*\}}\{|h(x)|\} > 0$ and $f(x^*) = min_{x \in \mathcal{N}_{dn}(x^*) \cup \{x^*\}}\{f(x)\}$, then regardless of how large $(cT)^\alpha$ in (2.3) or $\lambda(T)$ in (2.4) becomes, no feasible solution can be found.

Next, we review methods based on penalty formulations. These methods can be further classified into local search, global search and 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 [7, 128]. 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 (2.2) 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 have certain techniques to escape from the attraction of local minima or valleys in a search space. Typical methods include tabu search [75, 77, 15], multi-start [168, 165, 94, 191], heuristic repair methods [41], break-out strategies [139], guided local search (GLS) [201], and random walk [173].

Among all the global search methods, multi-start is the most straightforward approach to get out of local minima. The method works as follows. Assuming a search gets stuck at a local minimum and cannot find any improvement locally, the search process will be restarted from a randomly generated starting point. A major strength of multi-start is that it is very simple and efficient to implement. However, random restarts cause all valuable historical information to be lost and have little bearings in locating CLM_{dn} or CGM_{dn}.

Tabu search [75] is an effective global search method that records previously-seen solutions or points in its variable space using a so-called tabu list implemented using a simple data

structure. The solutions or points stored in a tabu list are prohibited in the future. (A trial point not in a tabu list is called an 'allowed move,' whereas a trial point inside the tabu list is called a 'prohibited move.') Note that the simplicity of a tabu list structure is very critical to the success of the algorithm because a simpler data structure is both faster to index and smaller in memory consumption. By maintaining a tabu list of previously-seen solutions or points, a search is forced to explore new regions in its search space. Therefore, local minima can be overcome and better solutions can be found.

There are many variations [75, 77] of tabu search. For example, a strict tabu search has no bound on the size of its tabu list, and a trial point is prohibited if and only if it has been visited before. A fixed tabu search, on the other hand, has a fixed-size tabu list that is updated in a FIFO manner. Both the fixed and strict tabu searches do not allow prohibited moves. Probabilistic tabu search, however, uses a large probability for allowed moves and a small probability for prohibited moves.

A tabu search is an effective and efficient global search method if it is implemented carefully. However, its effectiveness is very sensitive to the length of the tabu list, and the representation of previously-seen solutions that are placed in the tabu list. The stopping condition of tabu search methods is usually set to terminate after a desired solution is located, although the way of allowing or prohibiting a move has little bearings to the goal of finding CLM_{dn} or CGM_{dn} for constrained problems.

A guided local search (GLS) [201] is a heuristic strategy that uses problem-specific features to characterize different solution points and associates penalties with different features. If a region or point is revisited during a search, it will incur large penalties on some features, thereby forcing the search go to other regions. GLS defines the following *eval* function:

$$eval(s) = f(s) + \lambda \sum_{i=1,F} p_i \cdot I_i(s), \tag{2.5}$$

where s is a candidate solution, λ is a parameter to be determined, F is the number of features, p_i is the penalty for feature i, and I_i is an indication of whether feature i is associated with solution s. I_i is defined to be:

$$I_i(s) = \begin{cases} 1, & \text{if solution } s \text{ has feature } i, \\ 0, & \text{otherwise.} \end{cases}$$

GLS tries to find candidate solutions with better $eval(s)$ values. At the same time, p_i will be increased when the corresponding feature exhibits a local minimum. GLS has been applied successfully to some applications [201] and is efficient because each algorithmic step of decreasing $eval(x)$ meets the goal of locating problem solutions. The weakness of GLS, however, is that features are very problem specific and a poorly chosen feature may be detrimental to GLS.

A random walk [173, 172] performs greedy local descents most of the time and perturbs, occasionally, one or several variables in order to bring the search out of local traps. A probability, p, is used to govern the percentage of carrying out local descents versus performing random perturbations. Random walks have been found to be very successful in solving hard satisfiability problems, although the way of doing descents with occasional perturbations has little bearings on constraint satisfaction. The probability p and the level of perturbations are, however, difficult to tune for a new problem.

In heuristic repair methods, a network (called GENET) with a set of nodes and edges is constructed. Each variable in the original problem is represented by a group of nodes. Each node consists of a pair of a particular variable and an associated value assigned to that variable, thereby fixing a possible assignment to that variable. Two nodes are connected by an edge if and only if the two assignments decided by the two nodes violate the constraints. Note that an edge has an associated weight, or penalty, that is updated dynamically accord-

ing to constraint violations. The algorithm performs iterative repairs on a network, while trying to minimize constraint violations. It is obvious that the algorithmic steps of heuristic-repair methods actually guide a search gradually towards finding feasible solutions. A major problem with heuristic-repair methods is that a violated constraint is not necessarily caused by only two variable assignments but may be caused by multiple variables. Such scenarios cannot be coped with by networks that connect two variables at a time.

Learning-based approaches [29] attempt to predict good starting points for local search methods such as hill-climbing. The algorithm learns some relationships between starting points and solution quality of the local search method used. The method works well for simple functions but has difficulty in learning complex search terrains. For example, population-based incremental learning (PBIL) [11] incorporates the notion of a candidate solution population (used in GA) by replacing it with a probability vector. Each element in the vector represents the probability that the corresponding bit in a solution is on. During the learning process, the probability vector can be viewed as a model of the search terrain and is rederived after each generation. Note that the probabilistic model used in PBIL does not explore any inter-parameter dependency.

Mutual information maximization for input clustering (MIMIC) [25] analyzes the global structure of a variable space, uses knowledge of this structure to guide a randomized search in the variable space, and refines the estimation of the structure using new information collected. MIMIC improves PBIL by capturing a heuristically chosen set of pairwise dependencies among solution parameters. Combining optimizers with mutual information tree (COMIT) [12] further improves MIMIC by combining probabilistic models with fast local search techniques, like greedy search, in order to reduce computational costs. The problem of these algorithms is that, for NLPs with highly nonlinear constraints, the models used in

these algorithms are usually not accurate. As a result, the guidance provided by these models may be weak and does not have close bearings to the goal of locating feasible solutions.

In Bayesian methods [137, 202], the variables in an original problem are modeled by random variables. They try to find solutions by minimizing the deviation of the model from actual global solutions. The weaknesses of these methods are, again, in the difficulty in modeling accurately a search space determined by nonlinear objective and constraints functions and in the high cost of applying them to problems with more than twenty variables [196].

In general, global search methods based on penalty formulations can at best achieve 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 shall develop a systematic way to handle nonlinear constraints that can guide a search to achieve the goal of locating CLM_{dn}.

Global Optimization Based on Penalty Formulations. Given sufficiently large penalties on constraints and infinite time, 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) [118] and genetic algorithms (GA) [96] are two well-known global optimization algorithms for solving unconstrained NLPs.

Simulated annealing, a typical global optimization algorithm with asymptotic convergence, is motivated by real-world annealing processes that first heat up a system and then gradually cool it down in order to reach states with globally minimal energy. So far, SA has been applied successfully to solve unconstrained optimization problems [151, 122, 46, 4]. Its basic idea is as follows. An artificial temperature, T, is introduced to control the progress of

annealing. At the beginning of the annealing process, $T = T^0$ is large, and a starting point, x^0, in the search space is chosen. T is gradually deceased as annealing goes on. In each iteration of the process, a new trial point, x^{n+1}, in the search space is generated based on the current point, x^n. The trial point, x^{n+1}, is accepted or rejected based on the following Metropolis acceptance probability [73]:

$$A_T(x^n, x^{n+1}) = \exp(-\frac{(f(x^{n+1}) - f(x^n))^+}{T}) \tag{2.6}$$

where

$$a^+ = \begin{cases} a, & a > 0 \\ 0, & a \leq 0. \end{cases} \tag{2.7}$$

The form taken by $A_T(x^n, x^{n+1})$ reveals two facts. First, if x^{n+1} has a better (smaller) function value then x^n, then it will be accepted with $A_T(x^n, x^{n+1}) = 1$. However, if x^{n+1} has a worse (larger) function value, it will be accepted at a certain probability computed using (2.6). Second, T plays a very critical role. If, T is very large, which is the case at the beginning of the annealing process, then most of the trial points will be accepted because $(f(x^n) - f(x^{n+1}))^+/T$ is close to zero, regardless of function values of trial points. On the other hand, when T approaches zero at the end of the annealing process, only trial points with better function values will be accepted. A necessary and sufficient condition [66] for SA to converge to an unconstrained global optimum requires T to be decreased at a rate inversely proportional to a logarithmic function of time, given a sufficiently large initial temperature T^0:

$$T(t) = \frac{T^0}{log(1+t)}. \tag{2.8}$$

Many variations of SA have been proposed, like fast simulated annealing (FSA) [194], simulated annealing with extended neighborhood [230], adaptive simulated annealing [111, 222], and a combination of simulated annealing with local search heuristics [131, 13].

The advantage of SA lies in its global convergence property. Also, each step of probabilistic descents in its variable space matches the goal of locating optimal solutions. 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. Further, because SA allows up-hill moves, it is generally not as efficient as local search methods that only accept down-hill moves.

Genetic algorithm (GA) [80, 59, 133, 161, 142, 146, 93], a typical global optimization algorithm with reachability, roots itself in nature's rule of "fitness to survive." As summarized in [135, 133, 132], a genetic algorithm has five basic components: a) a genetic representation of solutions to the problem, b) a way to generate an initial population of solutions, c) a fitness function to rank different solutions, d) genetic operators to generate different genetic composition of offsprings, and e) parameters used by the algorithm.

A basic genetic algorithm [135, 133, 143] works as follows. A population of solutions, say $P(n)$ for the n^{th} generation, is maintained. ($P(0)$ is the starting population.) Each individual in $P(n)$ represents a possible solution to the problem and has a fitness rating according to a fitness function. In order to reproduce individuals for the next generation, two popular genetic operators are used. One is *mutation* that creates new individuals by changing or by perturbing the genetic composition of a single individual. The other is *crossover* that creates new individuals by combining the genetic compositions of two individuals. The

newly created individuals, together with all individuals in $P(n)$, are then rated according their fitness values, or objective values in case of unconstrained optimization problems. A new population, $P(n+1)$, is then formed by selecting from the top-fit individuals, according to the rule of "fitness to survive." After performing the above procedure for a significant number of generations, the algorithm is expected to converge to the best individual.

Some important issues for a genetic algorithm are the encoding of solutions, choosing suitable genetic operators, ranking and selection of individuals, and setting population size. A traditional and straightforward way to encode a solution is to use binary encoding, where a solution is represented by a string of binary bits (00011010010 for example). However, for many real-world problems, it is difficult and inefficient to use a binary representation. It has been found [54] that real-number encoding performs better than binary or Gray encoding for function and constraint optimization. The reason is that the topological structure of the coding space for a real-number encoding is the same as that of the original solution space. In this research, when we apply our discrete Lagrangian methods to solve continuous or mixed-integer constrained NLP problems, we also use real-number encoding (floating point representation) to represent continuous variables.

A conventional crossover operator simply swaps the genetic compositions of two individuals at a certain pivot point. The fusion operator [19], a generalized crossover operator, has been proposed in order to take into account the fitness information of parent individuals in a stochastic way. Note that the fusion operator will only reproduce one single offspring, in contrast to a traditional crossover operator that produces two offsprings. Assuming P_1 and P_2 to be the two parents selected, the child to be reproduced has a probability of $p = f(P_2)/(f(P_1) + f(P_2))$ to be P_1, and a probability of $1 - p$ to be P_2.

Together, the fitness-selection, crossover, and mutation operators of GA will guide a search gradually towards optimal solutions. However, 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.

Besides SA and GA, there are some other optimization strategies, although not as popular, that can be applied to solve discrete constrained NLPs using penalty formulations. Among those include random search, adaptive search [147, 30], controlled random search (CRS) [5, 153] and improving hit-and-run (IHR) [232].

A general sequential (or iterative) random search algorithm [232] works as follows:

$$\begin{aligned} &\text{Generate a trial point } p_k, \\ &\text{Advance the search by setting } \; x_{k+1} = \begin{cases} p_k, & \text{if } p_k \text{ is accepted} \\ x_k, & \text{if } p_k \text{ is rejected.} \end{cases} \end{aligned} \tag{2.9}$$

It was proved in [186] that under some generic conditions, a sequential random search method can asymptotically converge with probability one to a global minimum. Note that the way of generating and accepting trial points in (2.9) must not consistently exclude any region in a variable space. Otherwise, the global convergence property of the algorithm no longer holds.

Pure adaptive search (PAS) [147] is a typical sequential random search algorithm that consistently improves its solution in each iteration. Some analysis of PAS can be found in [147].

IHR combines the idea of PAS and a trial-point generator that can generate points with asymptotically uniform distribution. IHR was proved [233] to have a polynomial complexity of the number of dimensions of a class of elliptical programs.

The basic philosophy of CRS [5] resembles a contraction process in which a group of points are iteratively contracted by replacing the worst point by a better point. The replacement

is determined by a global or local technique. The global technique is for generating a new trial point based on two or more points selected from the group. In contrast, the local search technique searches in a local region around one selected point for a replacement. The iterative procedure stops when all points in the group are close enough to one another, i.e. $|f(x_a) - f(x_b)| < \epsilon_0$, where ϵ_0 is a predefined small positive value.

The issues for these random search methods are: a) that they depend heavily on a good choice of penalties in order to find feasible solutions; and b) that the trial point generator, normally based on some random distributions, has no bearing to the final goal of finding CGM_{dn} or CLM_{dn}. Therefore, these random search methods are usually not very efficient.

To summarize, transforming discrete constrained NLPs into unconstrained problems and solving them by many existing methods, although popular, has an inherent difficulty in choosing suitable penalties. If the penalties are too small, then a search might be trapped easily by deep local minima of the penalty function. On the other hand, if the penalties are too large, then the search terrain will be very rugged and feasible points may be hard to find.

2.1.3 Direct Solutions 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 [8, 9, 150] or repairing [114] methods in order to avoid infeasible points. This approach, however, has difficulty in handling nonlinear constraints whose feasible regions may be very hard to locate, leading to mostly infeasible points generated that are rejected.

The other approach is based on enumeration or randomized search techniques [105]. Enumerative algorithms [125, 221] belong to the class of complete methods that utilize branch-

and-bound techniques to find lower bounds of linearized constraints. In these algorithms, branching variables are chosen according to a rigid a priori ordering. (Better results may be obtained by choosing branching variables according to a rule that exploits the characteristics of the current subproblem.) The advantage of these enumerative methods is that optimality (CGM_{dn}) can be achieved and that the process is closely tied to the final goal of locating optimal feasible solutions. However, lower bounds found through linearized constraints may be inaccurate when constraints are highly nonlinear, and inaccurate bounds may lead to incorrect pruning and infeasible solutions when the algorithm terminates.

2.1.4 Lagrangian Relaxation

There is a class of algorithms called *Lagrangian relaxation* [72, 74, 64, 180, 16] 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:

$$\begin{aligned} z = \text{minimize}_x \quad & Cx \\ \text{subject to} \quad & Gx \leq b \quad \text{where } x \text{ is an integer vector of variables} \\ & x \geq 0 \qquad \text{and } C \text{ and } G \text{ are constant matrices} \end{aligned} \tag{2.10}$$

into the following form:

$$\begin{aligned} L(\lambda) = \text{minimize}_x \quad & (Cx + \lambda(b - Gx)) \\ \text{subject to} \quad & x \geq 0. \end{aligned} \tag{2.11}$$

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 [195] 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 [10] addressing nonlinear optimization problems. However, as pointed out in [195], 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 not for nonlinear functions.

Table 2.1 summarizes existing algorithms for solving discrete constrained NLPs based on the four evaluation criteria. Clearly, existing algorithms for solving *general* discrete constrained NLPs do not have a systematic way to handle nonlinear constraints except by utilizing penalty formulations. Our survey reveals a common difficulty with penalty formulations in choosing suitable penalties; penalties that are too small or too large are both detrimental to solution time and quality. Hence, there is usually no guarantee on final solution quality of these penalty-based methods. Further, there is no existing theory to characterize discrete-space constrained local minima (CLM_{dn}), resulting into a large disparity of different heuristics developed in the past. These observations motivate us to develop a theory to characterize constrained local minima (CLM_{dn}) and an associated search method to address systematically nonlinear constraints and guide a search to locate CLM_{dn}.

2.2 Prior Methods for Solving *Continuous* Constrained NLPs

A general continuous constrained NLP problem is defined in (1.1) in which x is a vector of continuous variables. Active research in the past three decades has produced a variety of methods [196, 107, 58, 91, 144, 133] to solve the general constrained continuous minimization problem defined in (1.1). Based on different problem formulations, existing methods can be classified into three categories: penalty formulations, direct solutions, and Lagrangian methods. Figure 2.2 classifies these methods according to their formulations.

Table 2.1: Summary of existing algorithms for solving discrete constrained NLPs. *Applicable domains* specify whether there exists limitations or special requirements on the type of problems that can be solved. The four criteria used for evaluation are described in the beginning of Chapter 2.

Problem Formulations	Search Strategies	Typical Methods	Applicable Domains	C_1			C_2	C_3	C_4
				Solution	Convergence	Time			
Constrained 0-1	Local Search	linearization	general	CLM_{dn}	none	finite	yes	restricted	no
		algebraic methods	general	CLM_{dn}	none	finite	yes	restricted	no
Programming		cutting-plane methods	general	CLM_{dn}	none	finite	yes	restricted	no
Penalty Formulations	Local Search	greedy	general	heuristic	none	bounded	yes	general	no
		hill-climbing	general	heuristic	none	bounded	yes	general	no
	Global Search	learning-based methods	general	heuristic	none	bounded	no	general	no
		multi-start	general	heuristic	none	bounded	no	general	no
		adaptive multi-start	general	heuristic	none	bounded	no	general	no
		Tabu search	general	heuristic	none	bounded	no	general	no
		GLS	general	heuristic	none	bounded	yes	general	no
		Random walk	general	heuristic	none	bounded	no	general	no
		Heuristic repair	general	heuristic	none	bounded	yes	general	no
		learning-based methods	general	heuristic	none	bounded	no	general	no
		Bayesian methods	general	heuristic	none	bounded	no	general	no
	Global Optimization	SA	general	optimal	asymptotic	infinite	yes	general	no
		GA	general	optimal	reachability	infinite	yes	general	no
		random search	general	optimal	reachability	infinite	no	general	no
		adaptive search	general	optimal	reachability	infinite	no	general	no
		CRS	general	optimal	reachability	infinite	no	general	no
		IHR	general	optimal	reachability	infinite	no	general	no
Direct Solutions	Global Search	repair methods	restricted	CLM_{dn}	none	bounded	yes	restricted	no
		reject/discarding	restricted	CLM_{dn}	none	bounded	yes	general	no
	Global Optimization	branch-and-bound	restricted	CGM_{dn}	none	finite	yes	restricted	no
		random search	restricted	CGM_{dn}	reachability	infinite	no	general	no
Lagrangian Formulations	Global Optimization	Lagrangian Relaxation	restricted	CGM_{dn}	none	finite	yes	restricted	no

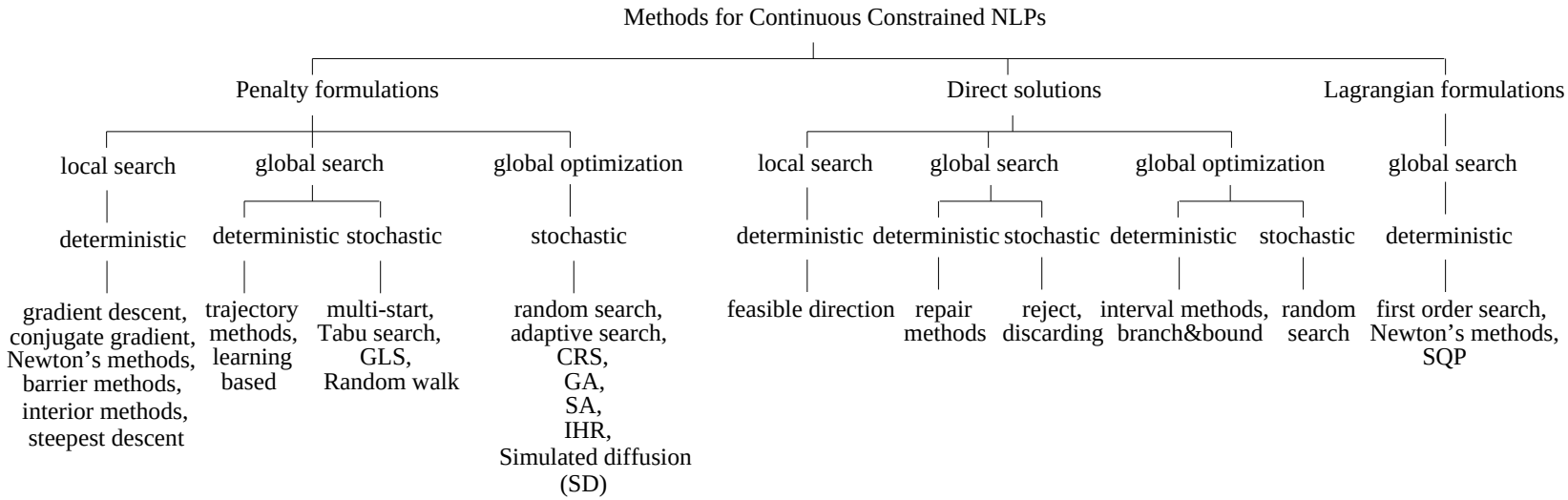


Figure 2.2: Classification of methods for solving continuous constrained NLPs

2.2.1 Penalty Formulations

In this section, we review methods for solving continuous constrained NLPs based on penalty formulations discussed in Section 2.1.2. These methods often rely on existing unconstrained optimization algorithms for finding solutions to unconstrained formulations. Depending on the strategies used to handle local minima, these penalty methods can further be classified into local search, global search and global optimization.

Local Search Methods Based on Penalty Formulations. These methods attempt to find solutions to unconstrained continuous minimization problems using only local information. Typical methods include gradient descent, conjugate gradient methods, Newton descent [128, 144, 158], and barrier or interior methods [58, 144, 224, 225]. Since they may get stuck in local minima and their solution quality is heavily dependent on their starting points, they are often used as components of other global-search or global-optimization methods.

Among unconstrained local-search methods, gradient descent is both simple and representative. Gradient descent in continuous space is analogous to greedy search in discrete space. It was derived from Taylor's theorem by using first-order approximation. Given function $f(x)$, gradient descent defines an iterative procedure:

$$x(n+1) = x(n) - \alpha \nabla f(x(n)), \tag{2.12}$$

where α is a positive real coefficient to be determined. Basically, (2.12) performs descent along the opposite gradient direction. A stopping criterion for (2.12) is measured by the norm of the gradient. If the norm value is small enough, say less than 10^{-8}, then a local minima is assumed to be reached. Obviously, each algorithmic step in gradient descent attempts to achieve the goal of reaching a local minimum.

In practice, step size α is hard to choose. A remedy is to use a dynamic step size computed as follows:

$$\alpha(n) = \text{argmin}_{c>0} f(x(n) + c\nabla f(x(n)). \tag{2.13}$$

This technique is employed in steepest descent [128].

Barrier or interior methods [58, 144, 225] try to confine a search inside feasible regions. Their major limitation is that they require a feasible starting point. In general, finding a feasible point for a given constrained NLP may be as difficult as solving the problem itself.

To summarize, existing local search methods based on penalty formulations for solving continuous constrained optimization problems can at best find a CLM_{cn} when started with large penalties. If the penalties are small or the search terrain is rugged, then local search methods may not find any feasible solution. Moreover, most unconstrained local search methods rely on gradients to provide search directions. Although gradients provide the steepest descent direction, they also unnecessarily guide a search to a local minimum too quickly.

Global Search Methods Based on Penalty Formulations. These methods improve local-search methods by utilizing some heuristics to help escape from local minima, by identifying good starting points, and by using local-search methods to find local minima. Among all the local minima found, the best one is returned as a search result. Examples of global search methods include multi-start [168, 165, 94, 191], trace or trajectory methods [203, 200], tabu search [77, 15], guided local search (GLS) [201], learning-based approaches [29], PBIL [11], MIMIC [25], COMIT [12], and random walk [173, 172]. (See Section 2.1.2 for details.) Although some of these methods were originally developed for solving discrete problems, they can be extended to solve continuous problems after some modifications. For example, random walk and GLS can be applied to solve continuous

problems when provided with a suitable trial-point generator. Assuming that the trial-point generator generates continuous points in a floating-point representation, then the algorithmic steps of random walk and GLS can actually search in a continuous space.

Although popular, existing unconstrained global search methods based on penalty formulations have no guarantee to converge to feasible solutions (CLM_{cn}) for constrained NLPs. If the penalties are too small, then no feasible solution may be found. On the other hand, penalties that are too large lead to a very rugged search space in which deep local minima may trap these methods.

Global Optimization Methods Based on Penalty Formulations. These methods can be classified into unconstrained algorithms with reachability and those with asymptotic convergence. Some typical methods in the first category include random search [147, 30, 5] and GA [80, 146, 93]. Examples of methods in the second category include covering methods [107, 55, 91, 138], simulated annealing (SA) and its variants [230, 151, 122, 232, 46, 3, 4, 80, 59, 133, 161, 142, 146, 93] discussed in Section 2.1.2. Similar to the discrete case, if penalties were not chosen properly, then these methods may not be able to find feasible solutions (CLM_{cn}) to the original constrained optimization problem.

Random search includes pure random search, pure adaptive search [147], hesitant adaptive search [30], controlled random search (CRS) and IHR [153, 2, 5]. As mentioned before, when applied to solve constrained optimization problems based on a penalty formulation, these techniques are usually not very efficient because their way of generating trial points does not usually lead to constraint satisfaction.

GA can be applied to solve continuous, mixed-integer, and discrete problems by using suitable representations and genetic operators. For example, a *Gaussian* mutation opera-

tor [8, 59] adds a random noise to the current solution point as follows:

$$x(n+1) = x(n) + N(0, \sigma) \tag{2.14}$$

where $N(0, \sigma)$ stands for a random Gaussian distribution with a zero mean and a standard deviation of σ. A possible crossover operator produces two offsprings, $x'(n+1)$ and $y'(n+1)$, by a linear combination [44] of two parents $x(n)$ and $y(n)$ using:

$$x'(n+1) = \alpha x(n) + (1-\alpha) y(n) \tag{2.15}$$

$$y'(n+1) = (1-\alpha) x(n) + \alpha y(n), \tag{2.16}$$

where α is in the range $(0, 1)$.

Obviously, using a Gaussian mutation operator and a linear-combination based crossover operator, GA can be applied easily to solve continuous and even mixed-integer problems.

Another method worth mentioning is *simulated diffusion* (SD) [67, 162]. SD utilizes the physical fact that a particle placed in a given potential and with Brownian motion is diffused into a global minimum of the given potential profile. The diffusion process can be described by the following differential equation:

$$dx = -\nabla f(x) dt + \sqrt{2T} dw, \tag{2.17}$$

where t is time, x is the physical location of the particle, $f(x)$ is a potential function in which the particle is put, dw is Gaussian random noise and T is temperature.

The first term on the right-hand side of (2.17) gives a descent direction for the particle, whereas the next term signifies the Brownian movement. When temperature T is high, $\sqrt{2T} dw$ dominates dx, and the movement is carried out in a stochastic fashion. This is quite similar to the high-temperature case of SA. After T is gradually reduced to a small value, $\nabla f(x)$ dominates dx; consequently, local descents are performed most of the time. Also,

the second term now serves as a force to bring the search process out of local minima. This low-temperature scenario is also similar to that of SA.

With a proper temperature cooling schedule [67], $P(x)$, the probability distribution of solution x, is independent of the initial value and is peaked around the global minimum of $f(x)$. Thus, given sufficiently long time, SD tends to converge to a global minimum. Compared to SA and GA, SD requires gradient information. It is, therefore, limited to continuous problems with first-order derivatives.

Existing unconstrained continuous global optimization methods based on penalty formulations for solving constrained optimization problems can converge to a CGM_{cn} when given sufficiently large penalties on constraints and infinite time. However, these conditions are hard to meet in practice. The difficulty, again, lies in the choice of suitable penalties: penalties that are too small will lead a search to a CLM_{cn} or even infeasible points, whereas penalties that are too large will lead a search to only a CLM_{cn}.

2.2.2 Direct Solutions for Solving Continuous Constrained NLPs

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

Local Search methods, like feasible-direction methods [114, 133], require a search to start from a feasible point. Each algorithmic step will keep the search within feasible regions, while trying to improve solution quality at the same time. Obviously, for problems with highly nonlinear constraints, finding a feasible starting point may be as difficult as solving the original constrained problem. Further, keeping a search inside a feasible region does not work well when feasible regions are disconnected. Note that although not very practical,

each algorithmic step of these methods does guide a search to achieve the goal of finding feasible solutions.

Global Search methods introduce techniques to overcome local minima. Typical global search methods include rejecting methods, discarding methods [156, 150], repair methods [114, 143] and preserving feasibility [134, 76]. Rejecting and discarding methods have been discussed in Section 2.1.3. Typical repair methods have some techniques to transform or repair infeasible points into feasible ones. These techniques, however, are quite limited and have difficulties in handling nonlinear constraints. As mentioned before, when feasible regions are disconnected, it will be hard for these methods to move from one feasible region to another in order to improve their solutions.

Global Optimization methods either take deterministic approaches, like interval methods, or stochastic approaches, like randomized search, to locate a CGM_{cn}. Deterministic methods [107, 20] are in general too expensive to apply for large problems except for linear problems and for problems that can linearized effectively. A typical deterministic approach divides a search space recursively into subregions, keeps promising subregions, and drops unpromising ones; examples include branch-and-bound and interval methods [91, 138]. These methods are usually computationally expensive and have difficulties in handling highly nonlinear constraints. When constraints are highly nonlinear, lower bounds cannot be found accurately, and the search space cannot be reduced effectively.

On the other hand, stochastic approaches, like random search and adaptive random search [105], probe a search space in a random fashion. (Random search has been discussed in Section 2.1.2.) Although optimality can be achieved given sufficiently long time, the chance of hitting a feasible point by random sampling is very small for a large constrained

NLP instance. Moreover, random sampling of a search space has little bearing to constraint satisfaction. Hence, random search techniques are generally not efficient for solving continuous constrained NLPs.

2.2.3 Lagrangian Formulations

Lagrangian methods generally work on equality constraints, and inequality constraints are first transformed into equivalent equality ones before applying Lagrangian methods. For instance, an inequality constraint can be transformed into an equality constraint by adding a slack variable [128] or by using the *MaxQ* method [219, 220, 212]. A general continuous equality-constrained minimization problem is formulated as follows:

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad h(x) = \quad [h_1(x), \ldots, h_m(x)]^T = 0. \end{aligned} \tag{2.18}$$

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

The *augmented Lagrangian function* in continuous space of (2.18) is defined as:

$$L_c(x, \lambda) = f(x) + \lambda^T h(x) + \frac{1}{2}||h(x)||^2, \tag{2.19}$$

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

Comparing (2.19) to penalty formulations defined in (2.1) and (2.2), there is no apparent difference. Indeed, one can view Lagrangian methods as *a special kind of penalty methods*. The major difference that distinguishes Lagrangian methods from other penalty-based methods is that CLM_{cn} in continuous Lagrangian space are characterized by first-order necessary

conditions with regularity and differentiability assumptions [128], whereas CLM_{cn} in other penalty-based methods are not.

Various continuous Lagrangian methods have been developed to locate CLM_{cn}. They are all based on the first-order necessary conditions. To state these conditions, we first introduce the concept of regular points. A point x satisfying constraints $h(x) = 0$ is said to be a *regular point* [128] if gradient vectors

$$\nabla h_1(x),\ \nabla h_2(x),\ \cdots,\ \nabla h_m(x) \tag{2.20}$$

at point x are linearly independent.

First-order necessary conditions for continuous problems [128]. Let x be a local extremum point of $f(x)$ subject to constraints $h(x) = 0$. Further, assume that x is a regular point. Then there exists $\lambda \in R^m$ such that

$$\nabla_x f(x) + \lambda^T \nabla_x h(x) = 0. \tag{2.21}$$

Based on the definition of Lagrangian function, the necessary conditions for x to be a constrained local extremum can be written as follows:

$$\begin{aligned} \nabla_x L_c(x, \lambda) &= 0, \\ \nabla_\lambda L_c(x, \lambda) &= 0. \end{aligned} \tag{2.22}$$

To ensure that the equilibrium point is a local minimum, *second-order sufficient conditions* are used to check that the solution is a strict relative minimum subject to constraints [128]. In the *second-order sufficient conditions*, second-order derivatives are required and the Hessian matrix of the Lagrangian function needs to satisfy certain conditions [128] in order to make a solution to (2.22) a strict CLM_{cn}.

Based on the first-order necessary conditions in continuous space, a number of search methods have been developed for solving constrained minimization problems. These [22, 128] include the first-order method, Newton's method, modified Newton's methods, quasi-Newton methods, and sequential quadratic programming (SQP) [24, 48, 108]. A major advantage of these methods is that solving the first-order conditions matches exactly the goal of locating a CLM_{cn}. Therefore, these algorithms are usually efficient for solving continuous constrained NLPs. Next, we briefly discuss two popular methods: first-order method and sequential quadratic programming.

The *first-order method* is a simple and straightforward method and is represented as an iterative process:

$$x^{k+1} = x^k - \alpha_k \nabla_x L_c(x^k, \lambda^k), \tag{2.23}$$

$$\lambda^{k+1} = \lambda^k + \beta_k h(x^k) \tag{2.24}$$

where α_k and β_k are step-size parameters to be determined. Intuitively, these two equations represent two counter-acting forces to resolve constraints and find constrained local minima. When any of the constraints is violated, its degree of violation is used in the second equation to increase the penalty on the unsatisfied constraint and to force it into satisfaction. When a constraint is satisfied, its associated Lagrange multiplier stops to grow. Finally, the first equation performs descents in the objective space when all the constraints are satisfied and stops at a local minimum in feasible space.

Sequential quadratic programming (SQP) is actually a generalization of Newton's method [128] for unconstrained optimization in the sense that it finds a step away from the current search point by solving a quadratic model of the original problem. In its simplest form, an SQP algorithm replaces $f(x)$ in the Lagrangian function by a quadratic approximation and

the weighted constraint functions $\lambda h(x)$ by their linear approximations:

$$q(d) = \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2_{xx} L_c(x, \lambda) d. \tag{2.25}$$

Here, step d is a solution to the following quadratic problem, assuming equality constraints only:

$$\begin{aligned} \text{minimize} \quad & q(d) \\ \text{subject to} \quad & h_i(x) + \nabla h_i(x)^T d = 0. \end{aligned} \tag{2.26}$$

The local convergence property of SQP is well defined when (x, λ) satisfies the second-order sufficient conditions [128]. That is, if point (x, λ) is sufficiently close to solution (x^*, λ^*), then the sequence generated using step size d will converge to x^* at a second-order rate.

The SQP method described here requires the computation of second-order derivatives, Hessian matrix more precisely, $\nabla^2_{xx} L(x^k, \lambda^k)$, at each step. However, in implementation, it is often replaced with BFGS approximation B_k, which is updated at each iteration. A simple update strategy defines:

$$s_k = x_{k+1} - x_k \tag{2.27}$$

$$y_k = \nabla_x L_c(x_{k+1}, \lambda_k) - \nabla_x L_c(x_k, \lambda_k) \tag{2.28}$$

and updates matrix B_k using the BFGS formula:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \tag{2.29}$$

Although SQP methods are generally efficient, they usually require functions to be differentiable and, therefore, cannot be applied to solve NLPs containing discrete variables.

In the theory of continuous Lagrange multipliers, a concept not commonly used but very critical to this research is that of saddle points (SP_{cn}). A *saddle point* (x^*, λ^*) of function

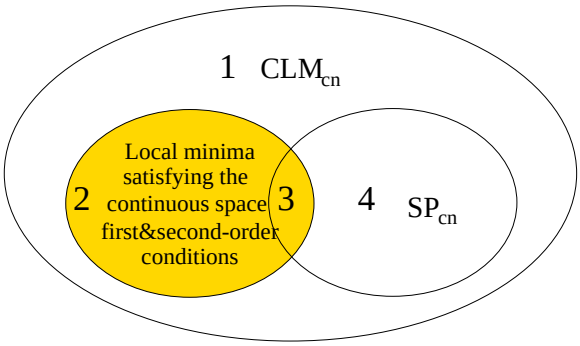


Figure 2.3: Properties of CLM_{cn} in a general continuous constrained NLP. Note that CLM_{cn} and SP_{cn} stand for continuous-space constrained local minima and continuous-space saddle points, respectively.

L_c is defined as a point that satisfies:

$$L_c(x^*, \lambda) \leq L_c(x^*, \lambda^*) \leq L_c(x, \lambda^*), \tag{2.30}$$

for all λ in $\mathcal{R}$ and all $x \in \mathcal{N}_{cn}(x^*)$, the continuous neighborhood of x^* defined in (1.3).

In general, there is no efficient method to find saddle points in continuous space. Moreover, given a point in Lagrangian space, it is very difficult to decide whether a given point (x, λ) is a saddle point unless the second-order derivative is positive or the Hessian matrix is positive definite.

Figure 2.3 summarizes the relationships among CLM_{cn} to a general continuous constrained NLP. In general, the set of all saddle points and the set of all regular points satisfying continuous-space first-order necessary and second-order sufficient conditions are only subsets of the set of CLM_{cn}. Hence, a common limitation of methods based on the first-order conditions (including the first-order, SQP and other Newton's methods) is that they cannot find CLM_{cn} that do not satisfy the first-order conditions. Moreover, such CLM_{cn} cannot be found by any existing method. The relationships in Figure 2.3 are demonstrated by the following example problems whose solution falls in each subset.

The first example constrained NLP, defined in (2.31), demonstrates that a CLM_{cn} is not necessarily a SP_{cn}. See [226] for a proof that $x^* = 0$ is a CLM_{cn}, but there does not exist

any λ to make $(0, \lambda)$ a SP_{cn}.

$$\begin{aligned} \text{minimize} \quad & f(x) = -x^2 \\ \text{subject to} \quad & h(x) = x^5 = 0. \end{aligned} \tag{2.31}$$

The example shows that $x^* = 0$ satisfies the first- and second-order conditions but is not a SP_{cn}.

The second example constrained NLP, defined in (2.32), demonstrates that a CLM_{cn}, $x^* = 0$ in this example, is not necessarily a solution to the continuous-space first-order necessary conditions.

$$\begin{aligned} \text{minimize} \quad & f(x) = (x+1)^4 \\ \text{subject to} \quad & h(x) = 0 \\ \text{where} \quad & h(x) = \begin{cases} x^4 & x < 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2.32}$$

Finally, the example constrained NLP defined in (2.33), demonstrates that a $SP_{cn}(0,0)$ for this particular problem does not satisfy the first-order necessary and second-order sufficient conditions:

$$\begin{aligned} \text{minimize} \quad & f(x) = \begin{cases} -x & x < 0 \\ 2x & \text{otherwise} \end{cases} \\ \text{subject to} \quad & h(x) = 0, \end{aligned} \tag{2.33}$$

where $h(x)$ is satisfied at and only at point 0.

However, there are also example constrained NLPs for which the three solution spaces are equal. Consider a continuous constrained problem defined as follows:

$$\begin{aligned} \text{minimize} \quad & f(x) = x^2 && (2.34) \\ \text{subject to} \quad & h(x) = 0 \\ \text{where} \quad & h(x) = 1 - x. && (2.35) \end{aligned}$$

It can be verified easily that $x = 1$ is a CLM_{cn}, a solution to first-order necessary and second-order sufficient conditions, and that $(x = 1, \lambda = 2)$ is a SP_{cn}. Therefore, the three solution spaces for this particular NLP are equal.

To characterize precisely the different regions in Figure 2.3, let C_{sp} be the saddle-point conditions (2.30), C_{reg} be the conditions for regular points [128] and the first- and second-order conditions, and C_{clm} be the conditions for constrained local minima. Obviously, points in Region 3 in Figure 2.3 satisfy condition $C_{reg} \wedge C_{sp} \wedge C_{clm}$; points in Region 2 satisfy $C_{reg} \wedge C_{clm} \wedge \neg C_{sp}$; points in Region 4 satisfy $C_{sp} \wedge C_{clm} \wedge \neg C_{reg}$; and points in Region 1 satisfy $C_{clm} \wedge \neg C_{reg} \wedge \neg C_{sp}$.

Table 2.2 summarizes existing methods for solving continuous constrained NLPs based on the four evaluation criteria mentioned in the beginning of this chapter. Obviously, direct solution methods have difficulties in addressing general nonlinear NLPs; methods based on penalty formulations may end up finding only CLM_{cn} or even infeasible solutions if penalties were chosen poorly; and methods based on Lagrangian formulations can at best locate solutions in Regions 2 and 3 of Figure 2.3.

Based on these observations, our goal in this research is to first develop a theoretical foundation that is able to handle discrete constrained NLPs as well as continuous and mixed-integer constrained NLPs in a unified way. Second, our proposed theory and methods should be able to characterize *all* CLM in the solution space and not part of them, as exemplified in

Table 2.2: Summary of existing algorithms for solving continuous constrained NLPs. *Applicable domains* specify whether there exists limitations or special requirements on the type of problems that can be solved. The four criteria used for evaluation are described in the beginning of Chapter 2.

Problem Formulations	Search Strategies	Typical Methods	Applicable Domains	C_1			C_2	C_3	C_4
				Solution	Convergence	Time			
Penlaty Formulations	Local Search	gradient descent	general	heuristic	none	bounded	yes	general	yes
		steepest gradient	general	heuristic	none	bounded	yes	general	yes
		conjugate gradient	general	heuristic	none	bounded	yes	general	yes
		Newton descent	general	heuristic	none	bounded	yes	general	yes
		barrier methods	general	heuristic	none	bounded	yes	general	yes
		interior methods	general	heuristic	none	bounded	yes	general	yes
	Global Search	multi-start	general	heuristic	none	bounded	no	general	no
		trajectory methods	general	heuristic	none	bounded	yes	general	yes
		Tabu search	general	heuristic	none	bounded	no	general	no
		GLS	general	heuristic	none	bounded	yes	general	no
		learning-based methods	general	heuristic	none	bounded	no	general	no
		Random walk	general	heuristic	none	bounded	no	general	no
	Global Optimization	SA	general	optimal	asymptotic	infinite	yes	general	no
		GA	general	optimal	reachability	infinite	yes	general	no
		random search	general	optimal	reachability	infinite	no	general	no
		adaptive search	general	optimal	reachability	infinite	no	general	no
		CRS	general	optimal	reachability	infinite	no	general	no
		IHR	general	optimal	reachability	infinite	no	general	no
		simulated diffusion	general	optimal	asymptotic	infinite	yes	general	yes
Direct Solutions	Local Search	feasible-direction methods	restricted	CLM_{cn}	none	finite	yes	restricted	yes
	Global Search	reject/discarding	restricted	CLM_{cn}	none	bounded	yes	general	no
		repair methods	restricted	CLM_{cn}	none	bounded	yes	restricted	no
		preserving feasibility	restricted	CLM_{cn}	none	bounded	yes	restricted	no
	Global Optimization	interval methods	restricted	CGM_{cn}	none	finite	yes	restricted	yes
		random search	restricted	CGM_{cn}	reachability	infinite	no	general	no
		branch and bound	restricted	CGM_{cn}	none	finite	yes	restricted	yes
Lagrangian Formulations	Global Search	first-order methods	general	CLM_{cn}	none	bounded	yes	general	yes
		Newton's methods	general	CLM_{cn}	none	bounded	yes	general	yes
		SQP	general	CLM_{cn}	none	bounded	yes	general	yes

methods based on the first- and second-order conditions in continuous Lagrangian methods (see Figure 2.3).

2.3 Prior Methods for Solving *Mixed-Integer* Constrained NLPs

A general mixed-integer constrained NLP is formulated as follows.

$$\begin{aligned} \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & h(x,y) = 0 \qquad x = (x_1, x_2, \ldots, x_{n_1}) \\ & g(x,y) \leq 0 \qquad y = (y_1, y_2, \ldots, y_{n_2}) \end{aligned} \tag{2.36}$$

where $f(x,y)$ is the objective function, $g(x,y) = [g_1(x,y), \ldots, g_k(x,y)]^T$ is a k-component vector representing inequality constraints, $h(x,y) = [h_1(x,y), \ldots, h_m(x,y)]^T$ is an m-component vector representing equality constraints, x is a vector of continuous variables, and y is a vector of discrete variables. In general, $f(x,y)$, $g(x,y)$ and $h(x,y)$ are nonlinear functions that are either continuous or discrete, convex or non-convex, and analytic or procedural (evaluated through simulations). Therefore, the functions of (2.36) are not assumed to be differentiable or even continuous.

In mixed-integer space, neighborhoods are actually a combination of discrete neighborhoods in discrete subspace and continuous neighborhoods in continuous subspace. Consequently, constrained local minima and saddle points are redefined based on such hybrid neighborhoods.

Definition 2.1 Given point (x,y) in mixed-integer space, where x represents the continuous subspace and y the discrete subspace, the neighborhood of (x,y) is defined to be:

$$\mathcal{N}_{mn}(x,y) = \mathcal{N}_{cn}(x) \ \cup \ \mathcal{N}_{dn}(y) \tag{2.37}$$

where $\mathcal{N}_{cn}$ and $\mathcal{N}_{dn}$ are, respectively, the sets of neighboring points of x in continuous subspace defined in (1.3) and the sets of neighboring points of y in discrete subspace defined in Definition 1.2.

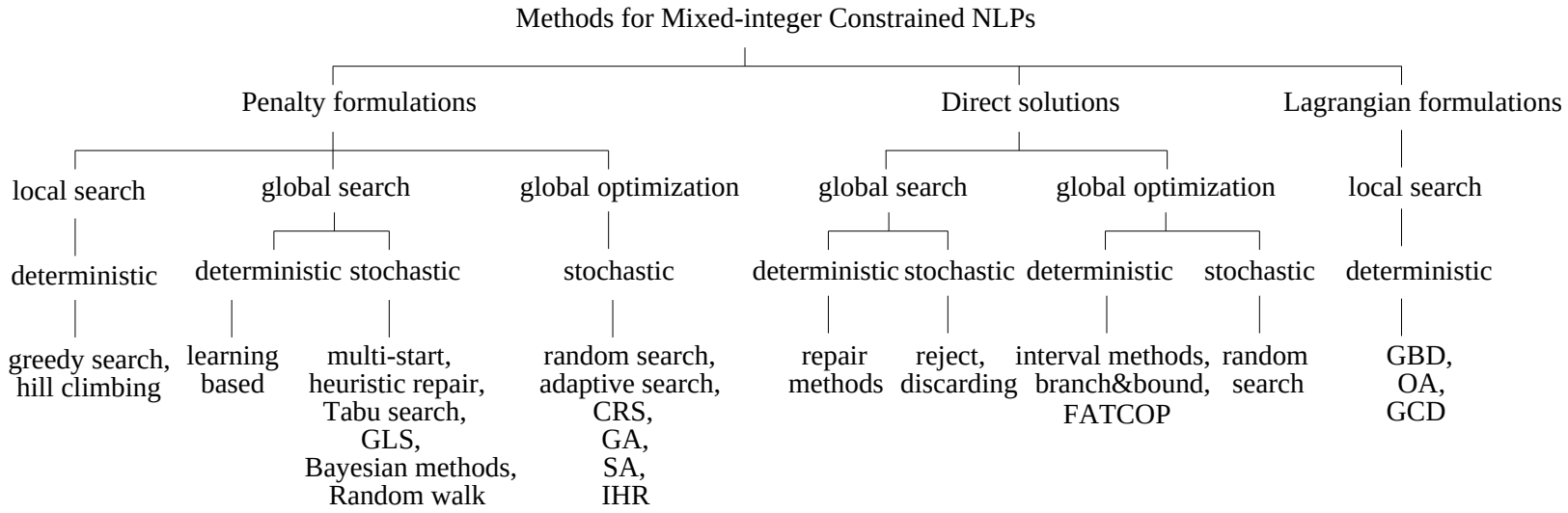


Figure 2.4: Classification of methods for solving mixed-integer constrained NLPs

The definitions of mixed-integer-space saddle points and constrained local minima are the same as those in discrete or continuous space, except that $\mathcal{N}_{mn}(x, y)$ is used instead. As before, to avoid confusion, we denote saddle points and constrained local minima in mixed-integer space by SP_{mn} and CLM_{mn}, respectively.

Existing MINLP (mixed-integer nonlinear programming) methods for solving (2.36) can be classified into three major approaches, as shown in Figure 2.4.

2.3.1 Penalty Formulations

Penalty-based methods transform a constrained MINLP into a sum of objective and constraints weighted by penalties and solve it using existing unconstrained search algorithms. (Various penalty formulations have been discussed in Section 2.2.1.) Based on a penalty formulation, many local search, global search, and global optimization techniques can be applied to solve MINLPs.

Local Search Methods Based on Penalty Formulations. These methods, like greedy descent and hill climbing [128, 155], perturb a search trajectory in the joint space of discrete and continuous variables, while trying to find improvements in the objective and decrease constraint violations at the same time. Convergence to CLM_{mn} may not be guaranteed if penalties were not chosen correctly. Similar to the discrete and continuous cases, a local search trajectory may be trapped easily by local minima in its variable space.

Global Search Methods Based on Penalty Formulations. These techniques, like GLS [201], Tabu search [75, 77], heuristic repair methods [41], learning-based approach [28, 29], Bayesian methods [136, 202, 137], multi-start [168, 165, 94, 191], and random walk [173, 172], can be applied after a constrained MINLP problem has been transformed into an unconstrained problem using penalty formulations. Detailed explanations of these

techniques can be found in Section 2.1. Similar to the discrete and continuous cases, the success of these methods depends heavily on proper choices of penalties.

Global Optimization Methods Based on Penalty Formulations. These methods include methods that can guarantee reachability, like GA [126, 231, 164, 164, 38, 31], pure random/adaptive search [147], hesitant adaptive search [30], CRS [153, 2, 5] and IHR [232], and methods that can ensure asymptotic convergence, like SA [234, 1, 52] and many of its variants [230, 151, 232, 46, 3, 4]. See Section 2.1 for detailed explanations on these methods.

For example, GA can be applied by constructing a fitness function, usually based on a weighted sum of the constraints and penalties, and by using genetic operators to minimize the penalty function. A general formulation may use a binary representation to represent real variables or a real-number encoding to represent variables whose precision and range are not known beforehand. Genetic operators used include crossovers and mutations. Crossovers take two chromosomes and create a new one based on a weighted sum. If the original variable is discrete, then the newly created chromosomes are truncated to discrete. Non-uniform mutations [132] may be used for manipulating floating-point numbers and for genes that are required to be integers. As the search aims at minimizing a penalty function, the overall function value will decrease gradually as constraint violations are suppressed in the search. The limitations of these penalty-based methods for solving constrained problems in mixed-integer space are similar to those in discrete and continuous spaces. See Sections 2.1.1 and 2.2 for further explanation.

2.3.2 Lagrangian Formulations

Methods exploiting the convexity of functions generally formulate a constrained MINLP in a Lagrangian formulation before decomposing 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. There are three classes of these algorithms.

a) *Generalized Benders Decomposition (GBD)* [56, 71, 21] is used to solve a subclass of constrained MINLPs under some convexity assumptions. For example, it requires the continuous subspace to be a nonempty and convex set and the objective and constraint functions to be convex. Its basic idea is to generate in each iteration an upper bound on the solution sought by solving a *primal* problem and a lower bound on a *master* problem. The primal problem corresponds to the original optimization problem with fixed discrete variables; the solution of which provides information on upper bounds and Lagrange multipliers associated with the equality and inequality constraints. The master problem is derived via nonlinear duality theory, making use of Lagrange multipliers obtained in the primal problem. Its solution provides information on lower bounds, as well as the set of fixed discrete variables to be used in the next primal problem. As the process iterates, it can be shown that the sequence of upper bounds are non-increasing, that the sequence of lower bounds are non-decreasing, and that the sequences converge in finite time.

b) *Outer Approximation (OA)* [51, 50] is similar to GBD except that it formulates the master problem using primal information and outer linearization. With similar restrictions as GBD, it requires the continuous subspace to be a nonempty, compact, and convex set, and the objective and constraint functions to be convex in the continuous subspace.

c) *Generalized Cross Decomposition (GCD)* [56, 97, 98, 160] iterates between two phases: Phase 1 solving the primal and dual subproblems, and Phase 2 solving the master problem. The solution of the primal subproblem in Phase 1 is similar to that in GBD and provides an upper bound on the solution of the original optimization problem and Lagrange multipliers for the dual subproblem. The dual subproblem provides a lower bound on the solution of the original problem and supplies solutions to the discrete variables of the primal subproblem.

After solving the primal and dual subproblems, a primal convergence test is applied on the discrete variables, while a dual convergence test is applied on the Lagrange multipliers. If any convergence test fails, then Phase 2 is entered in which the master problem is solved using cuts generated by the primal and dual subproblems. Similar to OA and GBD, GCD requires the objective and constraint functions to be proper convex functions.

In short, methods like GBD, OA, and GCD work well when a constrained MINLP can be decomposed into a sequence of continuous convex problems that can be solved easily. They have difficulties when the continuous subproblems are non-convex and cannot be decomposed.

2.3.3 Direct Solutions for MINLPs

Typical direct solution methods for solving MINLPs include: a) reject/discarding [109, 8, 156, 150] or repair methods [114, 143] that try to avoid infeasible points or repair infeasible points into feasible ones and that can at best find CLM_{mn}; b) random search techniques, like pure random/adaptive search [147], hesitant adaptive search [30], CRS [153, 2, 5] and IHR [232], that try to satisfy all the constraints and improve the objective by random sampling; and c) branch-and-bound based methods, like FATCOP [40, 39], that utilize linear programming relaxation, depth-first-search, and cutting planes to handle nonlinear constraints in problems that can be modeled effectively by linear relationships [40]. All these methods have difficulties in handling highly nonlinear constraints and in finding feasible solutions.

Table 2.3 summarizes the properties of existing methods for solving mixed-integer constrained NLPs based on the four evaluation criteria defined in the beginning of this chapter.

Table 2.3: Summary of existing algorithms for solving mixed-integer constrained NLPs. *Applicable domains* specify whether there exists limitations or special requirements on the type of problems that can be solved. The four criteria used for evaluation are described in the beginning of Chapter 2.

Problem Formulations	Search Strategies	Typical Methods	Applicable Domains	C_1			C_2	C_3	C_4
				Solution	Convergence	Time			
Penlaty Formulations	Local Search	greedy search	general	heuristic	none	bounded	yes	general	no
		hill-climbing	general	heuristic	none	bounded	yes	general	no
	Global Search	GLS	general	heuristic	none	bounded	yes	general	no
		Tabu search	general	heuristic	none	bounded	no	general	no
		heuristic repair	general	heuristic	none	bounded	yes	general	no
		learning-based methods	general	heuristic	none	bounded	no	general	no
		Bayesian methods	general	heuristic	none	bounded	no	general	no
		multi-start	general	heuristic	none	bounded	no	general	no
		Random walk	general	heuristic	none	bounded	no	general	no
	Global Optimization	GA	general	optimal	reachability	infinite	yes	general	no
		random search	general	optimal	reachability	infinite	no	general	no
		adaptive search	general	optimal	reachability	infinite	no	general	no
		CRS	general	optimal	reachability	infinite	no	general	no
		IHR	general	optimal	reachability	infinite	no	general	no
		SA	general	optimal	asymptotic	infinite	yes	general	no
Direct Solutions	Global Search	repair methods	restricted	CLM_{mn}	none	bounded	yes	restricted	no
		reject/discarding	restricted	CLM_{mn}	none	bounded	yes	general	no
	Global Optimization	random search	restricted	CGM_{mn}	reachability	infinite	no	general	no
		branch and bound	restricted	CGM_{mn}	none	finite	yes	restricted	no
		FATCOP	restricted	CGM_{mn}	none	finite	yes	restricted	no
Lagrangian Formulations	Local Search	GBD	restricted	CLM_{mn}	none	bounded	yes	restricted	yes
		OA	restricted	CLM_{mn}	none	bounded	yes	restricted	yes
		GCD	restricted	CLM_{mn}	none	bounded	yes	restricted	yes

2.4 Summary

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

The first class of methods try to solve constrained NLPs directly. Typical methods include enumeration, branch-and-bound, linearization, repair/discarding, keeping feasibility, and randomized search. In general, these methods cannot cope with highly nonlinear constraint functions and are only applicable to NLPs with simple nonlinear functions.

The second class of methods are based on penalty formulations. By combining both the objective and constraint functions through penalty coefficients, many unconstrained optimization techniques can be applied. Typical methods include: greedy search, gradient descent, tabu search, trajectory methods, genetic algorithm, simulated annealing, learning based methods, guided local search, and random/adaptive search. The common difficulty of these methods lies in their dependence on suitably chosen penalties. It is undesirable to use penalties that are either too large or too small. To address this issue, various dynamic penalty methods, heuristic repair, and weighting schemes have been proposed, although many of them are developed in an ad hoc fashion.

The third class of methods are based on Lagrangian formulations that can be viewed as a special kind of penalty formulations founded upon some first-order necessary conditions. Currently, these methods are only good for solving continuous constrained NLPs whose objective and constraint functions are assumed to be continuous and differentiable and whose solution points must be regular points [128]. (Lagrangian relaxation is basically developed for solving linear problems.) Moreover, as pointed out in Section 2.2.3, the set

of CLM_{cn} satisfying the continuous first-order necessary conditions is only a subset of all CLM_{cn} for general continuous constrained NLPs.

The previous work surveyed leads us to conclude that, given a general constrained NLP defined in discrete, continuous or mixed-integer space and assuming no requirement of continuity, differentiability and convexity on its objective and constraint functions, there is: a) no simple necessary and sufficient conditions to characterize its solution points that satisfy all the constraints; and b) no unified efficient systematic procedure to solve such a constrained NLP. As a result, our goal in this research is to develop a complete theory and efficient methods that can address in a unified fashion NLPs with: a) nonlinear constraints; b) discrete, continuous or mixed-integer variable space; and c) functions without continuity, differentiability and convexity requirements.

Starting from the Chapter 3, we present the theory of discrete constrained optimization using Lagrange multipliers. The theory was originally developed for solving discrete constrained NLPs. Using a floating-point representation of continuous variables, we are able to extend the theory to solve continuous and mixed-integer NLPs.

Chapter 3

Nonlinear Constrained Optimization Using Lagrange Multipliers

In this chapter, we present our proposed theory of discrete constrained optimization using Lagrange multipliers [217, 226, 179] for solving general constrained optimization problems whose (objective and constraint) functions may not be differentiable.

The theory was first developed for solving SAT problems [179, 207] but was incomplete in the sense that it only provided a sufficient, but not necessary, condition for a point to be a discrete-space constrained local minimum [217, 226]. In this chapter, we propose a new generalized augmented Lagrangian formulation with a transform function H that facilitates the proof of the necessary and sufficient conditions for CLM_{dn} in discrete space. The proposed first-order conditions prove a one-to-one correspondence among a discrete-space constrained local minimum, a discrete-space saddle point, and a point satisfying the first-order conditions. Therefore, discrete-space constrained local minima can be located by solving the discrete-space first-order conditions or by looking for discrete-space saddle points. An efficient procedure (DLM) to look for saddle points and its implementation details are presented in Chapter 4.

Although the proposed theory was originally developed for solving discrete problems, it can be generalized to solving constrained NLPs with continuous variables after coding continuous variables by a floating-point representation. Our proposed theory, therefore, can be applied to solve continuous and mixed-integer constrained NLPs.

In Section 3.1, we study the errors of using a floating-point representation to code continuous variables in continuous and mixed-integer constrained NLPs. We then develop the theory for NLPs with equality constraints and extend the theory in Section 3.4 to NLPs with inequality constraints. Section 3.5 briefly reviews constrained simulated annealing (CSA), an algorithm developed by Wah and Wang [213, 211] based on the theoretical results in this chapter.

3.1 Floating-Point Representations of Continuous Variables in Constrained NLPs

Our work is motivated by the facts that floating-point representations are accepted as the de facto standard on digital computers to represent real numbers, and that algorithms for solving constrained NLPs, whether using closed-form formulae or not, use floating-point representations to represent real numbers when implemented on digital computers. When $\mathcal{P}_{org}$, the original problem with continuous variables, is redefined to be $\mathcal{P}_{sub}$, a new problem over a finite discrete variable space (as specified by a floating-point representation), the solutions to $\mathcal{P}_{org}$ are generally different from those to $\mathcal{P}_{sub}$.

Intuitively, we like to bound the difference between the optimal solution to $\mathcal{P}_{org}$ and that to $\mathcal{P}_{sub}$. There are two approaches to derive such a bound. First, a given constrained NLP can be analyzed to derive a tight bound between its original and discretized solutions. Besides being problem-specific, this approach only works in some simple cases because there

is no systematic approach today for finding tight error bounds for general constrained NLPs. Second, a loose bound can be derived based on some problem-dependent and problem-independent attributes that can be measured easily. Although not as tight as problem-specific bounds, such loose bounds are useful to assess the quality of solutions in $\mathcal{P}_{sub}$. In this research, we adopt the second approach.

The problem-specific attribute that we use is the minimum Lipschitz constant based on the original Lipschitz constant [149]. Given a general n-dimensional continuous function $f(x)$ defined over a bounded variable space:

$$[B_1^l, B_1^u] \times \cdots \times [B_n^l, B_n^u], \tag{3.1}$$

we define the *minimum Lipschitz constant* ℓ^{min} of $f(x)$ as the minimum of all possible Lipschitz constants [149] ℓ that satisfy:

$$|f(x) - f(y)| \leq \ell \cdot ||x - y|| \qquad \forall\ x, y\ \in [B_1^l, B_1^u] \times \cdots \times [B_n^l, B_n^u], \tag{3.2}$$

where $||\cdot||$ is the norm operator. Note that, if $f(x)$ has first-order derivatives everywhere in its variable space, then ℓ^{min} is actually the absolute value of the maximum gradient. However, (3.2) is general enough to apply to cases in which $f(x)$ is not continuous or does not have first-order derivatives everywhere.

Next, we characterize some properties of floating-point representations and analyze formally the effect on solution qualities of using floating-point numbers to code continuous variables in constrained NLPs.

3.1.1 Characteristics of Floating-Point Representations

A typical floating-point number y on a digital computer consists of the mantissa d and the exponent e, each represented by a finite number of bits [95]:

$$y = \pm\beta^e \times .d_1 d_2 \ldots d_t, \tag{3.3}$$

where β is the base (also called the radix), t is the precision, and e is the exponent in a range determined by $[e_{min}, e_{max}]$. Each digit, d_i, in (3.3) satisfies

$$0 \leq d_i \leq \beta - 1. \tag{3.4}$$

Among the digits, d_1 is called the most significant bit and d_t, the least significant bit. For instance, the IEEE double-precision standard has $\beta = 2$, $t = 53$, $e_{min} = -1021$, and $e_{max} = 1024$. Clearly, (3.3) defines a discrete, finite set of numbers, whereas $\mathcal{R}$, the set of real numbers, is an infinite uncountable set. Hence, the set of all y is a *discrete, finite, proper subset* of $\mathcal{R}$.

To illustrate that exact constrained solutions may not be representable by floating-point numbers, consider a simple continuous constrained NLP:

$$\begin{aligned} \text{minimize} \quad & f(x) = x \\ \text{subject to} \quad & h(x) = 3x - 2^{-1074} = 0. \end{aligned} \tag{3.5}$$

Obviously, the best solution with the minimum constraint violation is $x = \frac{1}{3} \cdot 2^{-1074}$ that cannot be represented in the IEEE double-precision format.

Another property unique in floating-point representations is that the level of discretization is not uniform across the range of possible numbers [95]. For example, the closest positive point to $x = 0$ in an IEEE double-precision representation is $x = 2^{-1074}$ (equal to the multiplication of 2^{-53}, the contribution from the least significant bit, and 2^{-1021} that is calculated from $\beta^{e_{min}}$), whereas the closest point to $x = (2^{1000} + 2^{948})$ is $x = 2^{1000}$. [1] The distance between the two closest points in floating-point space is 2^{-1074} in the first case and 2^{948} in the second. Figure 3.1 illustrates a similar scenario in a two dimensional space, where

[1] Simply represent 2^{1000} using the format in (3.3) and then perturb the last digit of the mantissa will verify the result.

grid points represent floating point numbers. It is obvious that the spacing between two adjacent floating point numbers increases along both the x and y dimensions.

Next, we study the effect of floating-point representations on solution qualities.

3.1.2 Worst-Case Error Bounds on CGM

Consider a general (continuous or mixed-integer) constrained NLP whose continuous variables are represented in floating-point types (single or double precision, for example). Obviously, such a representation is independent of any particular numerical algorithm used to solve the problem. Let c^* be the constrained optimum solution to $\mathcal{P}_{org}$, the original NLP, and d^* be the constrained optimum solution to $\mathcal{P}_{sub}$, the NLP with its continuous variables represented by floating-point numbers. Note that d^* is a discrete solution and c^* is usually not, and that c^* can be further decomposed into a vector form $c^* = (c_1^*, c_2^*, \ldots, c_n^*)$. It is obvious that c_i^* must reside inside an interval determined by two adjacent floating-point numbers, B_j^l and B_j^u, $j = 1, \ldots, n$. Denote $s_{grid,j}$ to be the interval size calculated using $B_j^u - B_j^l$. Let γ^* be the hypercube containing c^*, where

$$\gamma^* = [B_1^l, B_1^u] \times \cdots \times [B_n^l, B_n^u]. \tag{3.6}$$

Denote the minimum Lipschitz constants of the objective and constraint functions to be, respectively, ℓ_f^{min} and $\ell_{h_i}^{min}, i = 1, \ldots, m$, computed from continuous points inside γ^*. The following theorem proves the relationship between c^* in $\mathcal{P}_{org}$ and d^* in $\mathcal{P}_{sub}$ found by any numerical algorithm.

Theorem 3.1 *Worst-case error on objective function due to floating-point representation of continuous variables.* Assume the following for $\mathcal{P}_{org}$ and $\mathcal{P}_{sub}$.

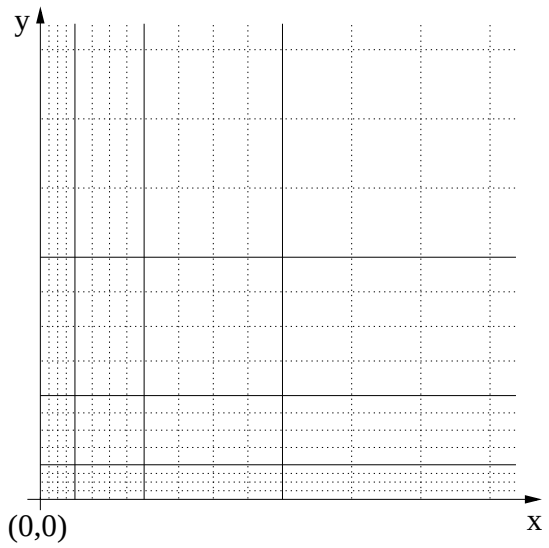


Figure 3.1: The non-uniform spacing between consecutive floating point numbers.

1. ℓ_f^{min}, and $\ell_{h_i}^{min}, i = 1, \ldots, m$, are finite. Intuitively, this condition limits the range of fluctuations in objective and constraint-function values in the variable space bounded by γ^*.

2. Constraint $h_i(x), i = 1, \ldots, m$, is considered to be satisfied if $|h_i(x)| \leq \Phi$, where Φ is a pre-specified maximum violation tolerance. This assumption is reasonable because it may not be possible to satisfy equality constraints exactly when there are precision errors in floating-point representations.

3. The interval sizes for all dimensions satisfy

$$G = \frac{1}{2}\sqrt{\sum_{j=1}^{n} s_{grid,j}^2} \leq \frac{\Phi}{max_{i=1}^{m} \ell_{h_i}^{min}}. \tag{3.7}$$

Then the error in objective f due to a floating-point representation of d^* is:

$$f(d^*) - f(c^*) \leq \ell_f^{min} \cdot G. \tag{3.8}$$

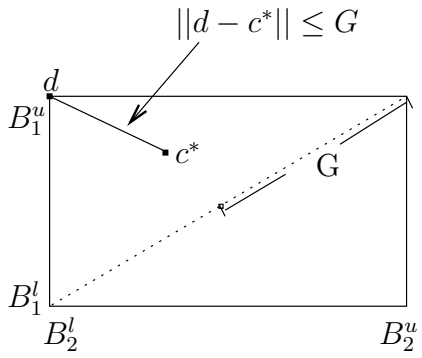


Figure 3.2: Illustration of Theorem 3.1.

Proof. For any infeasible discrete point x, from the second condition, we have $|h(x)| > \Phi$. Also, from (3.7), we conclude that

$$\forall x' \text{ such that } |x' - x| \leq G, \quad h(x') \neq 0, \tag{3.9}$$

holds true. (3.9) states the fact that for any continuous point x' close enough to a discrete infeasible point, x' cannot be the true CGM_{cn} because it cannot even be feasible.

Therefore, for any discrete point, d, that has a distance less than or equal to G to the true CGM_{cn} c^*, d is a feasible solution to P_{sub} ($|h_i(d)| \leq \Phi$), and $f(d)$ satisfies:

$$|f(d) - f(c^*)| \leq \ell_f^{min} \cdot ||d - c^*||. \tag{3.10}$$

Such a discrete point, d, is guaranteed to exist based on the special definition of G.

Thus, it holds that

$$|f(d) - f(c^*)| \leq \ell_f^{min} \cdot G. \tag{3.11}$$

Consequently,

$$f(d) \leq f(c^*) + \ell_f^{min} \cdot G \tag{3.12}$$

Since $f(d^*) \leq f(d)$, we have therefore verified that:

$$f(d^*) - f(c^*) \leq \ell_f^{min} \cdot G \tag{3.13}$$

Figure 3.2 illustrates the above proof in a two-dimensional space. In Figure 3.2, G is half of the diagonal distance of the rectangle bounded by $[B_1^l, B_1^u] \times [B_2^l, B_2^u]$, and d is the nearest discrete point to CGM_{cn} c^*.

It follows from the proof that d is feasible and $f(d) \leq f(c^*) + \ell_f^{min} \cdot G$. ■

Note that Theorem 3.1 only gives an upper bound on $f(d^*) - f(c^*)$, not on $|f(d^*) - f(c^*)|$. This is true because the constraints for $\mathcal{P}_{sub}$ are relaxed and $\mathcal{P}_{sub}$ can have better objectives than $\mathcal{P}_{org}$.

Although the bound in (3.8) is general and applies to all continuous and mixed-integer constrained NLPs and all search algorithms, its quality is problem dependent. In the following, we show three examples to illustrate this fact.

The first example shows that the bound is tight in some cases. Consider a one-dimensional convex, continuous, finite function $f(x)$ satisfying:

$$\ell_f^{min} \;=\; max(\ell_{h_1}^{min}, \ell_{h_2}^{min}, \ldots, \ell_{h_m}^{min}).$$

Let c^* be located in the center of two adjacent discrete grid points, and

$$-f'(x_1) = f'(x_2) = \ell_f^{min} \qquad \forall x_1 < c^* \text{ and } \forall x_2 > c^*.$$

Hence,

$$f(d^*) - f(c^*) = \ell_f^{min} \cdot ||d^* - c^*|| \;=\; \ell_f^{min} \cdot G. \tag{3.14}$$

The last equality is true because $f(x)$ is convex, and the two points d^* and c^* satisfy $||d^* - c^*|| = G$.

The second example shows that the bound is small in some cases. Consider the simple program in (3.5). Obviously, the CGM_{cn} is at $c^* = \frac{1}{3} \cdot 2^{-1074}$ and $f(c^*) = c^*$. Using a double-precision representation, c^* is inside the interval $[0, 2^{-1074}]$. Within this interval, $\ell_f^{min} = 1$,

$\ell_h^{min} = 3$, and $G = \frac{1}{2} \cdot 2^{-1074} = 2^{-1075}$. Let Φ be $\frac{3}{2} \cdot 2^{-1074}$. One can easily verify that the three conditions in Theorem 3.1 hold true. Therefore, according to Theorem 3.1,

$$f(d^*) - f(c^*) \leq \ell_f^{min} \cdot G = 1 \cdot 2^{-1075} = 2^{-1075}. \tag{3.15}$$

The bound is obviously correct because $f(d^*) = f(0) = 0$.

The third example shows that the bound is loose in some other cases. Consider the following constrained program with a large c^*.

$$\begin{aligned} \text{minimize} \quad & f(x) = x \\ \text{subject to} \quad & h(x) = x - (2^{1000} + 2^{947}) = 0. \end{aligned} \tag{3.16}$$

It is clear that $f(c^*) = c^* = (2^{1000} + 2^{947})$, and that c^* is inside $[2^{1000}, (2^{1000} + 2^{948})]$. Note that $(2^{1000} + 2^{948})$ is the minimum floating-point number larger than 2^{1000}, and $(2^{1000} + 2^{947})$ is not representable by a floating-point number. Inside the interval $[2^{1000}, (2^{1000} + 2^{948})]$, $\ell_f^{min} = \ell_h^{min} = 1$, and $G = \frac{1}{2} \cdot 2^{948} = 2^{947}$. In order to have a numerical solution based on a double-precision representation of x, we have to set $\Phi \geq 2^{947}$. The three conditions in Theorem 3.1 can be verified to be true for $\Phi = 2^{947}$. Hence,

$$f(d^*) - f(c^*) \leq \ell_f^{min} \cdot G = 1 \cdot 2^{947} = 2^{947}. \tag{3.17}$$

The results in this section reveal that solving continuous or mixed-integer constrained NLPs $\mathcal{P}_{org}$ by digital computers is equivalent to solving discrete constrained NLP $\mathcal{P}_{sub}$ mapped on a finite, discrete space represented by floating-point numbers. The precision of such a representation is adequate in most cases, although in degenerate cases in which the variable range is exceedingly large, the precision will be poor due to the large G. (A possible remedy is to use scaling on the variable range.) Given a discrete representation, we develop in the rest of this chapter the theory and methods to solve discrete constrained NLPs.

3.2 General Augmented Lagrangian Formulation of Discrete Constrained NLPs

A general discrete constrained NLP with equality constraints defined on discrete variable space X is formulated as follows:

$$\begin{array}{lll} \text{minimize} & f(x) & x = (x_1, \ldots, x_n) \text{ is a vector} \\ \text{subject to} & h(x) = 0 & \text{of discrete variables,} \end{array} \tag{3.18}$$

where $f(x)$ is the objective function and $h(x) = [h_1(x), \ldots, h_m(x)]^T$ is a set of m equality constraints. As stated before, our formulation has no requirements on convexity, differentiability, or continuity of the objective and constraint functions. Further, the objective and constraint functions in (3.18) are not necessarily required to have closed-forms and can even be evaluated procedurally.

Next we define a new augmented Lagrangian function based on a weighted sum of the objective function $f(x)$ and the constraint functions $h(x)$. A new transformation function $H(\cdot)$ is introduced in order to facilitate the proof of our proposed theory. Note that the Lagrangian function $L_d(x, \lambda)$ defined next is independent of whether the neighborhood is discrete or not. Moreover, its form is not unique and can take other forms without affecting the validity of our theory.

Definition 3.1 A *new generalized augmented Lagrangian function* of (3.18) is defined as:

$$L_d(x, \lambda) = f(x) + \lambda^T H\left(h(x)\right) + \frac{1}{2}||h(x)||^2, \tag{3.19}$$

where H is a non-negative (or non-positive) continuous transformation function satisfying $H(y) = 0$ if and only if $y = 0$, and

$$||h(x)||^2 = \sum_{i=1}^{m} h_i^2(x). \tag{3.20}$$

The definition of augmented Lagrangian function follows directly from the continuous version except the transformation function that is essential in proving our theory. We cannot use L_d to derive first-order necessary conditions in discrete space similar to those in continuous space [128] because L_d is a non-differentiable discrete function. Without differentiability, the results in continuous space cannot be extended to those in discrete space.

An understanding of gradients in continuous space shows that they define directions in a small neighborhood in which function values increase. To this end, we use a concept similar to that in traditional discrete-space search and define a descent direction in discrete space as follows.

A *direction of maximum potential drop (DMPD)* defined on discrete neighborhoods of $L_d(x, \lambda)$ at point x for fixed λ is a vector[2] that points from x to $x' \in \mathcal{N}_{dn}(x)$ with the minimum L_d:

$$\Delta_x L_d(x, \lambda) = \vec{\nu}_x = y \ominus x = (y_1 - x_1, \ldots, y_n - x_n) \tag{3.21}$$

$$\text{where} \quad y \in \mathcal{N}_{dn}(x) \cup \{x\} \text{ and } L_d(y, \lambda) = \min_{x' \in \mathcal{N}_{dn}(x) \cup \{x\}} L_d(x', \lambda).$$

Here, $\ominus$ is the vector-subtraction operator for moving from x to a point in $\mathcal{N}_{dn}(x) \cup \{x\}$. Intuitively, vector $\vec{\nu}_x$ points from x to y, the point with the minimum L_d among all neighboring points in $\mathcal{N}_{dn}(x)$, including x itself. That is, if x itself has the minimum L_d, then $\vec{\nu}_x = \vec{0}$. As an illustration, the DMPD of A in Figure 3.3 points to C, the neighbor of A with the minimum L_d.

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

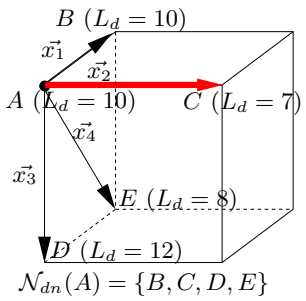


Figure 3.3: $DMPD(A) = C$

Based on this definition, it is easy to show that DMPDs cannot be added/subtracted [226]. Hence, we cannot use DMPDs in a similar proof [128] of first-order conditions in continuous space in order to prove the corresponding conditions in discrete space.

Note that in finite calculus [81], a *difference* operator Δ is defined to be:

$$\Delta f(x) = f(x + \delta) - f(x) \tag{3.22}$$

where $x+\delta$ is the closest grid point in discrete space to x and $\delta > 0$. This *difference* operator cannot be used to define first-order conditions similar to those in continuous space for the following reasons. First, it does not necessarily define a direction pointing to a grid point in discrete space because δ is the difference, which can be a real number, between function values of two discrete points. Second, it does not define a descent direction because it is associated with only two neighboring points $x + \delta$ and x, while ignoring all other points in $\mathcal{N}_{dn}(x)$. Consequently, it cannot be used to guide a search in discrete space. Finally, there is no chain rule [128] based the definition in (3.22), which is critical to the proof of the first-order conditions in continuous space [128].

A *discrete-space saddle point (SP_{dn})* [179, 217, 226] (x^*, λ^*) is defined in the same way as in continuous space [128] with the following property:

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*), \quad \text{for all } x \in \mathcal{N}_{dn}(x^*) \text{ and all } \lambda \in R. \tag{3.23}$$

The first inequality only holds if all constraints are satisfied and must be true for all λ. Further, (x^*, λ') is a SP_{dn} if $\lambda' \geq \lambda^*$, where $\lambda' \geq \lambda^*$ means that every element of λ' is no less than the corresponding element of λ^*. Last, for a reason similar to that of CLM_{dn}, whether a point is a SP_{dn} depends on the choice of $\mathcal{N}_{dn}(x)$. That is, x may be a SP_{dn} for $\mathcal{N}_{dn}(x)$ but not for $\mathcal{N}'_{dn}(x)$.

3.3 First-Order Necessary and Sufficient Conditions for CLM_{dn}

The concept of saddle points is very important to discrete problems because, starting from them, we can derive first-order necessary and sufficient conditions for CLM_{dn} that lead to efficient search procedures. These conditions are stronger than their continuous counterparts because they are necessary and sufficient (rather than necessary alone). Moreover, they are derived in an entirely different way using the concept of discrete-space saddle points rather than that of regular points [128].

Theorem 3.2 *First-order necessary and sufficient conditions for* CLM_{dn}. If H in (3.19) is a continuous function satisfying $H(x) = 0$ if and only if $x = 0$ and is non-negative (or non-positive), then a point in the search space of (3.18) is a CLM_{dn} if and only if

- it satisfies the discrete-space saddle-point condition (3.23) for any $\lambda \geq \lambda^*$, where $\lambda' \geq \lambda^*$ means that each element of λ' is not less than the corresponding element of λ^*; or
- it satisfies the following discrete-space first-order conditions:

$$\Delta_x L_d(x, \lambda) = 0, \tag{3.24}$$

$$\text{and } h(x) = 0, \tag{3.25}$$

where Δ_x is the DMPD operator defined on discrete neighborhoods.

Proof. The proof is done in three parts. In the first part, we prove that the saddle-point condition is necessary and sufficient for (3.24) and (3.25).

"$\Rightarrow$" part: Given a SP_{dn} (x^*, λ^*), we want to prove it to be a solution to (3.24) and (3.25). Eq. (3.24) is true because L_d cannot be improved among $\mathcal{N}_{dn}(x^*)$ from the definition of saddle points. Hence, $\Delta_x L_d(x^*, \lambda^*) = 0$ holds true, according to the definition of *DMPD*. Eq. (3.25) is true because $h(x) = 0$ must be satisfied at any solution point.

"$\Leftarrow$" part: Given a solution (x^*, λ^*) to (3.24) and (3.25), we like to prove it to be a SP_{dn}. The first condition $L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*)$ holds for all $x \in \mathcal{N}_{dn}(x^*)$ because $\Delta_x L_d(x^*, \lambda^*) = 0$. Hence, no improvement of L_d can be found in the neighborhood of x^*. The second condition $L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*)$ is true for all λ because $h(x^*) = 0$ according to (3.25). Thus, (x^*, λ^*) is a SP_{dn}.

In the second part of the proof, we prove that if $H(x)$ is a continuous function satisfying $H(x) = 0 \Leftrightarrow x = 0$ and is non-negative (or non-positive), then for any CLM_{dn} x^*, there exists a finite λ^* to make (x^*, λ^*) a SP_{dn}. We only prove the case in which $H(x)$ is non-negative, and the case in which $H(x)$ is non-positive can be proved similarly.

To prove this part, we construct λ^* for every CLM_{dn} x^* in order to make (x^*, λ^*) a SP_{dn}. This λ^* must be bounded and be found in finite time in order for the procedure to be useful.

a): Constructing λ^*. Given x^*, consider $x \in \mathcal{N}_{dn}(x^*)$. Let $h(x) = (h_1(x), \ldots, h_m(x))$ be an m-element vector, and the initial $\lambda^* = (\lambda_1^*, \ldots, \lambda_m^*) = (0, \ldots, 0)$. For every x such that $H(h(x)) > 0$, there is at least one constraint that is not satisfied, say $H(h_i(x)) > 0$. For this constraint, we set:

$$\lambda_i^* \rightarrow \max\left(\lambda_i^*, \frac{f(x^*) - f(x)}{H(h_i(x))}\right). \tag{3.26}$$

The update defined in (3.26) is repeated for every unsatisfied constraint of x and every $x \in \mathcal{N}_{dn}(x^*)$ until no further update is possible. Since $\mathcal{N}_{dn}(x^*)$ has a finite number of elements in discrete space, (3.26) will terminate in finite time and result in finite λ^* values.

b) Proving that (x^*, λ^*) *is a* SP_{dn}. To prove that (x^*, λ^*) is a SP_{dn}, we need to prove that:

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*) \quad \forall x \in \mathcal{N}_{dn}(x^*). \tag{3.27}$$

The first inequality is trivial because

$$L_d(x^*, \lambda) = f(x^*) = L_d(x^*, \lambda^*). \tag{3.28}$$

In the second inequality, for all $x \in \mathcal{N}_{dn}(x^*)$ such that $h(x) = 0$, it is clear that

$$L_d(x^*, \lambda^*) = f(x^*) \leq f(x) = L_d(x, \lambda^*) \tag{3.29}$$

holds since x^* is a CLM_{dn}. For all $x \in \mathcal{N}_{dn}(x^*)$ such that $h(x) \neq 0$, there must be at least one constraint that is not satisfied, say $H(h_i(x)) > 0$. Moreover, from the construction method, we know that

$$\lambda_i^* \geq \frac{f(x^*) - f(x)}{H(h_i(x))}. \tag{3.30}$$

Therefore,

$$L_d(x^*, \lambda^*) = f(x^*) \leq f(x) + \lambda_i^* H(h_i(x)) \tag{3.31}$$

holds. Further, since $\sum_{j=1, j\neq i}^{m} \lambda_j^* H(h_j(x))$ is non-negative (assuming all constraints are transformed by H into non-negative functions), it is clear that

$$\begin{aligned} L_d(x^*, \lambda^*) &= f(x^*) \leq f(x) + \sum_{j=1}^{m} \lambda_j^* H(h_j(x)) \\ &\leq f(x) + \sum_{j=1}^{m} \lambda_j^* H(h_j(x)) + \sum_{j=1}^{m} \frac{1}{2} H^2(h_j(x)) = L_d(x, \lambda^*). \end{aligned} \tag{3.32}$$

Hence, (x^*, λ^*) is a SP_{dn}.

In the last part, we prove that (x^*, λ') is a SP_{dn} for any $\lambda' \geq \lambda^*$. The proof is a straightforward extension of the second part and is not shown here.

The theorem follows after combining the three parts of the proof. ■

We like to point out a few key features of the theorem.

The first condition in Theorem 3.2 states that finding any $\lambda \geq \lambda^*$ suffices for finding a SP_{dn}. This is important in practice because a search procedure may not be able to find the exact λ^* but some $\lambda \geq \lambda^*$.

The theorem also requires a transformation H that is non-negative or non-positive, but not both. Examples of such transformation are the absolute-value function and the square function. The use of such transformations is not allowed in the traditional theory of Lagrange multipliers that works in continuous space because the transformed constraint function $H(h(x))$ is not differentiable at $h(x) = 0$. This is not an issue in the Lagrange-multiplier theory for discrete problems because it does not rely on derivatives to find descent directions.

Although not necessary, transformation H is sufficient to ensure that a CLM_{dn} is a SP_{dn} based on the Lagrangian function with transformed constraints. To illustrate the point, we show in the following example that CLM_{dn} x^* may not be a SP_{dn} when $h(x)$ can have both positive and negative values. In this case, it is not always possible to find λ^* to make (x^*, λ^*) a SP_{dn} even when x^* is a CLM_{dn} in discrete space.

Example 3.1 Consider a two-dimensional discrete equality-constrained problem with objective $f(x)$ and constraint $h(x) = 0$, where

$$f(0,0) = 0, \quad f(0,1) = 1, \quad f(0,-1) = 0, \quad f(1,0) = 0, \quad f(-1,0) = -1,$$
$$h(0,0) = 0, \quad h(0,1) = 0, \quad h(0,-1) = \tfrac{1}{2}, \quad h(1,0) = 0, \quad h(-1,0) = -\tfrac{1}{2}.$$

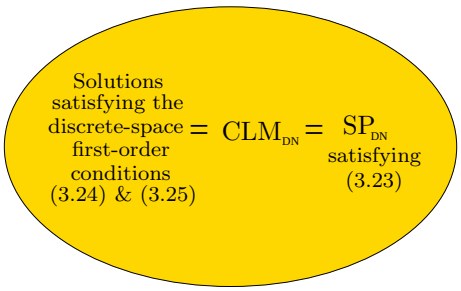


Figure 3.4: Relationships among solution sets of discrete constrained NLPs defined over discrete neighborhoods. CLM_{dn} and SP_{dn} stand for CLM and saddle points defined over discrete neighborhoods, respectively.

We like to show that $(0, 0)$ is a CLM_{dn} but not a SP_{dn}.

Obviously, $(x^*, y^*) = (0, 0)$ is a CLM_{dn} based on the values of its objective and constraint functions and those of its neighboring points. Further, from the definition of Lagrangian function, we know that $L_d((0, 0), \lambda) = 0$ holds true for any λ because $h(0, 0) = f(0, 0) = 0$.

To draw a contradiction, assume that $(0, 0)$ is a SP_{dn}. Hence, there exists λ^* such that

$$L_d((0,0),\lambda^*) \leq L_d((-1,0),\lambda^*), \tag{3.33}$$

$$\text{and} \quad L_d((0,0),\lambda^*) \leq L_d((0,-1),\lambda^*). \tag{3.34}$$

After substitution, we get the following equations:

$$0 \leq f(-1,0) + \lambda^* \cdot h(-1,0) + \frac{1}{2}h^2(-1,0) = -1 - \lambda^* \cdot \frac{1}{2} + \frac{1}{8}, \tag{3.35}$$

$$0 \leq f(0,-1) + \lambda^* \cdot h(0,-1) + \frac{1}{2}h^2(0,-1) = 0 + \lambda^* \cdot \frac{1}{2} + \frac{1}{8}.$$

Adding the above two inequality equations yields a contradiction that $0 \leq -\frac{3}{4}$.

Figure 3.4 depicts the results in Theorem 3.2. Comparing Figures 3.4 and 2.3 that summarizes the relationships among solution sets based on the traditional theory of Lagrange multipliers defined over continuous neighborhoods, we find the following three notable differences.

a) The traditional theory of Lagrange multipliers is defined over continuous neighborhoods (1.3) and requires the differentiability of the objective and constraint functions and CLM_{cn} to be regular points (points with linearly independent derivatives along all dimensions) [128]. The theory does not apply in discrete and (discretized) mixed-integer problems defined over discrete neighborhoods and whose functions may not be differentiable.

b) The set of CLM_{cn} satisfying the first-order and second-order sufficient conditions is only a subset of all CLM_{cn} (Figure 2.3). CLM_{cn} whose derivatives do not exist or that are not regular points cannot be found by existing algorithms in continuous space. Hence, global optimization of points that satisfy the first-order necessary and second-order sufficient conditions does not always lead to a CGM_{cn} of the original problem. In contrast, the theory of discrete constrained optimization using Lagrange multipliers shows a one-to-one correspondence between SP_{dn} and CLM_{dn}. Hence, in discrete space, a global-optimization strategy looking for saddle points (SP_{dn}) with the minimum objective value will result in a constrained global minimum (CGM_{dn}). This property is utilized in the constrained simulated annealing algorithm [213] described in Section 3.5.

c) Theorem 3.2 reduces the hard problem of finding CLM_{dn} to the easier problem of finding discrete-space saddle points (SP_{dn}) or points satisfying (3.24) and (3.25). These conditions are much easier to implement in practice than the search of points that explicitly satisfy multiple nonlinear constraints simultaneously. In Chapter 4, we describe a first-order search method that looks for discrete-space saddle points.

3.4 Handling Inequality Constraints

The results discussed so far apply only to discrete optimization problems with equality constraints. We handle (3.18) with inequality constraints by transforming them into equality

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 $\mathcal{N}_{dn}(\mathbf{x})$ using $G(\mathbf{x}, \mathbf{x}')$;
8. accept $\mathbf{x}'$ with probability $A_T(\mathbf{x}, \mathbf{x}')$
9. **end_for**
10. reduce temperature by $T \longleftarrow \alpha \times T$;
11. **end_while**
12. **end_procedure**

Figure 3.5: CSA: constrained simulated annealing procedure.

constraints using a maximum function.

$$g_j(x) \leq 0 \Longleftrightarrow \max(g_j(x), 0) = 0. \tag{3.36}$$

Obviously, the new equality constraint is satisfied if and only if $g_j(x) \leq 0$. Since all inequality constraints can be transformed into equality constraints in a similar way, we do not explicitly represent inequality constraints in the rest of this thesis.

3.5 CSA for General Constrained NLPs

In this section, we describe constrained simulated annealing (CSA) [213, 211, 218], an application of Theorem 3.2 to look for CGM_{dn} with asymptotic convergence. CSA looks for discrete-space saddle points by performing both probabilistic descents in the original variable space and probabilistic ascents in the Lagrange-multiplier space in order to satisfy all the constraints in (3.18).

3.5.1 CSA Procedure

Figure 3.5 shows the basic procedure of CSA. A detailed discussion of each line in Figure 3.5 can be found in [213, 211]. The fundamental idea is explained as follows. The annealing process starts 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 function values and constraint violations; otherwise, T^0 is set to be a smaller value.

The number of trials at each temperature, N_T, is related only to the number of constraint functions and the number of dimensions of the original problem. In [213, 211] 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.

A trial point x' in the neighborhood of the current point, x, is generated using a continuous stochastic distribution, such as Cauchy or Gaussian. Trial points generated by these continuous distributions can be used directly if the associated variable in the original NLP is continuous. Otherwise, the trial points generated are rounded to the closest discrete points.

$\mathcal{N}_{csa}(\mathbf{x})$, the neighborhood of a point in the joint space of the original variables and Lagrange multipliers, is defined as follows:

$$\mathcal{N}_{csa}(\mathbf{x}) = \{(x', \lambda) \in S \text{ where } x' \in \mathcal{N}_{dn}(x)\} \cup \{(x, \lambda') \in S \text{ where } \lambda' \in \mathcal{N}_{cn}(\lambda)\} \quad (3.37)$$

$$\begin{aligned} \mathcal{N}_{cn}(\lambda) = {} & \{\mu \in \Lambda \mid \mu < \lambda, \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\} \bigcup \\ & \{\mu \in \Lambda \mid \mu > \lambda, \text{ and } \mu_i = \lambda_i \text{ if } h_i(x) = 0\}, \end{aligned} \quad (3.38)$$

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 subspace X, and (x, λ') is

a neighbor to (x, λ) if λ' is a neighbor to λ in 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.*, $h_i(x) = 0$.

As in SA, an acceptance probability is used in CSA to decide whether a trial point generated is to be accepted or rejected. The difference in CSA, however, is that an acceptance probability needs to be defined for descents as well as ascents. The acceptance probability for CSA is defined to be:

$$A_T(\mathbf{x}, \mathbf{x}') = \begin{cases} exp\left(-\frac{(L_d(\mathbf{x}')-L_d(\mathbf{x}))^+}{T}\right) & \text{if } \mathbf{x}' = (x', \lambda) \\ exp\left(-\frac{(L_d(\mathbf{x})-L_d(\mathbf{x}'))^+}{T}\right) & \text{if } \mathbf{x}' = (x, \lambda') \end{cases} \tag{3.39}$$

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

The probability defined in (3.39) allows a search to find points with smaller Lagrangian value in a stochastic fashion. A smaller Lagrangian value implies either a smaller objective-function value or a smaller total constraint violation, both of which are desirable. In contrast, probabilistic ascents, which are absent from traditional SA, try to increase the Lagrange multipliers on violated constraints stochastically in order to force them into satisfaction.

3.5.2 Asymptotic Convergence of CSA

CSA has been proved [211, 213] 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) [197], and showing that the virtual energy is at its minimum at any CGM_{dn}. Details of the proofs can be found in [218].

The main result of CSA [211, 213] is summarized in the following theorem.

Theorem 3.3 *Asymptotic convergence of CSA* [213, 211]. The Markov chain modeling CSA converges asymptotically to a CGM_{dn} $x^* \in X_{opt}$ with probability one.

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. To this end, Wah and Chen [204] have developed an anytime schedule to allow a search to find an optimal (but finite) cooling schedule with high probability. CSA has been tested to work well in solving nonlinear benchmark problems [211].

3.6 Summary

In summary, the analysis in Section 3.1 shows that numerical algorithms implemented on digital computers aiming to solve $\mathcal{P}_{org}$ actually solve, instead, $\mathcal{P}_{sub}$ defined over a *finite, discrete subset* of the original variable space. As shown in Theorem 3.1, the constrained optimum solution to $\mathcal{P}_{sub}$ is generally different, but within a prescribed upper bound, from the constrained optimum solution to $\mathcal{P}_{org}$. The three examples illustrate that the upper bound is closely related to the value of c^* and will be small if c^* is close to zero. In contrast, if the absolute value of c^* is large, then the upper bound is poor due to the large spacing between adjacent floating-point numbers that bound c^*. Note that Theorem 3.1 only gives an upper bound. It is possible for $\mathcal{P}_{sub}$ to have better objectives than $\mathcal{P}_{org}$ because the constraints for $\mathcal{P}_{sub}$ are relaxed. The results allow us to treat discrete, continuous, and mixed-integer constrained NLPs in a unified fashion.

The theory of discrete constrained optimization using Lagrange multipliers, defined in Theorem 3.2, is theoretically complete in the sense that we prove the one-to-one correspon-

dence among the sets of CLM_{dn}, SP_{dn}, and points satisfying the first-order necessary and sufficient conditions defined over discrete neighborhoods. The theory leads to efficient search algorithms that look for saddle points in discrete neighborhoods in order to implicitly satisfy multiple nonlinear constraints simultaneously, instead of looking for points that attempt to satisfy those constraints explicitly.

Finally, Table 3.1 lists the differences between the theory and methods of Lagrange multipliers in continuous space and those in discrete space. [3]

[3]For a formal comparison, check the various definitions, examples and proofs in Chapters 1 and 3.

Table 3.1: Differences between the theory and methods of Lagrange multipliers in continuous space and those in discrete space.

Continuous constrained NLPs with differentiable functions	**Discrete constrained NLPs with non-differentiable functions**
Neighborhoods, $\mathcal{N}_{cn}(x)$, are open spheres whose radius approaches zero asymptotically; the number of neighboring points cannot be enumerated.	*Neighborhoods*, $\mathcal{N}_{dn}(x)$, are finite, user-defined sets of discrete neighboring points that may be very large but finite in size.
Constrained local minima (CLM_{cn}) are feasible points with the smallest objective value in a neighborhood of an open sphere.	*Constrained local minima* (CLM_{dn}) are defined similarly, except that their neighborhoods are finite and user-defined.
Augmented Lagrangian function is the sum of the objective and the constraints weighted by Lagrange multipliers.	Similar, except that a transformation is applied to each constraint function to convert it to non-negative.
Saddle point (SP_{cn}) is a local minimum in the original-variable space and a local maximum in the Lagrange-multiplier space (based on a neighborhood of an open sphere); there is no systematic procedure to look for SP_{cn} in continuous space.	*Saddle point* (SP_{dn}) is defined similarly, except that its neighborhood is finite and user-defined; SP_{dn} may be found by descents of the augmented Lagrangian function in the original-variable space and ascents in the Lagrange-multiplier space.
Direction of descent is found by differentiation.	Direction of descent is found by (limited) enumeration or by sampling.
CLM_{cn} that are regular points can be found by looking for points that satisfy the first-order necessary and second-order sufficient conditions; methods require the differentiability of functions.	CLM_{dn} are the same as SP_{dn} (necessary and sufficient); solution methods rely on sampling or (limited) enumeration.
No existing constrained global optimization procedures.	*Constrained simulated annealing (CSA)* converges asymptotically to SP_{dn} with the minimum objective values (CGM_{dn}).
Solution techniques cannot be generalized to nonlinear optimization problems in discrete or mixed-integer space.	Solution techniques can be generalized to mixed-integer and continuous constrained NLPs without differentiability or convexity assumptions on functions.

Chapter 4

Discrete Space First-Order Search Methods

This chapter extends the discrete-space first-order conditions defined in (3.24) and (3.25) into a first-order search and presents DLM, our proposed first-order search procedure. DLM is able to locate, in a systematic way, feasible points (or SP_{dn} when DMPD is truly implemented for selecting candidate points) for solving discrete constrained optimization problems. By employing floating-point representations of continuous variables, DLM is further able to solve continuous and mixed-integer constrained NLPs.

The framework and outline of DLM are introduced first in Section 4.1, followed by careful explorations of five major components of DLM, including neighborhood search, dynamic weight adaptation [215], global search, relax-and-tighten, and duration of run. In order to choose suitable strategies and parameters in various components of DLM, we test twelve benchmark problems on various combinations of parameters and strategies. Finally, we present experimental results of DLM on discrete, continuous and mixed-integer constrained NLP benchmarks in Section 4.3.

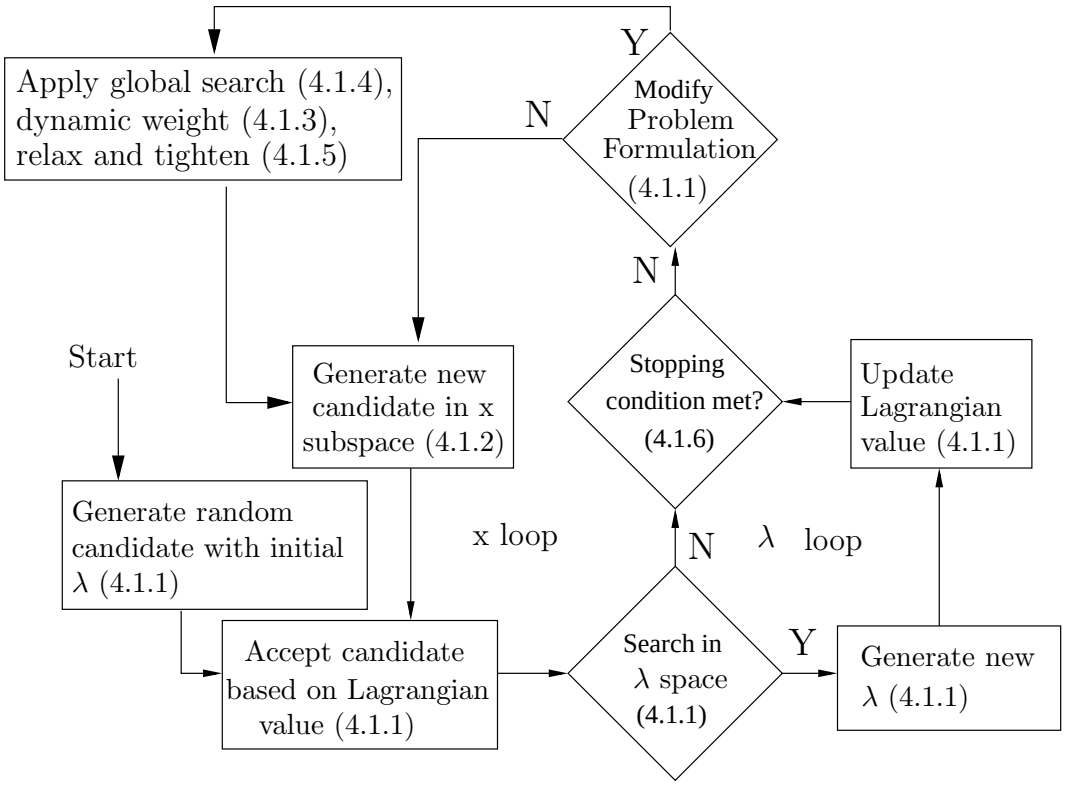


Figure 4.1: Framework of DLM, a first-order search for SP_{dn}. Each component is discussed in detail in the section marked in parenthesis.

4.1 A Discrete-Space First-Order Search Framework

The first-order necessary and sufficient conditions (3.24) and (3.25) provide a stopping condition for a search to look for a saddle point but do not specify the mechanism to arrive at such points. There are many ways to look for discrete saddle points, such as using simulated annealing [213], genetic algorithms, and hill climbing. In this section, we first propose a general framework (Figure 4.1) of first-order search method for locating feasible solutions (or SP_{dn}). We then present *DLM-General*, an implementation of the discrete-space first-order search framework in Figure 4.1, and explore carefully five components of *DLM-General*.

procedure *DLM-General*

0. set ϱ_1 to be a positive real constant; set random seed; j:=0;
1. set starting point x; set initial value of λ (set to 0 in the experiments);
2. **if** using dynamic weight adaptation **then** *weight_initialization*;
3. **while** stopping condition not satisfied **do** {
4. **if** modifying problem formulation **then** {
5. **if** using dynamic weight adaptation **then** *dynamic_weight_adaptation*;
6. **if** using global search **then** *perform_global_search*;
7. **if** using relax-and-tighten **then** *perform_relax_and_tighten*; }
8. x ***Loop:*** update x to x' only if this will result in $L_d(x', \lambda) < L_d(x, \lambda)$;
9. λ ***Loop:*** **if** condition for updating λ is satisfied **then** $\lambda_i \leftarrow \lambda_i + \varrho_1 h_i$;
10. j++; }

Figure 4.2: *DLM-General*: An implementation of the general discrete first-order local-search method. (The initial values of parameters are indicated here unless specified otherwise in the text.)

4.1.1 DLM: An Implementation of First-Order Search Framework

The framework in Figure 4.1 consists of two major loops. One loop is the "x loop" that generates new candidate points (or trial points) in the original-variable space and accepts them based on their Lagrangian values. The other loop, the "λ loop," updates the Lagrange multipliers in order to suppress constraint violations, if they exist.

Figure 4.2 shows an implementation of the framework in Figure 4.1. In the following, we describe some of the considerations and trade-offs in implementing the procedure.

a) *Initialization* (Lines 0-2). We choose either a fixed or a randomly generated starting point using a fixed initial seed. Both allow our results to be reproducible by others. We initialize all Lagrange multipliers to zero. An optimal initial setting of x and λ is difficult because it depends on the amount of constraint violation. Also, if dynamic weight adaptation is to be used, then *weight_initialization* is done. Variable j used in dynamic weight adaptation

counts the number of round robins in the search. It is initially set to zero in Line 0 and is increased in Line 10.

b) *Duration of each run* (Line 3). We use the idea of iterative deepening proposed in [204] to decide on the suitable duration of a run. See Section 4.1.6 for details.

c) *Modification of problem formulations* (Line 4). Many heuristics in DLM can be characterized as some kind of modification of problem formulations. Three such modifications are included in DLM and are discussed next.

d) *Dynamic weight adaptation* (Lines 2 and 5). Dynamic weight adaptation changes the Lagrangian formulation by adding a weight to the objective and adjusts the weight dynamically in order to improve convergence of DLM. Section 4.1.3 examines the issues and alternatives of weight adaptation. This approach addresses a similar issue as our previous approach [179] that scales the Lagrange multipliers periodically.

e) *Global search* (Lines 6). A search trajectory generated by DLM may be stuck in an infeasible local minimum. For example, when the trajectory is at a local minimum of both the objective and the constraint functions, then increasing the Lagrange multipliers at this point will not help bring the trajectory out of the local minimum. Global search will be performed to enable the search trajectory traverse wider regions in the search space. In Section 4.1.4, we propose to add a *distance_penalty* term to the Lagrangian formulation to implement the above idea.

f) *Relax-and-tighten* (Lines 7). In applying DLM to solve constrained problems with many equality constraints, we find that it is difficult to find feasible solutions. Section 4.1.5 addresses this issue by presenting relax-and-tighten, a strategy that relaxes the original equality constraints into inequality constraints and that gradually tightens the relaxed constraints.

g) *x Loop* (Line 8) performs neighborhood search. Here, we evaluate some possible neighboring points of x in order to find improvements in its Lagrangian value. We try

$x_1, \ldots x_n$ in a round-robin fashion, one variable at a time, and compare the Lagrangian value of x with that of its neighbor. To save time, we apply a greedy strategy rather than a hill-climbing strategy, switching from one variable to the next once any improvement in its Lagrangian value has been found. For solving general nonlinear constrained NLPs, neighborhoods are generated using a random distribution like Gaussian or Cauchy. Details are discussed in Section 4.1.2.

A consequence of applying a greedy instead of a hill-climbing strategy is that the associated search trajectory is not guided by the DMPD of L_d. Hence, when the algorithm stops at a feasible point x, the point may not be a CLM_{dn} because not all neighboring points in $\mathcal{N}_{dn}(x)$ have been examined. In general, our proposed search algorithm will only find a feasible solution when it stops and has no guarantee that it will reach a CLM_{dn} (or SP_{dn}).

h) λ *Loop* (Line 9). The Lagrange multipliers are updated when the search reaches a local minimum in the objective space. We do not update the multipliers more frequently due to instability of the trajectory. The amount of update is controlled by an application-dependent constant ϱ_1 (> 0).

A search based on *DLM-General* can be considered a *local search* in the Lagrangian space because the search stops when all the constraints are satisfied and when there is no improvement in the Lagrangian value of the neighboring points probed. However, the search can be considered a *global search* in the original-variable space because a trajectory can overcome local basins and minima in the original-variable space by manipulating its Lagrange multipliers. When some of the constraints are not satisfied, Lines 8 and 9 of *DLM-General* perform, respectively, descents in the original-variable space and ascents in the Lagrange-multiplier space. Obviously, descents in the original-variable space will stop after a certain number of iterations; that is, $x^{k+1} = x^k$. However, Line 9 of *DLM-General* will never stop as long as there are violated constraints, and λ will continue to increase to

suppress the unsatisfied constraints. Increases in λ allows Line 8 in *DLM-General* to move on and get out of local minima in the original-variable space.

The following theorem ensures that when *DLM-General* stops, a feasible point will be located.

Theorem 4.1 *Termination condition.* A feasible point x^* is reached when Procedure *DLM-General* stops.

Proof. When the procedure shown in Figure 4.2 stops at a point x^*, it is obvious that the Lagrange multipliers stop to grow. Mathematically, it implies that $\lambda_i = \lambda_i + \varrho_1 h_i$. Hence, $h_i(x^*) = 0$ for $i \in \{1, 2, \ldots, m\}$, since $\varrho_1 \neq 0$. It follows that x^* is a feasible point. ■

It is important to note that a global search in the original-variable space does not imply convergence in finite time. Similar to continuous Lagrange-multiplier methods, the time for *DLM-General* to find a feasible solution may be unbounded, even if feasible solutions exist.

The framework of *DLM-General* is very general and can be implemented in many ways. For example, in modifying a problem formulation, new objectives, additional constraints, or relaxed constraints may be added; in generating trial points in the x subspace, deterministic or stochastic neighborhoods may be selected; in determining the stopping condition, fixed or adaptive strategies may be employed. Within these various possibilities, we select five of the most important components of *DLM-General* and explore them carefully in the following five subsections.

4.1.2 Neighborhood Search

The choice of neighborhoods is very critical to the efficiency of *DLM-General* because they determine the probability that a trial point is accepted. Intuitively, one may tempt to

choose a small neighborhood in close vicinity to the current point because it allows descent direction to be found efficiently and because it resembles the way that gradients are defined in continuous space. This was done in our early studies [179] that use a small neighborhood consisting of points with Hamming distance one away. A small neighborhood allows a hill-climbing strategy to enumerate all neighboring points and find its DMPD.

Our recent work indicates that small neighborhoods unnecessarily limit the ability of a strategy to find points with better Lagrangian values. In this thesis we propose to use a large neighborhood that covers all the points of a variable in a search space and refines the probabilities of reaching specific neighboring points depending on the current progress. We also use a greedy strategy that stops after finding the first point with improved Lagrangian value rather than enumerating all neighboring points. Specifically, we try $x_1, \ldots x_n$ in a round-robin fashion, generate a point in the neighborhood of the variable picked based on a probability distribution, round the point to the closest discrete point if the variable picked is discrete, accept the new point if the Lagrangian value of the new point is better than that of x, and change to the next variable in the round robin. (A round robin can be viewed as a sequential examination of the n dimensions of a point.)

For neighborhood generation, we consider the following three choices of random distributions:

- *Cauchy distribution* with a density function of $f_d(x) = \frac{1}{\pi}\frac{\sigma}{\sigma^2+x^2}$, where σ is a parameter to be determined,

- *Gaussian distribution* with a density function of $f_d(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{\frac{(x-\mu)^2}{2\sigma^2}}$, μ is set to zero in our implementation and σ is to be determined,

- *Uniform distribution* in the range $[-\sigma, \sigma]$.

The spread/shape of the above three distributions are controlled by σ. Because there are n dimensions, σ is a vector $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ in which σ_i determines the spread in the i^{th} dimension. Once a variable is selected (say x_i), we use one of the three random distributions to generate a point centered around x_i and bounded in $[R_i^l, R_i^u]$.

Intuitively, σ_i should be decreased when most of the neighboring points of x_i generated do not lead to smaller Lagrangian values. In contrast, σ_i should be increased when there is a large chance of finding points with improved Lagrangian values. Such considerations lead to the following way of adapting σ_i.

At the beginning, we set $\sigma_i = 0.1, i = 1, \ldots, n$, and initialize n counters $\vartheta_i, i = 1, \ldots, n$, to zero. We use ϑ_i to measure the progress of neighborhood search with respect to x_i and increase it by one when the neighboring point x_i' generated has a smaller Lagrangian value with respect to that of x_i. We update σ_i after 50 round robins of neighborhood search as follows:

$$\sigma_i = \begin{cases} \sigma_i \cdot 1.001 & \text{if } \vartheta_i \geq 40 \\ \sigma_i / 1.001 & \text{if } \vartheta_i \leq 2 \\ \sigma_i & \text{otherwise,} \end{cases} \tag{4.1}$$

and clear all the counters to zero. The constants 2 and 40 in (4.1) were chosen experimentally. The effects of these three random distributions on solution time and quality of DLM are discussed in Section 4.2.

4.1.3 Dynamic Weight Adaptation

Lagrangian methods rely on ascents in the Lagrange-multiplier space and descents in the objective space in order to reach equilibrium. The convergence speed and solution quality, however, depends on the balance between $f(x)$, $h(x)$, and $g(x)$. Although changes in λ and μ lead to different balance between ascents and descents, convergence can be improved by

introducing a weight on $f(x)$. These considerations lead to a new Lagrangian function as follows.

$$L_d^{'}(x, \lambda) = w\ f(x) + \lambda^T H(h(x)) + \frac{1}{2}||h(x)||^2, \tag{4.2}$$

where $w > 0$ is a user-controlled weight on the objective. By using (4.2) in *DLM-General* in Figure 4.2 with different w, we observe four possible behaviors of the search trajectory:

- The trajectory converges without oscillations.
- The trajectory gradually reduces in oscillations and eventually converges.
- The trajectory oscillates within some range but never converges.
- The magnitude of oscillations increases, and the trajectory eventually diverges.

Obviously, the first two cases are desirable, and the last two are not. Moreover, we like to reduce the amount of oscillations and improve convergence time.

To demonstrate the four behaviors, we evaluate Problem 2.6 in [57] under different weights. The original problem is a nonlinear, continuous minimization problem with 10 variables and 25 inequality constraints. Since it uses continuous variables in the range $[0, 1]$, we convert it into a discrete minimization problem by multiplying each variable x_i by 1000 and restricting the new variable $1000x_i$ to integer values.

Table 4.1 shows the average results in evaluating the discretized problem from 20 random starting points. By changing w, results of different quality and convergence time can be obtained. There is, however, no effective method for choosing a fixed w except by trial and error.

Next we present a strategy to adapt w based on run-time search progress in order to obtain high-quality solutions and short convergence time. This approach is more general than our previous approach [179] that scales the Lagrange multipliers periodically in order

Table 4.1: Effects of static and dynamic weights on convergence time and solution quality from 20 randomly generated starting points for the discretized version of Problem 2.6 in [57]. (Weight w is the initial weight in the dynamic case.)

Fixed/Initial	Avg. Conv. Time		Fraction Converged		Average Solution		Best Soluiton	
weight w	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic
0.00001	1099	1249	100%	100%	-24.7	-29.0	-39.0	-39.0
0.0001	1247	1249	100%	100%	-29.4	-29.0	-39.0	-39.0
0.001	1249	1248	100%	100%	-29.3	-29.3	-39.0	-39.0
0.01	1254	1249	100%	100%	-29.4	-29.3	-39.0	-39.0
0.1	1267	1265	100%	100%	-28.3	-28.8	-39.0	-39.0
1	1230	1289	90%	100%	-27.4	-27.4	-39.0	-39.0
10	1203	1377	75%	100%	-27.8	-28.1	-39.0	-39.0
100	972	1902	40%	100%	-34.8	-29.3	-39.0	-39.0
1000	679	1409	15%	100%	-39.0	-38.5	-39.0	-39.0
10000	–	2182	0%	100%	–	-38.4	–	-39.0
100000	–	2289	0%	100%	–	-33.3	–	-39.0

to prevent them from growing to be very large when all constraint functions are positive. The Lagrange multiplier of a non-negative constraint may grow without limit because its value is always non-decreasing according to Line 9 of *DLM-General*, and a Lagrangian space with large Lagrange multipliers is more rugged and more difficult to search by local search methods. In our previous approach [179], the period between scaling and the scaling factor are application dependent and chosen in an ad hoc fashion. Our current approach adjusts the weight between the objective and the constraints, which is equivalent to scaling the Lagrange multipliers. It is more general because it adjusts the weight according to the convergence behavior of the search.

Figure 4.3 outlines the procedures for weight initialization and adaptation in order to speed up convergence. Its basic idea is to first estimate the initial weight $w(0)$ (Line 1), measure the performance of the search trajectory $(x(t), \lambda(t))$ periodically, and adapt $w(t)$ to improve convergence time or solution quality.

procedure *weight_initialization*
1. set $w(0)$ (initial weight, set to 0.00001 in the experiments);
2. set N_u (major window for changing w, set to 30 in the experiments);
3. set δ_t (minor window for changing w, set to 5 in the experiments);
4. j (number of iterations since last divergence) is initialized in Line 0 of *DLM-General*

procedure *dynamic_weight_adaptation*
5. record useful information for calculating performance;
6. **if** ($j \mod \delta_t = 0$) **then**
7. **if** trajectory diverges **then** { reduce w; $j \leftarrow 0$ }
8. **if** ($j \mod N_u = 0$) **then** {
9. compute performance metrics based on data collected;
10. change w when certain conditions are satisfied (see text) }

Figure 4.3: Procedures for weight initialization and adaptation in Figure 4.2. (The initial values of parameters are indicated here unless specified otherwise in the text.)

Let (x_j, λ_j) be the point in the j^{th} iteration, and $v_{max}(j)$ be its maximum violation:

$$v_{max}(j) = \max\left[\max_{1\le s\le m}\{|h_s(x(t))|\}, \max_{1\le s\le k}\{g_s(x(t)), 0\}\right] \tag{4.3}$$

To monitor the progress of the search, we divide iterations into non-overlapping major windows of N_u iterations (Line 2), each of which is then divided into minor windows of δ_t iterations (Line 3). We further record some statistics, such as $v_{max}(j)$ and $f_j(x)$, that will be used to calculate the performance in each minor/major window (Line 5).

At the beginning of a minor window (Line 6), we test whether the trajectory diverges or not (Line 7). Divergence happens when $v_{max}(j)$ is larger than an extremely large value (say 10^{20}). If it happens, we reduce w, say $w \leftarrow \frac{w}{10}$, and restart the window markers by resetting j to 0.

At the beginning of a major window (Line 8), we compute some metrics to measure the progress of the search relative to that of previous major windows (Line 9). In general, application-specific metrics, such as the number of oscillations of the trajectory, can be used. In our current implementation, we compute the averages of $v_{max}(j)$ and objective $f_j(x)$ in

the u^{th} major window:

$$\bar{v}_u = \frac{1}{N_u} \sum_{j=(u-1)N_u+1}^{uN_u} v_{max}(j);$$

$$\bar{f}_u = \frac{1}{N_u} \sum_{j=(u-1)N_u+1}^{uN_u} f_j(x); \qquad u = 1, 2, \cdots \tag{4.4}$$

Based on these measurements, we adjust w accordingly (Line 10). Note that when comparing values between two successive major windows $u-1$ and u, both must use the same w; otherwise, the comparison is not meaningful because the terrain may be totally different. Hence, after adapting w, we should wait at least two major windows before changing it again.

To understand how weights should be updated in Step 10, we examine all possible behaviors of the search trajectory in successive major windows. We have identified four possible cases.

First, the trajectory does not stay within a feasible region, but goes from one feasible region to another through an infeasible region. During this time, $v_{max}(i)$ is zero when the trajectory is in the first feasible region, increased when it travels from the first feasible region to an infeasible region, and decreased when going from the infeasible region to the second feasible region. No oscillations will be observed because oscillations normally occur around an equilibrium point in one feasible region. In this case, w is not changed.

Second, the trajectory oscillates around an equilibrium point in a feasible region. This can be detected when the number of oscillations in each major window is larger than a threshold, the trajectory is not always in a feasible region, and the trend of v_{max} does not decrease. To determine whether the oscillations will subside eventually, we compute $\bar{v}_u - \bar{v}_{u+1}$, the difference of the average of $v_{max}(i)$ for two successive major windows u and $u+1$. If the

difference is not reduced reasonably, then we assume that the trajectory has not converged and decrease w accordingly.

Third, the search trajectory moves very slowly in a feasible region. This happens when w is very small, and the constraints dominate the search process. As a result, the objective value is improving very slowly and may eventually converge to a poor value. This situation can be identified when the trajectory remains within a feasible region in two successive major windows and is improving in successive major windows, but the improvement of the objective is not fast enough and is below a threshold. Obviously, we need to increase w in order to speed up the improvement of the objective. If the objective remains unchanged, then the trajectory has converged and w will not be changed.

Finally, the trajectory does not oscillate when it starts from a point in a feasible region, but rather goes outside the feasible region and then converges to a point on the boundary of the feasible region. In this case, a large w on the objective makes it more difficult to satisfy the constraints, causing the trajectory to move slowly back to the feasible region. At this time, an appropriate decrease of w will greatly shorten the convergence time.

Table 4.1 shows the results after applying dynamic weight adaptation in *DLM-General* to solve the same benchmark problem. Comparing the results with those of static weights, we see that all the searches can now converge with better average solutions.

4.1.4 Global Search

We apply global search in DLM by guiding a trajectory away from points visited in the past [229]. The key idea is to store in a queue, $\mathcal{Q}$, a set of points visited recently in a trajectory and to avoid these points explicitly in the future by imposing penalties when the trajectory gets close to any of them. Since these penalties are to be minimized, a search is therefore pulled away from those previously visited points and local minima are overcome

correspondingly. The idea is similar to that of the local-minimum penalty methods [34, 65] that prevent multiple visits to the same local minimum by adding to the objective a penalty term on each local minimum found.

We have modified the original objective function, leading to the following new augmented Lagrangian function:

$$L'_d(x, \lambda) = L_d(x, \lambda) - |f(x)| \cdot distance\_penalty(x) \tag{4.5}$$

For efficiency in implementation, we limit $\mathcal{Q}$ to contain a fixed number q of points, $s_1, \ldots s_q$, visited previously and manage them in a FIFO fashion. Further, we update $\mathcal{Q}$ by inserting the point reached after examining all n dimensions at the end of each round robin. The size of the queue, q, needs to be selected properly: if it is too small, then its effect on *distance_penalty* is small; if it is too large, then the overhead in updating $\mathcal{Q}$ and in calculating (4.5) is high.

To avoid bias when some dimensions have larger ranges, we normalize the Manhattan distance in the j^{th} dimension computed between the current trajectory x_j and $s_{i,j}$ by dividing the distance by the size of the variable range $R_j^u - R_j^l$. Experimentally, we determine *distance_penalty* to be as follows:

$$distance\_penalty(x) = \psi \cdot tan^{-1}\left(100 \cdot \frac{\sum_{i=1}^{q}\sum_{j=1}^{n}\frac{||x_j - s_{i,j}||}{R_j^u - R_j^l}}{n \cdot q}\right), \tag{4.6}$$

where $\psi = 0.018$ was determined experimentally. It should be obvious that $distance\_penalty(x)$ will have a large penalty on $L'_d(x, \lambda)$ when x is close to one of the previously visited points, and will have a small value when x is far away from all previously visited points. The effects of the size of $\mathcal{Q}$ on selected benchmark problems are tested in the following section.

4.1.5 Relax-and-Tighten Strategy for Handling Equality Constraints

In solving benchmark constrained NLPs, we found some benchmarks with equality constraints (not the same as equality constraints transformed from inequality constraints using (3.36)) that are very hard to satisfy. (An equality constraint is considered satisfied if its violation $\Phi \leq 10^{-5}$.) This is expected because it is difficult for a neighborhood search based on random sampling to pinpoint exactly feasible solutions that satisfy multiple equality constraints.

To address this difficulty, we propose a strategy called *relax-and-tighten*. Its key idea is to first relax equality constraint $h(x) = 0$ to inequality constraint $|h(x)| \leq \varepsilon$, where ε is a positive threshold to be determined. The relaxed problem should be easier to solve by DLM as it has a larger feasible space. After obtaining a feasible point x^* to the relaxed problem, we update ε to $\min(V_{max}, \varepsilon)/v$, where v is a positive constant larger than 1, and V_{max} is the maximum violation of the current solution x^* with respect to the original problem with equality constraints. The relaxed inequality constraints are then tightened gradually until the original equality constraints are satisfied.

In order for relax-and-tighten to work well, the initial amount of constraint relaxation must be chosen carefully. If ε is too large, then solutions to the relaxed problem are not likely to be good starting points for problems with tightened constraints. In contrast, if ε is too small, then the relaxed inequality constraints may be as hard to satisfy as the original equality constraints. Similarly, it is important to determine the amount that constraints should be tightened after a feasible solution to the relaxed problem has been found. If the step size for constraint tightening is too small, then it will take a long time to find a feasible solution that satisfies the equality constraints. In contrast, if this step size is too large, then DLM may not be able find a feasible solution. Empirically, we found good performance by

setting the initial ε to V^0_{max}/v and by updating ε to $\min(V_{max}, \varepsilon)/v$ after a feasible solution to the relaxed problem has been found, where V^0_{max} is the initial maximum violation of the equality constraints and v is a constant to be determined.

As an example, consider Problem 7.4 in [57]. With 38 variables, 23 equality constraints, and $V^0_{max} > 100.0$, the problem is quite difficult to solve. If we apply DLM to solve a MINLP version of the problem from a randomly generated starting point, the best solution found has $V_{max} > 0.1$ after running more than 20 hours on a Pentium III 500-MHz computer. In contrast, applying the relax-and-tighten strategy led to a solution with $V_{max} < 10^{-5}$ in less than 1000 seconds with $v = 1.2$. Our experience shows that relax-and-tighten allows better solutions with smaller maximum violations to be found each time after the constraints are tightened.

4.1.6 Duration of Each Run

A general goal of *DLM-General* is to find high-quality solutions efficiently. Specifically, given a particular objective value f^*, we want minimize the average time of multiple DLM runs in order to find feasible solutions with objectives equal to or better than f^*.

Let N be the maximum number of iterations (or round robins) allowed in DLM, and P_R be the probability of locating a solution with prescribed solution quality f^* in N iterations. Clearly, P_R is monotonically non-decreasing with increasing N. The *average completion time* of finding a solution with quality f^* using multiple runs is, therefore [204]:

$$\sum_{i=1}^{\infty} i \cdot N \cdot (1 - P_R)^{i-1} \cdot P_R = \frac{N}{P_R} \tag{4.7}$$

Intuitively, if N is very small, then it is very unlikely for *DLM-General* to find a desired solution, leading to a small P_R and large $\frac{N}{P_R}$. On the other hand, when N increases, P_R

will increase until it approaches one. At that point, further increases in N will cause $\frac{N}{P_R}$ to increase monotonically. Figure 4.4 illustrates the behavior of (4.7) when applying *DLM-General* to solve a typical continuous constrained NLP G5 [135, 121] with a desired solution quality set to be 1.25 times the optimal solution. The curve demonstrates that $\frac{N}{P_R}$ is convex with respect to $log(N)$.

Wah and Chen [204] have explored a novel way of designing an optimal anytime constrained simulated annealing (CSA) algorithm. In their algorithm (CSA_{AT-ID}), iterative deepening is applied to the cooling schedule. Using a fast initial cooling schedule, the cooling schedule is doubled after a certain number of consecutive CSA runs until eventually a feasible solution with a prescribed solution quality has been located. They [204] have proved that the total time spent on CSA_{AT-ID} using iterative deepening is of the same order as the time of one CSA run with an optimal cooling schedule, where an optimal cooling schedule [204] is defined as a schedule that leads to the shortest average total CPU time of multiple CSA runs in order to find a solution of prescribed quality.

Due to the convexity of $\frac{N}{P_R}$ with respect to $log(N)$, as demonstrated in Figure 4.4, the same idea of iterative deepening can be applied to determine the duration of each DLM run in order to minimize the total average completion time of multiple DLM runs in locating solutions with prescribed quality.

In our implementation, we start a DLM run with a maximum number, N^0_{max}, of iterations. If a solution of prescribed quality is not found after three consecutive restarts, the maximum number of iterations will be doubled. We define a *DLM run* as a sequence of DLM restarts, each using a number of iterations that is increasing in a geometric fashion from the previous restart. Mathematically, the $(3i + j)^{th}$ DLM restart in a DLM run will have a maximum number of iterations equal to $2^i \cdot N^0_{max}$ for $i \geq 0$, and $j \in \{0, 1, 2\}$. In CSA_{AT-ID}, this kind of

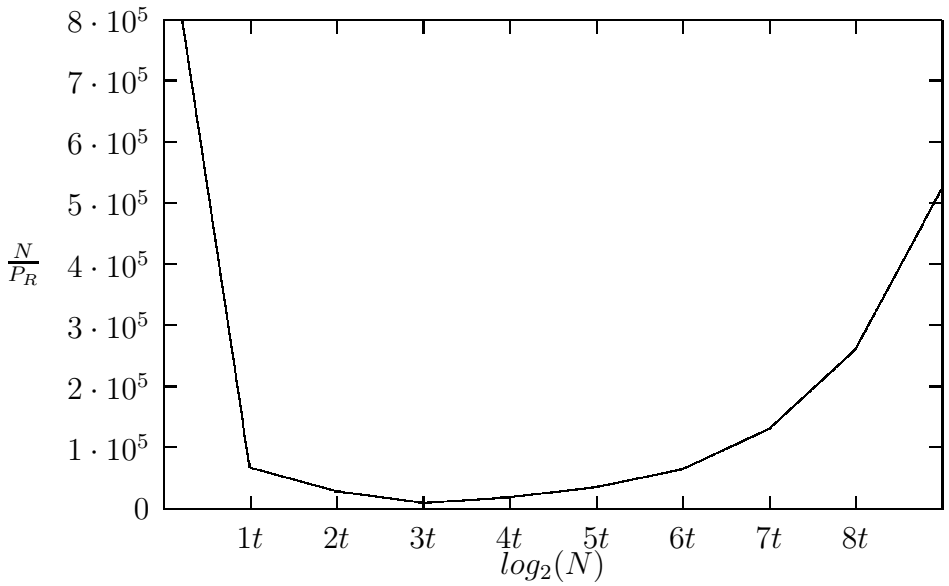


Figure 4.4: The average time to find a solution 1.25 times the optimal solution in $G5$ depicts a convex behavior with respect to the number of iterations of each run of *DLM- General*.

iterative deepening is stopped once a solution with prescribed quality has been found. This strategy, however, requires the knowledge of solution quality that is generally not available for new problems. In practice, we use the following heuristic stopping condition. A DLM run is stopped when either of the following two conditions is true: a) no improvement is found on the objective value after a consecutive number, $\mathcal{A}$, of restarts; b) no improvement is found on the maximum constraint violation after a consecutive number, $\mathcal{B}$, of restarts. Experimentally, we found $\mathcal{A} = 3$ and $\mathcal{B} = 6$ are good choices for most of the benchmark problems tested.

4.2 Performance Comparisons of Various Strategies

In this section, we apply *DLM-General* to solve 12 difficulty benchmark problems in order to determine the effects of different combinations of strategies on solution time and quality. Our goal is to select, if there exists, one combination of parameters or strategies that can generalize to these 12 test problems in terms of solution time and quality. These 12 problems

are: G1, G2 and G5 from [135, 121], and 2.1, 2.7.5, 5.2, 5.4, 6.2, 6.4, 7.2, 7.3 and 7.4 from [57]. Since these benchmark problems are continuous constrained NLPs, we have created the corresponding discrete and mixed-integer versions. These versions are done in such a way that allow solutions in the discrete or mixed-integer problems to be compared directly to their continuous counterparts.

We transform a continuous constrained NLP with n variables, $x_1, \ldots, x_n$, to a discrete or mixed-integer version as follows. In creating a discrete NLP, we discretize all variables in the original problem, whereas in creating an MINLP, we let variables with odd indices be continuous and those with even indices be discrete. In discretizing continuous variable x_j in the range $[R_j^l, R_j^u]$, we force it to take discrete values from the set:

$$A_j = \begin{cases} \left\{ R_j^l + (R_j^u - R_j^l)\frac{i}{s}, i = 0, 1, \ldots, s \right\} & \text{if } R_j^u - R_j^l < 1 \\ \left\{ R_j^l + \frac{i}{s}, i = 0, 1, \ldots, \lfloor (R_j^u - R_j^l)s \rfloor \right\} & \text{if } R_j^u - R_j^l \geq 1 \end{cases} \tag{4.8}$$

where $s = 10^7$. For example, if $R_j^u - R_j^l = 1$, then x_j will be discretized into a set of 10^7 discrete points. Obviously, given an MINLP with n dimensions, the discrete subspace created has at least $10^{7\lfloor n/2 \rfloor}$ points, a space so huge that it is impossible for any algorithm to enumerate. Using such a finely discretized space allows us to compare directly the solutions in the original continuous versions and those found by DLM in the transformed discrete and mixed-integer versions.

Next, we compare various combinations of parameters used in *DLM-General*. For neighborhood search, we tested three different neighborhood generators governed by Cauchy (denoted by $N1$), Gaussian (denoted by $N2$), and uniform (denoted by $N3$) distributions. For global search, we tested three different tabu-list ($\mathcal{Q}$) sizes: 0 (denoted by $S1$), 6 (denoted by $S2$) and 10 (denoted by $S3$). For the relax-and-tighten strategy, we tested three different factors for tightening and relaxing constraints, v: ∞ (denoted by $T1$), 1.2 (denoted by

$T2$) and 1.5 (denoted by $T3$). Note that when $S1$ is adopted, global search is actually not performed because the size of the tabu list $\mathcal{Q}$ is zero. Further, when $T1$ is employed, relax-and-tighten is not used because $V^0_{max}/v = V^0_{max}/\infty = 0$. For simplicity, we use N?-S?-T? to represent a combination of parameters/strategies chosen. As an example, N1-S3-T2 means that *DLM-General* takes a Cauchy neighborhood generator, uses a tabu list $\mathcal{Q}$ of size 6, and sets v to 1.2.

For each combination of parameters, we evaluate the aforementioned 12 problems (mixed-integer versions) from randomly generated starting points until a feasible solution is found. We call one of the above runs a feasible run that consists of one or more DLM runs (defined in Section 4.1.6) and that finds a feasible solution in the last DLM run. We repeated until 100 feasible runs were performed and recorded the CPU times and corresponding solution qualities. For a given combination of parameters, let $t_x(i)$ and $f_x(i)$ be the CPU time and objective of the i^{th} feasible run. In order to compare all these combinations of parameters, we perform normalization by using one set of parameters as a reference. In our experiments, we select N1-S3-T2 as the reference for normalizing all (27 in total) combinations of parameters as follows:

$$r_t(i) = \begin{cases} t_x(i)/t_r(i) - 1.0, & \text{if} \quad t_x(i) > t_r(i) \\ 1.0 - t_r(i)/t_x(i), & \text{if} \quad t_x(i) \leq t_r(i) \end{cases} \tag{4.9}$$

$$r_f(i) = (f_x(i) - f_r(i))/|f_r(i)| \tag{4.10}$$

where $t_r(i)$ and $f_r(i)$ are the CPU time and objective-function value of the i^{th} feasible run for the reference strategy N1-S3-T2, $i = 1, 2, \ldots, 100$. In our normalizations, (4.9) measures the symmetric speedup [205] in order to give equal weights to speedups and slowdowns in CPU times. On the other hand, because objective-function values might be negative, (4.10) measures relative improvements and degradations in objectives. The average r_t and r_f of all

the 100 normalized CPU times and solution qualities are computed as follows:

$$r_f = \frac{1}{100}\sum_{i=1}^{100} r_f(i), \qquad r_t = \frac{1}{100}\sum_{i=1}^{100} r_t(i) \tag{4.11}$$

A combination of parameters is considered better than reference N1-S3-T2 if both r_f and r_t are negative.

Figures 4.5 thru 4.7 show the results on evaluating the 12 difficult mixed-integer constrained NLP benchmarks. The left three diagrams (indexed by a, d and g) of Figures 4.5 thru 4.7 lead us to conclude that, without relax-and-tighten, some problems with many equality constraints (7.4 for example) elude solutions even after 100 runs. On the other hand, if v is set to be too large ($v = 1.5$ for instance), *DLM-General* cannot find solutions to all the 12 benchmark problems, as shown in Figure 4.6. Global search truly improves the solution quality. For example, when compared to the reference, N1-S2-T2 finds much better CLM_{dn} than N1-S1-T2. Moreover, solution times based on a Cauchy distribution outperform those based on the other two distributions. This is expected because the Cauchy distribution has a long, flat tail that enables a search to explore more effectively wider regions in the search space. Overall, we conclude that strategies/parameters combination N1-S2-T2 gives the best performance; hence, we use it to solve general constrained NLPs in the rest of the experiments.

Finally, Figures 4.8 thru 4.10 plot the performance of *DLM-General* using N1-S2-T2 (Cauchy, tabu-list $\mathcal{Q}$ size $= 6$ and $v = 1.2$) on the 12 benchmark problems in discrete, continuous and mixed-integer forms, respectively. Each point in a graph represents a pair of CPU time and solution quality for one feasible run of *DLM-General*. As a local search method in Lagrangian space, *DLM-General* usually finds different solutions when using different

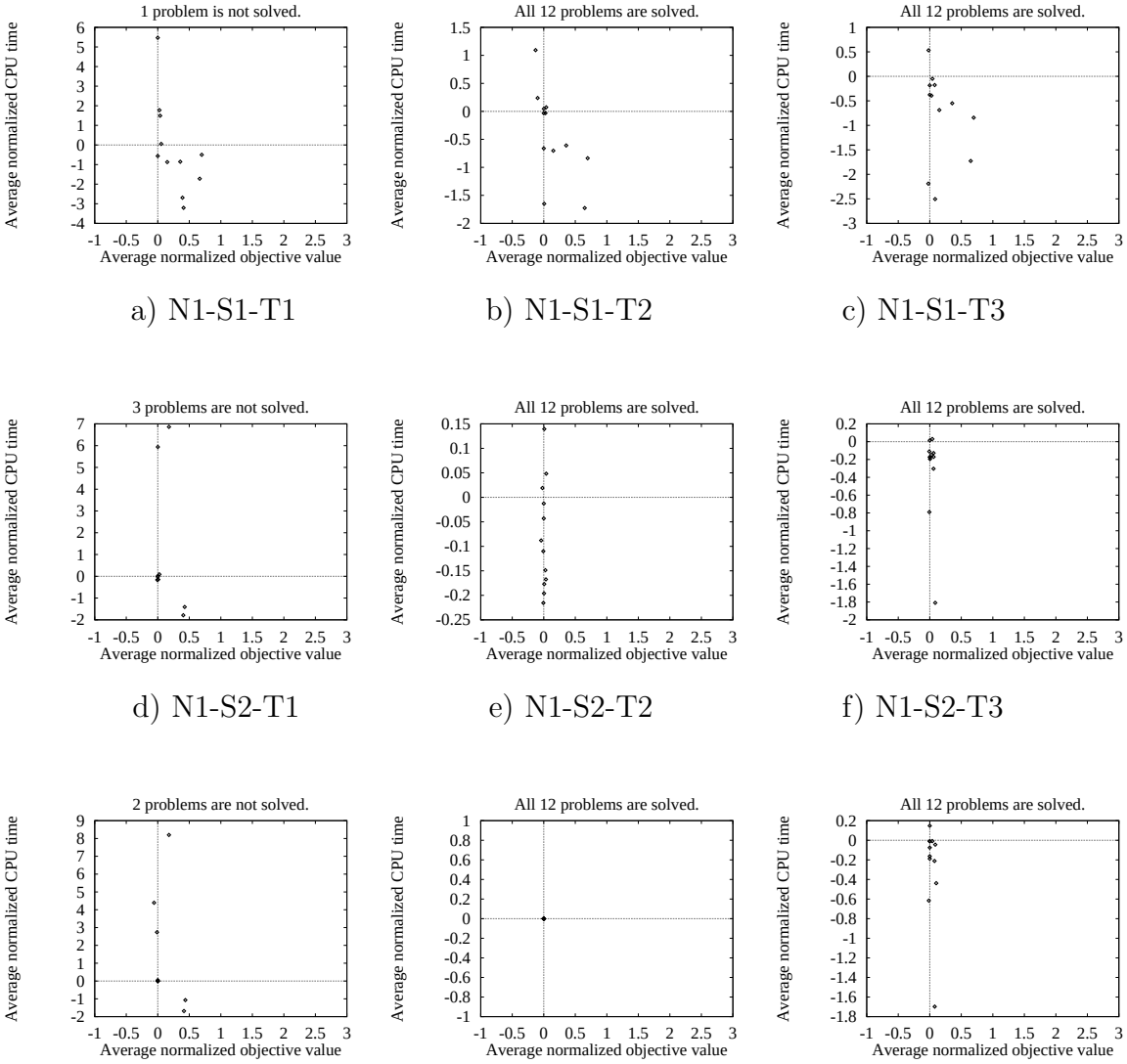
randomly generated starting points. Within 100 feasible runs, *DLM-General* is able to find good solutions for most of the problems tested.

4.3 Experimental Results on Constrained NLP Benchmarks

In this section, we fully evaluate *DLM-General* on a comprehensive set of constrained NLP benchmarks: G1 thru G10 developed in the GA community [135, 121], all the 29 Floudas and Pardalos' benchmarks in [57], and all nonlinear problems from CUTE [26], a constrained and unconstrained testing environment. These problems have objective functions of various types (linear, quadratic, cubic, polynomial, and nonlinear) and linear/nonlinear constraints of equalities and inequalities.

Tables 4.2 thru 4.4 present, respectively, the performance of *DLM-General* on discrete, continuous and mixed-integer versions of G1 thru G10 [135, 121]. The first four columns of Tables 4.2 and 4.4 list, respectively, the problem identifier, number of variables, total number of constraints, and number of equality constraints. The fifth column shows the best-known solutions for the original continuous version. The next twelve columns list, respectively, the success ratios (SR) and average CPU times needed to find solutions that differ within 0%, 1%, 5%, 10%, 15% and 25% from the best-known continuous solution. As expected, the CPU times increase and success ratios decrease as better solution qualities are specified. Finally, the last column of Tables 4.2 thru 4.4 lists the CPU times required by CSA on the same set of NLP benchmarks.

Tables 4.2 thru 4.4 also compare the performance of DLM and CSA [213, 211]. For a fair comparison, CSA was run multiple times on the same computer using the same sequence of starting points. We do not report the quality of solutions found by CSA as it always found

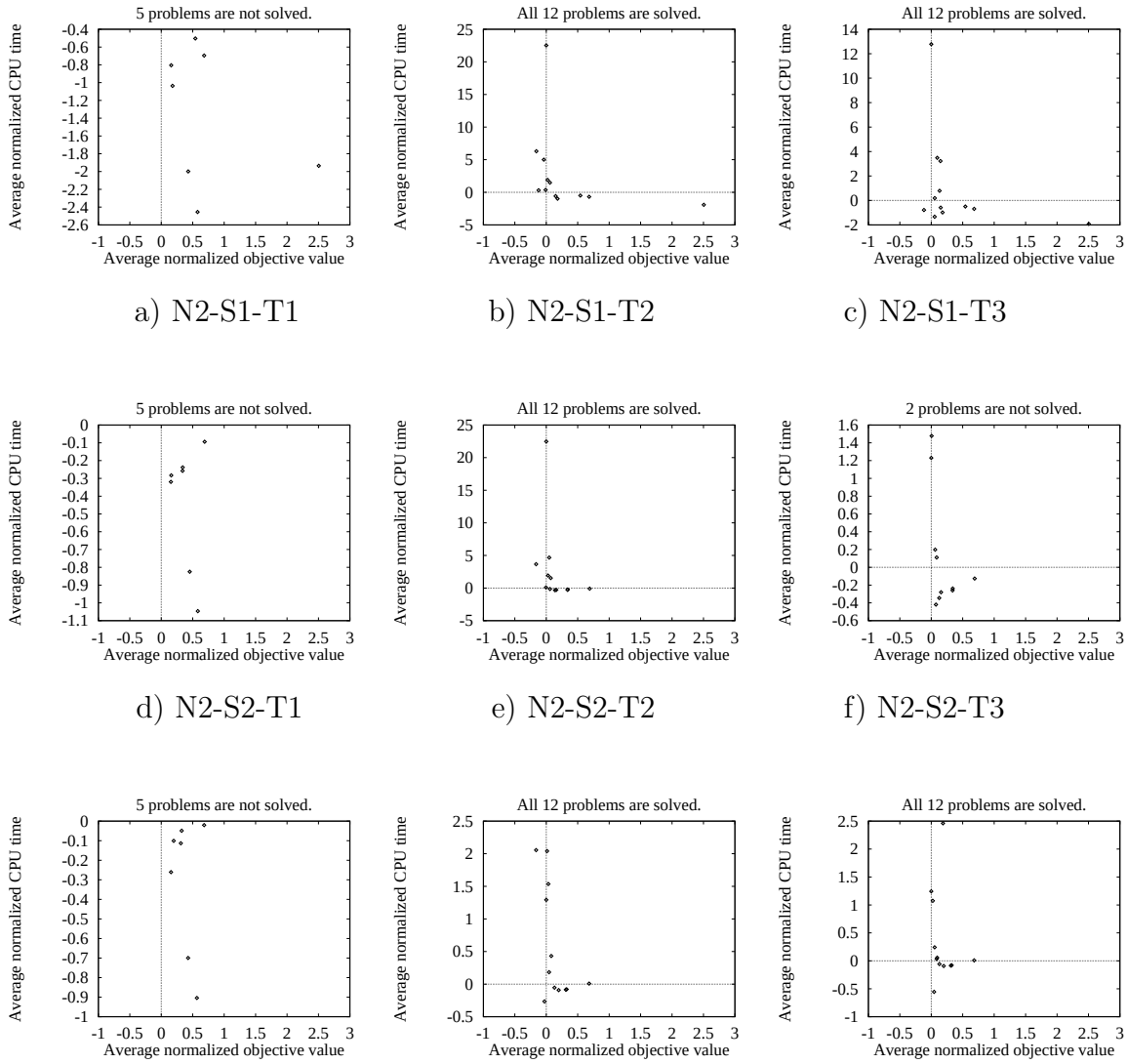


a) N1-S1-T1 b) N1-S1-T2 c) N1-S1-T3

d) N1-S2-T1 e) N1-S2-T2 f) N1-S2-T3

g) N1-S3-T1 h) N1-S3-T2 i) N1-S3-T3

Figure 4.5: Comparisons of average relative CPU times and average relative solution qualities under different parameters/strategies (using Cauchy distribution) normalized with respect to the reference strategy N1-S3-T2 (Cauchy distribution, tabu-list $\mathcal{Q}$ size = 10, $v = 1.2$) in solving 12 difficult mixed-integer constrained NLPs. All runs were made on a Pentium III 500MHz computer with Solaris 7.

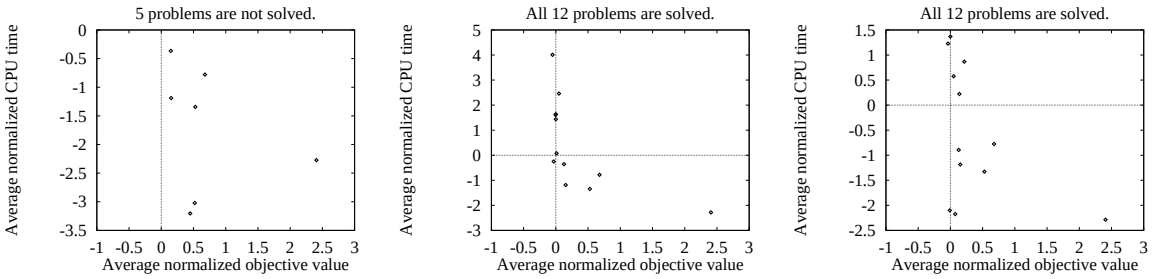


a) N2-S1-T1 b) N2-S1-T2 c) N2-S1-T3

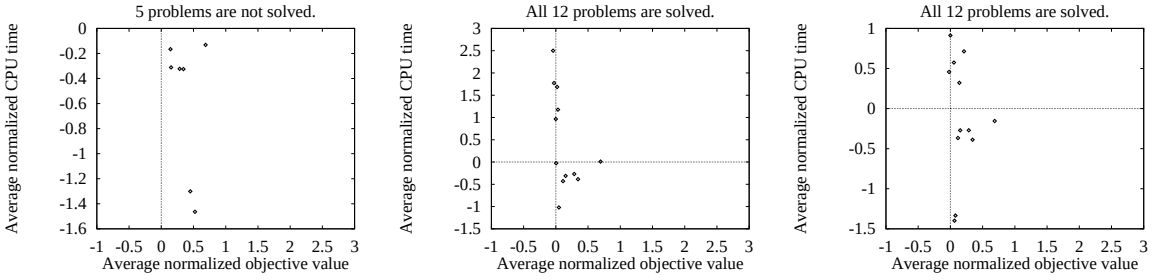
d) N2-S2-T1 e) N2-S2-T2 f) N2-S2-T3

g) N2-S3-T1 h) N2-S3-T2 i) N2-S3-T3

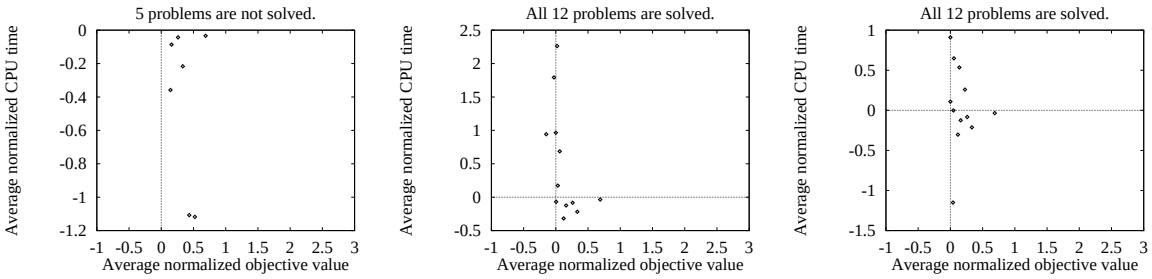
Figure 4.6: Comparisons of average relative CPU times and average relative solution qualities under different parameters/strategies (using Gaussian distribution) normalized with respect to the reference strategy N1-S3-T2 (Cauchy distribution, tabu-list $\mathcal{Q}$ size = 10, $v = 1.2$) in solving 12 difficult mixed-integer constrained NLPs. All runs were made on a Pentium III 500MHz computer with Solaris 7.



a) N3-S1-T1 b) N3-S1-T2 c) N3-S1-T3



d) N3-S2-T1 e) N3-S2-T2 f) N3-S2-T3



g) N3-S3-T1 h) N3-S2-T2 i) N3-S3-T3

Figure 4.7: Comparisons of average relative CPU times and average relative solution qualities under different parameters/strategies (using uniform random distribution) normalized with respect to the reference strategy N1-S3-T2 (Cauchy distribution, tabu-list $\mathcal{Q}$ size $= 10$, $v = 1.2$) in solving 12 difficult mixed-integer constrained NLPs. All runs were made on a Pentium III 500MHz computer with Solaris 7.

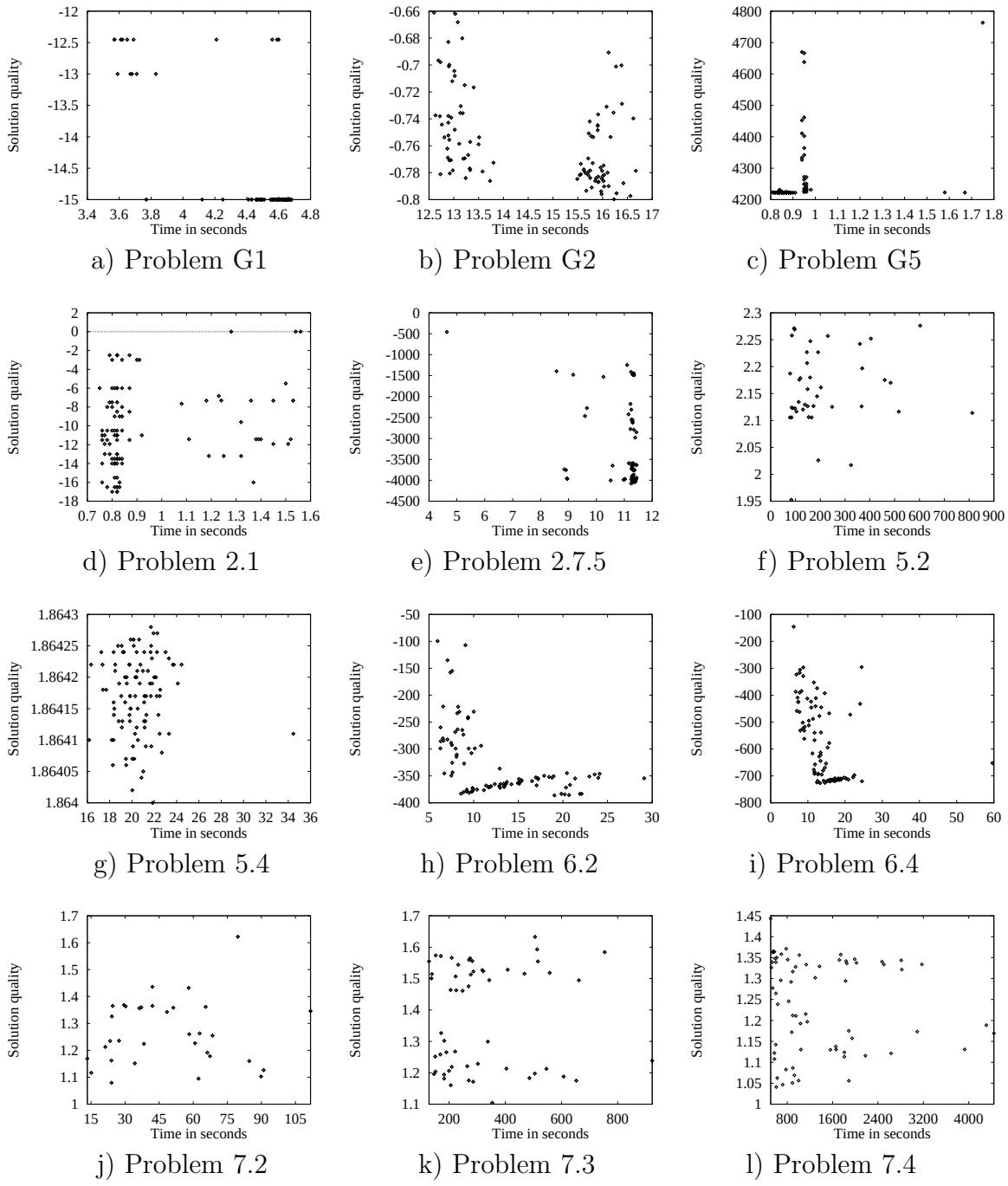


Figure 4.8: Performance of *DLM-General* using N1-S2-T2 (Cauchy, tabu-list $\mathcal{Q}$ size = 6 and $v = 1.2$) on 12 difficult derived *discrete* constrained NLPs. All runs were made on a Pentium III 500MHz computer with Solaris 7. The solutions in Problems 7.2-7.4 have been normalized by their best-known solutions.

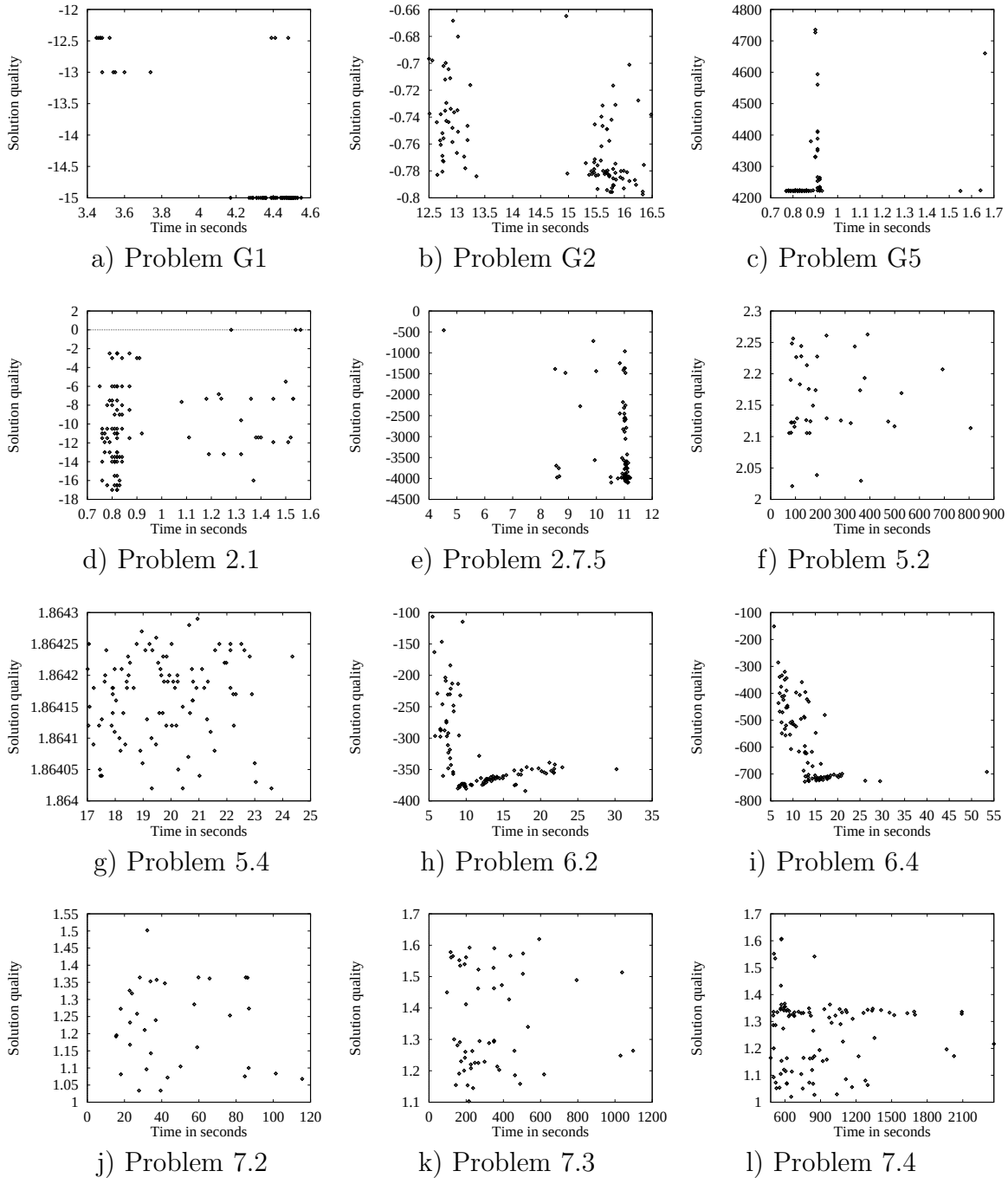


Figure 4.9: Performance of *DLM-General* using N1-S2-T2 (Cauchy, tabu-list $\mathcal{Q}$ size = 6 and $v = 1.2$) on 12 difficult *continuous* constrained NLPs. All runs were made on a Pentium III 500MHz computer with Solaris 7. The solutions in Problems 7.2-7.4 have been normalized by their best-known solutions.

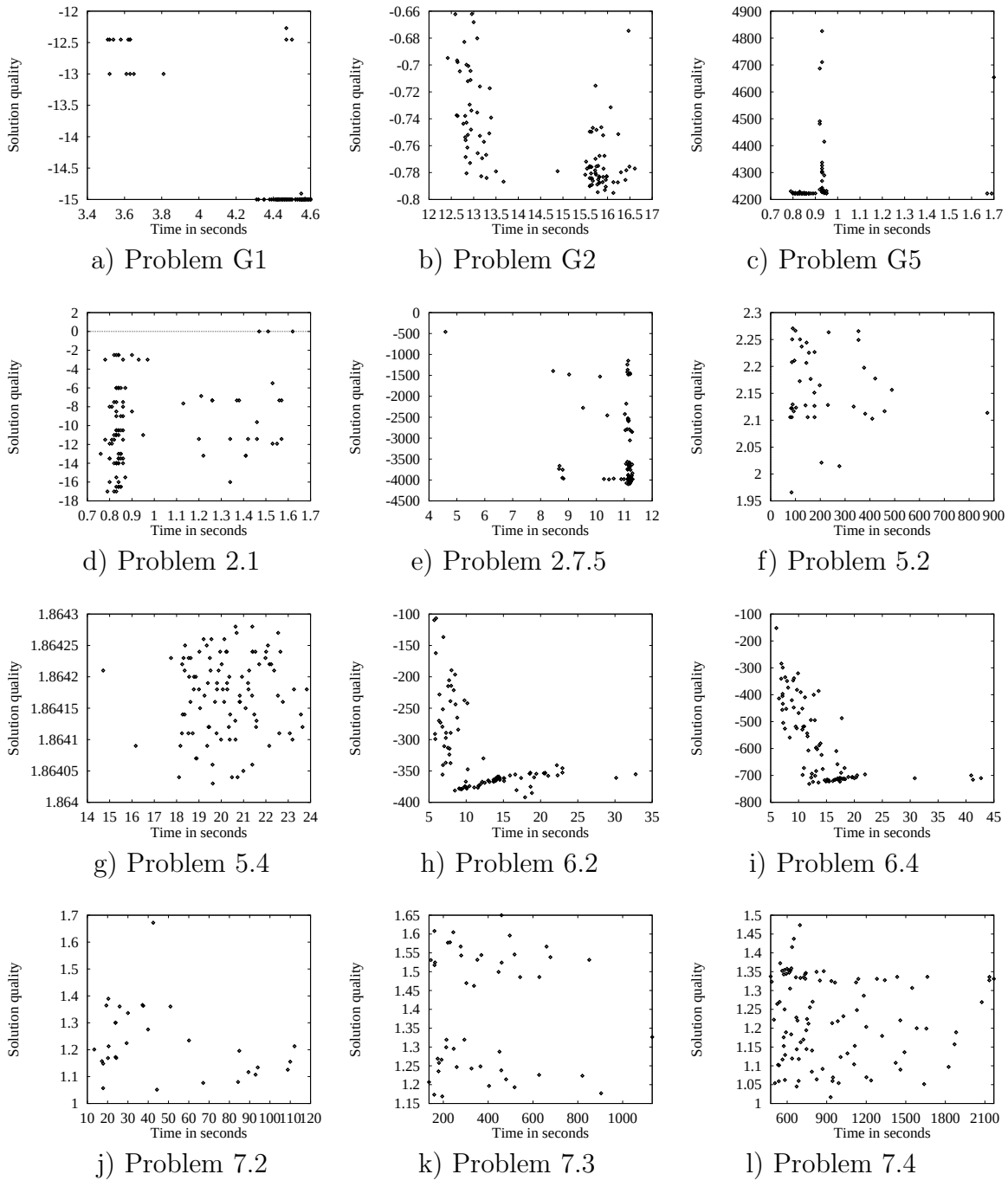


Figure 4.10: Performance of *DLM-General* using N1-S2-T2 (Cauchy, tabu-list $\mathcal{Q}$ size = 6 and $v = 1.2$) on 12 difficult derived *mixed-integer* constrained NLPs. All runs were made on a Pentium III 500MHz computer with Solaris 7. The solutions in Problems 7.2-7.4 have been normalized by their best-known solutions.

Table 4.2: Performance comparison of *DLM-General* and CSA in solving discrete constrained NLPs derived from continuous constrained NLPs G1-G10 [135, 121]. All timing results in seconds were collected on a Pentinum III 500-MHz computer with Solaris 7. For all problems except G2, CSA was able to find the optimal solutions in the times reported. For G2, CSA has a 97% success ratio. '-' stands for no solution found for the solution quality specified within 100 feasible DLM runs. 'SR' stands for success ratio of finding solutions with specified quality within 100 feasible DLM runs.

Problem	Var.	Constraints		Best.	0%		1%		5%		10%		15%		25%		CSA
ID	no.	total	eq.	known	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	time
G1	13	35	0	-15	85	5.26	85	5.26	85	5.26	85	5.26	90	4.97	100	4.47	19.5
G2	20	42	0	-0.80362	0	-	3	486.53	52	28.07	83	17.59	95	15.36	100	14.60	54.0
G3	20	41	1	1.0	92	12.43	100	11.43	100	11.43	100	11.43	100	11.43	100	11.43	37.1
G4	5	16	0	-30665	7	14.87	46	2.26	100	1.04	100	1.04	100	1.04	100	1.04	3.12
G5	4	13	3	4221.9	20	4.40	85	1.03	93	0.95	96	0.92	99	0.89	99	0.89	1.95
G6	2	6	0	-6961.81	1	13.30	49	0.27	49	0.27	49	0.27	49	0.27	49	0.27	0.644
G7	10	28	0	24.3062	51	6.11	100	3.12	100	3.12	100	3.12	100	3.12	100	3.12	16.1
G8	2	6	0	0.095825	0	-	0	-	96	0.30	96	0.30	97	0.30	97	0.30	0.792
G9	7	18	0	680.63	0	-	100	1.55	100	1.55	100	1.55	100	1.55	100	1.55	4.75
G10	7	18	0	1.00	0	-	5	39.40	66	2.98	91	2.16	95	2.07	98	2.01	5.82

the best solution. The results show that if the requirement on solution quality is around 5% to 25%, DLM usually takes less time than CSA. DLM, however, has no advantage over CSA in terms of CPU times when locating optimal or near-optimal solutions. This is expected as DLM is a local search, whereas CSA is a global-optimization algorithm. Besides CSA and DLM, we know of no other algorithms in the literature that solve these problems in their discrete and mixed-integer versions.

Next, Tables 4.5 thru 4.7 report, respectively, the performance of DLM on solving continuous and the corresponding discrete and mixed-integer versions of Floudas and Pardalos' constrained NLP [57]. Besides listing columns with similar meaning as those in Table 4.2, Tables 4.5 thru 4.7 also list the timing results of 100 runs of CSA, as CSA did not find the best solution in each run. The results lead to similar observations as those on evaluating G1 thru G10.

Table 4.3: Performance comparison of *DLM-General* and CSA in solving continuous constrained NLPs: G1-G10 [135, 121]. All timing results in seconds were collected on a Pentinum III 500-MHz computer with Solaris 7. For all problems except G2, CSA was able to find the optimal solutions in the times reported. '-' stands for no solution found for the solution quality specified within 100 feasible DLM runs. 'SR' stands for success ratio of finding solutions with specified quality within 100 feasible DLM runs.

Problem	Var.	Constraints		Best.	0%		1%		5%		10%		15%		25%		CSA
ID	no.	total	eq.	known	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	time
G1	13	35	0	-15	85	5.09	85	5.09	85	5.09	85	5.09	90	4.81	100	4.33	15.2
G2	20	42	0	-0.80362	0	-	3	488.30	56	26.16	87	16.84	97	15.10	100	14.65	53.8
G3	20	41	1	1.0	99	11.30	100	11.19	100	11.19	100	11.19	100	11.19	100	11.19	37.2
G4	5	16	0	-30665	10	10.01	45	2.22	100	1.00	100	1.00	100	1.00	100	1.00	1.76
G5	4	13	3	4221.9	20	4.22	85	0.99	94	0.90	96	0.88	99	0.85	99	0.85	1.62
G6	2	6	0	-6961.81	1	12.80	49	0.26	49	0.26	49	0.26	49	0.26	49	0.26	0.518
G7	10	28	0	24.3062	62	4.83	100	3.00	100	3.00	100	3.00	100	3.00	100	3.00	14.0
G8	2	6	0	0.095825	0	-	0	-	93	0.30	93	0.30	94	0.30	94	0.30	0.798
G9	7	18	0	680.63	0	-	100	1.47	100	1.47	100	1.47	100	1.47	100	1.47	4.48
G10	7	18	0	1.00	1	195.30	8	24.41	59	3.31	85	2.30	93	2.10	95	2.06	4.69

Finally, we apply DLM to solve continuous constrained NLP benchmarks in CUTE [26]. We select from CUTE those problems with at least a nonlinear objective or one or more nonlinear constraints. These CUTE problems are all minimization problems; some of which were constructed specifically by researchers to test optimization algorithms, while others were from real applications, such as semiconductor analysis in physics, chemical reactions in chemistry, economic equilibrium in economic analysis, and computer production planning in operations research. Both the number of variables and the number of constraints in CUTE can be as large as several thousands.

Table 4.8 reports the results on solving the selected nonlinear constrained problems in CUTE [26] using the given starting point specified for each problem. The first two columns show, respectively, the problem IDs and the number ($n_v = n$) of variables. 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

Table 4.4: Performance comparison of *DLM-General* and CSA in solving constrained MINLPs based on continuous constrained NLPs G1-G10 [135, 121]. All timing results in seconds were collected on a Pentinum III 500-MHz computer with Solaris 7. For all problems except G2, CSA was able to find the optimal solutions in the times reported. For G2, CSA has a 95% success ratio. '-' stands for no solution found for the solution quality specified within 100 feasible DLM runs. 'SR' stands for success ratio of finding solutions with specified quality within 100 feasible DLM runs.

Problem	Var.	Constraints		Best.	0%		1%		5%		10%		15%		25%		CSA
ID	no.	total	eq.	known	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	time
G1	13	35	0	-15	84	5.23	85	5.17	85	5.17	85	5.17	90	4.88	100	4.40	18.8
G2	20	42	0	-0.80362	0	-	0	-	57	25.62	82	17.81	94	15.54	100	14.61	53.4
G3	20	41	1	1.0	99	11.43	100	11.32	100	11.32	100	11.32	100	11.32	100	11.32	36.2
G4	5	16	0	-30665	9	11.41	46	2.23	100	1.03	100	1.03	100	1.03	100	1.03	2.98
G5	4	13	3	4221.9	15	5.77	85	1.02	94	0.92	96	0.90	100	0.87	100	0.87	1.88
G6	2	6	0	-6961.81	0	-	49	0.27	49	0.27	49	0.27	49	0.27	49	0.27	0.605
G7	10	28	0	24.3062	59	5.17	97	3.14	100	3.05	100	3.05	100	3.05	100	3.05	15.6
G8	2	6	0	0.095825	0	-	0	-	94	0.30	95	0.30	96	0.29	96	0.29	0.759
G9	7	18	0	680.63	0	-	100	1.50	100	1.50	100	1.50	100	1.50	100	1.50	4.61
G10	7	18	0	1.00	0	-	9	22.48	65	3.11	89	2.27	96	2.11	96	2.11	5.43

constraints (n_{ni}). The last eight columns show the solutions and CPU times that we obtain by using LANCELOT [42, 123], DONLP2 [192], CSA and DLM, respectively.

From the CPU times and solution qualities listed in Table 4.8, we conclude that methods based on sampling, like CSA and DLM, cannot compete in CPU times with methods utilizing gradients, like DONLP2 and LANCELOT. However, for problems like AVION2, CRESC4, HIMMELBJ, HS109, LAUNCH, and NET1, neither DONLP2 nor LANCELOT can solve them while CSA and DLM were able to find feasible solutions.

Table 4.5: Performance comparison of *DLM-General* and CSA in solving discrete constrained NLPs based on Floudas and Pardalos' continuous constrained NLPs [57]. All timing results in seconds were collected on a Pentinum III 500-MHz computer with Solaris 7. '-' stands for no solution found for the solution quality specified within 100 feasible DLM runs. 'SR' stands for success ratio of finding solutions with specified quality within 100 feasible DLM runs.

Benchmark Continuous NLPs					DLM												CSA			
Problem	Var.	Constraints		Best.	0%		1%		5%		10%		15%		25%		best	avg.	frac.	avg.
ID	no.	total	eq.	known	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	diff.	diff.	best	time
2.1	5	11	0	-17	3	32.87	3	32.87	8	12.32	13	7.58	13	7.58	33	2.99	0	0.4	44	1.59
2.2	6	2	0	-213	5	22.76	8	14.22	100	1.14	100	1.14	100	1.14	100	1.14	0	0	100	2.18
2.3	13	35	0	-15	85	3.42	85	3.42	85	3.42	85	3.42	90	3.23	99	2.93	0	0	100	17.4
2.4	6	17	0	-11	1	123.20	9	13.69	69	1.79	69	1.79	69	1.79	97	1.27	0	0	100	3.09
2.5	10	31	0	-268	46	5.29	99	2.46	100	2.44	100	2.44	100	2.44	100	2.44	0	0	100	24.2
2.6	10	31	0	-39	18	11.27	18	11.27	18	11.27	26	7.80	27	7.51	52	3.90	0	0.1	96	11.0
2.7.1	20	50	0	-394.75	0	-	3	395.17	26	45.60	26	45.60	27	43.91	37	32.04	0	1.96	98	105.1
2.7.2	20	50	0	-884.75	0	-	23	50.42	38	30.52	60	19.33	69	16.81	100	11.60	0	6.52	92	102.7
2.7.3	20	50	0	-8695.0	0	-	0	-	3	377.07	62	18.25	95	11.91	100	11.31	0	0	100	98.9
2.7.4	20	50	0	-754.75	0	-	6	195.62	6	195.62	7	167.67	7	167.67	98	11.98	0	0	100	102.7
2.7.5	20	50	0	-4150.4	0	-	0	-	46	24.00	62	17.81	74	14.92	74	14.92	0	47.4	75	115.6
2.8	24	10	10	15639	0	-	0	-	0	-	0	-	0	-	0	-	0	0	100	81.7
3.1	8	22	0	7049.33	0	-	7	308.61	28	77.15	92	23.48	100	21.60	100	21.60	0	0	100	5.82
3.2	4	16	0	-30665.5	53	1.65	98	0.89	100	0.88	100	0.88	100	0.88	100	0.88	0	0	100	3.12
3.3	6	18	0	-310	0	-	10	14.62	33	4.43	48	3.05	58	2.52	67	2.18	0	0	100	2.70
3.4	3	9	0	-4	2	16.45	92	0.36	94	0.35	94	0.35	94	0.35	96	0.34	0	0	100	1.05
4.3	4	9	1	-4.51	7	13.43	7	13.43	12	7.83	17	5.53	20	4.70	30	3.13	0	0	100	1.78
4.4	4	9	1	-2.217	0	-	0	-	22	4.12	22	4.12	27	3.36	36	2.52	0	0.003	92	1.94
4.5	6	15	3	-13.4	9	20.20	11	16.53	29	6.27	75	2.42	80	2.27	85	2.14	0	0	100	3.95
4.6	2	6	0	-5.51	46	0.57	46	0.57	46	0.57	46	0.57	46	0.57	61	0.43	0	0	100	0.656
4.7	2	5	1	-16.74	28	0.89	99	0.25	99	0.25	99	0.25	99	0.25	99	0.25	0	0	100	0.601
5.2	46	86	36	1.567	0	-	0	-	0	-	0	-	0	-	1	9089.00	0.0	0.157	1	639.0
5.4	32	58	26	1.86	0	-	99	20.57	99	20.57	99	20.57	99	20.57	99	20.57	0	0.16	3	172.0
6.2	5	17	4	400	0	-	0	-	8	155.50	43	28.93	64	19.44	70	17.77	0	0	100	5.54
6.3	5	17	4	600	0	-	0	-	0	-	2	383.35	6	127.78	18	42.59	0	1.0	94	5.82
6.4	5	17	4	750	0	-	0	-	23	56.37	41	31.62	47	27.59	54	24.01	0	0	100	6.92
7.2	16	41	13	1.0	0	-	0	-	0	-	3	543.57	7	232.96	18	90.59	0.03	0.25	1	48.6
7.3	27	64	19	1.0	0	-	0	-	0	-	0	-	1	17395.80	20	869.79	0.15	0.46	1	153.8
7.4	38	90	23	1.0	0	-	0	-	1	99885.00	7	14269.29	15	6659.00	25	3995.40	0.03	0.15	1	332.3

Table 4.6: Performance comparison of *DLM-General* and CSA in solving Floudas and Pardalos' continuous constrained NLPs [57]. All timing results in seconds were collected on a Pentinum III 500-MHz computer with Solaris 7. '-' stands for no solution found for the solution quality specified within 100 feasible DLM runs. 'SR' stands for success ratio of finding solutions with specified quality within 100 feasible DLM runs.

Benchmark Continuous NLPs					DLM												CSA			
Problem	Var.	Constraints		Best.	0%		1%		5%		10%		15%		25%		best	avg.	frac.	avg.
ID	no.	total	eq.	known	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	diff.	diff.	best	time
2.1	5	11	0	-17	3	31.53	3	31.53	8	11.82	13	7.28	13	7.28	33	2.87	0	0.32	52	1.26
2.2	6	2	0	-213	12	9.04	17	6.38	100	1.08	100	1.08	100	1.08	100	1.08	0	0	100	1.82
2.3	13	35	0	-15	85	3.26	85	3.26	85	3.26	85	3.26	90	3.08	100	2.77	0	0	100	15.2
2.4	6	17	0	-11	2	58.20	8	14.55	70	1.66	70	1.66	70	1.66	99	1.18	0	0	100	2.75
2.5	10	31	0	-268	46	5.13	99	2.39	100	2.36	100	2.36	100	2.36	100	2.36	0	0	100	23.1
2.6	10	31	0	-39	21	9.37	21	9.37	21	9.37	29	6.79	29	6.79	51	3.86	0	0.1	96	10.4
2.7.1	20	50	0	-394.75	0	-	0	-	24	47.89	24	47.89	24	47.89	34	33.81	0	0	100	79.5
2.7.2	20	50	0	-884.75	0	-	23	49.53	35	32.55	69	16.51	75	15.19	97	11.74	0	0	100	98.2
2.7.3	20	50	0	-8695.0	0	-	0	-	6	180.50	67	16.16	96	11.28	100	10.83	0	28.9	98	98.7
2.7.4	20	50	0	-754.75	0	-	5	230.40	5	230.40	5	230.40	6	192.00	98	11.76	0	0	100	98.1
2.7.5	20	50	0	-4150.4	0	-	0	-	50	21.56	62	17.39	72	14.98	74	14.57	0	37.6	78	108.0
2.8	24	10	10	15639	0	-	0	-	0	-	0	-	0	-	0	-	0	0	100	77.2
3.1	8	22	0	7049.33	0	-	1	2043.40	22	92.88	99	20.64	100	20.43	100	20.43	0	0	100	4.69
3.2	4	16	0	-30665.5	43	1.96	99	0.85	100	0.84	100	0.84	100	0.84	100	0.84	0	0	100	1.76
3.3	6	18	0	-310	2	69.95	11	12.72	34	4.11	47	2.98	57	2.45	66	2.12	0	0	100	3.04
3.4	3	9	0	-4	3	10.43	92	0.34	93	0.34	93	0.34	93	0.34	96	0.33	0	0	100	0.859
4.3	4	9	1	-4.51	5	18.36	6	15.30	11	8.35	15	6.12	22	4.17	32	2.87	0	0	100	1.51
4.4	4	9	1	-2.217	0	-	0	-	22	4.05	23	3.87	27	3.30	36	2.47	0	0.003	92	1.68
4.5	6	15	3	-13.4	8	22.38	11	16.27	32	5.59	74	2.42	78	2.29	87	2.06	0	0	100	3.63
4.6	2	6	0	-5.51	43	0.57	43	0.57	43	0.57	43	0.57	44	0.56	59	0.42	0	0	100	0.538
4.7	2	5	1	-16.74	26	0.92	98	0.24	98	0.24	98	0.24	98	0.24	98	0.24	0	0	100	0.481
5.2	46	86	36	1.567	0	-	0	-	0	-	0	-	0	-	0	-	0	0.153	1	485.1
5.4	32	58	26	1.86	0	-	100	19.79	100	19.79	100	19.79	100	19.79	100	19.79	0	0.17	2	176.1
6.2	5	17	4	400	0	-	0	-	3	393.70	42	28.12	64	18.45	70	16.87	0	0	100	4.71
6.3	5	17	4	600	0	-	0	-	0	-	4	188.10	7	107.49	17	44.26	0	0.3	98	5.37
6.4	5	17	4	750	0	-	0	-	26	49.07	44	29.00	47	27.15	54	23.63	0	0	100	5.95
7.2	16	41	13	1.0	0	-	0	-	2	816.90	10	163.38	13	125.68	20	81.69	0.02	0.24	1	41.83
7.3	27	64	19	1.0	0	-	0	-	0	-	0	-	2	9129.55	19	961.01	0.07	0.48	1	134.5
7.4	38	90	23	1.0	0	-	0	-	3	29468.00	16	5525.25	22	4018.36	39	2266.77	0.02	0.24	1	443.1

Table 4.7: Performance comparison of *DLM-General* and CSA in solving constrained MINLPs based on Floudas and Pardalos' continuous constrained NLPs [57]. All timing results were collected on a Pentinum III 500-MHz computer with Solaris 7. '-' stands for no solution found for the solution quality specified within 100 feasible DLM runs. 'SR' stands for success ratio of finding solutions with specified quality within 100 feasible DLM runs.

Benchmark Continuous NLPs					DLM												CSA			
Problem	Var.	Constraints		Best.	0%		1%		5%		10%		15%		25%		best	avg.	frac.	avg.
ID	no.	total	eq.	known	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	SR	sec	diff.	diff.	best	time
2.1	5	11	0	-17	3	32.50	3	32.50	8	12.19	13	7.50	13	7.50	33	2.95	0	0.4	41	1.48
2.2	6	2	0	-213	15	7.39	20	5.54	100	1.11	100	1.11	100	1.11	100	1.11	0	0	100	2.01
2.3	13	35	0	-15	85	3.37	85	3.37	85	3.37	85	3.37	90	3.18	99	2.89	0	0	100	16.8
2.4	6	17	0	-11	4	29.95	13	9.22	70	1.71	70	1.71	70	1.71	99	1.21	0	0	100	2.89
2.5	10	31	0	-268	49	4.87	97	2.46	100	2.39	100	2.39	100	2.39	100	2.39	0	0	100	23.6
2.6	10	31	0	-39	18	11.10	18	11.10	18	11.10	26	7.68	26	7.68	48	4.16	0	0.1	93	10.6
2.7.1	20	50	0	-394.75	0	-	0	-	25	46.56	25	46.56	25	46.56	37	31.46	0	0	100	103.4
2.7.2	20	50	0	-884.75	0	-	33	35.04	53	21.82	70	16.52	79	14.64	99	11.68	0	2.89	96	101.2
2.7.3	20	50	0	-8695.0	0	-	0	-	6	182.62	55	19.92	97	11.30	100	10.96	0	28.9	98	97.3
2.7.4	20	50	0	-754.75	0	-	5	233.14	5	233.14	6	194.28	6	194.28	98	11.89	0	0	100	101.1
2.7.5	20	50	0	-4150.4	0	-	0	-	48	22.67	58	18.76	73	14.91	73	14.91	0	31.1	81	114.0
2.8	24	10	10	15639	0	-	0	-	0	-	0	-	0	-	0	-	0	0	100	79.5
3.1	8	22	0	7049.33	0	-	5	432.24	27	80.04	94	22.99	100	21.61	100	21.61	0	0	100	5.43
3.2	4	16	0	-30665.5	46	1.89	99	0.88	100	0.87	100	0.87	100	0.87	100	0.87	0	0	100	2.98
3.3	6	18	0	-310	1	143.10	10	14.31	33	4.34	47	3.04	58	2.47	66	2.17	0	0	100	2.57
3.4	3	9	0	-4	3	10.70	93	0.35	93	0.35	93	0.35	93	0.35	96	0.33	0	0	100	1.01
4.3	4	9	1	-4.51	7	13.33	8	11.66	11	8.48	15	6.22	19	4.91	32	2.92	0	0	100	1.69
4.4	4	9	1	-2.217	0	-	0	-	22	4.12	22	4.12	27	3.36	34	2.66	0	0.003	92	1.86
4.5	6	15	3	-13.4	11	16.38	11	16.38	31	5.81	73	2.47	80	2.25	85	2.12	0	0	100	3.78
4.6	2	6	0	-5.51	43	0.59	44	0.57	44	0.57	44	0.57	44	0.57	58	0.43	0	0	100	0.618
4.7	2	5	1	-16.74	33	0.74	99	0.25	99	0.25	99	0.25	99	0.25	99	0.25	0	0	100	0.548
5.2	46	86	36	1.567	0	-	0	-	0	-	0	-	0	-	0	-	0.02	0.167	1	632.8
5.4	32	58	26	1.86	0	-	100	20.27	100	20.27	100	20.27	100	20.27	100	20.27	0	0.17	3	168.5
6.2	5	17	4	400	0	-	0	-	4	301.57	44	27.42	64	18.85	71	16.99	0	0	100	5.26
6.3	5	17	4	600	0	-	0	-	0	-	2	387.60	6	129.20	18	43.07	0	0.8	94	5.52
6.4	5	17	4	750	0	-	0	-	20	65.94	43	30.67	46	28.67	55	23.98	0	0	100	6.46
7.2	16	41	13	1.0	0	-	0	-	0	-	5	296.78	13	114.15	24	61.83	0.03	0.26	1	47.7
7.3	27	64	19	1.0	0	-	0	-	0	-	0	-	0	-	15	1194.05	0.17	0.46	1	153.3
7.4	38	90	23	1.0	0	-	0	-	2	46620.50	13	7172.38	25	3729.64	47	1983.85	0.02	0.17	1	333.5

Table 4.8: Comparison results of LANCELOT, DONLP2, CSA and DLM in solving selected continuous problems from CUTE using specified starting points in the benchmark. All timing results are in seconds and were collected on a Pentium-III 500-MHz computer running Solaris 7. $'-'$ means that no feasible solution was found, and $'*'$ means that solutions were obtained by the commercial version of LANCELOT (by submitting problems through [124]) but no CPU times were available. Numbers in bold represent the best solutions among the four methods if they have different solutions. Note that the objective functions $f(x)$ in CUTE can be linear, quadratic or nonlinear. For simplicity, we use L, Q and N to denote them respectively.

Problem	n_v	f	$h(x)$		$g(x)$		LANCELOT		DONLP2		CSA		DLM	
			n_{le}	n_{ne}	n_{li}	n_{ni}	sol.	CPU	sol.	CPU	sol.	CPU	sol.	CPU
ALJAZZAF	3	Q	0	1	0	0	**75.0**	0.46	-	-	**75.0**	1.38	$7.3683 \cdot 10^{9}$	0.07
ALLINITC	4	N	0	0	0	1	**30.44**	*	31.75	0.02	**30.44**	4.76	$2.6733 \cdot 10^{17}$	1.22
ALSOTAME	2	N	0	1	0	0	0.082	0.57	0.082	0.03	0.082	0.80	0.0821	0.24
AVION2	49	N	15	0	0	0	-	-	-	-	$\mathbf{9.47 \cdot 10^{7}}$	946.7	$9.7153 \cdot 10^{7}$	205.11
BATCH	46	N	12	0	60	1	-	-	-	-	$\mathbf{2.59 \cdot 10^{5}}$	12465	-	-
BRAINPC0	6907	N	0	6900	0	0	**0.0015**	55.5	-	-	-	-	-	-
BRAINPC1	6907	N	0	6900	0	0	**0.0**	84.8	-	-	-	-	-	-
BRAINPC2	13807	N	0	13800	0	0	$4.1 \cdot 10^{-8}$	93.2	-	-	-	-	-	-
BRAINPC3	6907	N	0	6900	0	0	$1.687 \cdot 10^{-4}$	89.4	-	-	-	-	-	-
BRAINPC4	6907	N	0	6900	0	0	$1.288 \cdot 10^{-3}$	79.1	-	-	-	-	-	-
BRAINPC5	6907	N	0	6900	0	0	$1.362 \cdot 10^{-3}$	143.7	-	-	-	-	-	-
BRAINPC6	6907	N	0	6900	0	0	$5.931 \cdot 10^{-5}$	85.2	-	-	-	-	-	-
BRAINPC7	6907	N	0	6900	0	0	$3.82 \cdot 10^{-5}$	109.4	-	-	-	-	-	-
BRAINPC8	6907	N	0	6900	0	0	$1.652 \cdot 10^{-4}$	112.8	-	-	-	-	-	-
BRAINPC9	6907	N	0	6900	0	0	$8.27 \cdot 10^{-4}$	68.2	-	-	-	-	-	-
BRIDGEND	2734	L	1304	1423	0	0	-	-	-	-	-	-	-	-
BRITGAS	450	N	0	360	0	0	0	8.3	-	-	-	-	-	-
BT11	5	N	1	2	0	0	**0.825**	0.62	**0.825**	0.02	**0.825**	3.73	$1.4127 \cdot 10^{14}$	0.31
BT12	5	Q	0	3	0	0	**6.188**	0.47	**6.188**	0.02	**6.188**	3.25	3401.0662	7.61
BT6	5	N	0	2	0	0	**0.277**	0.56	**0.277**	0.03	**0.277**	4.61	$5.0386 \cdot 10^{10}$	0.72
BT7	5	N	0	3	0	0	**306.5**	0.51	360.4	0.03	**306.5**	3.38	$4.7796 \cdot 10^{10}$	4.96
BT8	5	Q	0	2	0	0	**1.0**	0.57	**1.0**	0.02	**1.0**	2.93	$1.4459 \cdot 10^{6}$	8.43
C-RELOAD	342	L	26	174	0	84	-1.027	51.1	-	-	-	-	-	
CB2	3	L	0	0	0	3	1.952	0.60	1.952	0.03	1.952	2.49	1.9542	0.35
CRESC4	6	N	0	0	0	8	-	-	-	-	**0.872**	37.9	29.6295	17.00
CSFI1	5	L	0	2	0	2	**-49.07**	0.63	0.0	0.02	**-49.07**	4.56	-4.3916	2.39
DEMBO7	16	Q	0	0	0	20	174.9	*	-	-	174.9	342.9	**174.8034**	61.70
DIPIGRI	7	N	0	0	0	4	**680.6**	0.68	**680.6**	0.03	**680.6**	9.19	$3.9665 \cdot 10^{11}$	0.04
DIXCHLNG	10	N	0	5	0	0	**0.0**	1.12	2471.9	0.04	**0.0**	44.12	$4.3100 \cdot 10^{6}$	59.94
DNIEPER	61	N	0	24	0	0	$\mathbf{1.87 \cdot 10^{4}}$	0.83	-	-	$\mathbf{1.87 \cdot 10^{4}}$	3703	18947.0180	1353.94
ERRINBAR	18	L	0	8	1	0	**28.05**	*	-	-	**28.05**	83.3	238.5364	127.40
EXPFITA	5	N	0	0	22	0	**0.0011**	0.65	**0.0011**	0.07	$\mathbf{1.13 \cdot 10^{-3}}$	89.44	$7.0745 \cdot 10^{5}$	0.70
FEEDLOC	90	L	4	15	166	74	0.0	251.7	-	-	-	-	0.554	159347.0
FLETCHER	4	Q	0	1	3	0	19.53	0.57	-	-	**12.18**	3.46	$7.9692 \cdot 10^{5}$	1.74
GAUSSELM	14	L	0	5	6	0	**-2.25**	0.55	-	-	-2.007	56.19	-1.9797	21.24
GIGOMEZ2	3	L	0	0	0	3	1.952	0.59	1.952	0.03	1.952	2.35	1.9554	0.02
HELSBY	1408	L	658	741	0	0	-	-	-	-	-	-	-	-
HIMMELBI	100	N	0	0	12	0	**-1735.6**	1.23	-	-	**-1735.6**	14114	-1710.7390	1383.32
HIMMELBJ	45	N	14	0	0	0	-	-	-	-	**-1910.3**	2001	-378.0590	1835.66
HIMMELP2	2	N	0	0	0	1	-62.05	0.63	-62.05	0.03	-62.05	1.76	-62.0539	0.02
HIMMELP6	2	N	0	0	2	3	**-59.01**	0.69	-57.85	0.02	**-59.01**	2.88	-18.2407	0.02
HONG	4	N	1	0	0	0	22.57	0.50	22.57	0.03	22.57	2.90	22.5707	1.02
HS100	7	N	0	0	0	4	**680.6**	0.72	**680.6**	0.03	**680.6**	9.19	$3.9665 \cdot 10^{11}$	0.06
HS101	7	N	0	0	0	5	**1809.7**	*	-	-	**1809.7**	75.9	1810.6711	5.89
HS102	7	N	0	0	0	5	**911.9**	*	-	-	**911.9**	74.2	**911.8949**	24.91
HS103	7	N	0	0	0	5	-	-	**543.7**	0.13	**543.7**	75.0	830.2559	0.28
HS104	8	N	0	0	0	5	3.95	0.58	3.95	0.03	3.95	30.62	3.9590	0.02
HS107	9	N	0	6	0	0	**5055**	0.59	5085.5	0.04	**5055**	42.6	12041.9850	2.11
HS108	9	Q	0	0	0	13	**-0.866**	0.58	-0.675	0.1	**-0.866**	52.69	-0.8652	14.71
HS109	9	N	0	6	2	2	-	-	-	-	**5362**	48.38	5395.0698	5.90
HS111	10	N	0	3	0	0	-47.76	0.83	-47.76	0.05	-47.76	79.4	-39.0618	1.37
HS114	10	Q	1	2	4	4	**-1768.8**	1.64	-	-	**-1768.8**	43.4	-1486.5426	30.64
HS117	15	N	0	0	0	5	**32.35**	0.60	2400.0	0.04	**32.35**	54.7	$1.1923 \cdot 10^{5}$	0.64
HS119	16	N	8	0	0	0	**244.9**	0.54	-	-	**244.9**	421	252.7982	48.28
HS12	2	Q	0	0	0	1	**-30.0**	0.46	**-30.0**	0.03	**-30.0**	0.95	-27.1171	0.03
HS18	2	N	0	0	0	2	**5.0**	0.65	**5.0**	0.03	**5.0**	0.96	52.4232	0.01

continued on next page

continued from previous page

Problem	n_v	f	$h(x)$		$g(x)$		LANCELOT		DONLP2		CSA		DLM	
			n_{le}	n_{ne}	n_{li}	n_{ni}	sol.	CPU	sol.	CPU	sol.	CPU	sol.	CPU
HS19	2	N	0	0	0	2	**-6961.8**	0.58	**-6961.8**	0.03	**-6961.8**	1.27	-6960.1521	0.23
HS20	2	N	0	0	0	3	40.2	0.52	39.17	0.03	**38.19**	1.64	40.4699	0.02
HS23	2	Q	0	0	0	5	**2.0**	0.54	9.47	0.03	**2.0**	1.94	2.0002	0.19
HS24	2	N	0	0	3	0	-1.0	0.55	-1.0	0.03	-1.0	1.33	-0.9998	0.03
HS26	3	N	0	1	0	0	0.0	0.65	0.0	0.03	0.0	1.28	$6.1697 \cdot 10^{6}$	0.47
HS27	3	N	0	1	0	0	**0.04**	0.49	**0.04**	0.03	**0.04**	1.27	$1.9316 \cdot 10^{13}$	0.45
HS29	3	N	0	0	0	1	**-22.6**	0.53	**-22.6**	0.03	**-22.6**	1.34	-1.0279	0.04
HS30	3	Q	0	0	0	1	**1.0**	0.52	3.0	0.02	**1.0**	1.44	1.0000	0.01
HS32	3	N	1	0	0	1	**1.0**	0.54	1.02	0.03	**1.0**	1.72	8.0936	0.05
HS33	3	N	0	0	0	2	-4.0	0.55	-3.0	0.03	**-4.59**	2.05	-4.5858	0.32
HS34	3	L	0	0	0	2	-0.834	0.38	-0.834	0.03	-0.834	2.12	-0.8339	0.12
HS36	3	N	0	0	1	0	**-3300**	0.55	-1000	0.03	**-3300**	1.26	-2936.3940	0.02
HS37	3	N	0	0	2	0	-3456	0.48	-3456	0.02	-3456	1.54	-2842.8167	0.03
HS39	4	L	0	2	0	0	**-1.0**	0.52	**-1.0**	0.03	**-1.0**	2.11	11.7132	1.84
HS40	4	N	0	3	0	0	**-0.25**	0.58	**-0.25**	0.03	**-0.25**	2.83	4.7297	1.60
HS41	4	N	1	0	0	0	**1.926**	0.52	**1.926**	0.03	**1.926**	1.37	1.9516	0.01
HS42	4	N	1	1	0	0	**13.86**	0.56	**13.86**	0.02	**13.86**	2.36	22.5895	0.15
HS43	4	Q	0	0	0	3	**-44.0**	0.49	**-44.0**	0.03	**-44.0**	4.48	-25.9903	0.04
HS46	5	N	0	2	0	0	**0.0**	0.54	**0.0**	0.02	**0.0**	4.28	$4.6960 \cdot 10^{8}$	0.11
HS54	6	N	1	0	0	0	0.0	0.58	-0.156	0.03	**-0.908**	3.87	-0.3599	0.06
HS55	6	N	6	0	0	0	6.667	0.49	-	-	**6.333**	6.91	6.8050	1.59
HS56	7	N	0	4	0	0	**-3.456**	0.55	**-3.456**	0.06	-3.31	8.35	-0.0001	10.98
HS57	2	N	0	0	0	1	0.03065	0.57	**0.02846**	0.03	**0.02846**	20.45	97.2118	4.72
HS59	2	N	0	0	0	3	**-7.803**	0.88	-6.75	0.03	**-7.803**	5.45	-6.7495	0.10
HS60	3	N	0	1	0	0	**0.0326**	0.62	**0.0326**	0.03	**0.0326**	1.65	6.1245	0.27
HS61	3	Q	0	2	0	0	**-143.65**	0.57	**-143.65**	0.04	**-143.65**	1.54	$6.5567 \cdot 10^{6}$	5.37
HS62	3	N	1	0	0	0	**-26273**	0.61	**-26273**	0.03	**-26273**	2.20	-26225.9180	0.33
HS63	3	Q	1	1	0	0	**961.72**	0.55	**961.72**	0.03	**961.72**	1.87	963.2815	1.33
HS64	3	N	0	0	0	1	6299.8	0.43	6299.8	0.03	6299.8	1.68	6299.9009	0.08
HS68	4	N	0	2	0	0	**-0.9204**	0.72	-0.7804	0.04	**-0.9204**	6.76	-0.4048	0.03
HS69	4	N	0	2	0	0	**-956.71**	0.80	**-956.71**	0.04	**-956.71**	4.95	-949.3953	0.04
HS7	2	N	0	1	0	0	**-1.732**	0.56	**-1.732**	0.03	**-1.732**	0.94	1.6128	0.17
HS71	4	N	0	1	0	1	**17.01**	0.62	31.64	0.04	**17.01**	2.60	18.2356	0.02
HS73	4	L	1	0	1	1	**29.9**	0.52	29.98	0.03	**29.9**	3.12	33.9782	0.64
HS74	4	N	0	3	2	0	**5126.5**	0.50	**5126.5**	0.04	**5126.5**	5.25	5192.1899	0.12
HS75	4	N	0	3	2	0	**5174.4**	0.56	**5174.4**	0.04	**5174.4**	5.30	5228.0036	2.67
HS77	5	N	0	2	0	0	**0.2415**	0.56	**0.2415**	0.03	**0.2415**	4.09	$3.0985 \cdot 10^{16}$	0.31
HS78	5	N	0	3	0	0	**-2.92**	0.58	**-2.92**	0.03	**-2.92**	3.79	-0.0580	5.32
HS79	5	N	0	3	0	0	**0.0788**	0.57	**0.0788**	0.03	**0.0788**	4.10	$4.7342 \cdot 10^{10}$	0.84
HS80	5	N	0	3	0	0	0.054	0.58	0.054	0.03	0.054	4.25	0.0539	2.60
HS83	5	Q	0	0	0	3	**-30666**	0.52	-	-	**-30666**	5.68	-29818.0540	0.01
HS84	5	Q	0	0	0	3	-	-	$-2.35 \cdot 10^{6}$	0.03	$\mathbf{-5.28 \cdot 10^{6}}$	7.69	$-5.2122 \cdot 10^{6}$	0.50
HS87	6	N	0	4	0	0	-	-	8997	0.04	**8926**	8.43	8997.4358	0.30
HS93	6	N	0	0	0	2	-	-	**135.1**	0.03	**135.1**	4.96	135.9548	0.11
HS99	7	N	0	2	0	0	-	-	$\mathbf{-8.31 \cdot 10^{8}}$	0.06	$\mathbf{-8.31 \cdot 10^{8}}$	12.8	$-8.0426 \cdot 10^{8}$	44.40
HUBFIT	2	N	0	0	1	0	**0.0169**	0.46	**0.0169**	0.02	**0.0169**	1.21	474.9145	0.01
HYDROELL	1009	N	0	0	1008	0	$\mathbf{-3.582 \cdot 10^{6}}$	70.5	-	-	-	-	-	-
HYDROELM	505	N	0	0	504	0	$\mathbf{-3.582 \cdot 10^{6}}$	29.3	-	-	-	-	-	-
HYDROELS	169	N	0	0	168	0	$\mathbf{-3.582 \cdot 10^{6}}$	2.7	-	-	-	-	$-3.4673 \cdot 10^{6}$	3.46
LAKES	90	Q	60	18	0	0	-	-	-	-	-	-	-	-
LAUNCH	25	N	6	3	12	7	-	-	-	-	**9.0**	1941	10.8817	628.72
LEAKNET	156	N	73	80	0	0	8.0	25.7	-	-	-	-	-	-
LHAIFAM	99	N	0	0	0	150	-	-	-	-	-	-	-	-
LIN	4	N	2	0	0	0	**-0.02**	0.70	-0.0176	0.02	**-0.02**	6.69	-0.0196	0.01
LOADBAL	31	N	11	0	20	0	**0.453**	0.69	1.546	0.08	1.546	1712	-	-
LOOTSMA	3	N	0	0	0	2	-	-	-	-	**1.414**	2.10	1.4143	0.12
MADSEN	3	L	0	0	0	6	**0.616**	0.55	**0.616**	0.03	**0.616**	5.13	0.6184	0.12
MARATOS	2	Q	0	1	0	0	-1.0	0.40	-1.0	0.02	-1.0	0.839	-1.0000	0.54
MATRIX2	6	Q	0	0	0	2	**0.0**	0.52	**0.0**	0.04	**0.0**	3.93	$1.3946 \cdot 10^{5}$	0.16
MESH	41	N	4	20	24	0	-	-	0.0	0.16	$\mathbf{-1.0 \cdot 10^{5}}$	4009	-	-
MISTAKE	9	Q	0	0	0	13	-1.0	0.58	-1.0	0.06	-1.0	55.0	-0.9989	15.92
MRIBASIS	36	L	1	8	43	3	**18.218**	1.88	-	-	**18.218**	7612	19.2445	941.77
MWRIGHT	5	N	0	3	0	0	24.97	0.56	24.97	0.02	**1.318**	3.92	29146.6630	6.13
NET1	48	N	21	17	16	3	-	-	-	-	$\mathbf{9.41 \cdot 10^{5}}$	6776	$9.7000 \cdot 10^{5}$	1146.23
NET2	144	non.	64	59	32	5	-	-	-	-	$\mathbf{1.187 \cdot 10^{6}}$	226271	-	-
NET3	464	non.	195	199	110	17	-	-	-	-	-	-	-	-
NGONE	8	Q	0	0	2	6	**-0.5**	0.51	0.0	0.03	**-0.5**	24.7	-0.4998	1.47
ODFITS	10	N	6	0	0	0	**-2380**	0.50	**-2380**	0.04	**-2380**	26.8	-2333.0398	0.92
OPTCNTRL	32	Q	10	10	0	0	**550**	0.51	-	-	**550**	432	-	-

continued on next page

continued from previous page

Problem	n_v	f	$h(x)$		$g(x)$		LANCELOT		DONLP2		CSA		DLM	
			n_{le}	n_{ne}	n_{li}	n_{ni}	sol.	CPU	sol.	CPU	sol.	CPU	sol.	CPU
OPTPRLOC	30	Q	0	0	5	25	**-16.42**	4.02	-	-	**-16.42**	1674	-14.3597	65.52
ORTHREGB	27	Q	0	6	0	0	0.0	0.76	0.0	0.04	0.0	218.3	-	-
PENTAGON	6	N	0	0	15	0	$1.51 \cdot 10^{-4}$	0.56	$\mathbf{1.37 \cdot 10^{-4}}$	0.03	$\mathbf{1.37 \cdot 10^{-4}}$	32.1	0.0001	2.60
POLAK1	3	L	0	0	0	2	2.718	0.53	2.718	0.03	2.718	2.02	2.7196	1.69
POLAK3	12	L	0	0	0	10	**5.933**	0.82	-	-	**5.933**	417.2	-	
POLAK5	3	L	0	0	0	2	**50.0**	0.52	**50.0**	0.02	**50.0**	1.87	6501.9919	1.24
POLAK6	5	L	0	0	0	4	-44.0	0.74	-44.0	0.04	-44.0	11.8	669.4584	0.04
QC	9	N	0	0	4	0	-956.5	0.58	-	-	-956.5	28.5	**-1046.5118**	12.97
READING6	102	N	0	50	0	0	-	-	-	-	**-132.3**	28530	-	-
READING7	1002	N	0	500	0	0	-	-	-	-	-	-	-	-
READING8	2002	N	0	1000	0	0	-	-	-	-	-	-	-	-
RK23	17	L	4	7	0	0	**0.0833**	0.75	-	-	0.675	96.1	-	-
ROBOT	14	Q	0	2	0	0	**5.463**	0.55	-	-	**5.463**	34.8	5.5344	0.95
S316-322	2	Q	0	1	0	0	334.3	0.48	334.3	0.02	334.3	0.83	334.3219	0.40
SARO	4754	N	0	4015	0	0	252.3	3739.0	-	-	-	-	-	-
SAROMM	5120	N	365	4015	730	0	57.35	9147.5	-	-	-	-	-	-
SINROSNB	2	N	0	0	0	1	0.0	0.56	0.0	0.04	0.0	1.36	0.0001	2.08
SNAKE	2	L	0	0	0	2	-	-	**0.0**	0.02	**0.0**	1.43	1.5297	0.70
SPIRAL	3	L	0	0	0	2	**0.0**	0.71	0.121	0.31	**0.0**	3.46	**0.0006**	1.06
STANCMIN	3	N	0	0	2	0	**4.25**	0.58	-	-	**4.25**	1.72	**4.2515**	0.56
SVANBERG	10	N	0	0	0	10	**15.73**	0.59	16.5	0.03	**15.73**	85.3	**15.7316**	6.70
SYNTHES1	6	N	0	0	4	2	**0.759**	0.55	10.0	0.04	**0.759**	9.97	0.7593	4.10
SYNTHES2	11	N	1	0	10	3	**-0.554**	0.60	-	-	**-0.554**	94.3	0.4365	42.93
SYNTHES3	17	N	2	0	17	4	**15.08**	0.51	-	-	**15.08**	261.7	**15.0822**	138.83
TENBARS4	18	L	0	8	1	0	**368.5**	*	-	-	**368.5**	84.9	-	-
TWIRISM1	343	N	50	174	5	84	-1.01	136.1	-	-	-	-	-	
TWIRIMD1	1247	N	143	378	5	186	-1.034	10158	-	-	-	-	-	
TWIRIBG1	3127	N	292	630	5	312	-	-	-	-	-	-	-	
TWOBARS	2	N	0	0	0	2	1.51	0.53	1.51	0.03	1.51	1.36	1.5107	0.03
WOMFLET	3	L	0	0	0	3	0.0	0.51	0.0	0.02	0.0	2.5	0.0000	0.41
ZAMB2-10	270	N	0	96	0	0	**-1.58**	2.99	-	-	-	-	-1.4404	30147.80
ZAMB2-11	270	N	0	96	0	0	**-1.116**	1.83	-	-	-	-	-	-
ZAMB2-8	138	N	0	48	0	0	**-0.153**	1.20	-	-	**-0.153**	46156	-0.1338	16278.72
ZECEVIC3	2	Q	0	0	0	2	**97.31**	0.54	**97.31**	0.03	**97.31**	1.32	104.2501	0.01
ZECEVIC4	2	Q	0	0	1	1	7.558	0.59	7.558	0.02	7.558	1.23	7.5589	0.42
ZY2	3	N	0	0	0	2	**2.0**	0.46	7.165	0.03	**2.0**	2.35	6.2498	0.03

In comparing DLM and CSA, *DLM-General* generally cannot find as good solutions as CSA because *DLM-General* is a local search and there is only one starting point provided. This is the reason why *DLM-General* found feasible solutions with very large objective values for problems like ALJAZZAF, BT11, BT12 and BT6. However, DLM is usually faster than CSA. For example, for problems like DNIEPER, HIMMELBI, HIMMELBJ, MRIBASIS, NET1, ZAMB2-8 and ZAMB2-10 in which CSA took over over 1000 CPU seconds, DLM performs better in terms of solution times. Moreover, for problems DEMBO7 and QC, DLM outperforms any of the other three methods in terms of solution quality.

4.4 Summary

In this chapter, we have presented a general search framework to look for saddle points and have proposed DLM, an implementation of the framework utilizing local search. We have further investigated and explored in detail strategies for neighborhood search, dynamic weight adaptation, global search, relax-and-tighten, and duration of run. We have tested and compared various heuristics and trade-offs in implementing the method. By testing our strategy on 12 difficult benchmark problems, we have selected one suitable combination of parameters that can be generalized to other constrained NLPs. Finally, we have applied DLM to solve three sets of constrained NLP benchmarks. Our experimental results on these benchmark NLPs show that our search method is able to find high-quality solutions efficiently.

Chapter 5

Application I - Designing Multiplierless Filter Banks

In this chapter, we apply discrete-space first-order search method (DLM) to design multiplierless QMF (quadrature mirror filter) banks [210]. The filter coefficients in these filter banks are in powers-of-two (PO2), where numbers are represented as sums or differences of powers of two (also called Canonical Signed Digit–CSD–representation), and multiplications are carried out as additions, subtractions and shifts. We formulate the design problem as a nonlinear discrete constrained optimization problem, using the reconstruction error as the objective, and the stopband and passband energies, stopband and passband ripples and transition bandwidth as constraints. Using the performance of the best existing designs as constraints, we search for designs that improve over the best existing designs with respect to all the performance metrics. We apply DLM to find good designs, and dynamic weight adaptation to improve the convergence speed of Lagrangian methods without affecting their solution quality. Our method can find designs that improve over Johnston's benchmark designs using a maximum of three to six ONE bits in each filter coefficient, instead of using

Filter	Minimization Objectives
Overall Filter Bank	Amplitude distortion
	Aliasing distortion
	Phase distortion
Single Filter	Stopband ripple (δ_s)
	Passpand ripple (δ_p)
	Stopband energy (E_s)
	Passband energy (E_p)
	Transition bandwidth (T_t)

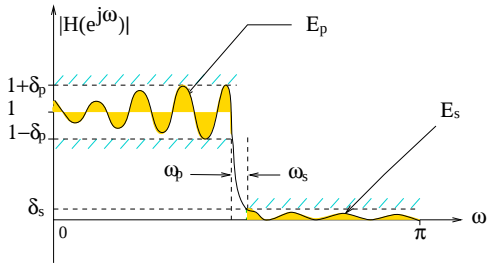


Figure 5.1: Possible design objectives of filter banks and an illustration of the design objectives of a single low-pass filter. ($[0, \omega_p]$ is the pass band; $[\omega_s, \pi]$, the stop band; $[\omega_p, \omega_s]$, the transition band.)

floating-point representations. The approach here is quite general and is applicable to the design of other types of multiplierless filter banks [216].

5.1 Introduction

Digital filter banks have been applied in many engineering fields. Figure 5.1 summarizes the various design objectives for measuring quality. In general, filter-bank design problems are multi-objective, continuous, nonlinear optimization problems.

Algorithms for designing filter banks are either optimization-based or non-optimization based. In optimization-based methods, a design problem is formulated as a multi-objective nonlinear optimization problem [198] whose form may be application- and filter-dependent. The problem is then converted into a single-objective optimization problem and solved by existing optimization methods, such as gradient-descent, Lagrange-multiplier, quasi-Newton, SA, and GA [112, 104]. On the other hand, filter bank-design problems have been solved

by non-optimization-based algorithms, which include spectral factorization [120, 199] and heuristic methods (as in IIR-filter design). These methods generally do not continue to find better designs once a suboptimal design has been found [199].

In this chapter, we apply DLM to designing multiplierless QMF banks. These filter banks are an important class of filter banks that have been studied extensively. In a two-band QMF bank, the reconstructed signal is:

$$\begin{aligned} \hat{X}(z) \;=\; & \frac{1}{2}\left[H_0(z)F_0(z) \;+\; H_1(z)F_1(z)\right]\; X(z) \\ & +\; \frac{1}{2}\left[H_0(-z)F_0(z) \;+\; H_1(-z)F_1(z)\right]\; X(-z) \end{aligned} \tag{5.1}$$

where $X(z)$ is the original signal, and $H_i(z)$ and $F_i(z)$ are, respectively, the response of the analysis and synthesis filters. To perfectly reconstruct the original signal based on $\hat{X}$, we have to eliminate aliasing, amplitude, and phase distortions. QMF banks with FIR filters implement perfect reconstruction by setting:

$$\begin{cases} F_0(z) = & H_1(-z), \\ F_1(z) = - & H_0(-z), \\ H_1(z) = & H_0(-z), \end{cases} \tag{5.2}$$

leading to a filter bank with one prototype filter $H_0(z)$, linear phase, and no aliasing distortions.

Traditional FIR filters in QMF banks use real numbers or fixed-point numbers as filter coefficients. Multiplications of such long floating point numbers generally limit the speed of FIR filtering. To overcome this limitation, *multiplierless* (*powers-of-two* or *PO2*) filters have been proposed. These filters use filter coefficients that have only a few bits that are ones. When multiplying a filter input (multiplicand) with one such coefficient (multiplier),

the product can be found by adding and shifting the multiplicand a number of times corresponding to the number of ONE bits in the multiplier. For example, the multiplication of y by 0100001001 can be written as the sum of three terms,

$$y \cdot 2^8 + y \cdot 2^3 + y \cdot 2^0, \tag{5.3}$$

each of which can be obtained by shifting y. A limited sequence of shifts and adds are usually much faster than full multiplications. Without using full multiplications, each filter tap takes less area to implement in VLSI, and more filter taps can be accommodated in a given area to implement filter banks of higher performance.

The frequency response of a PO2 filter, $H(z)$, is

$$H(z) = \sum_{i=0}^{\gamma-1} x_i z^{-i} = \sum_{i=0}^{\gamma-1} \left(\sum_{j=0}^{d-1} e_{i,j} 2^j \right) z^{-i} \quad \text{where} \tag{5.4}$$

$$\sum_{j=0}^{d-1} |e_{i,j}| \leq l \text{ for all } i, \quad e_{i,j} = -1, 0, 1.$$

Here, γ is the length of the PO2 filter, l is the maximum number of ONE bits used in each coefficient, and d is the number of bits in each coefficient.

The design of multiplierless filters has been solved by integer programming that optimizes filter coefficients with restricted values of powers-of-two. Other techniques used include combinatorial search [163], SA [32], GA [167], linear programming [119], and continuous Lagrange-multiplier methods in combination with a tree search [182].

This chapter is organized as follows. We formulate in Section 5.2 the design problem as a single-objective constrained optimization problem. In Section 5.3, we present our DLM-QMF that finds saddle points in discrete space and examines the issues related to the implementation of DLM-QMF to design multiplierless filter banks. Finally, Section 5.4 presents experimental results, and conclusions are drawn in Section 5.5.

5.2 Problem Formulation

The design of QMF banks can be formulated as a multi-objective unconstrained optimization problem or as a single-objective constrained optimization problem.

5.2.1 Multi-Objective Unconstrained Formulation

In a multi-objective formulation, the goals can be to:

- Minimize the amplitude distortion (reconstruction error) of the overall filter bank, or
- Optimize the individual performance measures of the prototype filter $H_0(z)$.

One possible formulation using a subset of the measures in Figure 5.1 is as follows:

$$\begin{aligned} &\text{minimize} \quad E_r \quad \text{and} \quad E_s \\ &\text{where} \quad E_r = \int_{\omega=0}^{\frac{\pi}{2}} (|H_0(e^{j\omega})|^2 + |H_0(e^{j(\omega-\pi)})|^2 - 1)^2 \; d\omega \\ &\text{and} \quad E_s = \int_{\omega=\omega_s}^{\pi} |H_0(e^{j\omega})|^2 d\omega \end{aligned} \tag{5.5}$$

Unfortunately, optimal solutions to (5.5) are not necessarily optimal solutions to the original problem that considers all the performance measures. Oftentimes, performance measures not included in the formulation are compromised. Note that in QMF banks, E_r is non-zero. A multi-rate filter bank that enforces perfect reconstruction ($E_r = 0$) can be formulated as a constrained optimization problem with a goal of minimizing E_s [110, 104].

In general, optimal solutions of a multi-objective problem form a *Pareto optimal frontier* such that one solution on this frontier is not dominated by another. One approach to find a point on the frontier is to optimize a weighted sum of all the objectives [112, 53, 198, 36, 141]. This approach has difficulty when frontier points of certain characteristics are desired, such as

those with certain transition bandwidth. Different combinations of weights must be tested by trial and error until a desired solution is found. When the desired characteristics are difficult to satisfy, trial and error is not effective in finding feasible designs. Instead, constrained formulations should be used.

5.2.2 Single-Objective Constrained Formulation

Another approach to solve a multi-objective problem is to turn all but one objectives into constraints, and define the constraints with respect to a reference design. The specific measures constrained may be application- and filter-dependent [198].

Constraint-based methods have been applied to design QMF banks in both the frequency [112, 36, 43, 120, 183, 187] and time domains [140, 185]. In the frequency domain, the most often considered objectives are E_r (reconstruction error) and δ_s (stopband ripple). As stopband ripples cannot be formulated in closed form, stopband attenuation is used instead (represented as E_s in Figure 5.1). In the time domain, Nayebi [140] gave a time-domain formulation with constraints in the frequency domain and designed filter banks using an iterative time-domain design algorithm.

Next we formulate the design of QMF banks in the most general form as a nonlinear constrained optimization problem using the reconstruction error as the objective and other measures (stopband ripple, stopband energy, passband ripple, passband energy and transition bandwidth) as constraints:

$$\begin{aligned} \text{minimize} \quad & E_r \\ \text{subject to} \quad & E_p \le \theta_{E_p}, \quad E_s \le \theta_{E_s}, \\ & T_t \le \theta_{T_t}, \quad \delta_p \le \theta_{\delta_p}, \\ & \delta_s \le \theta_{\delta_s} \end{aligned} \tag{5.6}$$

where θ_{E_p}, θ_{E_s}, θ_{δ_p}, θ_{δ_s} and θ_{T_t} are constraint bounds found in the best-known design (with possibly some bounds relaxed or tightened in order to obtain designs of different trade-offs). The goal here is to find filter banks of a finite word length whose performance measures are better than or equal to those of the reference design. Since the objective and the constraints are nonlinear, the problem is multi-modal with many local minima.

The original optimization problem with inequality constraints (5.6) can be transformed into an optimization problem with equality constraints as follows:

$$\text{minimize} \qquad f(x) = V_{E_r} = \frac{E_r - \theta_{E_r}}{\theta_{E_r}} \tag{5.7}$$

$$\begin{aligned}
\text{subject to} \quad V_{E_p} &= \max\left(\frac{E_p - \theta_{E_p}}{\theta_{E_p}}, 0\right) = 0, \qquad V_{E_s} = \max\left(\frac{E_s - \theta_{E_s}}{\theta_{E_s}}, 0\right) = 0, \\
V_{\delta_p} &= \max\left(\frac{\delta_p - \theta_{\delta_p}}{\theta_{\delta_p}}, 0\right) = 0, \qquad V_{\delta_s} = \max\left(\frac{\delta_s - \theta_{\delta_s}}{\theta_{\delta_s}}, 0\right) = 0, \\
V_{T_t} &= \max\left(\frac{T_t - \theta_{T_t}}{\theta_{T_t}}, 0\right) = 0,
\end{aligned} \tag{5.8}$$

where x is a vector of discrete coefficients, θ_{E_r} is the reconstruction error of the best-known design, and all functions have been normalized with respect to the values of the best-known design.

5.3 DLM-QMF: An Implementation of Discrete First-Order Search Method

Based on (5.7) and (5.8), the discrete Lagrangian function for optimizing PO2 filter banks is:

$$L_d(x, \lambda) = f(x) + \sum_{i \in \{E_p, E_s, \delta_p, \delta_s, T_t\}} \lambda_i \cdot V_i + \frac{1}{2} \sum_{i \in \{E_p, E_s, \delta_p, \delta_s, T_t\}} V_i^2, \tag{5.9}$$

procedure *DLM-QMF*

1. set c (positive real constant for controlling the speed of change of Lagrange multipliers);
2. set i_{max} (maximum number of iterations);
3. set starting point x;
4. set initial value of λ (set to 0 in the experiments);
5. **if** using dynamic weight adaptation **then** *weight_initialization*;
6. **while** stopping condition not satisfied **do** {
7. x ***Loop:*** update x to x' only if this will result in $L_d(x', \lambda) < L_d(x, \lambda)$;
8. λ ***Loop:*** **if** condition for updating λ is satisfied **then** $\lambda_i \leftarrow \lambda_i + c \cdot \max(0, g_i)$;
9. **if** using dynamic weight adaptation **then** *dynamic_weight_adaptation* }

Figure 5.2: DLM-QMF: An implementation of discrete first-order method for designing PO2 filter banks. (The initial values of parameters are indicated here unless specified otherwise in the text.)

where x is a vector of coefficients, each of which is in CSD form of the sum of several signed binary bits, such as $2^{-1} + 2^{-3} - 2^{-6}$. Since we have only equality constraints transformed from inequality constraints, we use λ as our Lagrange multipliers in the following discussion.

Figure 5.2 shows an implementation of the discrete-space first-order conditions for designing PO2 filter banks formulated as nonlinear discrete constrained minimization problems. The procedure shows several aspects that can be tuned in order to improve its performance.

- *Starting points* (Line 3). We choose a starting point based on a discrete approximation of an existing QMF bank with real coefficients (Section 5.3.1).
- *Initial Lagrange-multiplier values* (Line 4). We initialize all Lagrange multipliers to zero in order to allow our results to be reproduced easily. An optimal initial setting is difficult because it depends on the amount of constraint violation.
- *Time constraint* (Lines 2 and 6). The search algorithm will terminate if it has converged or the number of iterations is larger than i_{max}.

- *x Loop* (Line 7). Here, we evaluate all possible neighboring points of x in order to find improvements in its Lagrangian value (Section 5.3.2).
- λ *Loop* (Lines 1 and 8). The Lagrange multipliers are updated when the search reaches a local minimum in the objective space. We do not update the multipliers more frequently due to instability of the trajectory. The amount of update is controlled by an application-dependent constant c and other filter-related parameters (Section 5.3.3).
- *Dynamic weight adaptation* (Lines 5 and 9). Weight adaptation adjusts the weight between the objective and the constraints in order to adjust their relative importance and to improve convergence (Section 4.1.3).

5.3.1 Generating a Starting Point

There are two alternatives in selecting a starting point (Line 3 in Figure 5.2): using the parameters of an existing PO2 QMF bank, or using a discrete approximation of an existing QMF bank with real coefficients. The first alternative is not always possible because not many such filter banks are available in the literature. In this section, we discuss the second alternative.

In the second approach, we first transform the real coefficients of the best-known design to PO2 forms using a CSD representation. Given a real coefficient and b, the maximum number of ONE bits to represent the coefficient, we apply Booth's algorithm [27] to represent consecutive 1's using two ONE bits and then truncate the least significant bits of the coefficients. This approach generally allows a number to be represented in a few ONE bits. As an example, consider a binary fixed-point number 0.10011101100. After applying Booth's algorithm and truncation, we can represent the number in 2 ONE bits:

$$0.10011101100 \xrightarrow[\text{Booth's Algorithm}]{} 0.101000\bar{1}0\bar{1}00 \xrightarrow[\text{Truncation}]{} 2^{-1} + 2^{-3}.$$

Table 5.1: Comparison of a PO2 filter bank obtained by truncating the real coefficients of Johnston's 32e QMF bank [112] to 3 bits and a similar PO2 filter bank whose coefficients were scaled by 0.5565 before truncation. (Performance has been normalized with respect to the performance of the original filter bank.)

Performance Metrics	E_r	E_p	E_s	δ_p	δ_s	T_t
Filter bank A with Truncated Coefficients	6.93	9.61	1.09	1.89	1.05	1.00
Filter bank B with Scaling and Truncation	0.99	1.08	0.96	1.20	0.98	0.99

Previous work [127, 163, 37] shows that scaling has a significant impact on the optimization of coefficients in PO2 filters. That is, if each coefficient is scaled properly before the search starts (based on a heuristic objective), the quality of the final design can be improved significantly. In our case, the performance of a PO2 filter obtained by truncating its real coefficients to a fixed maximum number of ONE bits is not as good as one whose real coefficients were first multiplied by a scaling factor. We illustrate this observation in the following example.

Consider Johnston's 32e filter bank [112] as a starting point. Table 5.1 shows the metrics of two PO2 filters: Filter Bank A was obtained by truncating each of the original coefficients to a maximum of 3 ONE bits, whereas Filter Bank B was obtained by multiplying each of the coefficients by 0.5565 before truncation. Filter Bank B performs better and is almost as good as the original design with real coefficients. In fact, a design that is better than Johnston's 32e design can be obtained by using Filter B as a starting point, but no better designs were found using Filter A. This example illustrates that multiplying the filter coefficients by a scaling factor changes the bit patterns of the coefficients, which can improve the quality of the starting point when the coefficients are truncated.

Experiments also show that it is possible to find good designs without requiring the PO2 coefficients to have the same degree of precision as that of continuous coefficients. For instance, in our experiments, we restrict the minimum exponent of the ONE bits in each

procedure *find_scaling_factor*

1. $LeastSum = +\infty$;
2. **for** $ScaleFactor := 0.5000$ **to** 1.0 **step** 0.0001 **do** {
3. Multiply each filter coefficient by *ScaleFactor*;
4. Get the PO2 form of the scaled coefficients;
5. Compute the weighted sum of constraint violation: $sum := \sum_{i=1}^{5} w_i \cdot g_i$;
6. **if** ($sum < LeastSum$) **then** { $LeastSum := sum$; $BestScale := ScaleFactor$ }}
7. **return** *BestScale*

Figure 5.3: Algorithm for finding the best scaling factor, where w_i is the weight of constraint i.

coefficient (in the range $[-1, 1]$) to be -22, even though the real coefficients have a minimum exponent of -31.

To find the best scaling factor, we enumerate over different scaling constants and scale all the coefficients by a common constant before the search begins. Figure 5.3 shows a simple but effective algorithm to find the proper scaling factor to be multiplied before the coefficients are truncated. We evaluate the quality of the resulting starting point by a weighted sum of its performance metrics. Since under most circumstances, the constraint on transition bandwidth is more difficult to satisfy, we give it a weight of 100 and a weight of 1 for the other four metrics. Note that our objective in finding a good scaling factor is different from that in the previous work [127, 163, 37]. Further, note that the filter output in the final design will need to be divided by the same scaling factor.

Experimental results show that the algorithm in Figure 5.3 works fast and can complete in a few minutes, and that the scaling factors chosen are reasonable and suitable. It is important to point out that scaling does not help when the number of ONE bits allowed to represent each coefficient is large. For instance, when the maximum number of ONE bits

allowed is larger than 6, the performance of all the filters is nearly the same for all scaling factors.

As an illustration, consider the design of a PO2 QMF bank [209] based on Johnston's 32d design [112] as our constraints. Assuming a minimum exponent of -22 in each ONE bit, we enumerate and find the best scaling factor for all the coefficients to be 0.9474.

5.3.2 x Loop

The value of x is updated in Line 7 in Figure 5.2. There are two ways in which x can be updated: greedy update and hill climbing. In greedy updates, the update of x leading to the maximum improvement of $L_d(x, \lambda)$ is found before an update is made. This approach is very time consuming and may not lead to the best filter bank when DLM-QMF stops. On the other hand, in hill climbing, x is updated as soon as an improvement in $L_d(x, \lambda)$ is found. This approach is efficient and generally leads to good designs. For this reason, we use hill climbing as our update strategy.

We process all the bits of all the coefficients in a round-robin manner. Suppose γ is the filter length, l is maximum number of ONE bits that can be used for each coefficient, and the i^{th} coefficient is composed of l elements $b_{i,1}, b_{i,2}, \ldots, b_{i,l}$. We process the elements in the following order repetitively:

$$b_{1,1}, b_{1,2}, \ldots, b_{1,l}, b_{2,1}, \ldots, b_{\gamma,1}, \ldots, b_{\gamma,l}.$$

For each element $b_{i,j}$, we perturb it to be $b'_{i,j}$ that differs from $b_{i,j}$ by either the sign or the exponent or both, while maintaining $b'_{i,j}$ to be not the same in its exponent as another element of the i^{th} coefficient. Using $b_{i,1}, \cdots, b_{i,j-1}, b'_{i,j}, \cdots, b_{i,l}$ while keeping other coefficients the same, we compute the new value $L_d(x', \lambda)$ and accept the change if $L_d(x', \lambda) < L_d(x, \lambda)$.

5.3.3 λ Loop

Lines 1 and 8 in Figure 5.2 is related to the condition when λ should be updated. In traditional Lagrangian methods on continuous variables, λ is updated in every iteration. This approach does not work in DLM-QMF because if λ were updated after each update of x, then the search behaves like random probing and restarts from a new starting point even before a local minimum is reached. For this reason, λ for violated constraints should be updated less frequently, only when no further improvement in $L_d(x, \lambda)$ can be made in Line 7 of DLM-QMF for all the bits in all the coefficients. This is the approach we have taken in solving satisfiability problems [179, 226, 207]. However, we have found that more frequent updates of λ may lead to better PO2 filters. In our implementation, we update λ every time three coefficients have been processed. Since λ is updated before all the filter coefficients have been perturbed, the guidance provided by λ may not be exact.

When updating λ before the search reaches a local minimum of $L_d(x, \lambda)$, we set c in Line 8 of Figure 5.2 to be a normalized value as follows:

$$c = \frac{\theta_{speed}}{\max_{i=1}^{n} g_i} \tag{5.10}$$

where θ_{speed} is a real constant for controlling the speed of increasing λ. Experimentally, we have determined θ_{speed} to be 0.6818.

When the search reaches a local minimum of $L_d(x, \lambda)$, perturbing any single bit in any coefficient will result in no improvement of $L_d(x, \lambda)$. At this point, we need to update λ differently in order to bring the search out of the local minimum. This is done by choosing a proper value of c in Line 8 of DLM-QMF. If λ is increased too fast, then the search will restart from a random starting point. On the other hand, if λ is increased too slowly, then the trajectory will remain in the current local minimum, and updates of x in the next iteration

of DLM-QMF will bring the search to the same local minimum! Hence, we like to set c so that it will bring the search out of the current local minimum in one step, and local descents in the next iteration will head to an adjacent local minimum. This means that, after λ has been changed to λ', there exists x' in $\mathcal{N}_{dn}(x)$ such that

$$L_d(x,\lambda) \leq L_d(x',\lambda) \quad \text{and} \quad L_d(x',\lambda') < L_d(x,\lambda') \tag{5.11}$$

Replacing $L_d(x,\lambda)$ by

$$f(x) + \sum_{i=1}^{n} \lambda \cdot max(0, g_i(x)) \tag{5.12}$$

in (5.11), we get the condition before λ changes:

$$f(x) + \sum_{i=1}^{n} \lambda \cdot max(0, g_i(x)) \leq f(x') + \sum_{i=1}^{n} \lambda \cdot max(0, g_i(x')) \tag{5.13}$$

and that after λ is updated to

$$\lambda_i' = \lambda_i + c \cdot max(0, g_i(x)), \tag{5.14}$$

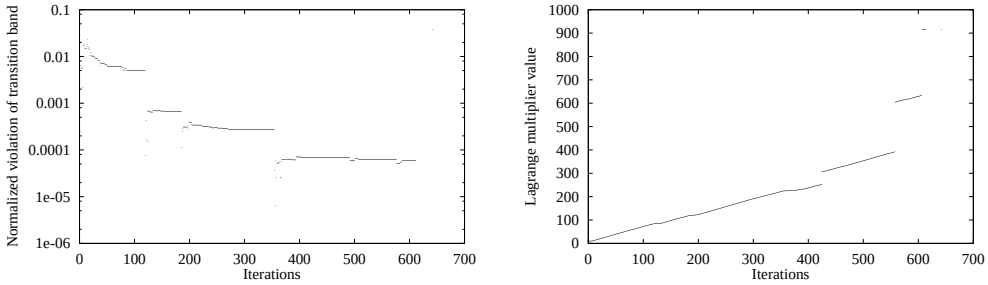
we have

$$f(x) + \sum_{i=1}^{n} \lambda_i' \cdot max(0, g_i(x)) > f(x') + \sum_{i=1}^{n} \lambda_i' \cdot max(0, g_i(x')) \tag{5.15}$$

where $max(0, g_i(x'))$ is the new violation of the i^{th} constraint at x'. After transformations, we get

$$c > \frac{L_d(x',\lambda) - L_d(x,\lambda)}{\sum_{i=1}^{n} max(0, g_i(x)) \cdot (max(0, g_i(x)) - max(0, g_i(x')))} \tag{5.16}$$

When c is large enough to satisfy (5.16) for all x', and λ is increased according to Line 8 of DLM-QMF, we are assured that there is a new x' that will cause L_d to decrease in the next iteration.



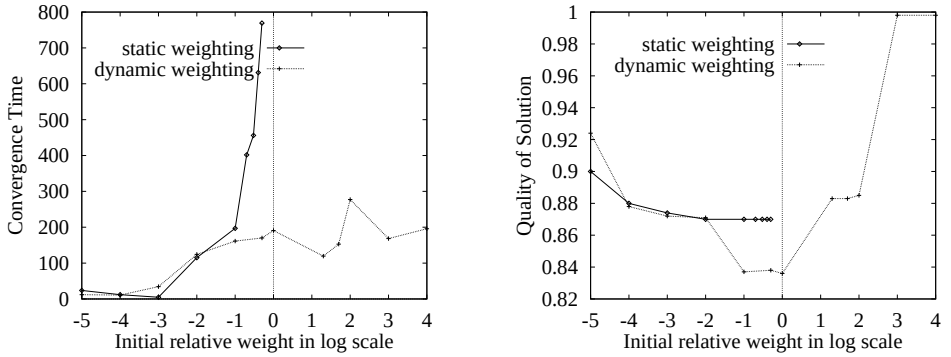
(a) Violation values of T_t (b) Corresponding λ_{T_t}

Figure 5.4: Performance progress measured during the search of Problem 32e.

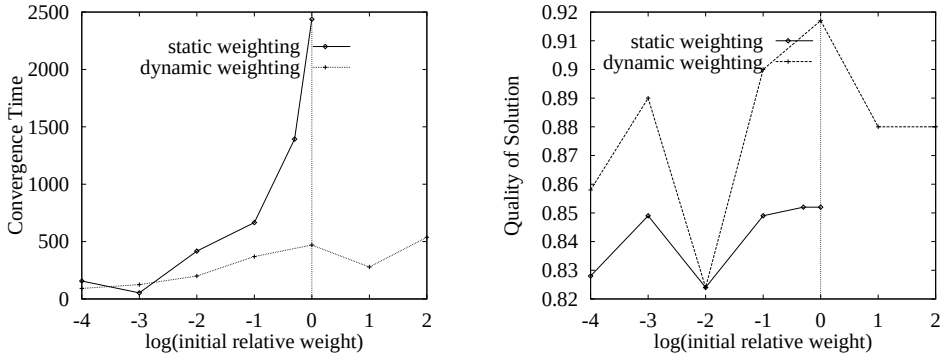
As an example, consider in Figure 5.4a the violation of transition bandwidth T_t in a typical search based on the constraints derived from Johnston's 32e filter bank [112]. Figure 5.4a shows that the value of the violation on T_t can be extremely small, on the order of 10^{-5} in the later part of the search. For such small violation values, the update of λ_{T_t} using c defined in (5.10) will result in a large number of iterations before the violation can be overcome. Using c defined in (5.16) to increase λ_{T_t}, we see in Figure 5.4b that λ_{T_t} jumps three times when the condition for updating λ was satisfied. These saved at least half of the total search time in order to find the solution.

5.4 Experimental Results

We have applied DLM-QMF to solve the QMF-bank design problems formulated by Johnston [112]. In this section, we compare the performance of designs found by DLM-QMF and those by Johnston [112], Chen *et al.* [37], *Novel* [208], simulated annealing (SA), and genetic algorithms (GA). All the experiments were run on Pentium Pro 200 computers with Linux unless specified otherwise. Note that all the filter-bank coefficients can be found at *ftp://manip.crhc.uiuc.edu/pub/papers/PostScript/J66/J66.coefficients.*



(a) Problem 32d with static weights (b) Problem 32d with dynamic weights



(c) Problem 48e with static weights (d) Problem 48e with dynamic weights

Figure 5.5: Comparison of convergence time and quality of solution between static weighting and dynamic weighting for multiplierless QMF-bank design problems 32d and 48e, where quality is measured by the ratio of the reconstruction error of our design to that of Johnston's design [112]. Hence, better designs have smaller values of solution quality.

Our goal is to find designs that are better than the baseline results across all six performance measures. Hence, we use (5.7) with the constraint bounds defined by those of the baseline designs.

5.4.1 Performance of DLM-QMF with Dynamic Weights

To design multiplierless QMF banks, we allow the maximum number of ONE bits to be 6 and the minimum exponent to be -22 for each filter coefficient. The Lagrangian method uses both static weights and dynamic weights to solve 32d and 48e problems. We compare both the convergence time and the quality of solution in terms of reconstruction error. The

starting points were obtained from Johnston's design, and the control parameters were the same as those used in the previous subsection except that the window size N_u is 10.

A comparison of DLM-QMF with static weights and that with dynamic weights is shown in Figure 5.5. Even though the initial weights have very large ranges, $[10^{-5}, 10^{4}]$ for 32d and $[10^{-4}, 10^{2}]$ for 48e, the dynamic weight-adaptation algorithm converges in less than 300 minutes for the 32d problem and 510 minutes for 48e. However, using DLM-QMF with static weights, when the initial w is larger than 1.0, the search cannot converge within 15 hours for 32d and 32 hours for 48e.

Note that, for 48e, the solution quality of DLM-QMF with static weights is slightly better than our dynamic weight-adaptation algorithm for some initial weights. This happens because the latter may change the terrain during the search and find different solutions.

Finally, Table 5.2 shows the results of solving all the Johnston's benchmarks using filter coefficients with a maximum of six ONE bits. Our results show that we were able to find designs that have better reconstruction errors, while the other performance metrics are either the same or better.

5.4.2 Comparisons of DLM-QMF and Johnston's Designs

In this section, we compare the performance of designs found by DLM-QMF and those by Johnston [112].

There are two parameters in a PO2 filter bank design: the maximum number of ONE bits in each filter coefficient and the number of filter taps. In our experiments, we have varied one while keeping the other fixed when evaluating a PO2 design with respect to a benchmark design.

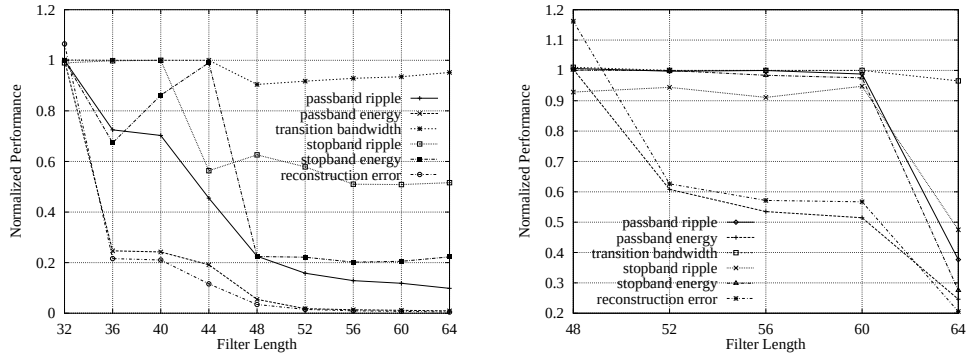
We have used closed-form integration to compute the performance values. In contrast, Johnston [112] used sampling to compute energies. Hence, designs found by Johnston are

Table 5.2: Experimental results of DLM-QMF in solving multiplierless QMF-bank design problems. The initial points of the run were from six ONE-BIT expressions of scaled Johnston's solutions.

Filter	E_r	δ_p	E_p	δ_s	E_s	T_r	Scaling Factor	Time (hrs)
16a	0.99	0.99	0.94	0.99	0.95	0.99	0.9747	1.6
16b	0.99	0.99	0.90	0.99	0.98	0.99	0.8524	2.1
16c	0.96	0.99	0.98	0.99	0.99	0.99	0.5967	3.0
24b	0.97	0.99	0.87	0.96	0.99	0.99	0.9661	5.6
24c	0.89	0.99	0.58	0.99	0.99	0.99	0.6413	12.0
24d	0.81	0.99	0.83	0.99	0.99	0.99	0.5342	13.1
32c	0.96	0.99	0.75	0.99	0.99	0.99	0.5706	12.0
32d	0.83	0.95	0.61	0.99	0.99	0.99	0.6971	3.1
32e	0.72	0.99	0.90	0.99	0.99	0.99	0.5019	7.2
48c	0.88	0.95	0.85	0.99	0.99	0.99	0.7914	18.0
48d	0.95	0.99	0.75	0.99	0.99	0.99	0.7138	23.0
48e	0.91	0.99	0.80	0.99	0.99	0.99	0.5793	8.0
64d	0.87	0.99	0.83	0.76	0.99	0.99	0.9955	9.5
64e	0.85	0.99	0.73	0.99	0.99	0.99	0.8026	24.1

not necessarily at the local minima in a continuous sense. To demonstrate this, we applied local search in a continuous formulation of the 24D design, starting from Johnston's design. We found a design with a reconstruction error of 3.83E-05, which is better than Johnston's result of 4.86E-05. By applying global search, we can further improve the design to have a reconstruction error of 3.66E-05.

We have evaluated PO2 designs obtained by DLM-QMF with respect to Johnston's designs whose coefficients are 32-bit real numbers. Using the performance of Johnston's 32e design as constraints [112], we ran DLM-QMF from 10 different starting points obtained



(a) Problem 32e (b) Problem 48e

Figure 5.6: Normalized performance for PO2 filter banks with a maximum of 3 ONE bits per coefficient and different number of filter taps.

by randomly perturbing 1% of all the coefficients of Johnston's design [112]. Each run was limited so that each ONE bit of the coefficient was processed in a round robin fashion 400 times. We then picked the best solution of the 10 runs and plotted the result in Figure 5.6, which shows the normalized performance of PO2 designs with increasing number of filter taps, while each filter coefficient has a maximum of 3 ONE bits. (The best design is one with the minimum reconstruction error if all the constraints are satisfied; otherwise, the one with the minimum violation is picked.) Our results show a design with 32 taps that is nearly as good as Johnston 32e's design. For filters with 32, 36, 40 and 44 taps, we used a starting point derived from Johnston's 32e design with filter coefficients first scaled by 0.5565 and truncated to a maximum of 3 ONE bits, and the filter coefficients of the remaining taps set to zeroes initially. Starting points for filters with longer than 44 taps were generated similarly, except that a scaling factor of 0.5584 was used instead. Our results show that, as the filter length is increased, all the performance metrics improve, except the transition bandwidth, which remains close to that of the benchmark design.

With respect to Johnston's 48e design [112], we set a limit so that each ONE bit of the coefficient was processed in a round-robin fashion 800 times, and ran DLM-QMF once from the truncated Johnston's 48e design. (The scaling factor was 0.5584 for filters with 48, 52,

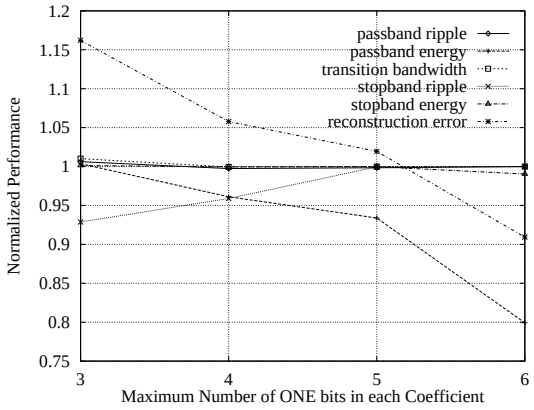


Figure 5.7: Normalized performance with respect to Johnston's 48e QMF bank [112] for PO2 filters with 48 taps and different maximum number of ONE bits per coefficient.

56, and 60 taps. The scaling factor was 0.6486 for filters with 64 taps.) Our results show that our 48-tap PO2 design is slightly worse than that of Johnston's, while PO2 designs with 52 taps or longer have performance that are either the same or better than those of Johnston's 48e design. In particular, the reconstruction error of our 52-tap PO2 design is 62% of Johnston's 48e design, while that of our 64-tap PO2 design is only 21% of Johnston's 48e design.

In the next set of experiments, we kept the same number of taps as Johnston's 48e design and increased the maximum number of ONE bits in each coefficient from 3 to 6. We set a limit so that each ONE bit of the coefficient was processed in a round-robin fashion 800 times, and ran DLM-QMF once from the truncated Johnston's 48e design. Figure 5.7 shows a design that is better than Johnston's 48e design when the maximum number of ONE bits per coefficient is 6. In this case, the reconstruction error is 91% of Johnston's 48e design. (The scaling factors used are 0.5584 for 3 bits, 0.8092 for 4 bits, 0.7409 for 5 bits, and 1.0 for 6 bits.)

Table 5.3: Comparison of normalized performance of filter banks with discrete coefficients designed by DLM-QMF and those with continuous coefficients designed by Johnston, Chen, *Novel*, simulated annealing (SIMANN), and genetic algorithms (EA-Ct and EA-Wt). Columns 2-4 show the performance of DLM-QMF using 3 ONE bits for 32-tap filters and 6 ONE bits for 64-tap filters normalized with respect to that of Johnston's 32e, 64d, and 64e filter banks [112]. Columns 5-6 show the performance of DLM-QMF using 3 ONE bits normalized with respect to that of Chen *et al.*'s 64-tap and 80-tap filter banks [37]. Columns 7-10 show the performance of 32-tap filter banks designed using *Novel* [208], SA, and EA, normalized with respect to that of Johnston's 32e filter bank and using Johnston's design as constraints.

Type	Discrete Coefficients					Continuous Coefficients			
Method	DLM-QMF					*Novel*	SA	EA-Ct	EA-Wt
Problem	J-32e	J-64d	J-64e	C-64	C-80	J-32e	J-32e	J-32e	J-32e
E_r	0.83	0.90	0.89	0.91	0.95	0.712	0.500	0.724	0.507
E_p	1.00	0.82	0.83	0.80	0.96	0.896	0.582	0.905	0.590
E_s	1.00	1.00	1.00	1.00	0.86	1.000	1.000	1.000	0.999
δ_p	1.00	0.97	1.00	1.00	1.00	1.000	1.000	1.000	0.997
δ_s	0.99	0.75	1.00	1.00	1.00	1.000	1.000	1.000	0.999
T_t	1.00	1.00	1.00	1.00	1.00	1.000	1.013	1.000	1.013

With respect to Johnston's 64d and 64e designs, Table 5.3 shows improved PO2 designs obtained by DLM-QMF using a maximum of 6 ONE bits per coefficient and 64 taps. No improvements were found when the maximum number of ONE bits is less than 6.

5.4.3 Comparisons of DLM-QMF and Other Optimization Methods

In this section, we compare the performance of designs found by DLM-QMF and those by Chen *et al.* [37], *Novel* [208], SA, and GA. Table 5.3 shows improved designs found by DLM-QMF with respect to Chen *et al.*'s designs with, respectively, 64 and 80 taps, all using a maximum of 3 ONE bits per coefficient. In these designs, we used Chen *et al.*'s designs as

starting points and ran DLM-QMF once with a limit so that each ONE bit was processed in a round-robin fashion 1,000 times.

We also compare in Table 5.3 the performance of 32e PO2 filter banks obtained by DLM-QMF with a maximum of 3 ONE bits per coefficient, and those obtained by *Novel*, simulated annealing (SA), and evolutionary algorithms (EAs). *Novel* uses a continuous trace function to bring a search out of local minima rather than restarting the search from a new starting point when the search finds a feasible design. The SA we have used is SIMANN from netlib that works on a weighted-sum formulation. The EA is Sprave's Lice (Linear Cellular Evolution) that can be applied to both constrained and weighted-sum formulations. SIMANN and EA-Wt use weighted-sum formulations with weight 1 for the reconstruction error and weight 10 for the remaining metrics. EA-Ct works on the same constrained formulation defined in (5.6). All methods were run significantly long with over 10 hours on a SUN SS20 workstation in each run.

We have tried various parameter settings and report the best solutions in Table 5.3. *Novel* improves Johnston's designs consistently. SIMANN and EA-Wt have difficulty in improving over Johnston's design across all measures and have found designs with larger transition bandwidth. EA-Ct found a design that improves Johnston's across all measures, although it is not as good as the one found by *Novel*. Note that all these designs have continuous coefficients that will need either a complex carry-save adder or a 32-bit multiplier in each tap in their hardware implementations. In contrast, DLM-QMF obtained a design that improves E_r, while the other metrics are either exactly the same or slightly better than those of Johnston's. Moreover, the design uses a maximum of five additions in each tap, leading to very cost-effective implementations.

Since existing optimization packages like SIMANN and EA works in continuous space, we have also constructed our own simulated annealing package call *discrete simulated annealing*

(DSA) that works directly in discrete space. As SA cannot handle constraints directly, we create a single objective based on a weighted sum of the objective and the constraints using static weights:

$$F = w_0 V_{E_r} + w_1 V_{\delta_p} + w_2 V_{E_p} + w_3 V_{T_t} + w_4 V_{\delta_s} + w_5 V_{E_s} \tag{5.17}$$

DSA first defines an initial temperature T_0, and selects a starting point and scaling factor in the same way as that in Section 5.3.1. It then generates a new x' in discrete space and accepts the new point at the current temperature T according to the following probability:

$$\text{probability of accepting } x' = e^{-\frac{((L(x')-L(x)))^+}{T}} \quad \text{where } a^+ = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.18}$$

Periodically, T is scaled down by $scale_T$ when the maximum violation does not decrease over a period of time (set to 10 round robins in our experiments). Finally, DSA reports the best solution when the search converges.

In our experiments using DSA, we found it very difficult to set T_0, $scale_T$, and the static weights in (5.17) that lead to better feasible PO2 designs. A set of improperly chosen parameters will lead to violations of certain constraints. This phenomenon is obvious because the weights define the relative importance of the constraints.

Our experience on DSA is illustrated in the search of a better design of Johnston's 24c filter bank. After extensive experimentation, we initialized the weights to be $w_0 = 1.0$, $w_1 = w_2 = w_4 = 5.0$, $w3 = 15.0$, $w_5 = 25.0$, and $scale_T = 0.95$. We further set the scaling factor to be 0.6413, the same as that in DLM-QMF for 24c. Table 5.4 lists the eight designs found by DSA. When the initial temperature was too high (≥ 5.0), DSA did not find any meaningful design, but found near feasible designs when the initial temperature is lower. When the initial temperature is 0.005, DSA found a feasible PO2 design with six ONE bits

Table 5.4: Experimental results of DSA in designing multiplierless QMF-bank problem 24c, starting from a six ONE-BIT expression of scaled Johnston's solutions.

T_0	E_r	δ_p	E_p	δ_s	E_s	T_r	Search Time(Hours)
5.0	-	-	-	-	-	-	1.5
1.0	0.81	0.81	0.65	1.04	0.83	0.94	0.9
0.5	0.99	1.008	0.99	0.99	0.99	0.99	1.0
0.1	0.91	0.76	0.72	1.01	0.94	1.01	1.5
0.05	0.96	0.97	0.88	1.009	0.90	1.01	1.6
0.01	0.88	0.91	0.75	1.006	0.97	0.99	1.2
0.005	0.98	0.96	0.93	0.99	0.99	0.99	2.4
0.001	0.87	0.94	0.70	1.003	0.96	0.99	1.7

that is slightly better than Johnston's 24c. Note that we did not find any feasible design after trying many other combinations of parameters.

In short, we found it difficult to use global search strategies, like SA and GA, to design PO2 filter banks formulated as weighted sum of the objective and the constraints. Without dynamically changing the weights as in DLM-QMF, it is hard to choose a proper set of weights (except by trial and error) that will allow SA or GA to converge to feasible designs. The best that SA and GA can find are designs with trade-offs on different metrics. For this reason, the method studied in this thesis represents a significant advance in solving discrete constrained optimization problems.

5.5 Summary

We have presented in this chapter an efficient implementation of discrete Lagrangian method (DLM-QMF) for designing multiplierless powers-of-two (PO2) QMF banks. Our results show that DLM-QMF can find better PO2 filter banks with very few ONE bits in each filter coefficient than other discrete and continuous optimization methods. Our design method

is unique because it starts from a constrained formulation, with the objective of finding a design that improves over a benchmark design. In contrast, existing methods for designing PO2 filter banks can only obtain designs with different trade-offs among the performance metrics and cannot guarantee that the final design is always better than the benchmark design with respect to all the performance metrics.

Chapter 6

Application II - Solving Hard Satisfiability Problems

In this chapter we study the solution of SAT problems formulated as discrete decision and discrete constrained optimization problems. Constrained formulations are better than traditional unconstrained formulations because violated constraints may provide additional forces to lead a search towards a satisfiable assignment. We examine in this chapter various formulations of the objective function, choices of neighborhood in DLM, strategies for updating Lagrange multipliers, and heuristics for avoiding traps. Experimental evaluations on hard benchmark instances pinpoint that traps contribute significantly to the inefficiency of DLM and force a trajectory to repeatedly visit the same set of or nearby points in the original variable space. To address this issue, we propose and study two trap-avoidance strategies. The first strategy adds extra penalties on unsatisfied clauses inside a trap, leading to very large Lagrange multipliers for unsatisfied clauses that are trapped more often and making these clauses more likely to be satisfied in the future. This is an *indirect* strategy because it only imposes conditions in such a way that makes it less likely for a trajectory to repeat the same traps, rather than identifying traps explicitly and avoiding them. The second strategy

is a *direct* strategy that stores information on points visited before, whether inside traps or not, and avoids visiting points close to points visited before. It can be implemented by modifying the Lagrangian function in such a way that, if a trajectory gets close to points visited before, an extra penalty will take effect and force the trajectory to a new region. It specializes to the first strategy because traps are special cases of points visited before. Finally, we show experimental results on evaluating benchmarks in the DIMACS and SATLIB archives and compare our results with existing results on GSAT, WalkSAT, LSDL, and Grasp. The results demonstrate that DLM with trap avoidance is robust as well as effective in solving hard SAT problems.

6.1 Introduction

Satisfiability (SAT) problems are the most fundamental discrete constraint-satisfaction problems among all NP-complete problems. Many real-world applications, like artificial intelligence, computer-aided design, database processing, and planning, can be formulated as SAT problems. These problems generally require algorithms of exponential complexity in the worst case in order to obtain satisfiable assignments.

A general *satisfiability* (SAT) problem is defined as follows. Given a set of n clauses $\{C_1, \cdots, C_n\}$ on m variables $x = (x_1, \cdots, x_m)$, $x_j \in \{0, 1\}$, and a Boolean formula in conjunctive normal form:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n, \tag{6.1}$$

find a truth assignment to x in order to satisfy (6.1), where a truth assignment is a combination of variable assignments that makes the Boolean formula true. For example, the assignment $(1, 0, 0, 0)$ is a solution to the following simple SAT problem with four variables

and four clauses:

$$(x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_1} \vee \bar{x_2} \vee x_4) \wedge (\bar{x_1} \vee \bar{x_3} \vee \bar{x_4}).$$

In this chapter, we formulate a SAT problem in two forms: the first as a discrete constrained decision problem, and the second as a discrete constrained optimization problem:

$$\text{Discrete Decision Problem:} \quad U_j(x) = 0 \quad \forall j \in \{1, 2, \ldots, n\}, \tag{6.2}$$

$$\begin{aligned} \text{Discrete Optimization Problem:} \quad & \text{minimize} \quad \text{objective} \\ & \text{subject to } U_j(x) = 0 \quad \forall j \in \{1, 2, \ldots, n\}, \end{aligned} \tag{6.3}$$

where $U_j(x)$ is a binary expression equal to zero when the j^{th} clause is satisfied and to one otherwise, and n is the number of clauses. The selection of a suitable objective in (6.3) is discussed in Section 6.3. We formulate SAT problems using the two formulations, transform them into discrete Lagrangian functions, and solve them using efficient global-search strategies.

We have used a constrained formulation instead of the traditional unconstrained formulation because constraints allow a search strategy to better focus its effort on unsatisfied clauses instead of searching blindly. We have found experimentally that some clauses are more difficult to satisfy than others and need to be handled differently. By choosing a constrained formulation and by assigning a dynamically changing penalty (Lagrange multiplier) according to the duration that the corresponding clause is unsatisfied, our search strategy has a better chance to focus on hard-to-satisfy clauses and find a satisfiable assignment. In contrast, all clauses in an unconstrained formulation have some fixed relative weights. Fixing the weights is undesirable because different clauses may have different degrees of difficulty to be satisfied at different times during a search, making it hard to choose suitable weights ahead of time.

This chapter is organized as follows. We survey existing work on solving SAT problems in Section 6.2. We discuss in Section 6.3 various considerations in choosing objective functions, neighborhoods, updates of Lagrange multipliers, and trap avoidance. Base on the alternatives, we then present a basic implementation of DLM used in [179] for solving SAT problems and explain why some hard instances cannot be solved. To solve those hard instances, we propose in Sections 6.4 and 6.5, respectively, efficient trap-avoidance strategies based on a constrained decision formulation and a constrained optimization formulation. The general idea of trap avoidance is to avoid visiting the same regions in the Lagrangian-function space repeatedly. Finally, Section 6.6 presents our experimental results and compares our algorithms to other well-known methods in this area.

6.2 Previous Work

Many algorithms and heuristics have been developed to solve SAT problems. In this section, we classify existing methods based on their problem formulations, first according to whether the problem variables are discrete, and second according to whether constraints are included. Methods in each class are further classified according to whether they are *complete* (proving feasibility as well as infeasibility) or *incomplete* (finding a feasible solution if one exists).

6.2.1 Discrete Formulations

In a discrete formulation, variables can only take two values, 0 or 1. This mapping is natural for SAT problems because a Boolean variable can be mapped to 0 (false) or 1 (true). The major benefit of discrete formulations lies in their efficiency of implementation.

a) Discrete constrained decision formulations, defined in (6.1), entail the search of solutions that can satisfy all the clauses. Existing methods in this class are generally complete

methods. Examples include resolution [14, 157, 68, 45], backtracking [154], and consistency testing [84, 89]. Due to the exhaustive nature of these search methods, they are expensive to use and normally have difficulty to address large-size problems.

There is little work on incomplete methods for solving problems using constrained decision formulations because incomplete methods generally focus on all the constraints in a single objective function, rather than individual constraints. However, we find that such a formulation is natural for SAT problems, after combining all the constraints into a Lagrangian function based on the theory of discrete constrained optimization using Lagrange multipliers. We show in Section 6.3.1 such a formulation and present in Section 6.4 the corresponding trap-avoidance strategies.

b) Discrete unconstrained formulations involve the minimization of $f(x)$, an objective function on the number of unsatisfied clauses. Obviously, a solution is reached when $f(x)$ is zero.

Methods in this class are generally incomplete methods and cannot prove infeasibility. They can be classified as local- and global-search strategies. Since local-search methods may be trapped by local minima in the objective space, various global-search strategies have been proposed. Next, we discuss briefly some existing methods using unconstrained formulations

Gu [87, 190, 86, 85] proposed a number of local search and parallel local search for solving SAT problems. The various strategies proposed include iterative perturbation of trajectories and randomized search for overcoming local minima [188, 189, 184, 86, 85, 89, 87, 88].

Selman *et al.* proposed GSAT [175], a randomized local search that takes the best possible moves whenever possible, and that randomly picks one variable and flips its assignment when there are several moves (or flips of variables) with the same effect. Flat moves, or sideway-

moves, are allowed to better explore plateaus in the variable space. GSAT can quickly solve randomly generated 3-SAT problems with up to 2000 variables.

WalkSAT [174, 173, 172] adds random walks ('noise') in GSAT by picking a variable in some unsatisfied clauses with probability p and flips its assignment, and by performing greedy local search (GSAT) with probability $1 - p$. It resembles simulated annealing (SA) that accepts descents with a predetermined probability and allows a certain level of ascents in variable space. WalkSAT has been very successful in solving many hard SAT problems. Some theoretical analysis of GSAT/WalkSAT can be found in [101, 70].

Tabu search [75] is a method that records previously-seen patterns using a simple data structure and tries to avoid those patterns in the future. A possible implementation is to maintain a tabu list in order to force a search to explore unknown/unvisited regions in the variable space. Its underlying idea of avoiding visits of historical points is quite general and can be found in many global-search heuristics. In fact, the two trap-avoidance strategies proposed in Sections 6.4 and 6.5 can be classified as Tabu search, although we use constrained formulations instead of unconstrained ones. An adaptive way of performing Tabu search can be found in [15].

There are algorithms that address each clause individually by introducing a weight on each clause [172, 139, 61] and by updating the weights when descents cannot be performed. In general, they can help a search overcome local minima and find solutions quickly.

Stochastic methods, such as GA and SA, have more systematic mechanisms to bring a search out of local minima. However, as reported in [173] and based on our experience in applying SA and CSA [213], they are not effective in solving large SAT problems. The major difficulty of SA lies in its requirement of an exceedingly slow cooling schedule in solving large SAT problems in order for the search to converge to satisfiable assignments. Recently,

there is some research on combining global optimization schemes, like GA, with local-search methods [60].

c) Discrete constrained optimization formulations represent a SAT instance as a constrained optimization problem in which each clause is defined as a constraint. In a typical approach, a SAT instance is formulated as an integer linear programming (ILP) problem and solved by existing algorithms, like branch-and-bound [23], cutting-plane [100], and interior-point [115, 181] methods. Although these methods sometimes perform better than resolution, they are computationally expensive and cannot solve hard-to-satisfy instances.

A second approach adds an artificial objective function $N(x)$ to the constraints defined on the clauses. The following formulation adds an objective on the number of unsatisfied clauses [179, 207]:

$$\min_{x\in\{0,1\}^m} \quad N(x) = \sum_{i=1}^{n} U_i(x) \tag{6.4}$$

$$\text{subject to} \quad U_i(x) = 0 \quad \forall i \in \{1, 2, \ldots, n\}.$$

Based on the this formulation, the discrete Lagrange-multiplier method has successfully solved many hard SAT instances in the DIMACS archive [179]. However, this objective does not add any new information in the formulation because it is simply a summation of all the constraint functions. Hence, its benefit in guidance during a search is doubtful.

6.2.2 Continuous Formulations

In continuous formulations, a SAT problem is transformed in such a way that a solution in continuous space will also be a solution to the original problem in discrete space. Such a formulation may be helpful because a search does not have to commit a variable to either 0 or 1 prematurely but allows it to take continuous values within a specified range.

However, search algorithms that work in continuous space are very expensive to apply and can only solve small SAT instances. Algorithms in this class have been developed for both unconstrained and constrained formulations.

a) Continuous unconstrained formulations define an objective as follows:

$$\min_{x \in R^m} f(x) = \sum_{i=1}^{n} C_i(x), \tag{6.5}$$

where R is the set of real numbers, and $C_i(x)$ is a function of clause C_i:

$$C_i(x) = \prod_{j=1}^{m} a_{i,j}(x_j) \tag{6.6}$$

and

$$a_{i,j}(x_j) = \begin{cases} (1 - x_j)^2 & \text{if } x_j \text{ in } C_i \\ x_j^2 & \text{if } \bar{x_j} \text{ in } C_i \\ 1 & \text{otherwise} \end{cases}$$

Many existing methods can be applied to solve (6.5). Typical local-search methods include gradient descent, conjugate gradient, Quasi-Newton methods, and sequential quadratic programming (SQP). They are fast but may be trapped easily by local minima [128, 86, 87, 88] and do not work well for large SAT instances with thousands of variables. Global-search techniques, such as clustering, generalized-gradient, Bayesian, stochastic, and trajectory methods [203, 177] can also be applied; however, they are usually much more computationally expensive than descent methods.

b) Continuous constrained-optimization formulations define a new objective together with the set of constraints defined in (6.6):

$$\begin{aligned} \min_{x \in R^m} \quad & f(x) \\ \text{subject to} \quad & C_i(x) = 0 \quad \forall i \in \{1, 2, \ldots, n\}. \end{aligned} \tag{6.8}$$

Existing approaches generally construct $f(x)$ as a (redundant) combination of all the constraint functions.

Typical methods for solving continuous constrained optimization problems include traditional Lagrange-multiplier methods [128], SQP [108, 192], CSA [213, 211], and genetic algorithms. Our experience with continuous formulations is that they do not reduce the number of local minima, and continuous algorithms are an order-of-magnitude more expensive to apply than the corresponding discrete algorithms [35].

Previous results in the area motivate us to study global-search strategies based on constrained decision/optimization formulations. A SAT instance can be formulated naturally as a constrained decision problems or as a constrained optimization problem, provided that a meaningful objective function can be defined. We discuss in Sections 6.4 and 6.5 two such formulations. In the next section, we present the implementation details of DLM for solving SAT problems.

6.3 Solving SAT using Lagrange-Multiplier Formulations in Discrete-Space

We describe the solution of SAT as a discrete Lagrangian search in this section. We first present alternative ways of formulating the heuristic objective function. This is followed by a discussion of various components of a basic implementation of DLM and its performance in solving some standard benchmark SAT instances.

Although the overall strategy for updating Lagrange multipliers may resemble existing weight-update heuristics [61, 139], our proposed formulation is based on a solid mathematical foundation of discrete constrained optimization using Lagrange multipliers. The Lagrangian search, when augmented by new heuristics presented in the following sections, provides a powerful tool to solve hard-to-satisfy SAT instances.

6.3.1 Formulations of Objective Function

Since the original discrete decision problem (6.2) does not have an objective function $f(x)$, an artificial objective function needs to be created before the problem can be transformed into a Lagrangian function (3.19). We have studied the following alternative objective functions:

a) *Constant* $f(x)$. Variations include setting $f(x)$ to zero and to the number of violated constraints [179]. In this case, the objective does not have any effect on search in a Lagrange-multiplier formulation. Although results in [179] demonstrate good performance using this formulation, better performance can be accomplished by using a suitably chosen objective. Section 6.4 presents global-search strategies based on setting $f(x)$ to zero.

b) $f(x) = -\sum_i v_i(x)$, where $v_i(x)$ is the number of variables in Clause C_i that can make C_i satisfied. For example, $v_1(x) = 1$ when $C_1 = x_1 \vee x_2$ and $x = (x_1, x_2) = (1, 0)$, because only x_1 can make this clause satisfied. Note that $v_i(x) = 0$ when C_i is unsatisfied. The intuitive reason for choosing this objective is to maximize the number of variables that can make at least one clause satisfied, since a trajectory will be quite close to a solution when there are many variables that can satisfy different clauses. However, this choice does not perform well experimentally when applied to solve hard SAT benchmarks. A possible reason is that there is no direct connection between such $f(x)$ and constraint satisfaction, and a small $f(x)$ does not necessarily mean that it is better in terms of constraint satisfaction.

c) *Setting $f(x)$ to be the sum of a few hard-to-satisfy constraints.* This approach aims to emphasize a few specific constraints in the objective. It does not work well because whether a constraint is hard to satisfy may depend on its current assignment, and it is hard to identify all of them ahead of time.

d) $f(x) = \sum_i T(u_i(x))$, where $u_i(x)$ is the number of times that Clause C_i is unsatisfied, and T is an exponential function. The goal here is penalize clauses that are unsatisfied more often than others by assigning a large weight to them in order to force them into satisfaction in the future. This approach does not work well because the constraint part in the Lagrangian function will carry smaller and smaller relative weight as more constraints are satisfied, making the search biased too much towards its historical information.

e) *Setting $f(x)$ to measure the coverage of variable space by a search trajectory.* The approach here is to find a trajectory that covers the variable space evenly, while avoiding the repetition of the same subspaces searched before. It is intuitively sound because it aims to explore the variable space efficiently, although its success depends on a suitable coverage measure that can be implemented efficiently. We study in Section 6.5, $distance\_penalty(x)$, a new objective that measures the sum of Hamming distances between the current point and points visited recently in the trajectory.

In short, our experimental results show that the first and last choices of the objective function work well in solving SAT problems. There are other choices that may be intuitively sound but do not lead to constraint satisfaction.

6.3.2 Major Components in Discrete-Space Lagrange-Multiplier Method

There are three components in DLM that may affect performance:

A) Choice of neighborhood. The neighborhood of a point defines the set of points with different original-variable assignments that need to be evaluated in order to carry out a greedy search (or hill climbing) in the Lagrangian-function space. Its choice involves trade-offs between the quality of descent directions found and the overhead in finding them.

The main advantage of using small neighborhoods is its low cost in finding descent directions. However, small neighborhoods with only nearby points may not be as good as large neighborhoods with a combination of nearby and distant points because the latter may lead to better descent directions and have a smaller chance of getting stuck in a trap (defined later) or in a flat region.

In this research, we have studied four different neighborhoods $x' \in \mathcal{N}(x)$:

a) $\mathcal{N}_1(x)$ is the set of variable assignments that differ from x by one variable. That is, x can be changed to x' by flipping just one variable, and vice versa.

b) $\mathcal{N}_2(x)$ is the set of variable assignments that differ from x by one variable, and the flip must be a variable in an unsatisfied clause. That is, $\mathcal{N}_2(x)$ only allows flips in $\mathcal{N}_1(x)$ that will change at least one unsatisfied clause into satisfaction.

c) $\mathcal{N}_3(x)$ is the set of variable assignments that differ from x by multiple variables. That is, x will change to x' by flipping one or more variables, and vice versa. Normally, we set an upper bound on the number of flips that can be performed all at once.

d) $\mathcal{N}_4(x)$ is the set of variable assignments that differ from x by multiple variables, and each of those flips must be a variable in an unsatisfied clause. This is similar to $\mathcal{N}_2(x)$ except that it allows multiple flips at the same time.

Obviously, $\mathcal{N}_2(x)$ is smaller in size than $\mathcal{N}_1(x)$, and $\mathcal{N}_4(x)$ is smaller than $\mathcal{N}_3(x)$. Both $\mathcal{N}_3(x)$ and $\mathcal{N}_4(x)$ allow multiple flips at the same time, hence providing more effective exploration of a variable space but at a significant overhead.

Our experimental results show that $\mathcal{N}_2(x)$ is better than $\mathcal{N}_1(x)$ because the latter is more closely related to unsatisfied clauses and is more pertinent to our goal of constraint satisfaction. Similarly, $\mathcal{N}_3(x)$ is better than $\mathcal{N}_4(x)$ for the same reason. Both $\mathcal{N}_3(x)$ and $\mathcal{N}_4(x)$ are much more expensive to apply in a neighborhood search than $\mathcal{N}_2(x)$, with only a marginal gain in terms of quality of descent directions found. For this reason, we select $\mathcal{N}_2(x)$ in our algorithms. Note that our implementation of neighborhood search is similar to that in GSAT because they both carry out greedy local searches.

Based on $\mathcal{N}_2(x)$, we define a *trap* as a combination of x and λ such that a point in it has one or more unsatisfied clauses, and any change to a *single* variable in x will cause the associated Lagrangian-function value L_d to increase. Note that a satisfiable assignment is not a trap because all its clauses are satisfied, even though its L_d may increase when x is perturbed.

B) Strategies for updating Lagrange multipliers. As discussed earlier, updates of Lagrange multipliers are critical in bringing a trajectory out of traps in the original variable space.

We have tried to increase Lagrange multipliers of unsatisfied clauses periodically and when trajectories get stuck in flat regions or in traps. Experimentally, we have found that periodic increases is not beneficial and may actually lead to oscillations in a trajectory. The reason is that a search may not have enough time to explore a region in detail before it is forced to leave after increasing its Lagrange multipliers. As a result, we increase Lagrange multipliers only when a search reaches a trap or a flat region.

Another observation of DLM is that a direct implementation of the discrete-space first-order conditions will cause Lagrange multipliers to grow without bound if some constraints remain unsatisfied. Large Lagrange multipliers result in a rugged Lagrangian-function space, making it difficult for a trajectory to escape from infeasible local minima. Hence, Lagrange

multipliers need to be decreased periodically in order to change the relative weights of clauses. So far, we have tried the following alternatives:

a) *Setting all Lagrange multipliers to zero periodically.* This strategy does not work well because all valuable historical information accumulated in a trajectory will be lost after resets.

b) *Controlling the ratio of the maximum to the average values of Lagrange multipliers.* The idea here is to limit the "ruggedness" of a Lagrangian-function space by scaling all Lagrange multipliers or by subtracting a common factor when the ratio is larger than a threshold. It does not work well because it is hard to choose a suitable problem-dependent threshold.

c) *Subtracting a common factor from all Lagrange multiplier periodically.* This strategy generalizes well to various SAT instances. We have adopted this strategy in our current implementation.

d) *Subtracting a common factor from all Lagrange multipliers of satisfied clauses periodically.* Intuitively, decreasing the Lagrange multipliers of all satisfied clauses actually increases the relative weights of unsatisfied clauses, thereby making them easier to be satisfied. Unfortunately, this strategy does not prove to be useful experimentally.

e) *Scaling all Lagrange multipliers by a common factor if their average is larger than a threshold.* This strategy is similar to the last two strategies except that it is triggered when the average is larger than a threshold, rather than periodically. It is used in our implementation to solve the *hanoi* problems but does not generalize well to other problems.

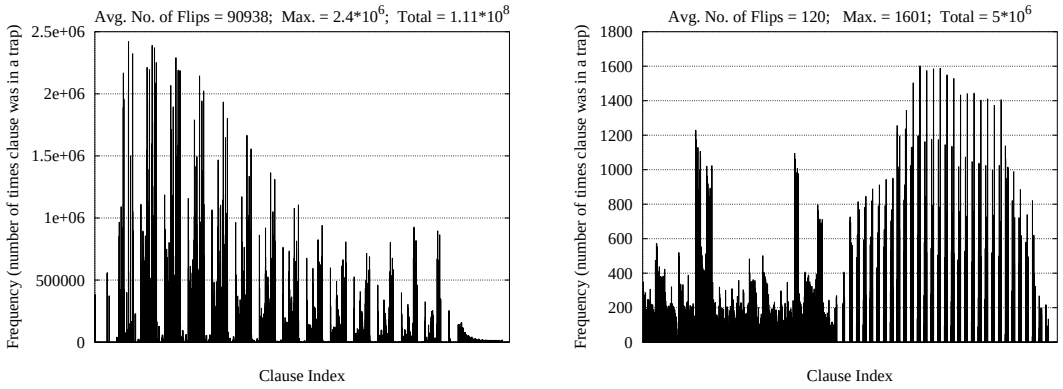
In short, a simple strategy to reduce all Lagrange multipliers periodically works well. More complex strategies do not because they have problem-dependent and difficult-to-tune parameters.

C) Strategies for trap avoidance. The occurrence of traps is a consequence of the interleaved application of descents in the original-variable space and ascents in the Lagrange-multiplier space, and is specific to our implementation of discrete-space first-order conditions. Due to the use of neighborhood function $\mathcal{N}_2(x)$, we flip one variable at a time in our implementation if the flip can decrease L_d and continue until no new single-variable flips can be found.

In general, traps cannot be prevented, and it is undesirable to get into the same set of traps repeatedly because it is obviously wasteful. To this end, we study trap-avoidance strategies using indirect controls in Section 6.4 and direct controls in Section 6.5.

A typical scenario is as follows. A clause is initially unsatisfied but becomes satisfied after a few flips due to increases of its λ. It then becomes unsatisfied again after a few more flips due to increases of λ of other unsatisfied clauses. Such cyclic state changes on a set of clauses are tremendously inefficient because the trajectory remains in an unsatisfiable state. Note that, although DLM performs global search in the variable space and can escape from traps after getting there, they cannot prevent a trajectory from revisiting the same set of traps in the future.

The occurrence of traps is illustrated in a simple implementation of the discrete-space first-order conditions. The output profiles show that some clauses are flipped frequently from being satisfied to unsatisfied, meaning that the trajectory traverses in cycles in a small region or is stuck in the original-variable space for an indefinite period of time. To demonstrate that some clauses are more likely to be unsatisfied, we plot the number of times a clause is in a trap. This is not the same as the number of times a clause is unsatisfied because a clause may be unsatisfied when outside a trap. We do not consider the path a search takes to reach a trap, during which a clause may be unsatisfied, because the different paths to reach a trap are not crucial in determining the strategy to escape from it.



a) *Hanoi4* b) *Par-16-1*

Figure 6.1: Large disparity between the maximum and the average numbers of times a clause is in traps.

Figure 6.1 shows that some clauses reside in traps more often than average in using DLM to solve *hanoi4* and *par16-1*, two hard SAT instances in the DIMACS archive. This behavior leads to an inefficient search because the trajectory may be trapped in a small region for a long time.

Ideally, we like a trajectory to never visit the same point twice in solving an optimization problem. This can be achieved if we can store all the points traversed by a trajectory in the past. It is, however, impractical in terms of memory usage and computation overhead in calculating DMPD for the ever-increasing number of points visited before.

We present in the following five approximate methods to force a trajectory not to repeat itself. The first three methods identify explicitly when a trajectory is in traps before taking corrective actions. The remaining two methods involve steps that are carried out periodically and not specific to the case when a trajectory is inside traps.

a) *Flipping multiple variables at a time.* Since a trap is defined with respect to the perturbation of one variable, we can perturb multiple variables at a time to see if L_d decreases. This is impractical due to the large number of neighborhood points that need to be considered.

procedure *DLM-BASIC-SAT*

1. Reduce the original SAT problem;
2. Generate a random starting point using a fixed seed;
3. Initialize $\lambda_i \longleftarrow 0$;
4. **while** solution not found and time not used up **do**
5. x ***Loop:*** pick $x_j \notin$ *TabuList* that reduces L_d the most;
6. Maintain *TabuList*;
7. Flip x_j;
8. λ ***Loop:*** **if** $\#_{FlatMoves} > \theta_1$ **then**
9. $\lambda_i \longleftarrow \lambda_i + \delta_o$;
10. **if** $\#_{Adjust}\%\ \theta_2 = 0$ **then**
11. $\lambda_i \longleftarrow \lambda_i - \delta_d$ **end_if**
12. **end_if**
13. **end_while**

end

Figure 6.2: *DLM-BASIC-SAT* [179]: An implementation of the basic discrete-space first-order method for solving SAT.

b) *Random restarts.* Although restarts are effective to bring a trajectory out of traps, valuable historical information accumulated in a trajectory will be lost. It is also impractical to maintain historical information on all the traps visited by a trajectory.

c) *Extra penalties on clauses in traps.* The idea is to add extra penalties on unsatisfied clauses inside traps, leading to very large Lagrange multipliers for unsatisfied clauses that are trapped more often and making these clauses less likely to be unsatisfied in the future. This is an *indirect* strategy because it only imposes conditions to make it less likely for a trajectory to repeat the same traps, rather than identifying traps explicitly and pushing a trajectory away from them. The strategy has been found to be very useful and is evaluated in Section 6.4.

d) *Keeping historical information on trajectories.* The idea is to avoid visiting points, whether inside traps or not, close to points that have already been visited before in a trajectory. It can be implemented by modifying the Lagrangian function in such a way that,

if a trajectory gets close to points visited before, then an extra penalty will take effect and force the trajectory to a new region. Of course, one can only keep track of a part of a search trajectory due to space and computation limits. Experimental results show that even keeping a small part of the historical information is very helpful in reducing the time to find satisfiable assignments. This is a *direct* strategy because it stores explicit information on points visited before. It is investigated thoroughly in Section 6.5.

e) *Periodic decreases of Lagrange multipliers* can bring a trajectory to a new region in a Lagrangian-function space. Hence, if a trajectory is inside a trap, it will be out of the trap once the space changes. This *indirect* trap-avoidance strategy has been studied in the basic DLM in [179].

6.3.3 Basic DLM for Solving SAT Problems

Figure 6.2 shows a basic implementation of DLM [179] that uses some of the concepts discussed in this section: constant $f(x)$, $\mathcal{N}_2(x)$, and periodic reduction of all Lagrange multipliers by a common factor. In addition, it uses heuristics based on tabu lists [75] and flat moves [173]. We explain each step of this algorithm when we present our proposed trap-avoidance strategy in the next section.

Table 6.1 lists the average performance of our current implementation of *DLM-BASIC-SAT* in solving DIMACS/SATLIB benchmark problems, each evaluated from ten randomly generated starting points.

Table 6.1: Performance of *DLM-BASIC-SAT* in solving DIMACS/SATLIB SAT problems. All experiments were run on a 500-MHz Pentinum-III computer with Solaris 7. (*aim* is on artificially generated random-3-SAT; *ii* is from inductive inference; *jnh* is on random SAT with variable-length clauses; *par8* is for learning parity functions; *ssa* is on circuit fault analysis; *ais* is on all-interval series; *uf* is on uniform random-3-SAT; *flat* is on "flat" graph coloring; *logistics* is on logistics planning; and *sw* is on "morphed" graph coloring [69].)

Problem ID	Succ. Ratio	CPU Sec.	Num. of Flips	Problem ID	Succ. Ratio	CPU Sec.	Num. of Flips
aim-50-1-6-yes1-1	10/10	0.01	3645	aim-50-1-6-yes1-2	10/10	0.01	1466
aim-50-1-6-yes1-3	10/10	0.01	984	aim-50-1-6-yes1-4	10/10	0.01	2060
aim-50-2-0-yes1-1	10/10	0.00	987	aim-50-2-0-yes1-2	10/10	0.01	1169
aim-50-2-0-yes1-3	10/10	0.01	2736	aim-50-2-0-yes1-4	10/10	0.01	2292
aim-50-3-4-yes1-1	10/10	0.01	1639	aim-50-3-4-yes1-2	10/10	0.01	1126
aim-50-3-4-yes1-3	10/10	0.01	729	aim-50-3-4-yes1-4	10/10	0.01	665
aim-50-6-0-yes1-1	10/10	0.01	199	aim-50-6-0-yes1-2	10/10	0.01	197
aim-50-6-0-yes1-3	10/10	0.00	171	aim-50-6-0-yes1-4	10/10	0.00	148
aim-100-1-6-yes1-1	10/10	0.02	6031	aim-100-1-6-yes1-2	10/10	0.02	4512
aim-100-1-6-yes1-3	10/10	0.02	5768	aim-100-1-6-yes1-4	10/10	0.01	2912
aim-100-2-0-yes1-1	10/10	0.03	9460	aim-100-2-0-yes1-2	10/10	0.03	9473
aim-100-2-0-yes1-3	10/10	0.02	5077	aim-100-2-0-yes1-4	10/10	0.02	7797
aim-100-3-4-yes1-1	10/10	0.05	10503	aim-100-3-4-yes1-2	10/10	0.02	2783
aim-100-3-4-yes1-3	10/10	0.04	7667	aim-100-3-4-yes1-4	10/10	0.02	4898
aim-100-6-0-yes1-1	10/10	0.01	476	aim-100-6-0-yes1-3	10/10	0.01	680
aim-100-6-0-yes1-2	10/10	0.01	229	aim-100-6-0-yes1-4	10/10	0.01	819
aim-200-1-6-yes1-1	10/10	0.17	80877	aim-200-1-6-yes1-2	10/10	0.07	29595
aim-200-1-6-yes1-3	10/10	0.15	68990	aim-200-1-6-yes1-4	10/10	0.06	29865
aim-200-2-0-yes1-1	10/10	0.44	174356	aim-200-2-0-yes1-2	10/10	0.15	57462
aim-200-2-0-yes1-3	10/10	0.10	36183	aim-200-2-0-yes1-4	10/10	0.33	129955
aim-200-3-4-yes1-1	10/10	0.54	99393	aim-200-3-4-yes1-2	10/10	0.11	18354
aim-200-3-4-yes1-3	10/10	0.11	19583	aim-200-3-4-yes1-4	10/10	0.53	98180
aim-200-6-0-yes1-1	10/10	0.02	894	aim-200-6-0-yes1-2	10/10	0.04	1961

continued on next page

continued from previous page

Problem ID	Succ. Ratio	CPU Sec.	Num. of Flips	Problem ID	Succ. Ratio	CPU Sec.	Num. of Flips
aim-200-6-0-yes1-3	10/10	0.03	1700	aim-200-6-0-yes1-4	10/10	0.02	632
ii8a1	10/10	0.01	59	ii8a2	10/10	0.00	147
ii8a3	10/10	0.02	365	ii8a4	10/10	0.02	1068
ii8b1	10/10	0.01	78	ii8b2	10/10	0.01	429
ii8b3	10/10	0.03	772	ii8b4	10/10	0.03	796
ii8c1	10/10	0.01	433	ii8c2	10/10	0.04	1714
ii8d1	10/10	0.02	1098	ii8d2	10/10	0.05	2720
ii8e1	10/10	0.01	307	ii8e2	10/10	0.03	1767
ii16a1	10/10	0.20	9360	ii16a2	10/10	0.22	10116
ii16b1	10/10	0.32	6673	ii16b2	10/10	0.23	5395
ii16c1	10/10	0.18	2696	ii16c2	10/10	0.25	8303
ii16d1	10/10	0.22	12289	ii16d2	10/10	0.49	18479
ii16e1	10/10	0.14	1042	ii16e2	10/10	0.28	8132
ii32a1	10/10	0.13	5478	ii32b1	10/10	0.01	451
ii32b2	10/10	0.03	870	ii32b3	10/10	0.05	2139
ii32b4	10/10	0.12	6268	ii32c1	10/10	0.01	254
ii32c2	10/10	0.02	325	ii32c3	10/10	0.04	1490
ii32c4	10/10	0.41	7506	ii32d1	10/10	0.03	971
ii32d2	10/10	0.08	2904	ii32d3	10/10	0.33	8676
ii32e1	10/10	0.01	168	ii32e2	10/10	0.03	892
ii32e3	10/10	0.06	2426	ii32e4	10/10	0.04	2083
ii32e5	10/10	0.11	5083				
jnh1	10/10	0.01	899	jnh7	10/10	0.01	632
jnh12	10/10	0.02	1491	jnh17	10/10	0.02	1250
jnh201	10/10	0.01	155	jnh204	10/10	0.02	2401
jnh205	10/10	0.03	2732	jnh207	10/10	0.05	6829
jnh209	10/10	0.04	4146	jnh210	10/10	0.01	506

continued on next page

continued from previous page

Problem ID	Succ. Ratio	CPU Sec.	Num. of Flips	Problem ID	Succ. Ratio	CPU Sec.	Num. of Flips
jnh212	10/10	0.24	33197	jnh213	10/10	0.02	1459
jnh217	10/10	0.01	381	jnh218	10/10	0.02	1104
jnh220	10/10	0.08	9918	jnh301	10/10	0.10	11039
par8-1-c	10/10	0.03	7698	par8-2-c	10/10	0.05	14421
par8-3-c	10/10	0.83	271275	par8-4-c	10/10	0.07	21763
par8-5-c	10/10	0.09	26736	par8-1	10/10	0.13	41810
par8-2	10/10	0.17	57521	par8-3	10/10	0.38	122311
par8-4	10/10	0.15	48256	par8-5	10/10	0.40	135212
ssa7552-038	10/10	0.13	16250	ssa7552-158	10/10	0.07	8816
ssa7552-159	10/10	0.08	8084	ssa7552-160	10/10	0.10	13742
ais6	10/10	0.01	416	ais8	10/10	0.07	7242
ais10	10/10	0.23	18916	ais12	10/10	2.19	140294
uf200-01	10/10	0.14	11810	uf200-02	10/10	0.20	22446
uf200-03	10/10	0.07	1851	uf200-04	10/10	0.11	8248
uf200-05	10/10	0.17	16162	uf200-06	10/10	0.66	80804
uf200-07	10/10	0.13	12457	uf200-08	10/10	0.08	7199
uf200-09	10/10	0.17	15005	uf200-0100	10/10	0.12	12732
flat100-1	10/10	0.36	108069	flat100-2	10/10	0.17	49512
flat100-3	10/10	0.05	11072	flat100-4	10/10	0.47	150496
flat100-5	10/10	0.09	23146	flat100-6	10/10	0.17	46834
flat100-7	10/10	2.64	859110	flat100-8	10/10	2.77	900221
flat100-9	10/10	0.06	16428	flat100-100	10/10	0.07	18502
logistics-a	10/10	0.16	17427	logistics-b	10/10	0.16	18965
logistics-c	10/10	0.21	16870	logistics-d	10/10	1.65	48603
sw100-1	10/10	0.62	117577	sw100-2	10/10	1.43	288571
sw100-3	10/10	0.97	192017	sw100-4	10/10	1.02	203461
sw100-5	10/10	0.65	127605	sw100-6	10/10	1.75	352747

continued on next page

continued from previous page

Problem ID	Succ. Ratio	CPU Sec.	Num. of Flips	Problem ID	Succ. Ratio	CPU Sec.	Num. of Flips
sw100-7	10/10	0.83	160884	sw100-8	10/10	0.99	197997
sw100-9	10/10	0.89	171101	sw100-10	10/10	0.55	104940
sw100-11	10/10	0.83	163650	sw100-12	10/10	1.64	331097
sw100-13	10/10	0.68	130571	sw100-14	10/10	0.50	93632
sw100-15	10/10	0.52	98793	sw100-16	10/10	1.79	364416
sw100-17	10/10	0.71	136542	sw100-18	10/10	0.83	162783
sw100-91	10/10	1.57	312889	sw100-92	10/10	1.04	207944
sw100-93	10/10	0.89	172169	sw100-94	10/10	1.19	237928
sw100-95	10/10	0.45	84771	sw100-96	10/10	0.71	136622
sw100-97	10/10	0.77	150486	sw100-98	10/10	1.46	295163
sw100-99	10/10	1.28	247702	sw100-100	10/10	0.47	89026
sw100-8-p0-c5	10/10	1.00	191275				

Although quite simple, *DLM-BASIC-SAT* can find solutions within seconds to most satisfiable DIMACS benchmarks, such as all the problems in the *aim*, *ii*, *jnh*, *par8*, and *ssa* classes, and most problems in SATLIB, like uniform 3-SAT *uf*, flat graph coloring *flat*, and morphed graph coloring *sw*. *DLM-BASIC-SAT* is either faster than, or at least comparable to, competing algorithms like GSAT and Grasp [130]. However, it has difficulty in solving problems in the *par16*, *hanoi*, *g*, *f2000*, and *par32* classes.

6.4 Trap Avoidance based on Constrained Decision Formulations

The Lagrangian function when SAT is formulated as a constrained decision problem (6.2) is:

$$L_d(x, \lambda) = \sum_{i=1}^{n} \lambda_i U_i(x), \tag{6.9}$$

where $U_i(x)$ is a binary function equal to zero when the i^{th} clause is satisfied and to one otherwise. Here, we assume that a constant objective function is used.

Figure 6.3 shows the pseudo code of DLM with Strategy (c) on trap avoidance (discussed in Section 6.3) in which extra penalties are added to clauses when they are found in traps [228, 227, 229]. It defines a weight for each Lagrange multiplier, identifies the location of traps, and increases the weights of all unsatisfied clauses each time the search reaches a trap. When an undesirable imbalance happens in which some clauses are trapped more often than others, the Lagrange multipliers of clauses with the largest weight are increased in order to force these clauses into satisfaction. In our implementation, imbalance happens when the ratio of the largest weight to the average is larger than a predefined threshold. If the increments to the corresponding Lagrange multipliers of those trapped clauses are large enough, these clauses are likely to be satisfied in the future. Also, the chance for the trajectory to hit the same traps again is much lower. We explain each line of the procedures in Figure 6.3 in detail next.

- Line 1 performs some straightforward reductions on all clauses with a single variable. For all single-variable clauses, we set the value of that variable to make the clause satisfied and propagate the assignment. For example, if a clause has just one variable x_4, then x_4 must be true. Reduction stops when there are no single-variable clauses.

procedure *DLM-Trap-Avoidance-SAT*

/* An implementation of the discrete first-order method for solving SAT problems */

1. Reduce original SAT problem;
2. Generate a random starting point using a fixed seed;
3. Initialize $\lambda_i \longleftarrow 0$ and $t_i \longleftarrow 0$;
4. **while** solution not found and time not used up **do**
5. x ***Loop:*** pick $x_j \notin$ *TabuList* that reduces L_d the most;
6. **If** search is in a trap **then**
7. For all unsatisfied clauses u, $t_u \longleftarrow t_u + \delta_w$ **end_if**
8. Maintain *TabuList*;
9. Flip x_j;
10. λ ***Loop:*** **if** $\#_{FlatMoves} > \theta_1$ **then**
11. $\lambda_i \longleftarrow \lambda_i + \delta_o$;
12. **if** $\#_{Adjust} \% \theta_2 = 0$ **then**
13. call *DECREASE-LAMBDA* **end_if**;
14. call *SPECIAL-INCREASE*;
15 **end_if**
16. **end_while**

end

procedure *SPECIAL-INCREASE*

/* Special increases of λ on certain clauses when their weights are imbalanced */

17. Pick a set of clauses S;
18. **if** $\frac{\max_{i \in S} t_i}{\sum_{i \in S} t_i / n} \geq \theta_3$ **then**
19. For clause i in S with the largest t_i, $\lambda_i \longleftarrow \lambda_i + \delta_s$;
20. **end_if**

end

procedure *DECREASE-LAMBDA*

/* Two alternatives to decrease Lagrange multipliers */

21. Alternative-1: $\lambda_i \longleftarrow \lambda_i - \delta_d$ for all clauses
22. Alternative-2: $\lambda_i \longleftarrow \lambda_i / \delta_d$ for all clauses when
the average λ of all clauses is larger than θ_4

end

Figure 6.3: Pseudo code of *DLM-Trap-Avoidance-SAT*.

- Line 2 generates a random starting point using a fixed seed. Note that we use the long-period random-number generator of L'Ecuyer with Bays-Durham shuffle and added safeguards rather than the default generator provided in the C library in order to allow our results to be reproducible across different platforms.

- Line 3 initializes t_i (temporary weight) and λ_i for Clause i to zero in order to make the experiments repeatable. Note that increases to λ_i are faster if t_i is larger.

- Line 4 defines a loop that will stop when time (maximum number of flips) runs out or when a satisfiable assignment is found.

- Line 5 (x Loop) chooses x_j that will reduce L_d the most among all variables not in *TabuList*. If such a variable cannot be found, then it picks x_j that will not increase L_d. Such a flip is called a *flat move* [173]. We allow flat moves in order to help a trajectory explore flat regions.

- Lines 6-7 locate a trap and increase t_u by δ_w (= 1) for all unsatisfied clauses in that trap.

- Line 8 maintains *TabuList* [75] that is important in helping a search explore flat regions effectively. Each time a variable is flipped, it will be put in *TabuList* in a FIFO order, and the oldest element will be removed from *TabuList*.

- Line 9 flips the x_j chosen (from false to true or vice versa). It also records the number of times the trajectory is doing flat moves.

- Lines 10-11 (λ Loop) increase the Lagrange multipliers of all unsatisfied clauses by δ_o (= 1) when the sum of up-hill and flat moves exceeds a predefined threshold θ_1 (50 for f, 16 for *par*16 and *par*32, 26 for g, and 18 for *hanoi*4). After increasing the Lagrange multipliers of all unsatisfied clauses, we increase Counter $\#_{Adjust}$ by one.

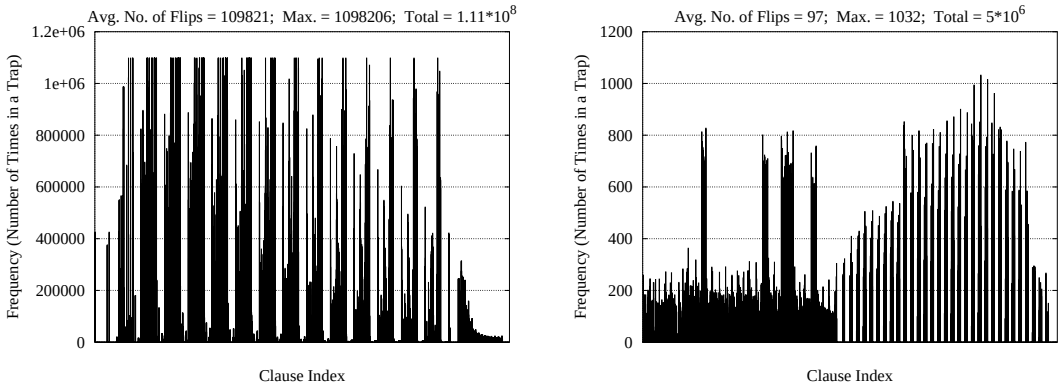
- Lines 12-13 implement Strategy (e) on trap avoidance (discussed in Section 6.3) that reduces the Lagrange multipliers of all clauses by calling Procedure *DECREASE-LAMBDA* when $\#_{Adjust}$ reaches threshold θ_2 (12 for f, 46 for $par16$, 56 for $par32$, 6 for g, and 40 for $hanio4$). This step helps change the relative weights of all the clauses and may allow the trajectory to go to another region in the Lagrangian-function space after the reduction.

- Line 14 calls Procedure *SPECIAL-INCREASE* when some clauses appear in traps more often than other clauses.

- Line 17 picks S, a problem-dependent set of clauses (for *par16-1* to *par16-5*, the set of all currently unsatisfied clauses; for others, the set of all clauses).

- Lines 18-19 compute the ratio between the maximum and the average weights to see if the ratio is imbalanced, where n is the number of clauses. If the ratio is larger than θ_3 (3 for $par16$, $par32$, and f, 1 for g, and 10 for $hanoi4$), then we increase the Lagrange multiplier of the clause with the largest weight by δ_s (1 for all problems).

- Lines 21-22 provide two alternatives to reducing Lagrange multipliers. The first alternative reduces Lagrange multipliers by a common integer δ_d (set to 1 in our experiments), whereas the second calculates the average of all Lagrange multipliers before decreasing them by a common factor δ_d ($= 2$) if the average is larger than θ_4 (set to 4.0 in our experiments). The second alternative was used in *DLM-Trap-Avoidance-SAT* when solving *hanoi4* and *hanoi4-simple* because it allowed satisfiable assignments to be found in time shorter by an order of magnitude when compared to using the first alternative [228].



a) *Hanoi4* b) *Par-16-1*

Figure 6.4: Reduced disparity between the maximum and the average numbers of times a clause is in traps using *SPECIAL-INCREASE*.

Intuitively, increasing the Lagrange multipliers of unsatisfied clauses in traps can reduce their chance to be in traps again. Figure 6.4 illustrates this point by plotting the number of times that clauses appear in traps after using *SPECIAL-INCREASE*. When compared to Figure 6.1, we see that *SPECIAL-INCREASE* has controlled the large imbalance in the number of times that clauses are unsatisfied. For *hanoi4* (resp. *par16-1*), the maximum number of times a clause is trapped is reduced by more than 50% (resp. 35%) after the same number of flips.

Note that imbalance is controlled by θ_3 and δ_s. If we use smaller θ_3 and larger δ_s, then better balance can be achieved. However, better balance does not always lead to better solutions because a search may leave a trap quickly, thereby missing some solutions for hard problems.

Table 6.2 lists the experimental results on all the hard instances solved by *DLM-Trap-Avoidance-SAT*. Besides listing the success ratios, average CPU times, and average number of flips, we also show the standard deviations of number of flips in the fifth and tenth columns. Obviously, the standard deviations of the number of flips are of the same order as the average

Table 6.2: Performance of *DLM-Trap-Avoidance-SAT* in solving some hard SAT instances and the *g*-class problems that were not solve well before [179]. Experiments were run on a 500-MHz Pentinum III computer with Solaris 7.

Problem ID	Succ. Ratio	CPU Sec.	Number of Flips		Problem ID	Succ. Ratio	CPU Sec.	Number of Flips	
			avg.	std. dev.				avg.	std. dev.
par16-1	10/10	96.5	$1.6 \cdot 10^7$	$1.3 \cdot 10^7$	par16-1-c	10/10	28.8	4850828	4990514
par16-2	10/10	95.7	$1.7 \cdot 10^7$	$1.5 \cdot 10^7$	par16-2-c	10/10	61.0	10138948	8953101
par16-3	10/10	125.7	$2.2 \cdot 10^7$	$2.4 \cdot 10^7$	par16-3-c	10/10	35.3	5920445	6394110
par16-4	10/10	54.5	$9.2 \cdot 10^6$	$1.3 \cdot 10^7$	par16-4-c	10/10	46.1	7786958	5056654
par16-5	10/10	178.5	$3.0 \cdot 10^7$	$4.2 \cdot 10^7$	par16-5-c	10/10	44.6	7386779	9140594
hanoi4	10/10	14744	$5.4 \cdot 10^8$	$3.0 \cdot 10^8$	hanoi4-simple	10/10	14236	$9.2 \cdot 10^8$	$8.3 \cdot 10^8$
par32-1-c	1/20	8622	$8.9 \cdot 10^8$	0	g125-17	10/10	144.8	754560	620193
par32-2-c	1/20	102590	$9.2 \cdot 10^9$	0	g125-18	10/10	3.98	6819	3519
par32-3-c	1/20	154607	$1.4 \cdot 10^{10}$	0	g250-15	10/10	12.9	2426	228
par32-4-c	1/20	115963	$1.1 \cdot 10^{10}$	0	g250-29	10/10	331.4	354453	257111
f600	10/10	0.664	39935	39182	f1000	10/10	3.7	217062	156537
f2000	10/10	16.2	655100	461838	bw-large-d	10/10	311.8	1328274	1073476

number of flips. *DLM-Trap-Avoidance-SAT* is, therefore, robust and insensitive to different starting points.

In Figure 6.5, we plot the distribution of number of flips for applying *DLM-Trap-Avoidance-SAT* to solve benchmark problem bw-large-d from 100 randomly generated starting points. The average number of flips for this particular problem is around $1.3 \cdot 10^6$, as shown in the last row of Table 6.2. Clearly, the maximal number of flips, $6.0 \cdot 10^6$, is of the same order as the average number of flips over 100 runs of *DLM-Trap-Avoidance-SAT*. It proves again the robustness of *DLM-Trap-Avoidance-SAT*. The averages and standard deviations presented are better than those by R-Novelty [102].

We leave comparisons of *DLM-Trap-Avoidance-SAT* to other algorithms in the area in Section 6.6. Note that *DLM-Trap-Avoidance-SAT* is one of the few methods that can solve

par16-1 to *par16-5*, *hanoi4*, *hanoi4-simple*, and *par32-1-c* to *par32-4-c*, among existing local- and global-search methods for solving SAT.

6.5 Trap Avoidance based on Constrained Optimization Formulations

The trap-avoidance strategy presented in the last section is an *indirect* strategy because it only imposes conditions in such a way that makes it less likely for a trajectory to repeat the same traps. In this section we present a *direct* strategy that remembers points visited recently in a trajectory and avoids revisiting them explicitly by imposing penalties on these points. The strategy specializes to trap avoidance because traps are special cases of points visited before. Since these penalties are to be minimized, we have included them in the objective of a constrained optimization problem, leading to the following Lagrangian function with a new heuristic objective:

$$L_d(x, \lambda) = -distance\_penalty(x) + \sum_{i=1}^{n} \lambda_i U_i(x), \tag{6.10}$$

where *distance_penalty(x)* is the sum of Hamming distances from the current point x in a trajectory to points visited in the recent past. Hence, if a trajectory is stuck in a trap, then the penalties from the current point to points in the trap will be large, leading to a large objective in (6.10) and steering the trajectory away from the trap. On the other hand, if the trajectory is not inside a trap, then the penalties to points visited before will be large, again steering the trajectory away from regions visited before.

The exact form of $distance\_penalty(x)$ is:

$$distance\_penalty(x) = (-C_L \cdot \prod_j (1 - U_j(x))) + \sum_i \min(\theta_t, |x - x_i^s|), \tag{6.11}$$

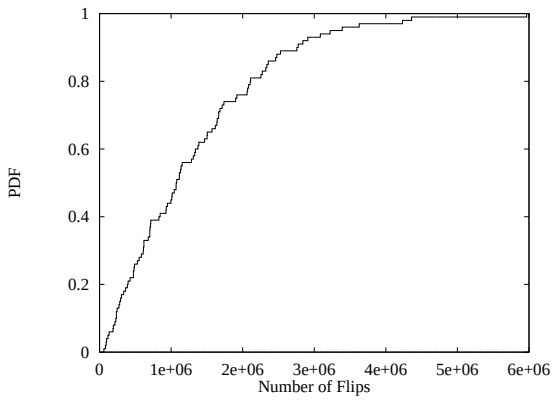


Figure 6.5: Distribution of number of flips for applying *DLM-Trap-Avoidance-SAT* to solve benchmark problem bw-large-d from 100 randomly generated starting points.

where θ_t is a positive threshold, $|x - x_i^s|$ is the Hamming distance between x and x_i^s visited before in a trajectory, i is an index to all the points saved, j is an index to all clauses, and C_L is a large positive constant. If x is not a solution, then $(-C_L \cdot \prod_j (1 - U_j(x)))$ in (6.11) will be zero, and $distance\_penalty(x)$ only measures the sum of Hamming distances. On the other hand, if x is a feasible solution that satisfies all the constraints, then $(-C_L \cdot \prod_j (1 - U_j(x)))$ will be $-C_L$, making the solution point a CLM as compared to neighboring feasible points.

Parameter θ_t is used to control the search and put an upper bound on *distance_penalty* so that it will not be a dominant factor in the new Lagrangian function. Without θ_t, the first-order search will prefer a far-away point than a point with less constraint violation, which is not desirable. In our experiments, we set θ_t to be 2. This means that, when the Hamming distance between the current point and each stored historical point of a trajectory is larger than 2, the impact of all the stored historical points on *distance_penalty* will be the same.

Due to limitations in memory usage and computation overhead in calculating DMPD on L_d, we only keep a fixed-size queue of size q_s of historical points and periodically update this queue in a FIFO manner. The period of update is based on w_s flips; namely, after w_s flips, we save the current search point in the queue and remove the oldest element from the queue.

procedure *DLM-Distance-Penalty-SAT*

1. Reduce the original SAT instance;
2. Generate a random starting point using a fixed seed;
3. Initialize $\lambda_i \longleftarrow 0$;
4. **while** solution not found and time not used up **do**
5. x ***Loop:*** pick $x_j \notin$ *TabuList* that reduces L_d the most;
6. Flip x_j;
7. **If** $\#_{Flips} \% w_s = 0$ **then**
8. Update the queue on historical points **end_if**
9. Maintain *TabuList*;
10. λ ***Loop:*** **if** $\#_{FlatMoves} > \theta_1$ **then**
11. $\lambda_i \longleftarrow \lambda_i + \delta_o$;
12. **if** $\#_{Adjust} \% \theta_2 = 0$ **then**
13. $\lambda_i \longleftarrow \lambda_i - \delta_d$; **end_if**;
14. **end_if**
15. **end_while**

end

Figure 6.6: Pseudo code of *DLM-Distance-Penalty-SAT*

Figure 6.6 shows the *DLM-Distance-Penalty-SAT* procedure for solving general SAT instances. Since a large part of it is similar to that in Figure 6.3, we only explain the differences next.

Line 3 does not initialize temporary weights t_i as in *DLM-Trap-Avoidance-SAT* because we need not identify explicitly the locations of traps here. Such situations will be handled by *distance-penalty(x)* automatically in the objective function.

Lines 7-8 maintain a queue on a fixed number of historical points. After a predefined number of flips, the algorithm inserts the current search point in the queue and deletes the oldest historical point. Note that this queue needs to be designed carefully in order to make the whole scheme efficient. In our experiments, we choose the queue size q_s to be in the range $[3, 20]$.

Lines 12-13 reduce λ_i of all clauses by δ_d (= 1) when $\#_{Adjust}$ reaches threshold θ_2 (12 for f, 46 for *par*16, 7 for g, and 40 for *hanio*4). These reductions are critical in our strategy because they help maintain the effect of *distance_penalty*(x) in L_d by keeping $\lambda^T h(x)$ in L_d in a suitable range, given that *distance_penalty* is in a fixed range that can be computed from w_s and θ_t. Without these reductions, *distance_penalty* will be relatively small when λ becomes too large and has no significant effect in avoiding regions visited before.

As compared to *DLM-Trap-Avoidance-SAT* in Figure 6.3, the new approach is simpler and has less parameters to tune. We do not show frequency diagrams like those in Figures 6.1 and 6.4 because we do not identify traps explicitly, and the Lagrangian function is different with the addition of *distance-penalty*. Note that there is more overhead in searching for a suitable variable to flip (Line 5); that is, each flip will take more CPU time than a similar flip in *DLM-Trap-Avoidance-SAT*. However, the overall CPU time is actually much shorter for most benchmark problems tested because the new trap-avoidance strategy can avoid visiting the same regions more effectively.

We have applied *DLM-Distance-Penalty-SAT* to solve some hard, satisfiable SAT instances in the DIMACS archive. Table 6.3 shows that it can solve quickly *f2000*, *par16-1-c* to *par16-5-c*, *par16-1* to *par16-5*, *hanoi4* and *hanoi4-simple* with 100% success ratio. Similar to the case of *DLM-Trap-Avoidance-SAT*, we apply *DLM-Distance-Penalty-SAT* to solve benchmark problem bw-large-d from 100 randomly generated starting points. The distribution of number of flips over 100 runs is plotted in Figure 6.7. From the distribution shown in Figure 6.7 and the standard deviations of number of iterations listed in Table 6.3, we conclude that *DLM-Distance-Penalty-SAT* is robust and insensitive to starting points.

Comparisons to other algorithms are shows in the following section.

Table 6.3: Performance of *DLM-Distance-Penalty-SAT* in solving hard SAT instances. Experiments were run on a 500-MHz Pentinum-III computer with Solaris 7.

Problem ID	Succ. Ratio	CPU Sec.	Number of Flips		Problem ID	Succ. Ratio	CPU Sec.	Number of Flips	
			avg.	std. dev.				avg.	std. dev.
par16-1	10/10	101.7	$1.3 \cdot 10^7$	$1.4 \cdot 10^7$	par16-1-c	10/10	20.8	2786081	4651849
par16-2	10/10	154.0	$2.1 \cdot 10^7$	$1.8 \cdot 10^7$	par16-2-c	10/10	51.6	6824355	6773141
par16-3	10/10	76.3	$9.8 \cdot 10^6$	$8.3 \cdot 10^6$	par16-3-c	10/10	27.5	3674644	3301287
par16-4	10/10	83.7	$1.1 \cdot 10^7$	$6.6 \cdot 10^6$	par16-4-c	10/10	35.8	4825594	3489535
par16-5	10/10	121.9	$1.5 \cdot 10^7$	$1.4 \cdot 10^7$	par16-5-c	10/10	32.4	4264095	3480451
hanoi4	10/10	6515	$6.3 \cdot 10^8$	$4.3 \cdot 10^8$	hanoi4-simple	10/10	9040	$1.1 \cdot 10^9$	$1.2 \cdot 10^9$
g125-17	10/10	41.4	434183	346459	bw-large-a	10/10	0.10	6176	4324
g125-18	10/10	4.8	22018	20025	bw-large-b	10/10	1.55	67946	61441
g250-15	10/10	17.7	2437	222	bw-large-c	10/10	72.36	1375437	1031397
g250-29	10/10	193.1	289962	200066	bw-large-d	10/10	146.28	1112332	660373
f600	10/10	0.80	73753	59421	anomaly	10/10	0.00	259	353
f1000	10/10	3.21	285024	345406	medium	10/10	0.02	1537	798
f2000	10/10	19.2	1102816	819142	huge	10/10	0.19	10320	8611

6.6 Performance Comparisons with Some Existing Algorithms

In this section, we compare our trap-avoidance strategies to some well-known global-search methods in solving DIMACS/SATLIB benchmark instances. As most other methods do not report the number of flips, we omit this measure in our comparisons. The results demonstrate the robustness and effectiveness of DLM and trap avoidance in solving satisfiable SAT instances.

Table 6.4 compares our results to GSAT/WalkSAT and *LSDL*, two incomplete stochastic search methods in the area.

When compared to GSAT/WalkSAT, DLM with trap avoidance can solve many hard-to-satisfy problems, like *par16-*, *par32-1-c* to *par32-4-c*, *hanoi4*, and *hanoi4-simple*, that cannot

Table 6.4: Performance comparisons of *DLM-Trap-Avoidance-SAT*, *DLM-Distance-Penalty-SAT*, *WalkSAT/GSAT*, and *LSDL* [41] on solving some hard SAT instances. Our experiments were run on a 500-MHz Pentinum-III computer with Solaris 7. WalkSAT/GSAT was evaluated on an SGI Challenge with MPIS processor, model unknown. The timing results of *LSDL*, using two different strategies GENET and MAX, were collected on a SUN Sparc classic, model unknown. So far, *DLM-Distance-Penalty* has not found any solution to par32-?-c, as denoted by '-' in the table. ("NR" in the table stands for "not reported.")

Problem	*DLM-Distance Penalty*		*DLM-Trap Avoidance*		*WalkSAT/ GSAT*		*LSDL*	
							GENET	MAX
ID	Succ. Ratio	Sec.	Succ. Ratio	Sec.	Succ. Ratio	Sec.	Sec.	Sec.
par16-1	10/10	101.7	10/10	96.5	NR	NR	NR	NR
par16-2	10/10	154.0	10/10	95.7	NR	NR	NR	NR
par16-3	10/10	76.3	10/10	125.7	NR	NR	NR	NR
par16-4	10/10	83.7	10/10	54.5	NR	NR	NR	NR
par16-5	10/10	121.9	10/10	178.5	NR	NR	NR	NR
par16-1-c	10/10	20.8	10/10	28.8	NR	NR	NR	NR
par16-2-c	10/10	51.6	10/10	61.0	NR	NR	NR	NR
par16-3-c	10/10	27.5	10/10	35.3	NR	NR	NR	NR
par16-4-c	10/10	35.8	10/10	46.1	NR	NR	NR	NR
par16-5-c	10/10	32.4	10/10	44.6	NR	NR	NR	NR
par32-1-c	-	-	1/20	9556	NR	NR	NR	NR
par32-2-c	-	-	1/20	113714	NR	NR	NR	NR
par32-3-c	-	-	1/20	171372	NR	NR	NR	NR
par32-4-c	-	-	1/20	128537	NR	NR	NR	NR
f600	10/10	0.80	10/10	0.664	NR	35*	NR	NR
f1000	10/10	3.21	10/10	3.7	NR	1095*	NR	NR
f2000	10/10	19.2	10/10	16.2	NR	3255*	NR	NR
hanoi4	10/10	6515	10/10	14744	NR	NR	NR	NR
hanoi4-s	10/10	9040	10/10	14236	NR	NR	NR	NR
g125-17	10/10	41.4	10/10	144.8	7/10**	264**	282.0	192.0
g125-18	10/10	4.8	10/10	3.98	10/10**	1.9**	4.5	1.1
g250-15	10/10	17.7	10/10	12.9	10/10**	4.41**	0.418	0.328
g250-29	10/10	193.1	10/10	331.4	9/10**	1219**	876.0	678.0
anomaly	10/10	0.00	10/10	0.01	NR	NR	NR	NR
medium	10/10	0.02	10/10	0.01	NR	NR	NR	NR
huge	10/10	0.19	10/10	0.18	NR	NR	NR	NR
bw-large-a	10/10	0.10	10/10	0.12	0.3***	NR	NR	NR
bw-large-b	10/10	1.55	10/10	2.43	22***	NR	NR	NR
bw-large-c	10/10	72.36	10/10	126.86	670***	NR	NR	NR
bw-large-d	10/10	146.28	10/10	254.80	937***	NR	NR	NR

*: Results from [173] for similar but not the same problems in the DIMACS archive
**: Results from [171]
***: Results from [116]

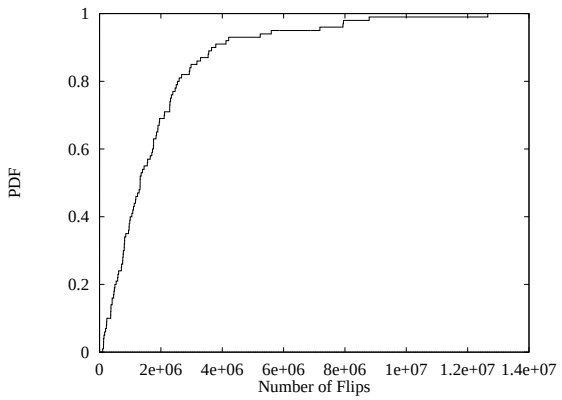


Figure 6.7: Distribution of number of flips for applying *DLM-Distance-Penalty-SAT* to solve benchmark problem bw-large-d from 100 randomly generated starting points.

be solved by GSAT/WalkSAT. Our results on the *f*-class problems are around 100 times faster and have 100% success ratio from all ten randomly generated starting points on the *g*-class problems. In contrast, GSAT/WalkSAT were not 100% successful on *g125-17* and *g125-29*. For problems in Blocksworld [166], DLM with trap avoidance also has significant improvements.

When compared to *LSDL* [41] on hard graph coloring problems, our algorithms outperform *LSDL* on *g125-17* and *g250-29*, using 60% less CPU time, whereas *LSDL* is faster in solving the easier *g125-18* and *g250-15* problems. The latter happens because DLM incurs additional overhead in maintaining information used in trap avoidance. Hence, there is no apparent advantage of using trap avoidance for very simple problems. Moreover, *LSDL* uses a more efficient representation specific to graph coloring by representing a constraint violation as a link between two nodes.

Table 6.4 shows that *DLM-Trap-Avoidance-SAT* performs worse than *DLM-Distance-Penalty-SAT* on *par16-*, *hanoi-*, *g-*, and Blocksworld problems. *DLM-Trap-Avoidance-SAT* takes 50% more CPU times to solve *hanoi4*, *g125-17*, and *g250-29*. However, it performs better on the *f*-class problems that are randomly-generated 3-SAT problems. Further, it has

Table 6.5: Performance comparisons of Grasp [130], *DLM-Trap-Avoidance-SAT* and *DLM-BASIC-SAT* on some typical DIMACS benchmarks. The timing results of Grasp were collected on a SUN SPARC 5/85 computer. '-' stands for 'not solved.'

Problem Class	Success Ratio	CPU Seconds	GRASP	
			Success Ratio	CPU Seconds
aim50-	10/10	0.01**	10/10	0.4
aim100-	10/10	0.02**	10/10	1.8
aim200-	10/10	0.32**	10/10	10.8
ii8-	10/10	0.02**	10/10	23.4
ii16-	10/10	0.25**	10/10	10311
ii32-	10/10	0.09**	10/10	7.0
ssa-	10/10	0.095**	10/10	6.5
f-	10/10	7.7*	-	-
g-	10/10	64.3*	-	-
par8-	10/10	0.25**	10/10	0.4
par16-	10/10	108*	10/10	9844
hanoi-	10/10	7778*	5/10	14480

*: Average computed using values in Table 6.3
**: Average computed using values in Tables 6.1

found solutions to *par32-1-c* to *par32-4-c* that cannot be found by *DLM-Distance-Penalty-SAT*.

Finally, we compare our algorithms to Grasp [130], one of the best complete algorithms. Since Grasp performs the best on most DIMACS benchmarks [130] when compared to other complete methods, such as POSIT [62], CSAT [49], H2R [152] and DPL—a recent implementation of the Davis-Putnam procedure [14], we compare our results with respect to Grasp only.

Since Grasp is a complete method that can prove unsatisfiability, we expect it perform not as well on satisfiable problems. Table 6.5 shows that Grasp cannot solve problems in the *f-* and *g-* classes, and that DLM have consistently better performance on the problems tested.

6.7 Summary

We have demonstrated in this chapter the theory of discrete constrained optimization using Lagrange multipliers that provides a solid mathematical foundation for handling nonlinear discrete constraints and its application in solving large yet satisfiable SAT problems. Two important observations that can be made from our results are the significance of historical information found in a search and the way this information is represented and used.

The first aspect in improving a search is to gather and maintain as much useful yet simple information as possible. In this chapter, we collect information on traps in *DLM-Trap-Avoidance-SAT* and locations of points visited in the past in *DLM-Distance-Penalty-SAT*. Although our experimental results demonstrate that such information is useful, it is by no means complete. Other information that can be collected include the locations of basins, their sizes, profiles on the difficulty of satisfaction of each clause/constraint, the relationship of one clause to another in terms of constraint satisfaction, and the lengths of cycles that a trajectory has experienced.

The second aspect in improving a search is to represent efficiently the information collected in order for it to be recalled in the future. The information can be represented as a new objective, or new constraints, or both, in a Lagrangian formulation. The secret, however, lies in the efficiency of implementation, since some of the unsolved yet feasible DIMACS benchmarks may require billions of flips, making it critical that each flip be very fast. Our experience in designing *DLM-Distance-Penalty-SAT* illustrates this point because early versions of our design were inefficient and performed poorly. Only careful re-implementations led to the current version that performs well.

In short, the keys to a successful SAT algorithm lie in the identification of useful historical information in a search and its efficient representation and recall.

Chapter 7

Conclusions and Future Work

In this chapter, we conclude our research on discrete constrained optimization using Lagrange multipliers and point out some possible future directions.

7.1 Summary of Work

In this thesis, we have developed a new theory of discrete constrained optimization using Lagrange multipliers and an associated first-order search procedure DLM for solving *general* discrete constrained NLPs without convexity or differentiability assumptions. Our proposed discrete-space first-order conditions are able to characterize all CLM_{dn} to a discrete constrained NLP. Based on the first-order conditions, we have developed a first-order search framework for locating CLM_{dn}. Together, our proposed theory and first-order methods provide a systematic way to solve discrete constrained NLPs.

In order to solve, in a unified fashion, discrete, continuous and mixed-integer constrained NLPs, we have explored carefully floating-point representations (or discretization) of continuous variables and have proved that there exists an upper bound on the error between a CGM_{cn} to a continuous or mixed-integer NLP and a CGM_{dn} to its discretized version. Such a characterization of the error introduced by discretization is very important because

it allows discrete, continuous, and mixed-integer NLPs to be solved in a unified way with bounded errors.

To make DLM an efficient search method, we have proposed various heuristics and have evaluated various combinations of these heuristics in solving constrained NLP benchmarks. Finally, we have demonstrated the power of our proposed theory and search methods by solving two real-world applications: the design of multiplierless filter banks and the solution of hard satisfiability problems.

There are two significant contributions of this research. First, we have established for the first time a systematic way to characterize and locate constrained solutions to *general* constrained optimization problems in discrete, continuous and mixed-integer space. Second, we have proposed an efficient DLM search framework that has been applied to solve some real-world applications successfully.

7.2 Future Work

It is our belief that there is a bright future for the theory of discrete constrained optimization using Lagrange multipliers and the first-order search method DLM. Our proposed theory and methods are general, simple and mathematically sound; and most important of all, they have been proved to be effective and efficient by the applications we have solved. However, there is always plenty of room for improvement.

A general direction of improvement is to reduce the CPU time required to find a desirable solution. Two such possibilities for achieving this goal are discussed next.

7.2.1 Development of More Efficient Heuristics

One promising future direction is to develop more efficient heuristics in order to locate saddle points more efficiently. For example, the neighborhood search used in this thesis has a special way to adapt its neighborhood size and is purely heuristic. One possible improvement is to use several neighborhoods of different sizes and select (in a random or a deterministic order) one in order to generate a trial point. Based on statistics collected on the quality of trial points over a period of time, the neighborhood size can then be adapted periodically.

The global search strategies adopted in this thesis are Tabu search. Although it is effective, the size of a tabu list needs to be tuned carefully in order to achieve good efficiency. Hence, better representations of previously-seen solutions and better ways of avoiding those solutions need to be developed. Also, trajectory or trace-based methods are potential candidate global-search strategies that are worth trying.

7.2.2 Reducing The Number of Probes

A second promising direction of improving solution time is to reduce the number of probes required for finding a solution with prescribed quality. One approach is to consider the tradeoffs between time and quality of a probe.

Intuitively, more probes should be performed in promising regions than in other regions. Although a pure random probe is efficient to implement, it is not intelligent in the sense that knowledge or information gathered from previous probes are ignored. Consider, for example, starting a search from inside a deep valley with no solution and a small neighborhood that does not cover solutions outside the valley. Clearly, random probing will not find anything useful. However, after a certain number of probes, the algorithm should learn that this region is not promising and should start over from other regions. Hence, an intelligent algorithm

should analyze its previous probes in order to decide where to probe next. As an example, Wah and Qian [206] have applied successfully Bayesian analysis to data sampling. In their approach, although each probe is more expensive due to Bayesian analysis, the overall CPU time is shortened significantly.

To further extend the above idea, we may generalize the concept of a *probe*. A *probe* is not necessarily *one* function evaluation of a given trial point x. Rather, it can be a local descent (greedy or gradient descent for example) from x or a certain number of algorithmic steps (iterations) by an existing search algorithm. Clearly, such a *probe* is much more expensive than a function evaluation. However, it may be able to locate very good solutions or guide a search to a promising region, leading to shorter overall CPU time. For example, some preliminary results have shown that, by combining LANCELOT [123] as an efficient search component in CSA, better results can be found than those of LANCELOT or CSA alone.

Another approach is to extend the idea of anytime search (CSA_{AT-ID}) in CSA [204] to other search heuristics. The basic philosophy of the anytime approach is to start a search with very short runs and prolong the length of subsequent runs using iterative deepening. Obviously, shorter runs usually have smaller probabilities in finding a desirable solution. The average CPU time, however, is not a monotonic function of the length of a run. If the average CPU time to find a desirable solution is convex (or has a finite minimum) to the length of a run, then it is possible to use iterative deepening to find an optimal duration of a run. This idea can be generalized to GA in order to decide a suitable population size and the number of generations performed in GA. Some preliminary results have already demonstrated that it is a very promising research direction.

Bibliography

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

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

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

[4] 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.

[5] 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.

[6] S. Anily and A Federgruen. Simulated annealing methods with general acceptance probabilities. *Journal of Appl. Prob.*, 24:657–667, 1987.

[7] 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.

[8] 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.

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

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

[11] 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.

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

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

[14] P. Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-boolean optimization. Technical Report MPI-I-95-2-003, Max-Planck-Institute fur Informatik, 1995.

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

[16] 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.

[17] Benchmarks on nonlinear mixed optimization problems. available through ftp (ftp://elib.zib-berlin.de/pub/mp-testdata/ip/index.html), 1992.

[18] 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.

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

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

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

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

[23] C. E. Blair, R. G. Jeroslow, and J. K. Lowe. Some results and experiments in programming techniques for propositional logic. *Computers and Operations Research*, 13(5):633–645, 1986.

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

[25] 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.

[26] 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.

[27] A. D. Booth. A signed binary multiplication technique. *Quart. J. Mech. Appl. Math.*, 4:236–240, 1951.

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

[29] 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.

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

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

[32] R. A. Caruana and B. J. Coffey. Searching for optimal FIR multiplierless digital filters with simulated annealing. *Technical report, Philips Laboratories*, 1988.

[33] The Optimization Technology Center. What is optimization. WWW site: http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/opt.html, 2000.

[34] 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.

[35] Y.-J. Chang and B. W. Wah. Lagrangian techniques for solving a class of zero-one integer linear programs. In *Proc. Computer Software and Applications Conf.*, pages 156–161, Dallas, TX, August 1995. IEEE.

[36] C.-K. Chen and J.-H. Lee. Design of quadrature mirror filters with linear phase in the frequency domain. *IEEE Trans. on Circuits and Systems - II*, 39(9):593–605, September 1992.

[37] C.-K. Chen and J.-H. Lee. Design of linear-phase quadrature mirror filters with powers-of-two coefficients. *IEEE. Trans. Circuits Syst.*, 41(7):445–456, 7 1994.

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

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

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

[41] 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.

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

[43] C. D. Creusere and S. K. Mitra. A simple method for designing high-quality prototype filters for m-band pseudo QMF banks. *IEEE Trans. on Signal Processing*, 43(4):1005–1007, April 1995.

[44] L. Davis. Adapting operator probabilities in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann Publishes, San Mateo, CA, 1989.

[45] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.

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

[47] DIMACS SAT benchmark suite. ftp://dimacs.rutgers.edu/pub/challenge/, 1994.

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

[49] O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier. SAT versus UNSAT. In *DIMACS Workshop on Satisfiability Testing*, New Brunswick, NJ, 1993.

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

[51] 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.

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

[53] M. H. Er and C. K. Siew. Design of FIR filters using quadrature programming approach. *IEEE Trans. on Circuits and Systems - II*, 42(3):217–220, March 1995.

[54] 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.

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

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

[57] 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.

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

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

[60] G. Folino, C. Pizzuti, and G. Spezzano. Combining cellular genetic algorithms and local search for solving satisfiability problems. *Proc. Tenth IEEE Int'l Conf. on Tools with Artificial Intelligence*, pages 192 –198, 1998.

[61] J. Frank. Learning short-term weights for GSAT. *Proc. 15'th Int'l Joint Conf. on AI*, pages 384–391, 1997.

[62] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. Ph.D. Thesis, Dept. of Computer and Information Science, Univ. of Pennsylvania, May 1995.

[63] M. R. Garey and D. S. Johnson. *Computers and intractability, a guide to the theory of NP-Completeness*. W. H. Freeman and Company, 1979.

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

[65] 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.

[66] 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.

[67] S. Geman and C-R Hwang. Diffusions for global optimization. *SIAM J. Constr. Optimization*, 24(5):1031–1043, 9 1986.

[68] M. R. Genesereth and N. J. Nilsson. *Logical Foundation of Artificial Intelligence*. Morgan Kaufmann, 1987.

[69] I. P. Gent, H. H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. *Proc. Sixteenth National Conf. on Artificial Intelligence*, pages 654–660, 1999.

[70] I. P. Gent and T. Walsh. An empirical analysis of search in GSAT. *Journal of Artificial Intelligence Research*, 1:25–37, 1993.

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

[76] 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.

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

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

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

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

[81] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Welsley publishing company, 1994.

[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] J. Gu. *Parallel Algorithms and Architectures for Very Fast AI Search*. PhD thesis, Dept. of Computer Science, University of Utah, August 1989.

[85] J. Gu. Efficient local search for very large-scale satisfiability problems. *SIGART Bulletin*, 3(1):8–12, January 1992.

[86] J. Gu. The UniSAT problem models (appendix). *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(8):865, Aug 1992.

[87] J. Gu. Local search for satisfiability (SAT) problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(4):1108–1129, 1993.

[88] J. Gu. Global optimization for satisfiability (SAT) problems. *IEEE Trans. on Knowledge and Data Engineering*, 6(3):361–381, Jun 1994.

[89] J. Gu and W. Wang. A novel discrete relaxation architecture. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(8):857–865, August 1992.

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

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

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

[93] 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.

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

[95] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1996.

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

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

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

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

[100] J. N. Hooker. Resolution vs. cutting plane solution of inference problems: some computational results. *Operations Research Letters*, 7:1–7, 1988.

[101] H. H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. *Proc. Sixteenth National Conf. on Artificial Intelligence*, pages 661–666, 1999.

[102] H. H. Hoos and T. Stutzle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24:421–461, 2000.

[103] J. J. Hopfield and D. W. Tank. Neural computation by concentrating information in time. In *Proc. National Academy of Sciences*, volume 84, pages 1896–1900, Washington, D.C., 1987. National Academy of Sciences.

[104] B. R. Horng and A. N. Wilson, Jr. Lagrange multiplier approaches to the design of two-channel perfect-reconstruction linear-phase FIR filter banks. *IEEE Trans. on Signal Processing*, 40(2):364–374, February 1992.

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

[106] R. Horst, P. M. Pardalos, and N. V. Thoai. *Introduction to Global Optimization*. Kluwer Academic Publishers, Amsterdam, 1995.

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

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

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

[110] V. K. Jain and R. E. Crochiere. Quadrature mirror filter design in the time domain. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 32(2):353–361, April 1984.

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

[112] J. D. Johnston. A filter family designed for use in quadrature mirror filter banks. *IEEE Proc. Int'l Conf. on ASSP*, pages 291–294, 1980.

[113] 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.

[114] 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.

[115] A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. Computational experience with an interior point algorithm on the satisfiability problem. *Annals of Operations Research*, 25:43–58, 1990.

[116] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. *Proc. the AAAI National Conf. on AI*, pages 1194 – 1201, 1996.

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

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

[119] D. Kodek and K. Steiglitz. Comparison of optimal and local search methods for designing finite wordlength FIR digital filters. *IEEE Trans. Circuits Syst.*, 28:28–32, 1 1981.

[120] R. D. Koilpillai and P. P. Vaidyanathan. A spectral factorization aparoach to pesudo-QMF design. *IEEE Trans. on Signal Processing*, 41(1):82–92, January 1993.

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

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

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

[124] LANCELOT Solver. http://www-neos.mcs.anl.gov/neos/solvers/NCO:LANCELOT/.

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

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

[127] Y. C. Lim and S. R. Parker. FIR filter design over a discrete power-of-two coefficient space. *IEEE Trans. Acoust. Speech, Signal Processing*, ASSP-31:583–519, June 1983.

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

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

[130] J. P. Marques-Silva and K. A. Sakalla. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Computers*, 48(5):506–521, May 1999.

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

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

[133] 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.

[134] 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.

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

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

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

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

[139] 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.

[140] K. Nayebi, T. P. Barnwell III, and M. J. T. Smith. Time-domain filter bank analysis: A new design theory. *IEEE Transactions on Signal Processing*, 40(6):1412–1429, June 1992.

[141] T. Q. Nguyen. Digital filter bank design quadratic-constrained formulation. *IEEE Trans. on Signal Processing*, 43(9):2103–2108, September 1995.

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

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

[144] 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.

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

[146] 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.

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

[148] 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.

[149] J. D. Pinter. *Global Optimization in Action*. Nonconvex Optimization and Its Applications. Kluwer Academic, 1996.

[150] 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.

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

[152] D. Pretolani. Efficiency and stability of hypergraph SAT algorithms. In D. S. Johnson and M. A. Trick, editors, *Second DIMACS Implementation Challenge*, volume 26, pages 479–498. American Mathematical Society, 1993.

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

[154] P. W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.

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

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

[157] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.*, pages 23–41, 1965.

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

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

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

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

[162] T. Sakurai, B. Lin, and A. R. Newton. Fast simulated diffusion: An optimization algorithm for multi-minimum problems and its application to MOSFET model parameter extraction. *IEEE Transactions on Computer-Aided Design*, 11(2):228–234, 2 1992.

[163] H. Samueli. An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients. *IEEE Transactions on Circuits and Systems*, 36(7):1044–1047, 1989.

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

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

[166] SATLIB suite. http://www.informatik.tu-darmstadt.de/AI/SATLIB.

[167] J. D. Schaffer and L. J. Eshelman. Designing multiplierless digital filters using genetic algorithms. In *Proc. Int'l Conf. on Genetic Algorithms*, pages 439–444, San Mateo, CA, 1993. Morgan Kaufmann.

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

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

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

[171] B. Selman. Private communcation, 1995.

[172] 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.

[173] 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.

[174] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. of 12th National Conf. on Artificial Intelligence*, pages 337–343, Seattle, WA, 1994.

[175] B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of AAAI-92*, pages 440–446, San Jose, CA, 1992.

[176] Y. Shang. *Global Search Methods for Solving Nonlinear Optimization Problems*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, August 1997.

[177] Y. Shang and B. W. Wah. Global optimization for neural network training. *IEEE Computer*, 29:45–54, March 1996.

[178] Y. Shang and B. W. Wah. Discrete Lagrangian-based search for solving MAX-SAT problems. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, pages 378–383. IJCAI, August 1997.

[179] 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.

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

[181] R. C. J. Shi, A. Vannelli, and J. Vlach. An improvement on Karmarkar's algorithm for integer programming. *COAL Bulletin of the Mathematical Programming Society*, 21:23–28, 1992.

[182] J.-J. Shyu and Y.-C. Lin. A new approach to the design of discrete coefficient FIR digital filters. *IEEE Trans. on Signal Processing*, 43(1):310–314, 1 1995.

[183] M. J. T. Smith and T. P. Barnwell III. Exact reconstruction techniques for tree-structured subband coders. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 34(3):434–441, June 1986.

[184] R. Socic and J. Gu. Fast search algorithms for the N-queen problem. *IEEE Trans. on Systems, Man, and Cybernetics*, 21(6):1572–1576, November 1991.

[185] Iraj Sodagar, Kambiz Naybei, and Thomas P. Barnwell III. Time-varying filter banks and wavelets. *IEEE Trans. on Signal Processing*, 42(11):2983–2996, November 1994.

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

[187] A. K. Soman, P. P. Vaidyanathan, and T. Q. Nguyen. Linear phase paraunitary filter banks: Theory, factorizations and designs. *IEEE Trans. on Signal Processing*, 41(12):3480–3496, December 1993.

[188] R. Sosič and J. Gu. A polynomial time algorithm for the n-queens problem. *SIGART Bulletin*, 1(3):7–11, October 1990.

[189] R. Sosič and J. Gu. 3,000,000 queens in less than one minute. *SIGART Bulletin*, 2(2):22–24, April 1991.

[190] R. Sosič and J. Gu. Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Trans. on Knowledge and Data Engineering*, 6(5):661–668, 1994.

[191] 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.

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

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

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

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

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

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

[198] T. E. Tuncer and T. Q. Nguyen. General analysis of two-band QMF banks. *IEEE Trans. on Signal Processing*, 43(2):544–548, February 1995.

[199] P. P. Vaidyanathan. *Multirate Systems and Filter Banks.* Printice-Hall Inc., 1993.

[200] 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.

[201] 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.

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

[203] 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.

[204] 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.

[205] B. W. Wah, A. Ieumwananonthachai, L. C. Chu, and A. Aizawa. Genetics-based learning of new heuristics: Rational scheduling of experiments and generalization. *IEEE Trans. on Knowledge and Data Engineering*, 7(5):763–785, October 1995.

[206] B. W. Wah and M.-L. Qian. Data sampling using Bayesian analysis and its applications in simulated annealing. In *Proc. Fifth Int'l Conf. on Computer Science and Informatics*, volume 1, pages 643–646, February 2000.

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

[208] B. W. Wah, Y. Shang, T. Wang, and T. Yu. QMF filter bank design by a new global optimization method. In *Proc. Int'l Conf. on Acoustics, Speech and Signal Processing*, volume 3, pages 2081–2084, Munich, Germany, April 1997. IEEE.

[209] B. W. Wah, Y. Shang, and Z. Wu. Discrete Lagrangian method for optimizing the design of multiplierless QMF filter banks. In *Proc. Int'l Conf. on Application Specific Array Processors*, pages 529–538. IEEE, July 1997.

[210] B. W. Wah, Y. Shang, and Z. Wu. Discrete Lagrangian method for optimizing the design of multiplierless QMF filter banks. *IEEE Transactions on Circuits and Systems, Part II*, 46(9):1179–1191, September 1999.

[211] 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.

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

[213] 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.

[214] B. W. Wah and T. Wang. Tuning strategies in constrained simulated annealing for nonlinear global optimization. *Int'l J. of Artificial Intelligence Tools*, 9(1):3–25, 2000.

[215] B. W. Wah, T. Wang, Y. Shang, and Z. Wu. Improving the performance of weighted Lagrange-multiplier methods for nonlinear constrained optimization. *Information Sciences*, 124(1-4):241–272, May 2000.

[216] B. W. Wah and Z. Wu. Discrete Lagrangian method for designing multiplierless two-channel PR-LP filter banks. *J. of VLSI Signal Processing*, 21(2):131–150, June 1999.

[217] 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.

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

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

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

[221] 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.

[222] 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.

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

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

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

[226] 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.

[227] 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.

[228] 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.

[229] Z. Wu and B. W. Wah. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In *Proc. 2000 National Conf. on Artificial Intelligence*, pages 310–315. AAAI, July-Aug. 2000.

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

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

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

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

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

[235] A. A. Zhigliavskii. *Theory of Global Random Search*. Kluwer Academic Publishers, Boston, 1991.

Vita

Zhe Wu received his B.E. degree from the Special Class for Gifted Young, University of Science & Technology of China in 1996. He received a M.S. degree in Computer Science from the University of Illinois at Urbana-Champaign in May, 1998. From 1996 to 2000, his research focused on discrete constrained optimization using Lagrange multipliers. Various applications have been solved during this period. Examples include very hard satisfiability problems, multiplierless QMF and PR-LP filter-bank designs, and discrete, continuous, and mixed-integer constrained NLP benchmarks.

His interests include nonlinear optimization, efficient algorithm designs, database, software engineering, computer networks and computer security.