GLOBAL SEARCH METHODS FOR SOLVING NONLINEAR OPTIMIZATION PROBLEMS

BY

YI SHANG

B.Engr., University of Science and Technology of China, 1988
M.Engr., Academia Sinica, 1991

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, 1997

Urbana, Illinois

© Copyright by Yi Shang, 1997

GLOBAL SEARCH METHODS FOR SOLVING NONLINEAR OPTIMIZATION PROBLEMS

Yi Shang, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1997
Benjamin W. Wah, Advisor

In this thesis, we present new methods for solving nonlinear optimization problems. These problems are difficult to solve because the nonlinear constraints form feasible regions that are difficult to find, and the nonlinear objectives contain local minima that trap descent-type search methods. In order to find good solutions in nonlinear optimization, we focus on the following two key issues: how to handle nonlinear constraints and how to escape from local minima. We use a Lagrange-multiplier-based formulation to handle nonlinear constraints, and develop Lagrangian methods with dynamic control to provide faster and more robust convergence. We extend the traditional Lagrangian theory for the continuous space to the discrete space and develop efficient discrete Lagrangian methods. To overcome local minima, we design a new trace-based global-search method that relies on an external traveling trace to pull a search trajectory out of a local optimum in a continuous fashion without having to restart the search from a new starting point. Good starting points identified in the global search are used in the local search to identify true local optima. By combining these new methods, we develop a prototype, called Novel (Nonlinear Optimization Via External Lead), that solves nonlinear constrained and unconstrained problems in a unified framework.

We show experimental results in applying Novel to solve nonlinear optimization problems, including (a) the learning of feedforward neural networks, (b) the design of quadrature-mirror-filter digital filter banks, (c) the satisfiability problem, (d) the maximum satisfiability problem, and (e) the design of multiplierless quadrature-mirror-filter digital filter banks. Our method achieves better solutions than existing methods, or achieves solutions of the same quality but at a lower cost.

DEDICATION

To my wife Lei, my parents, and my son Charles

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the support of many people. With much pleasure, I take the opportunity to thank all the people who have helped me through the course of graduate study.

First, I would like to thank my advisor, Professor Benjamin W. Wah, for his invaluable guidance, advice, and encouragement. His constant confidence in the face of research uncertainties and his persistent questioning of our methods have been and will always be inspiring to me. I would like to express my gratitude for the other members of my dissertation committee, Professors Michael T. Heath, Chung L. Liu, and Sylvian R. Ray, for their careful reading of the dissertation and valuable suggestions. I would like to thank Dr. Bart Selman for providing help and guidance in studying SAT problems. I would like to thank Ms. Jenn Wilson for carefully reading the final draft of this dissertation.

Many thanks are due to my colleagues who have contributed to this dissertation – Dr. Yao-Jen Chang for laying the foundation of this research work; Ms. Ting Yu for adapting our methods to digital filter-bank applications; Mr. Tao Wang for extending our methods to constrained optimization problems; and Mr. Zhe Wu for extending our methods to more discrete optimization applications and carrying on this work in the future. I would like to thank all my current and former colleagues – Lon-Chan Chu, Kumar Ganapathy, Arthur Ieumwananonthachai, Yao-Jen Chang, Chin-Chi Teng, Peter Wahle, Tao Wang, Chien-Wei Li, Yu Ting, Xiao Su, Jeff Monks, Zhe Wu, and Dong Lin – for their helpful discussions, useful advice, support, and friendship.

I would like to acknowledge the financial support of National Science Foundation Grants MIP 96-32316, MIP 92-18715, and MIP 88-10584, and National Aeronautics and Space Administration Grant NAG 1-613.

My family always surrounded me with love and support. I wish to thank my wife Lei for her love, understanding, encouragement, and patience. I wish to thank my parents, whose love and support have brought me this far. I want to thank my sister Ting and my brother Ying for their caring and support throughout my education. I want to thank my parents-in-law for their help in the final stage of my graduate education. I wish to dedicate this thesis to my wife, my parents, and my little boy, Charles.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

1. INTRODUCTION

Many application problems in engineering, decision sciences, and operations research are formulated as optimization problems. Such applications include digital signal processing, structural optimization, engineering design, neural networks, computer aided design for VLSI, database design and processing, nuclear power plant design and operation, mechanical engineering, and chemical process control [21, 66, 188, 246]. Optimal solutions in these applications have significant economical and social impact. Better engineering designs often result in lower implementation and maintenance costs, faster execution, and more robust operation under a variety of operating conditions.

Optimization problems are made up of three basic components: a set of unknowns or variables, an objective function to be minimized or maximized, and a set of constraints that specify feasible values of the variables. The optimization problem entails *finding values of the variables that optimize (minimize or maximize) the objective function while satisfying the constraints.*

There are many types of optimization problems. Variables can take on continuous, discrete, or symbolic values. The objective function can be continuous or discrete and have linear or nonlinear forms. Constraints can also have linear or nonlinear forms and be defined implicitly or explicitly, or may not even exist.

In this chapter, we first formally define optimization problems and identify the classes of problems addressed in this thesis. We then summarize the characteristics of nonlinear optimization problems and solution methods. Finally, we present the goals and approaches of this research, outline the organization of this thesis, and summarize our contributions.

1.1 Optimization Problems

A general minimization problem is defined as follows:

Given a set D and a function $f : D \to P$, find at least one point $\mathbf{x}^* \in D$ that satisfies $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in D$, or show the non-existence of such a point.

A mathematical formulation of a minimization problem is as follows:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in D. \end{aligned} \tag{1.1}$$

In this formulation, $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ is an n-dimensional vector of unknowns. The function f is the *objective* function of the problem, and D is the feasible domain of $\mathbf{x}$ specified by *constraints.*

A vector, $\mathbf{x}^* \in D$, satisfying $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in D$ is called a *global minimizer* of f over D. The corresponding value of f is called a *global minimum.* A vector $\mathbf{x}^* \in D$ is called a *local minimizer* of f over D if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in D$ close to $\mathbf{x}^*$. The corresponding value of f is called a *local minimum.* Note that since max $f(D) = -\min(-f(D))$, *maximization* problems can be transformed into *minimization* problems shown in (1.1). We use optimization and minimization interchangeably in this thesis.

Optimization problems occur in two levels: the problem-instance level and the meta level. In the problem-instance level, each problem instance is solved independently to obtain its optimal solution. In contrast, in the meta level, heuristics and algorithms are designed to solve a class of problems. The optimal solution in this case should generalize well to the whole group of problems, even if it is obtained based on one or a subset of problem instances.

Consider supervised learning of feedforward artificial neural networks (ANNs, to be studied in Chapter 4) as an example to illustrate these two levels of optimization. ANNs consist of a large number of interconnected simple processing units (neurons). Each processing unit is an elementary processor performing some primitive operations, like summing the weighted inputs coming to it and then amplifying or thresholding the sum. The output of each neuron is fed into neurons through connections, and weighted by connection weights. Hence, in ANNs, information is stored in synaptic connections and represented as connection weights. Generally speaking, a neural network performs a mapping from an input space to an output space. In supervised learning, the target outputs are known ahead of time, and neural networks are trained to perform the desired mapping functions.

Problem-instance-level optimization occurs when the architecture of a neural network, such as the number of neurons, activation function of each neuron, and connection pattern, is fixed, and only the values of connection weights are adjustable. In this optimization problem, the connection weights are variables, and the objective is to minimize the difference between the output of the neural network and the desired output.

Meta-level optimization occurs, on the other hand, when the architecture of a neural network is not fixed. For example, the neural network can be a feedforward or a feedback network, a multi-layer network with or without shortcuts, or a pyramid network. The activation function of each neuron can be a step function, a sigmoidal function, or a radial basis function. An appropriate architecture needs to be chosen in order to solve a class of problems well. The solution of meta-level optimization is a set of heuristics that produce appropriate network architectures based on the characteristics of application-specific training patterns.

The variables of a nonlinear problem can take on various values, such as discrete, continuous, or mixed values. In most problems, variables have well-defined physical meaning. However, it is not rare that variables are not well-defined or even undefined in many real-world applications. In those circumstances, variables take on noisy, approximate, or imprecise values.

Consider the previous example on supervised learning of neural networks. In most problem-instance-level optimization problems, the variables – connection weights – take on continuous real numbers. However, for finite-precision and multiplierless neural networks, connection weights are restricted to take on values from a discrete set, such as {-1, 0, or 1}. The meaning of variables in a problem-instance-level learning problem is well-defined.

It represents the exhilarating or inhibiting relationship between neurons and take continuous, discrete, or mixed values. On the other hand, the variables in meta-level learning are not well-defined. There are a large number of different architectures, and their performance measures on classes of tasks are usually noisy and inaccurate.

The search space of an optimization problem can be finite or infinite. Continuous optimization problems consist of real-value variables, and the search space is usually infinite. For instance, when the connection weights take on real values, supervised learning of neural networks have infinite search space.

Many discrete optimization problems have finite search spaces whose complexity can be polynomial or exponential as a function of the number of inputs. For example, the search space of finding the shortest path between two nodes in a graph is polynomial with respect to the number of nodes. However, the search space of a maximum satisfiability problem (to be studied in Chapter 6) is exponential with respect to the number of literals. Search spaces with exponential complexity can be enormously large when the problem size is large. For example, a maximum satisfiability problem with 1000 variables has $2^{1000} \approx 10^{300}$ possible assignments, although variables only take value 0 or 1.

The objective function of an optimization problem may or may not have a closed-form formula. Some objectives are evaluated deterministically and return the same value for the same set of variables every time. Other objectives are evaluated probabilistically, and could have different values every time they are evaluated.

Again, in the example on supervised neural-network learning, a problem instance-level optimization problem usually has a closed-form nonlinear objective function that is computed deterministically. A meta-level optimization problem often does not have a closed-form objective function and is evaluated through a procedure. Due to the randomness involved in evaluating noisy and limited training patterns, the results may be probabilistic.

Similar to objective functions, constraints may or may not have a closed-form formula. For example, in the QMF filter-bank design problem (to be studied in Chapter 5), some constraints, such as stopband and passband energy, have a closed-form formula. Other constraints, including stopband ripple, passband ripple and transition bandwidth, do not have a closed-form formula and have to be evaluated numerically.

In constrained optimization problems, some constraints are hard constraints, meaning that they must be satisfied by solutions. Others are soft constraints, for which some degree of violation is acceptable. In satisfiability problems (to be studied in Chapter 6), all constraints are hard constraints and must not be violated. In contrast, in neural-network learning problems, soft constraints are sometimes used to specify the desirable ranges of connection-weight values. In this case, some degree of violations are acceptable as long as the violations result in better neural networks.

In addition to constraints specifying the feasible domain of an optimization problem, constraints on computational resources are also important for solving real-world applications. For the application problems studied in Chapters 4, 5, and 6 (including feedforward neural-network learning, digital filter-bank design, satisfiability, maximum satisfiability, and multiplierless filter-bank design), constraints on computational resources, specifically, CPU-time constraints, are usually imposed in our experiments. Since it can be very time-consuming to solve large nonlinear optimization problems, time constraints are used to make sure that the solution process finishes in a reasonable amount of time.

As we have seen, there are many types of optimization problems with distinctive characteristics. In this thesis, we focus on problem-instance-level optimization problems whose objective and constraints are evaluated deterministically.

1.2 A Taxonomy of Optimization Problems

Figure 1.1 shows a taxonomy of optimization problems and the classes of problems addressed in this thesis. In this figure, optimization and decision problems are classified according to the attributes of variable type, presence of constraints, and complexity.

Although our focus is on optimization problems, decision problems are closely related to optimization problems, especially constrained optimization problems. In our research, we

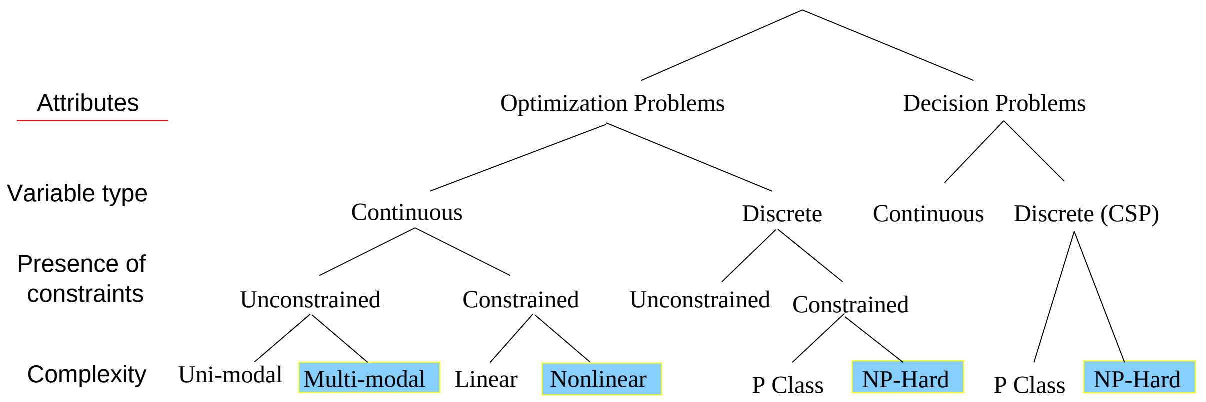


Figure 1.1: A classification of optimization and decision problems. The classes of problems addressed in this thesis are shaded.

solve decision problems through optimization approaches. Therefore, we show a classification of decision problems in Figure 1.1.

1.2.1 Continuous Optimization Problems

Optimization problems are classified into *continuous* and *discrete* problems. A problem is continuous if the unknowns (variables) take on continuous real values, i.e., D in (1.1) consists of real numbers. A problem is discrete if the unknowns take on discrete, usually integer values.

Continuous optimization problems are further classified into *constrained* optimization and *unconstrained* optimization based on the presence of constraints. Problems without constraints fall into the class of unconstrained optimization.

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in R^n \end{aligned} \tag{1.2}$$

There are two types of optimal points of an optimization problem: local minima and global minima. A local minimum has the smallest value in a local feasible region surrounding itself, whereas a global minimum has the smallest value in the whole feasible domain. In a continuous unconstrained optimization problem, an objective function is minimized in the real domain. An unconstrained optimization problem is *uni-modal* if its objective function is convex. A uni-modal problem has one local minimum, which is the global minimum at the same time. A problem is *multi-modal* if its objective function has more than one local minimum. General nonlinear functions are multi-modal and may have many local minima that are not global minima.

When each dimension of D in (1.1) consists of real values constrained by simple lower and/or upper bounds, the corresponding optimization problem is called a *simple-bounded* continuous optimization problem.

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & l \le \mathbf{x} \le u \\ & \mathbf{x} \in R^n \end{aligned} \tag{1.3}$$

where l and u are constants.

We put simple-bounded constrained problems in the class of unconstrained optimization because simple-bound constraints (for example, $1 \le x \le 5$) are easy to handle, and algorithms for problems without constraints and with simple-bound constraints are similar. Although variables in unconstrained optimization problems can be any real numbers, the search for global optima of multi-modal functions often takes place in some limited regions of interests. Hence, unconstrained nonlinear optimization problems and simple-bounded constrained problems are usually solved in similar ways. The remaining problems with nontrivial constraints belong to the class of constrained optimization.

The Rastrigin function from optimal control application is an example of a simple-bounded optimization problem [253]. The minimization problem is formulated as follows:

$$\min_{\mathbf{x} \in R^2} f(\mathbf{x}) = x_1^2 + x_2^2 - \cos 18x_1 - \cos 18x_2$$

$$-1 \le x_i \le 1, \quad i = 1, 2$$

Its global minimum is equal to -2 and the minimum point is at (0, 0). There are approximately 50 local minima in the region bounded by the two constraints.

As shown in Figure 1.1, constrained continuous problems are classified into linear and nonlinear depending on the form of constraint functions. When D in (1.1) is bounded by linear functions, the corresponding optimization problem is called a linear constrained problem. This class of problems is relatively easy to solve. Among these problems, two types of problems that have been studied extensively and solved well are linear programming and quadratic programming problems. If the objective f is a linear function of the unknowns and the constraints are linear equalities or inequalities in the unknowns, the corresponding optimization problem is called a *linear programming* problem.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} x_i \geq b_j \qquad \text{for } j = 1, 2, \cdots, m \\ & \mathbf{x} \geq 0 \\ & \mathbf{x} \in R^n \end{aligned} \tag{1.4}$$

where c_i, a_{ij}, and b_j are constant real coefficients. A linear programming problem has one local minimum, which is also the global minimum.

If the objective f is a quadratic function and the constraints are linear functions, the corresponding optimization problem is called a quadratic programming problem. As an example,

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_i x_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} x_i \geq b_j \qquad \text{for } j = 1, 2, \cdots, m \\ & \mathbf{x} \geq 0 \\ & \mathbf{x} \in R^n \end{aligned} \tag{1.5}$$

where c_i, d_{ij}, a_{ij} and b_j are constant real coefficients. Linear constrained problems, including linear programming and quadratic programming problems, have been studied extensively. Efficient algorithms have been developed to solve them very well [59, 84, 161, 216].

When D is characterized by nonlinear functions, the optimization problem is called a *nonlinear optimization* or a *nonlinear programming* problem.

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h(\mathbf{x}) = 0 \\ & g(\mathbf{x}) \leq 0 \\ & \mathbf{x} \in R^n \end{aligned} \tag{1.6}$$

where $h(\mathbf{x})$ represents a set of equality constraints, and $g(\mathbf{x})$, a set of inequality constraints. In a nonlinear optimization problem, the objective function as well as the constraint functions are nonlinear. Nonlinear constrained problems are difficult to solve because nonlinear constraints may constitute feasible regions that are difficult to find, and nonlinear objectives may have many local minima. Many application problems fall into this class [82, 126].

In this thesis, we focus on general multi-modal nonlinear optimization problems and propose some new methods. Uni-modal problems are relatively easy to solve. They have been studied extensively for decades. Many algorithms have been developed, including gradient descent, Newton's method, quasi Newton's methods, and conjugate gradient methods [63, 79, 80, 161]. These algorithms are very efficient and can solve optimization problems with tens of thousands of variables within seconds.

Multi-modal problems are much more difficult and yet plentiful in real applications. Although many methods have been developed for multi-modal problems, most of them are efficient only for subclasses of problems with specific characteristics. Methods for general multi-modal problems are far from optimal [82, 125, 188, 253].

1.2.2 Discrete Optimization Problems

When the feasible set D in (1.1) consists of discrete values, the problem is called a discrete optimization problem. Discrete optimization is a field of study in combinatorics, as well as in mathematical programming. A classification of combinatorial problems consists of four categories: (a) evaluation of required arrangements, (b) enumeration of counting of possible arrangements, (c) extremization of some measure over arrangements, and (d) existence of specific arrangements [190]. Discrete optimization usually refers to the third category. In our study, we also consider the last category a special case of discrete optimization after introducing an artificial objective function.

Some famous examples of discrete optimization problems are as follows:

- *Knapsack Problem.* Determine a set of integer values x_i, $i = 1, 2, \cdots, n$, that minimize $f(x_1, x_2, \cdots, x_n)$, subject to the restriction $g(x_1, x_2, \cdots, x_n) \geq b$ where b is a constant.
- *Traveling Salesman Problem.* Given a graph (directed or undirected) with specified weights on its edges, determine a tour that visits every vertex of the graph exactly once and that has minimum total weight.
- *Bin Packing.* Given a set of weights, w_i, $1 \leq i \leq n$, and a set of bins, each with fixed capacity W, find a feasible assignment of weights to bins that minimizes the total number of bins used.

Discrete optimization problems, including the above examples, can be expressed in the following *integer programming* (IP) formulation.

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h(\mathbf{x}) = 0 \\ & g(\mathbf{x}) \leq 0 \\ & \mathbf{x} \in I^n \end{aligned} \tag{1.7}$$

where I represents the integer space. Variable domain D consists of integers and is characterized by equality constraints $h(\mathbf{x})$ and inequality constraints $g(\mathbf{x})$. Notice the similarity of (1.7) to the formulation of continuous constrained optimization problem in (1.6).

As shown in Figure 1.1, we classify discrete optimization problems according to the existence of constraints and their computational complexity. When there is no constraint besides integral requirements of variables, the problem is called an *unconstrained* problem. With additional constraints, the problem is called a *constrained* problem. An unconstrained discrete optimization problem has the following form:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in I^n \end{aligned} \tag{1.8}$$

When D consists of integer values constrained by simple lower and/or upper bounds, the optimization problem is called a *simple-bounded discrete* optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \\ & \mathbf{x} \in I^n \end{aligned} \tag{1.9}$$

As in the continuous case, we associate simple-bounded discrete optimization problems with the class of unconstrained problems. Algorithms for these problems are similar. The majority of discrete optimization problems have some constraints, and few are unconstrained optimization problems.

When domain D and objective f are characterized by linear functions, the optimization problem is called an *integer linear programming* (ILP) problem. Here, the objective function is linear in the unknowns, and the constraints are linear equalities or inequalities in the unknowns:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i \geq b_j \qquad \text{for } j = 1, 2, \cdots, m \\ & \mathbf{x} \geq 0 \\ & \mathbf{x} \in I^n \end{aligned} \tag{1.10}$$

where c_i, a_{ij} and b_j are constant real/integer coefficients. Many discrete optimization problems can be formulated as ILPs by introducing additional variables and constraints.

An example of ILP problems is the following inequality-form knapsack problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_i x_i \geq b_j \\ & x_i = 0 \text{ or } 1, \quad a_i \text{ and } c_i \text{ are integers} \quad i = 1, 2, \cdots, n \end{aligned}$$

Discrete constrained optimization problems have been studied extensively in computer science and operations research [52,86,141,162,186]. Based on their computational complexity, there are two important classes of discrete optimization problems: Class P and Class NP. Class P contains all problems that can be solved by algorithms of polynomial-time complexity, where P stands for polynomial. Examples of Class P problems are matching, spanning trees, network flows, and shortest path problems. Class P problems have been well studied.

Many discrete problems in real-world applications do not have polynomial-time algorithms and are more difficult to solve than those in Class P. Among them, one important class is NP. Class NP includes all those problems that are solvable in polynomial time if correct polynomial-length guesses are provided. Examples are knapsack, traveling-salesman, and bin-packing problems. Problems to which all members of NP polynomially reduce are called NP-hard. The Class NP contains the Class P as well as a great many problems not belonging to P. The focus of this thesis is on NP-hard problems that are not polynomial-time solvable.

1.2.3 Decision Problems

A *decision problem* differs from a constrained optimization problem in the way that it has no objective function. Usually, a decision problem is specified by a set of constraints, and a feasible solution that satisfies all the constraints is desired, or infeasibility is derived.

Decision problems are classified as continuous or discrete according to their variable domains. In a continuous decision problem, each variable is associated with a continuous domain. There are not many continuous decision problems in real applications. In this thesis, we focus on NP-hard discrete decision problems.

In a discrete decision problem, each variable is associated with a discrete, finite domain. Discrete decision problems are referred to as *constraint satisfaction problems* (CSPs) in artificial intelligence. There has been extensive research on CSPs in artificial intelligence, resource scheduling, temporal reasoning, and many other areas [60,61,89,90,166,173,203,207,222,255].

An example of CSP is the well-known N-queen problem that is defined as follows: Given an integer N, place N queens on N distinct squares in an N $\times$ N chess board so that no two queens are on the same row, column, or diagonal.

Decision problems can be solved as optimization problems. For example, an algorithm that finds the minimal tour length in a traveling salesman problem will also determine whether there is a tour with a specific threshold for every possible threshold.

On the other hand, optimization problems can also be solved through one or a series of decision problems. For example, the minimal tour length in a traveling salesman problem can be found by solving a sequence of decision problems. In each decision problem, the existence of a tour within a specified length is solved. As the length decreases gradually, the minimal tour length can be found.

1.3 Challenges in Solving Nonlinear Optimization Problems

Except for trivial cases, nonlinear optimization problems do not have closed-form solutions and cannot be solved analytically. Numerical methods have been developed to search for optimal solutions of these problems. The process of solving an optimization problem becomes a process of searching for optimal solutions in the corresponding search space.

Finding global minima of a nonlinear optimization problem is a challenging task. Nonlinear constraints form feasible regions that are hard to find and difficult to deal with, and nonlinear objectives have many local minima that make global minima hard to find and verify.

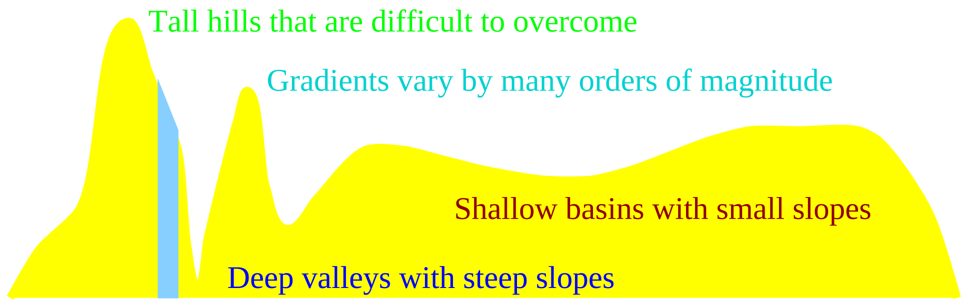

Tall hills that are difficult to overcome
Gradients vary by many orders of magnitude
Shallow basins with small slopes
Deep valleys with steep slopes

Figure 1.2: Illustration of difficulties in optimizing a nonlinear function. There may exist local minima in a multi-modal nonlinear function with different characteristics, shallow or deep, wide or narrow.

In solving nonlinear constrained optimization problems, there may not be enough time to find a feasible solution when nonlinear constraints constitute small feasible regions that are difficult to locate. For example, in our study of designing digital filter banks in Chapter 5, the feasible regions are small, and random searches like genetic algorithms can hardly find a feasible solution. Also, in satisfiability problems (studied in Chapter 6), feasible solutions can be very few and difficult to find due to the large number of constraints.

Second, nonlinear objective functions in both constrained and unconstrained optimization problems make global minima difficult to find. Figure 1.2 illustrates the challenges of a multi-modal nonlinear function. The terrains have different characteristics and cause problems for search algorithms that adapt to only a subset of the characteristics. Because of large slopes, tall hills are difficult to overcome in gradient-based search methods. In the search space, gradients can vary by many orders of magnitude, which make the selection of appropriate step-size difficult. Large shallow basins and flat plateaus provide little information for search direction, and may take a long time for a search algorithm to pass these regions if the step-size is small. Further, small local minima are difficult to find.

These difficulties happen in real-world applications, such as neural-network learning. In supervised learning of feedforward artificial neural networks (studied in Chapter 4), artificial neural networks are trained to perform desired mapping functions. The difference of desired outputs and actual outputs of a neural network forms an error function to be minimized. The

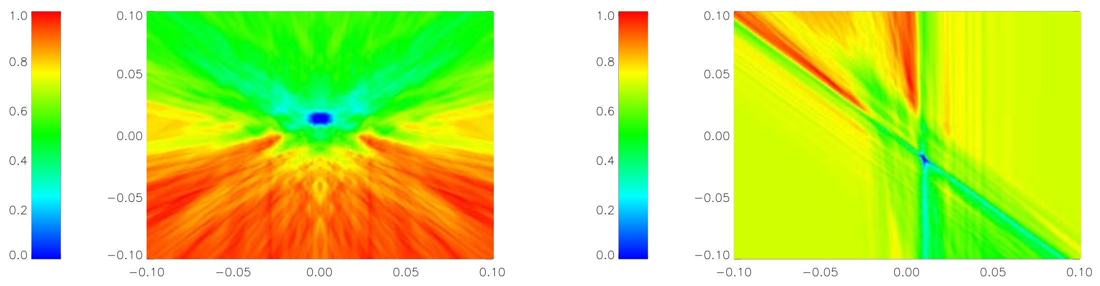


Figure 1.3: Two-dimensional projections of the 33-dimensional error surface for a five hidden-unit 33-weight feed-forward neural network with sigmoidal activation function. The terrain is around a solution found by our new global search method, Novel, to solve the two-spiral problem. These two graphs are plotted along two different pairs of dimensions.

neural-network learning problem is solved as an unconstrained optimization problem with the error function as the objective. Objective functions in neural-network learning problems are usually highly nonlinear and have many local minima.

Figure 1.3 shows the error function of a feedforward neural network trained to solve the two-spiral problem (to be discussed in Chapter 4). The 33-dimensional error function is projected to two different pairs of dimensions. The plots are around a solution found by our new global search method, Novel.

Figure 1.3 shows that the terrains have many different features. The left plot shows a large number of small and shallow local minima. The global minimum is in the middle, and its attraction region is small. The right graph shows a terrain divided into four parts by two narrow valleys. Two of them consist of large flat regions with gradients close to 0, whereas the other two of them have many local minima. There are also two long and narrow valleys across the terrain. On the edge of these valleys, steep slopes have very large gradients. Values at the bottom of these valleys changes gradually.

In nonlinear optimization problems, global optimal solutions are not only difficult to find, but also difficult to verify. There is no local criterion for deciding whether a local optimal solution is a global optimum. Therefore, nonlinear optimization methods cannot guarantee solution qualities for general nonlinear problems.

To summarize, the challenges of general nonlinear optimization include the following:

- Feasible regions bounded by nonlinear constraints may be difficult to find;
- The objective-function terrain of search space may be very rugged with many sub-optima;
- There may exist terrains with large shallow basins and small but deep basins;
- The dimension of optimization problems is large in many interesting applications;
- The objective and constraints are expensive to evaluate.

1.4 Characteristics of Nonlinear Optimization Algorithms

A large number of optimization methods have been developed to solve nonlinear optimization problems. Nonlinear optimization methods are classified into *local optimization* and *global optimization* methods. Global optimization methods have the ability to find global optimal solutions given long enough time, while local optimization methods do not.

Local-optimization methods include gradient descent, Newton's method, quasi Newton's method, and conjugate gradient methods [7, 13, 97, 127, 143, 170, 247]. They converge to a local minimum from some initial points. Such a local minimum is globally optimal only when the objective is quasi-convex and the feasible region is convex, which rarely happens in practice [82]. For nonlinear optimization problems, a local minimum can be much worse than the global minimum. To overcome local minima and search for global minima, global optimization methods have been developed.

Global optimization methods look for globally optimal solutions [82, 125, 188, 253]. As stated by Griewank [99], global optimization for nonlinear problems is mathematically ill-posed in the sense that a lower bound for the global optima of the objective function cannot be given after any finite number of evaluations of the objective function, unless the objective satisfies certain conditions, such as the Lipschitz condition, and the search space is bounded.

Global optimization methods perform global and local searches in regions of attraction and balancing the computation between global and local exploration. A global search method has the mechanism to escape from local minima, while a local search method does not. In an optimization problem, a region of attraction defines the region inside which there is a local minimum and the constraints are satisfied. The regions of attraction trap local search methods. In the general black-box model, global optimization is performed without *a priori* knowledge of the terrain defined by the objective and the constraints [168, 219, 253, 261]. Therefore, global optimization algorithms use heuristic global measures to search for new regions of attraction. Promising regions identified are further optimized by local refinement procedures, such as gradient descent and Newton's methods. In many real-world applications, the computational complexity of finding global optima is prohibitive. Global optimization methods usually resort to finding good sub-optimal solutions.

Nonlinear optimization methods employ various search algorithms to escape from local minima and search for global optima. The characteristics of search algorithms are summarized as follows:

(a) Representation of search space. A search space specifies the range of variable assignments that are probed during the process of searching for optimal solutions. The search space may contain only feasible regions specified by constraints, or may contain some infeasible regions as well. The search space of an optimization problem can be finite or infinite, which directly affects the computational complexity of the corresponding search algorithms. For example, the search space is finite in a traveling salesman problem with a finite number of cities and in a knapsack problem with finite number of objects. For continuous optimization problems in which variables take on real values, the search space is infinite [149, 190, 203, 250, 255].

The search space can have structural representations. For example, in a 0-1 binary integer programming problem, the search space can be represented as a binary search tree. Each node of the search tree represents the resolution of a variable, and the two branches coming from it represent the two alternative values of the variable. In another representation, the space space can be represented based on the permutation of variable values. All possible permutations constitute the search space.

The representation of the search space affects the complexity of a search problem. Structural representations can sometimes reduce the complexity significantly. For example, in solving a 0-1 integer programming problem, a representation based on a binary search tree provides the basis for branch-and-bound methods, which use efficient algorithms to find tight lower and upper bounds of search nodes and can prune the search tree significantly. The search space becomes much smaller than that in a representation based on permutations [107,185,188,202].

(b) Decomposition strategies. Some search algorithms work on the whole search space directly, while others decompose the large search space into smaller ones and then work on them separately. Divide-and-conquer and branch-and-bound are two decomposition strategies that have been applied in many search algorithms for both continuous and discrete problems [107,149,188,190,253]. Unpromising sub-spaces are excluded from further search, and promising ones are recursively decomposed and evaluated.

(c) Heuristic predictor or direction finder. During a search process, search algorithms have used many different heuristics to guide the search to globally optimal solutions. In branch-and-bound methods, heuristic lower bounds are associated with decomposed smaller sub-spaces to indicate their goodness. For example, in interval methods, interval analyses are used to estimate the lower bounds of a continuous region. Similarly, in solving integer programming problems, discrete problems are relaxed into continuous linear programming problems to obtain a lower bound [107,149,190]. The lower-bound information is further used to guide the search. In simulated annealing, the objective values of neighborhood points provide a direction for the search to proceed [144,208]. In genetic algorithms, search directions are obtained from fitness values of individuals in a population [95,121]. Components of individuals with high fitness are reinforced, and the search moves in directions formed by good building blocks.

(d) Mechanisms to help escape from local minima. To find globally optimal solutions, a search algorithm has to be able to escape from local minima. In discrete optimization, a back-tracking mechanism has been used extensively. A similar mechanism has also been applied in covering methods to solve continuous optimization problems [190,253]. Probabilistic methods [140,235] get out of local minima based on probabilistic decisions. For example, simulated annealing methods can move in a direction with worse solutions using adaptive probabilities [144,208]. Genetic algorithms escape from local minima through probabilistic recombination of individuals and random perturbation of existing solutions [95,121].

(e) Mechanisms to handle constraints. To solve constrained optimization problems, search algorithms have to handle constraints efficiently. Simple constraints, including simple bound and linear constraints, are relatively easy to handle. For example, variable substitution can eliminate linear constraints [164]. Two major classes of techniques, transformational and non-transformational approaches, have been developed to handle nonlinear constraints. In transformational approaches, the constraints and objectives are combined to form a single function [125,253]. Hence, constrained problems are converted into another form, usually an unconstrained form, before being solved. In non-transformational approaches, the search is performed in a relaxed search space that contains an infeasible region [164,208]. Infeasible solutions along a search path are either discarded or repaired to become feasible solutions.

(f) Stopping conditions. Some optimization problems can be solved optimally in a short amount of time. Unfortunately, most real-world applications are large and have high computational complexity, which makes optimal solutions impossible to find or verify. Hence, search algorithms have to terminate in a reasonable amount of time and settle on approximation solutions. The degree of approximation is usually proportional to the amount of time available. Better solutions can be found given more execution time. Pruning mechanisms have also been used to reduce the search space and find approximation solutions. For example, a search algorithm stops when all nodes of a search tree are pruned based on certain conditions. Many stopping criteria have been used to deal with the time-quality tradeoff. Some widely used stopping conditions include resource limit, e.g. physical time limit, of computer program execution, and degree of solution improvement [78,137,151,158,217,235].

(g) Resource scheduling strategies. To execute an optimization program on a computer system, resource scheduling determines the allocation of computer resources, including CPU and memory, which in turn affects resource utilization. Scheduling issues are especially important for parallel computer systems. Large optimization problems are computationally intensive and may require the most powerful computer systems, such as parallel processing systems. Good resource scheduling strategies make good use of the complex parallel systems. Many parallel search algorithms have been developed and studied theoretically and empirically. Significant speedups have been achieved on existing parallel computers [19,37–39,44,79,174,175].

Among the seven issues of search algorithms, we identify the two most critical ones for nonlinear optimization: handling nonlinear constraints and escaping from local minima.

The representation of a search space is derived from the feasible domain of an optimization problem, and is usually straightforward. For unconstrained or simple-bounded constrained problems, their search space corresponds to feasible regions. For nonlinear constrained problems, their search space is relaxed to contain infeasible regions as well.

Decomposition strategies have been studied extensively for both continuous and discrete nonlinear problems. They are effective to help reduce the complexity of relatively small problems. However, their computational complexity increases exponentially with problem size, which renders them not very helpful for larger problems.

Heuristic predictors are very useful when problem-specific characteristics are available and ready to be exploited. For general black-box nonlinear optimization, only limited information, such as gradient, is available. Few heuristic predictors work well in this situation.

For real-world applications, stopping conditions are usually derived from physical limitations, such as computation resources and time available. Good stopping conditions make better usage of limited resources and prevent unnecessary computation. They do not help in finding good optimal solutions.

Similarly, resource scheduling strategies schedule tasks to be executed in a computer system. They try to make high utilization of limited computational resources and finish the search tasks as fast as possible. For optimization problems, good resource scheduling strategies can help reduce the time to find solutions of the same quality. However, they are not helpful to improve the solution if the problem has exponential complexity.

In short, handling nonlinear constraints and escaping from local minima are the two key issues in nonlinear optimization. Any good nonlinear optimization method has to have good strategies to address them.

Nonlinear constraints make problems difficult by forming small, irregular, and hard-to-find feasible regions. Nonlinear functions may be complex and do not have obvious features to aid the search. Methods based on easy-to-find feasible regions do not work in this situation. Constraint handling techniques need to be robust and able to deal with general complex nonlinear constraints.

Local minima of nonlinear objective functions make local searches more difficult. General nonlinear optimization problems have many local minima. Search methods that are trapped by local minima are incapable of obtaining good solutions. The mechanism of escaping from local minima determines the efficiency of a global search algorithm and the solution quality it obtains, and has long been the central issue in developing global search methods.

1.5 Goals and Approaches of This Research

The goals of this research are to address the two key issues in nonlinear optimization, handling nonlinear constraints and escaping from local minima, and to develop efficient and robust global search methods. We develop methods to solve nonlinear continuous and discrete optimization problems that (a) achieve better solution quality than existing methods in a reasonable amount of time, or (b) execute faster for the same solution quality.

1.5.1 Approaches to Handle Nonlinear Constraints

To deal with general nonlinear constraints, we have developed adaptive Lagrangian methods (Chapter 2). A constrained optimization problem is first transformed into a Lagrangian function. Then gradient descents in the original variable space and gradient ascents in the Lagrange-multiplier space are performed. Eventually, the search process converges to a saddle point that corresponds to a local minimum of the original constrained optimization problem.

Adaptive control of relative weights between the objective and the constraints can speed up convergence. The process of searching for saddle points is governed by the combination of the two counteracting descent and ascent forces. Descents try to reduce the objective value whereas ascents try to satisfy the constraints. At a saddle point, the descent and ascent forces reach a balance through appropriate Lagrange-multiplier values.

The speed of convergence to a saddle point varies a lot depending on the relative magnitudes of the objective function and the constraints. When the objective value is too large as compared to the constraint values, descents in the original-variable space dominate the search process, and the Lagrange multipliers have to become very large in order to overcome the descent force and pull the search back to a feasible region. Since the Lagrange multipliers are updated according to constraint violations, it may take a very long time for the Lagrange multipliers to become large enough, which slows down the convergence to saddle points.

On the other hand, when the objective has little weight and the force due to descents is very small, constraint satisfaction dominates the search. Hence, the search process visits many feasible points, but improvements of objective values are slow, and so is convergence to a saddle point.

In order to improve convergence speed, we have developed mechanisms to adaptively control the relative weight of descent and ascent forces (Section 2.2). A weight coefficient is assigned to the objective function, and is adjusted adaptively during the search.

For example, in one of our adaptive methods, the initial values of weights are selected based on the gradient of the objective function at the starting point. The search is then performed for a small amount of time. The ratio of the change in the objective value to the time interval gives the speed of descent in the objective function. This ratio is used as the normalization factor to adjust the initial weight. Next, the search is re-started from the original starting point with the adjusted initial weight on the objective. As the search goes on, the weight is adaptively adjusted based on the magnitude of the maximum violation of the constraints and the change of the objective function.

The general control strategy is to use the constraint violation as the primary indicator and the change of the objective value as the secondary indicator. If any constraint is violated, the weight on the objective is reduced; otherwise, if the objective value is changing, then the weight on the objective is increased. The weight on the objective is adjusted adaptively so that convergence is more robust and faster when the objective and the constraints have different magnitude.

To handle constraints in discrete constrained optimization problems, we extend the Lagrangian theory for continuous problems to discrete problems, and develop the discrete Lagrangian method to solve discrete constrained problems (Section 2.2).

Similar to the continuous case, a Lagrangian function of discrete variables is formed by combining the objective and the constraints using Lagrange multipliers. Then, descents in the original-variable space and ascents in the Lagrange-multiplier space are performed to seek saddle points. In the continuous case, descents are based on continuous gradients. Variables in discrete optimization problems take on discrete values; therefore, the traditional definition of gradient cannot be used. We propose to use discrete gradients instead. A point in the neighborhood of the current point that has the most reduction in function value gives the discrete gradient-descent direction of the function at that point. We show that by using the discrete gradient descent, the search based on the Lagrangian function converges to a saddle point corresponding to a local minimum of the original discrete optimization problem. Also, adaptive control of objective weight can speed up convergence.

1.5.2 Approaches to Overcome Local Minima

Nonlinear optimization problems usually have many local minima in their search space. In unconstrained problems, the nonlinear objective function has many local minima. In constrained problems, the Lagrangian function formed by the combination of the objective and the constraints has many local minima. In order to find global optima, it is necessary for search methods to escape from local minima.

To overcome local minima, we have developed a new deterministic global search method, called the *trace-based* method (Chapter 3). Our trace-based method consists of a global search phase and a local search phase. The global search phase is further divided into a coarse-level and a fine-level global search. In the coarse-level global search, methods that explore the whole search space efficiently are applied to find promising regions. The output of the coarse-level global search is one or a set of starting points for the fine-level global search.

In the fine-level global search, the trace-based method uses a combined force of an external guidance and the local gradient to form the search trajectory. An external trace function is designed to lead the search trajectory around the search space and to pull the search out of local minima once it gets there. In the meantime, local gradient attracts the search towards a local minimum. The smaller the local minimum is, the larger the gradient force becomes. In other words, the trace force emphasizes exploration of the search space, while the gradient force detects local minima. The search trajectory formed by the combination of these two forces traverses the search space continuously, and reveals promising local minimal regions on the way. Good initial points for the local-search phase are identified along the search trajectory.

In the local-search phase, local-descent methods are started from the initial points generated from the fine-level global search. The local minimum corresponding to each initial point is located. The best ones among them are output as the final solution.

1.5.3 A Unified Framework to Solve Nonlinear Optimization Problems

Combining our methods of handing constraints and overcoming local minima, we can solve general nonlinear optimization problems. Figure 1.4 shows how various types of problems are solved in a unified framework.

Unconstrained optimization problems are solved directly by trace-based global optimization methods. Constrained problems, whether continuous or discrete, are first transformed into Lagrangian formulations, in which constraints are combined with the objective function using Lagrange multipliers.

Decision problems, including constraint satisfaction problems (CSPs), do not have objective functions. They are solved as optimization problems by first introducing artificial

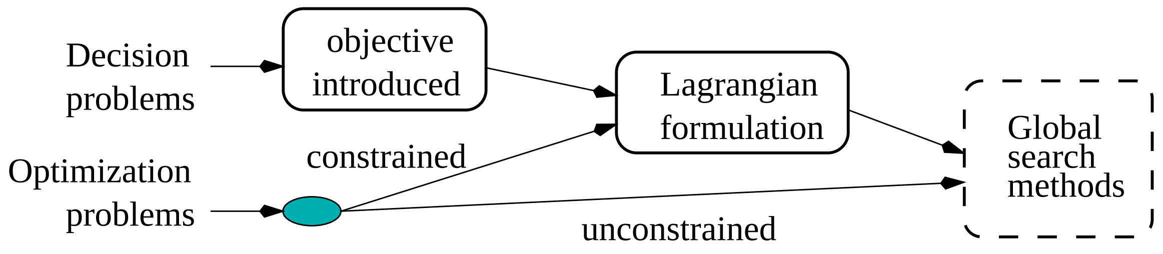


Figure 1.4: Transformation of various forms of nonlinear optimization problems into a unified formulation that is further solved by global search methods.

objectives. These objectives are helpful in providing guidance toward feasible solutions. For example, given a CSP as follows:

$$\text{Find a feasible solution that satisfies } h(\mathbf{x}) = 0 \tag{1.11}$$

The problem can be transformed into a constrained optimization problem by adding a merit function that measures the norm of the constraint set:

$$\begin{aligned} \text{minimize} \quad & N(h(\mathbf{x})) \\ \text{subject to} \quad & h(\mathbf{x}) = 0 \end{aligned} \tag{1.12}$$

where $N(\cdot)$ is a scalar function such that $N(h(\mathbf{x})) = 0$ iff $h(\mathbf{x}) = 0$. Although (1.12) is equivalent to the original constraint-only formulation (1.11), the objective (merit) function indicates how close the constraints are being satisfied, hence providing additional guidance in leading the search to a satisfiable assignment.

Note that a sufficient condition for solving the CSP defined in (1.11) is when there is an assignment such that the objective function in (1.12) is zero. However, optimizing the objective function in (1.12) alone without the constraints is less effective, as there may exist many local minima in the space of $N(\cdot)$. Strictly following a descent path of $N(\cdot)$ often ends up in a dent where $N(\cdot)$ is not 0, but its value cannot be improved by local refinement.

After a constrained problem is transformed into a Lagrangian formulation, our trace-based global search is performed based on the Lagrangian function to find good solutions.

1.5.4 Applications of Proposed Methods

We have applied our Lagrangian methods for handling nonlinear constraints and our trace-based global search method for overcoming local minima to several applications. These include artificial neural-network learning, digital filter-bank design, satisfiability and maximum satisfiability, and multiplierless filter-bank design.

We have applied our trace-based global search method to learn the weights of a feedforward neural network – a nonlinear unconstrained optimization problem (Chapter 4). Artificial neural networks have been used in many applications in recent years. However, the learning and design method is far from optimal. In our experiments, we formulate the supervised learning of feedforward neural networks as an unconstrained optimization problem. The objective function is highly nonlinear with many local minima. We compare our trace-based method with other global search methods and existing results in solving a set of benchmark problems. Our trace-based method improves the designs significantly, resulting in much smaller neural networks with better qualities.

We have applied our Lagrangian method with adaptive control to solve constrained optimization applications, specifically the design of digital filter banks (Chapter 5). Starting with a multiple-objective design problem, we formulate it into a constrained formulation constraining all objectives but one. The constrained continuous problem is then solved by our adaptive Lagrangian method and trace-based global-search method. Our experiments show that our method converges faster, is more robust, and improves existing solutions on all test problems.

We have applied our discrete Lagrangian method to solve several discrete optimization problems, including the satisfiability problem, the maximum satisfiability problem, and the design of a multiplierless digital filter bank (Chapter 6). These problems are formulated as discrete constrained optimization problems and solved by discrete Lagrangian methods. Using a large number of benchmark and test problems, we show very promising results of the discrete Lagrangian method. It either improves existing solutions, or finds optimal solutions much faster than other methods.

To summarize, we address two important issues of nonlinear optimization in this research: the handling of nonlinear constraints and methods to help escape from local minima. We use a Lagrange-multiplier-based formulation to handle nonlinear continuous and discrete constraints, and develop efficient Lagrangian methods to search for saddle points. To overcome local minima, we develop a new global search method that uses an external force to lead the search continuously across the search space and perform a combination of local and global searches. These methods have been applied to a large number of unconstrained and constrained application problems and have obtained very promising results.

1.6 Outline of Thesis

This thesis is divided into two parts. In the first part, which consists of Chapters 2 and 3, we review existing methods for handling nonlinear constraints and overcoming local minima and present our new methods. In the second part, which consists of Chapters 4, 5, and 6, our proposed methods are applied to solve some application problems formulated as unconstrained and constrained continuous and discrete optimization problems. Figure 1.5 shows the organization of this thesis.

Many problems in engineering can be formulated as optimization problems. Most optimization problems have multi-modal objective functions, i.e., there exists more than one local optimum, and possibly nonlinear constraints. Local optimization techniques, which stop at identifying local optima, are not suitable for such problems. Mechanisms of handling nonlinear constraints and overcoming local minima are two important aspects of solving nonlinear optimization problems.

In Chapter 2, we address issues in handling nonlinear constraints. We first review existing methods to deal with nonlinear constraints in continuous and discrete problems, and introduce the classical Lagrangian theory. Then, we propose a new method based on Lagrange multipliers, which employs adaptive control of relative weights between the objective and the constraints to achieve faster and more robust convergence. To solve discrete constrained problems, we extend Lagrangian theory for continuous problems to discrete problems, and present a new discrete Lagrangian method. For both continuous and discrete problems, we

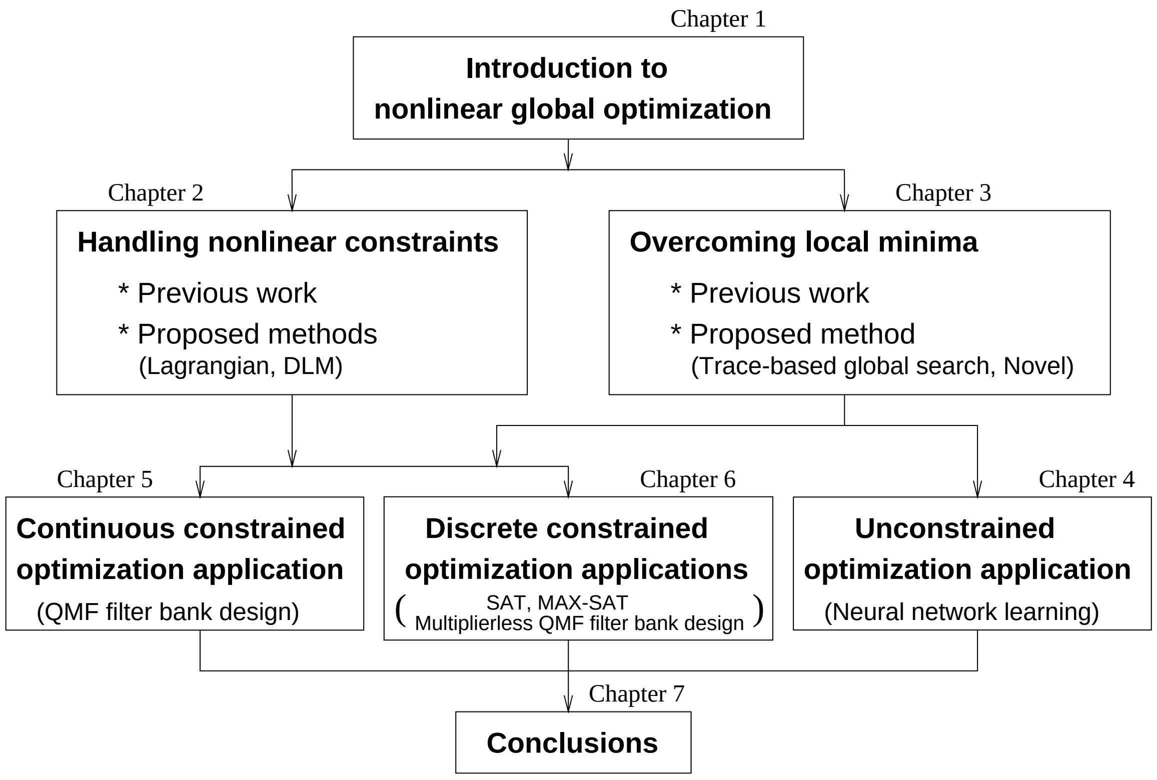

Chapter 1
Introduction to nonlinear global optimization
Chapter 2
Handling nonlinear constraints
* Previous work
* Proposed methods
(Lagrangian, DLM)
Chapter 3
Overcoming local minima
* Previous work
* Proposed method
(Trace-based global search, Novel)
Chapter 5
Continuous constrained optimization application
(QMF filter bank design)
Chapter 6
Discrete constrained optimization applications
(SAT, MAX-SAT Multiplierless QMF filter bank design)
Chapter 4
Unconstrained optimization application
(Neural network learning)
Chapter 7
Conclusions

Figure 1.5: Organization of this thesis

transform constrained problems into unconstrained problems based on Lagrange multipliers. The transformed unconstrained problems are further solved by local- and global-search methods.

In Chapter 3, we address issues in overcoming local minima. We first review existing nonlinear optimization techniques, including local- and global-search methods, and identify advantages and disadvantages of these optimization methods. Then, we present a new deterministic global search method, our proposed trace-based method. Beginning with an example, we introduce the overall framework of the method. We then present each component of the method and discuss them in detail. Our trace-based method employs an innovative global search strategy that combines global search and local search efficiently. Its unique trace-based global search techniques make it stand out among existing global search techniques and pose great potential in generating improved performance on real-world applications. An implementation of the trace-based method, Novel (an acronym for Nonlinear Optimization Via External Lead), is then presented.

In Chapter 4, we present the application of Novel to solve non-trivial unconstrained nonlinear optimization problems. The application is neural network learning. First, we describe the neural network learning problem and survey existing learning methods. The learning problem of many types of neural network can be formulated as an optimization problem. Also, many neural network learning algorithms can be traced back to optimization methods. Then, we present experimental results of Novel in solving some neural-network benchmark problems, and compare the results with those of other global search methods. The results obtained by Novel are phenomenal and show significant improvement in terms of learning error and network size.

In Chapter 5, we apply our adaptive Lagrangian method and Novel to solve nonlinear constrained optimization problems. The application is the design of quadrature-mirror-filter (QMF) digital filter banks. We first introduce the filter-bank design problem, which has multiple objectives to be optimized. Then, we summarize existing approaches and present our nonlinear constrained optimization formulation, which is solved by our adaptive Lagrangian method and Novel. In our experiments, we solve a set of QMF filter-bank design problems. For all the test problems, our results improve previous results, and are also better than those obtained by other methods. Using our constrained formulation, we have also studied performance tradeoffs among different design objectives.

In Chapter 6, we present the application of our discrete Lagrangian methods (DLM) to solve discrete optimization problems, including satisfiability (SAT), maximum satisfiability (MAX-SAT), and the design of multiplierless QMF filter bank problems. The satisfiability problem is the core of NP-complete problems and is significant in computer science research. We first summarize various formulations and the corresponding methods for solving this problem. Then we present three versions of DLM to solve SAT problems. A large number of SAT benchmark problems are used in our experiments. Our experimental results show that DLM usually performs better than the best existing methods and can achieve 1 to 3 order-of-magnitude speedup for many problems. MAX-SAT is a general case of SAT. We have also applied DLM to solve MAX-SAT test problems. In our experiments, DLM generally finds better solutions and is usually two orders of magnitude faster than competing methods. Finally, we apply DLM to design multiplierless QMF filter banks. DLM has obtained high-quality designs with less computation and hardware implementation costs than real-value designs.

Finally, in Chapter 7, we summarize the work we have presented in this thesis and propose future directions to extend and enhance this research.

1.7 Contributions of This Research

The following are the main contributions of this thesis:

- *Adaptive Strategy to Handle Nonlinear Constraints (Chapter 2).* We have proposed Lagrange-multiplier-based methods to handle nonlinear constraints, and have developed an adaptive and robust strategy to speed up the search for saddle points.

- *Discrete Lagrangian Method (Section 2.2).* We have extended the classical Lagrangian theory to discrete problems and have developed the discrete Lagrangian method (DLM) that works on discrete optimization problems directly and efficiently.

- *Innovative Trace-based Global Search to Overcome Local Minima (Chapter 3).* We have proposed a trace-based global search method consisting of a coarse-level and fine-level global-search phase and a local-search phase. Each phase addresses a different aspect of the global search. Collectively, they constitute a powerful method that finds global optima faster, or finds good solutions in a short amount of time. Our unique trace-based global search strategy overcomes local minima in a continuous fashion.

- *Global Search Approach for Neural-Network Learning (Chapter 4).* We have applied the new trace-based global search method to neural-network learning. The method has shown substantial performance advantage over existing neural-network learning algorithms and other global search methods.

- *Global Search Approach for Digital Filter-Bank Design (Chapter 5).* We have applied our adaptive Lagrangian method and trace-based global search to the design of quadrature-mirror-filter digital filter banks. We have found designs that are better than previous results, as well as better than those obtained by simulated annealing and genetic algorithms.

- *Discrete Lagrangian Method for Solving Discrete Problems (Chapter 6).* We have applied our discrete Lagrangian method, DLM, to solve a large number of satisfiability and maximum satisfiability problems. We have obtained significant performance improvement over the best existing methods. DLM has also been applied to the design of multiplierless QMF filter banks (mainly by Mr. Zhe Wu, another student in the group). We have found high-quality designs with a fraction of computation and implementation costs.

2. HANDLING NONLINEAR CONSTRAINTS

A nonlinear constrained optimization problem consists of a nonlinear objective function and a set of possibly nonlinear constraints. Compared with unconstrained optimization problems, constrained problems have the additional requirement of satisfying constraints. Naturally, how to handle constraints is the first issue to be addressed.

Constrained problems are continuous or discrete depending on their variable domains. Continuous problems are defined over continuous variables, while discrete problems are defined over discrete variables. Continuous and discrete problems have different characteristics, and their solution methods are also different.

Constraints in continuous constrained problems have linear or nonlinear forms. Linear constraints are relatively easy to handle. For example, variable substitution and other transformation methods can be used to eliminate linear constraints as well as some variables [164]. Some linear constrained optimization problems, including linear programming and quadratic programming, have been studied extensively and can be solved by efficient algorithms developed in the past [59, 84, 161, 216]. General nonlinear constrained problems are more difficult and have been an active research topic in recent years [83, 125, 126].

Discrete constrained problems can have linear or nonlinear constraints. In contrast to continuous problems, linear constrained discrete problems can have high computational complexity and be expensive to solve due to their large search space. Nonlinear constraints can be transformed into linear constraints after introducing dummy variables [190].

In this chapter, we focus on issues in handling nonlinear constraints. We first review existing methods for constrained continuous and discrete problems, and introduce classical Lagrangian theory for handling nonlinear constraints. Then, we propose a Lagrange-multiplier-based method with adaptive control. This method dynamically adjusts the relative weights between the objective and the constraints to achieve faster and more robust convergence to saddle points. To solve discrete problems, we extend Lagrangian theory from the continuous domain to the discrete domain, and present a new discrete Lagrangian method.

In general, our approach of handling nonlinear constraints is to transform both continuous and discrete constrained problems into unconstrained formulations using Lagrange multipliers, and solve them using local- and global-search methods.

2.1 Previous Work

Methods to handle nonlinear constraints take one of the two approaches: *transformational* or *non-transformational*. In non-transformational approaches, optimal solutions are searched directly based on the objective and the constraints. Examples are rejecting/discarding methods, repairing methods and reduced-gradient methods. In transformational approaches, the original constrained problem is transformed into another, usually a simple form, before being solved. Figure 2.1 shows a classification of nonlinear constrained methods.

2.1.1 Non-transformational Approaches

Non-transformational approaches work on the original problem directly by searching through its feasible regions for the optimal solutions. They stay within the feasible regions and try to improve the objective values. Methods in this category include rejecting/discarding methods, repairing methods, and reduce-gradient methods [3, 129, 135, 153, 161, 164, 276]. The advantages of these methods are (a) their feasible termination points and (b) their convergence to a local constrained minimum. Their disadvantages are (a) their requirement of an initial feasible point and (b) the difficulty to remain within a feasible region when constraints are nonlinear.

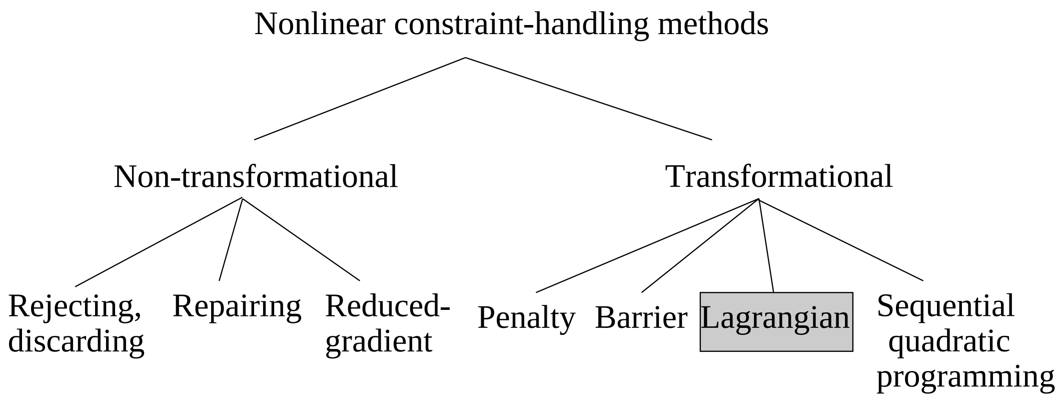


Figure 2.1: Classification of methods to handle nonlinear constraints. Our new method presented in this chapter is a variation of the Lagrangian method.

Rejecting/discarding methods take the general strategy that infeasible solutions are dropped during the search process for optimal solutions, and only feasible solutions are accepted [129, 164]. This strategy is simple, easy to understand, and easy to implement. Its disadvantage is that it is inefficient if a feasible region is difficult to find, which happens when the optimization problem is highly constrained and the feasible region is irregular. When a feasible region is small and not easy to find, these methods spend most of the time in generating and rejecting infeasible candidate solutions.

Repairing methods [135] attempt to maintain feasibility by repairing moves that go off a feasible region. Infeasible solutions are transformed into feasible ones with some repair. Usually, repairing methods are problem-specific. Restoring feasibility may be as difficult as the original optimization problem.

Reduced-gradient methods search along gradient or projected gradient directions within feasible regions [3, 153, 161, 276]. They perform well if constraints are "nearly linear." When possible, these methods first eliminate equality constraints and some variables. The original problem is reduced to a bound-constrained problem in the space of the remaining variables. Variables are divided into two categories, fixed and superbasic. Fixed variables are those that are at either upper or lower bounds, and that are to be held constant. Superbasic variables are free to move. A standard reduced-gradient algorithm searches along the steepest-descent direction in the superbasic variable space.

In short, non-transformational methods work directly on constraints of the constrained optimization problem. They explicitly handle infeasible points during the search and do not transform the original constrained problem. These methods do not work well if a feasible starting point is difficult to find and the feasible region is nonlinear and irregular.

2.1.2 Transformational Approaches

Transformational approaches convert a constrained problem into another form before solving it. Well-known methods include penalty methods, barrier methods, Lagrangian methods, and sequential quadratic programming methods [83, 126, 161, 165, 188, 208].

Penalty methods approximate a constrained problem by an unconstrained problem that assigns high cost to points far from the feasible region. Barrier methods, on the other hand, approximate a constrained problem by an unconstrained problem that assigns high cost to being near the boundary of the feasible region. Unlike penalty methods, these methods are applicable only to problems having a robust feasible region.

Penalty methods handle constraints based on penalty functions [161, 165, 188, 208]. A penalty term that reflects the violation of the constraints is added to the objective function to form a penalty function. A constrained optimization problem is transformed into a single or a sequence of unconstrained optimization problems minimizing penalty functions. Unconstrained optimization methods are used to find optimal solutions of the unconstrained problem. As the penalty parameter on each constraint is made large enough, the unconstrained problem has the same optimal solution as the original constrained problem.

Penalty methods are simple and easy to implement. However, finding the appropriate penalty parameters for a specific problem instance is not trivial. If the penalty parameters are too large, constraint satisfaction dominates the search, which converges to any feasible point that may be far from the optimum. Also, large penalty terms make the penalty function space very rugged and difficult to search for good solutions. On the other hand, if the penalty parameters are too small and constraint violations do not have large enough weights, the optimal solution based on the penalty function may not be feasible. Usually, penalty methods require tuning penalty coefficients either before or during the optimization process.

Barrier functions are similar to penalty functions except that barriers are set up to avoid solutions from going out of feasible regions [83, 126, 188]. Barriers are derived from constraints. Examples of barrier functions are those that introduce logarithms of the inequalities in the objective function. Barrier methods transform constrained problems into unconstrained problems and minimize barrier functions. Basically, they work inside feasible regions and search for optimal feasible points. Barrier methods usually require a feasible starting point and have difficulty in situations in which feasible regions of a set of nonlinear constraints are hard to identify.

Lagrangian and augmented Lagrangian methods use Lagrange multipliers to combine constraints with the objective function to form a Lagrangian function. The original constrained problem is transformed into an unconstrained problem based on the Lagrangian function. By introducing Lagrange multipliers, constraints can be gradually resolved through iterative updates. Lagrangian and augmented Lagrangian methods manage numerical stability and achieve high accuracy at a price of an increased number of problem dimensions [83, 126, 161, 188].

Penalty, barrier, and Lagrangian methods transform a constrained problem into an unconstrained form before solving it. In contrast, sequential quadratic programming (SQP) methods approximate the original problem using quadratic programming problems [31, 201]. They solve a series of successive quadratic approximations of a nonlinear constrained problem. In its purest form, SQP methods replace the objective function by a quadratic approximation and replace the constraint functions by linear approximations. A set of quadratic subprograms is solved. If the starting point is sufficiently close to the local minimum, SQP methods converge at a second-order rate.

Feasible sequential quadratic programming methods are variations of SQP methods in which all iterations work on feasible points [183, 184]. They are more expensive than standard SQP methods, but are useful when the objective function is difficult or impossible to calculate outside the feasible set, or when termination of the algorithm at an infeasible point is undesirable.

Among these transformational approaches of handling nonlinear constraints, Lagrangian methods are general, robust, and can achieve high degrees of precision. Penalty methods have difficulty in choosing appropriate penalty coefficients. Barrier methods do not work well when the feasible region is small and hard to find. SQP methods have to start close to local minima to perform well. Therefore, we use Lagrange multipliers for constraint relaxation in developing our algorithm. In the next section, we briefly introduce Lagrangian theory for continuous constrained problems and Lagrangian methods.

2.1.3 Existing Lagrangian Theory and Methods

Lagrangian methods are classical methods for solving continuous constrained optimization problems [161, 232, 269]. The basic form of the Lagrangian method works with equality constraints, while inequality constraints are first transformed into equality constraints. In this section, we first introduce Lagrangian theory for equality constrained problems and related Lagrangian methods. Then, we present methods that transform inequality constraints into equality ones.

Given an equality constrained optimization problem as follows:

$$\begin{aligned} \min_{\mathbf{x}\in R^n} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h(\mathbf{x}) = 0 \end{aligned} \tag{2.1}$$

where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, and $h(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_m(\mathbf{x}))$ are m constraints. The corresponding *Lagrangian function* $L(\mathbf{x}, \lambda)$ is defined by

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i h_i(\mathbf{x}) \tag{2.2}$$

where $\lambda = (\lambda_1, \cdots, \lambda_m)$ are *Lagrange multipliers.* The augmented Lagrangian function is a combination of Lagrangian and penalty function as follows:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i h_i(\mathbf{x}) + \frac{1}{2}||h(\mathbf{x})||_2^2 \tag{2.3}$$

The augmented Lagrangian function often provides better numerical stability and convergence.

Definition 2.1.1 *A point* $\mathbf{x}^*$ *satisfying the constraints* $h(\mathbf{x}^*) = 0$ *is said to be a* regular point *of the constraints if the gradient vectors* $\nabla h_i(\mathbf{x}^*)$, $i = 1, \cdots, m$ *are linearly independent.*

The tangent plane at regular points can be characterized in terms of the gradients of the constraint functions.

First-order necessary conditions. Let $\mathbf{x}^*$ be a local minimum of f subject to constraints $h(\mathbf{x}) = 0$, and $\mathbf{x}^*$ be a regular point of these constraints. Then there exists $\lambda \in R^m$ such that

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \lambda) = 0 \qquad \text{and} \qquad \nabla_{\lambda} L(\mathbf{x}^*, \lambda) = 0 \tag{2.4}$$

The conditions in (2.4) are not sufficient to have the existence of a constrained local minimum of $f(\mathbf{x})$ unless second- or higher-order derivatives of $f(\mathbf{x})$ also satisfy certain conditions. An example of second-order sufficient conditions is as follows [161]:

Second-order sufficient conditions. Suppose there exist points $\mathbf{x}^*$ and $\lambda^* \in R^m$ such that

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \lambda^*) = 0 \qquad \text{and} \qquad \nabla_{\lambda} L(\mathbf{x}^*, \lambda^*) = 0 \tag{2.5}$$

Suppose also that matrix $\nabla^2_{\mathbf{x}} L(\mathbf{x}^*, \lambda^*)$ is positive definite on the tangent plane $M = \{z : \nabla h(\mathbf{x}^*) z = 0\}$, that is, $\mathbf{y}^t \nabla^2_{\mathbf{x}}(\mathbf{x}^*, \lambda^*)\mathbf{y} > 0$ for all $\mathbf{y} \in M$ and $\mathbf{y} \neq 0$. Then $\mathbf{x}^*$ is a strict local minimum of $f(\mathbf{x}, \lambda)$ subject to $h(\mathbf{x}) = 0$.

The necessary conditions in (2.4) form a system of $n+m$ equations with $n+m$ unknowns. When the equations are nonlinear, it is difficult to solve them analytically. Numerical methods find solutions satisfying the necessary conditions through the search of saddle points.

Definition 2.1.2 *A* saddle-point $(\mathbf{x}^*, \lambda^*)$ *of Lagrangian function* $L(\mathbf{x}, \lambda)$ *is defined as one that satisfies the following condition:*

$$L(\mathbf{x}^*, \lambda) \leq L(\mathbf{x}^*, \lambda^*) \leq L(\mathbf{x}, \lambda^*) \tag{2.6}$$

for all $(\mathbf{x}^*, \lambda)$ *and all* $(\mathbf{x}, \lambda^*)$ *sufficiently close to* $(\mathbf{x}^*, \lambda^*)$.

The following theorem states the relationship between local minima and saddle points. Here, we present the proof for equality-constrained problems. Similar proofs have been derived for continuous problems with inequality constraints [161, 232, 269]. This proof will be used when we extend Lagrangian theory to discrete optimization problems in Section 2.2.4.

Saddle-Point Theorem. $\mathbf{x}^*$ is a local minimum to the original problem defined in (2.1) if there exists λ^* such that $(\mathbf{x}^*, \lambda^*)$ constitutes a saddle point of the associated Lagrangian function $L(\mathbf{x}, \lambda)$.

Proof: Since $(\mathbf{x}^*, \lambda^*)$ is a saddle point, $L(\mathbf{x}^*, \lambda) \leq L(\mathbf{x}^*, \lambda^*)$ for λ sufficiently close to λ^*. From the definition of the Lagrangian function, this implies

$$\sum_{i=1}^{m} \lambda_i h_i(\mathbf{x}^*) \leq \sum_{i=1}^{m} \lambda_i^* h_i(\mathbf{x}^*).$$

Our proof is by contradiction. Suppose there exists some k, $1 \leq k \leq m$, $h_k(\mathbf{x}^*) \neq 0$. If $h_k(\mathbf{x}^*) > 0$, then vector $\lambda = (\lambda_1^*, \cdots, \lambda_k^* + \delta, \cdots, \lambda_m^*)$ would violate the inequality for a positive δ. If $h_k(\mathbf{x}^*) < 0$, then vector $\lambda = (\lambda_1^*, \cdots, \lambda_k^* - \delta, \cdots, \lambda_m^*)$ would violate the inequality for a positive δ. Therefore, $h(\mathbf{x}^*) = 0$, and $\mathbf{x}^*$ is a feasible solution to the problem.

Since $(\mathbf{x}^*, \lambda^*)$ is a saddle point, $L(\mathbf{x}^*, \lambda^*) \leq L(\mathbf{x}, \lambda^*)$ for $\mathbf{x}$ sufficiently close to $\mathbf{x}^*$. From the definition of Lagrangian function,

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i^* h_i(\mathbf{x}).$$

Thus, for any feasible $\mathbf{x}$, $h(\mathbf{x}) = 0$, and we have

$$f(\mathbf{x}^*) \leq f(\mathbf{x}).$$

So $\mathbf{x}^*$ is a local minimum. ■

A saddle-point is a local minimum of Lagrangian function $L(\mathbf{x}, \lambda)$ in the $\mathbf{x}$ space and a local maximum of $L(\mathbf{x}, \lambda)$ in the λ space. One typical way to find saddle points is to descend in the $\mathbf{x}$ space and ascend in the λ space.

Lagrange multipliers λ can also be viewed as penalties associated with constraints, and Lagrangian function L corresponds to a penalty function. When some constraints are not satisfied, the sum of unsatisfied constraints, weighted by the corresponding Lagrange multipliers, are added to the objective function to form a penalty function. Ascents of L in the λ space, therefore, correspond to increasing the penalties associated with unsatisfied constraints. As L is minimized, the penalties will eventually increase to a point that pushes the constraints to be satisfied. Likewise, descents of L in the $\mathbf{x}$ space find a local minimum when all the constraints are satisfied.

Based on the Saddle-Point Theorem, numerical algorithms have been developed to look for saddle points that correspond to local minima in the corresponding search space. One typical method is to do descents in the original variable space of $\mathbf{x}$ and ascents in the Lagrange-multiplier space of λ [7, 50, 161, 283]. This method can be written as a system of ordinary differential equations (ODEs) as follows.

$$\frac{d\mathbf{x}}{dt} = -\nabla_{\mathbf{x}} L(\mathbf{x}, \lambda) \quad \text{and} \quad \frac{d\lambda}{dt} = \nabla_{\lambda} L(\mathbf{x}, \lambda) \tag{2.7}$$

where t is an autonomous time variable. This dynamic system evolves over time t and performs gradient descents in the original variable space of $\mathbf{x}$ and gradient ascents in the Lagrange-multiplier space of λ. When the system reaches an equilibrium point where all the gradients vanish, a saddle point of Lagrangian function L is found. In other words, given an initial point, the system will not stop until a saddle point (a local minimum of the constrained optimization problem) is reached.

Methods based on solving (2.7) are local search methods. When applied, an initial assignment to $\mathbf{x}$ and λ are first given, and the local solution will be the very saddle point reached from this initial point. After reaching the saddle point, the solution will not improve unless a new starting point is selected. Note that nonlinearity can cause chaos in which a small variation in the initial point can lead to a completely different solution. This happens when (2.7) are solved using finite-step discrete methods. In this case, over-shoots and under-shoots can lead to unpredictable and perhaps undesirable solutions.

Besides the method that solves (2.7) as a system of ordinary differential equations, other Lagrangian methods are based on successive minimization of the Lagrangian function with respect to the original variables, with updates of the Lagrange multipliers between iterations.

$$\max_{\lambda \in R^m} \min_{\mathbf{x} \in R^n} L(\mathbf{x}, \lambda) \tag{2.8}$$

Unconstrained optimization methods, such as Newton's method, are applied to solve the minimization problem with fixed λ. These methods are approximation of the approach to solve a system of ODEs. They are faster but not as accurate.

Lagrangian methods find local minima in the constrained space. From an initial assignment to $\mathbf{x}$ and λ, the search converges to a saddle point corresponding to a local solution. After reaching the saddle point, the solution will not change unless restarted from a new initial point. Note that nonlinearity can cause chaos in which a small variation in the initial point can lead to a completely different final solution.

We have discussed Lagrangian theory and methods that work on equality constraints. Inequality constraints are transformed into equality constraints to be solved. One transformation method, the *slack-variable method*, introduces slack variables for inequality constraints [161]. For example, an inequality constraint, $g(\mathbf{x}) \leq 0$, is transformed into equality constraint $g(\mathbf{x}) + z^2 = 0$ with the addition of slack variable $z \in R$.

For an inequality-constrained problem,

$$\begin{aligned} \min_{\mathbf{x} \in R^n} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_j(\mathbf{x}) \leq 0 \quad j = 1, \cdots, m, \end{aligned} \tag{2.9}$$

after introducing slack variables $\mathbf{z} = (z_1, z_2, \cdots, z_m)$, the corresponding augmented Lagrangian function is

$$L_z(\mathbf{x}, \mathbf{z}, \lambda) = f(\mathbf{x}) + \sum_{j=1}^{m} \left(\lambda_j (g_j(\mathbf{x}) + z_j^2) + \frac{1}{2} |g_j(\mathbf{x}) + z_j^2|^2 \right). \tag{2.10}$$

At a saddle point, $L_z(\mathbf{x}, \mathbf{z}, \lambda)$ is at a local minimum with respect to $\mathbf{x}$ and $\mathbf{z}$ and at a local maximum with respect to λ. From the local minimum condition of L_z with respect to $\mathbf{z}$ at a saddle point, the following solution can be derived.

$$z_j^2 = \max(0, -g_j(\mathbf{x}) - \lambda_j) \tag{2.11}$$

After substituting (2.11) into (2.10), we have

$$L_z(\mathbf{x}, \lambda) = f(\mathbf{x}) + \frac{1}{2}\sum_{j=1}^{m}(\max{}^2(0, \lambda_j + g_j(\mathbf{x})) - \lambda_j^2) \tag{2.12}$$

In this augmented Lagrangian function, slack variables $\mathbf{z}$ have been removed. Lagrangian methods find saddle points of this function by doing descents in the $\mathbf{x}$ space and ascents in the λ space.

2.1.4 Discrete Optimization

Discrete optimization problems usually have finite search spaces. However, they are difficult to solve due to the enormous size of their search spaces, which grow explosively with the number of discrete variables. The immense size of the search space rules out the possibility of completely enumerating all solutions, except for trivial problems. Often, only a small fraction of the search space can be enumerated.

Some discrete problems can be solved without enumeration at all. Unfortunately, the state of our present knowledge leaves most discrete optimization problems in the enumeration-required category [190].

The difficulty of discrete problems is addressed in complexity theory. Mathematicians and computer scientists have studied complexity for many years. In early 1970s, the seminal papers of Cook [51] and Karp [142] profoundly affected the research of complexity. Complexity theory tries to classify problems in terms of the mathematical order of the computational resources required to solve a problem. Categories of computational orders include (1) computation that grows logarithmically (or better) with problem size, (2) computation that grows polynomially, and (3) computation that grows worse than polynomially, e.g. exponentially.

Many complexity theories for discrete problems were developed on a class of succinctly defined problems called *decision problems*. Decision problems are solved by a yes or no answer to all input, rather than by a solution or an optimal value. Discrete optimization problems are converted to their decision-problem counterparts by asking the question of whether there is a feasible solution to a given problem with the value of the objective function equal to or smaller than a specified threshold.

The difficulty of decision problems ranges from the best solved Class P, to *undecidable* ones that are proved not solvable by any algorithm. Class P contains the set of all problems solved by polynomial-time complexity algorithms. Class NP (which stands for nondeterministic polynomial) contains more difficult problems. NP includes all decision problems that can be answered in polynomial time if the right polynomial-length guess is provided. Problems in NP can be solved by total enumeration of polynomial-length solutions. It is widely believed that P $\neq$ NP.

Class NP contains Class P, and also includes many problems not in P. Problems to which all members of NP polynomially reduce are called *NP-hard*. The first NP-hard problem is the *satisfiability problem* found by Cook in 1971. This is followed by the discovery of a rich family of NP-hard problems. NP-hard problems are as difficult to solve as any problem of NP. Problems that are both NP-hard and members of NP are called *NP-complete*. NP-complete problems form an equivalence class: if any NP-complete problem has a polynomial-time algorithm, then all NP-complete problems are solvable in polynomial time. The satisfiability problem is the first NP-complete problem.

NP-complete problems only contain decision problems. Many discrete optimization problems have an NP-complete threshold decision counterpart. For example, a traveling salesman problem is an optimization problem that finds the minimal tour length in a graph. Its corresponding decision problem is whether there exists a tour with a specific length in the graph. This decision problem is NP-complete. The original optimization problem is harder than the decision problem. If there is an algorithm that solves the original optimization problem in polynomial time, the decision problem can also be solved in polynomial time. Hence, optimization problems are NP-hard. In this thesis, we focus on NP-hard problems.

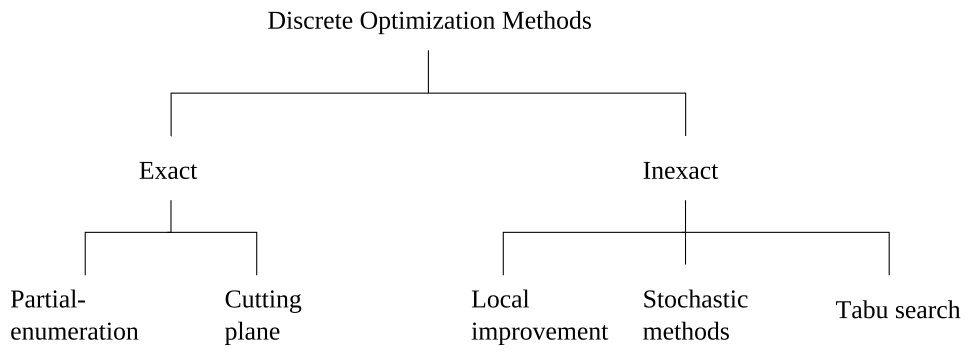


Figure 2.2: Classification of discrete optimization methods.

Methods to solve NP-hard problems are either exact or inexact. Exact methods guarantee solution quality [61, 89, 107, 203, 207], while inexact approaches return suboptimal solutions with no optimality guarantee [106, 222, 228]. NP-hard problems have exponential time complexity in the worst case. Figure 2.2 shows a classification of discrete optimization methods discussed in this section.

Two main classes of exact methods are partial enumeration and cutting-plane methods. Partial enumeration methods systematically decompose a large search space into smaller tractable ones. When feasible regions are easy to identify, each search space consists of only feasible points. On the other hand, when feasible points are not obvious, each search space often consists of infeasible points, and a sequence of infeasible points are enumerated to approach feasible and minimal solutions. Branch-and-bound methods are well-known partial enumeration methods [12, 150, 154, 176].

Cutting plane and polyhedral description methods proceed by re-defining the given problem better and better until it becomes tractable [11, 57, 122]. More constraints, usually linear constraints, are introduced to re-define the feasible region. Regions containing the optimal solution are refined, and unpromising regions are excluded from further search.

Almost all discrete optimization problems can be expressed in the form of integer linear programs (ILP). General discrete models are transformed into the ILP format at the cost of introducing a large number of variables and constraints. Therefore, the transformed ILP formulations could be more expensive to solve than the original problems. Many discrete problems have been solved efficiently by problem-specific algorithms that exploit the underlying combinatorial structure, which is not present in general ILP formulations and thus not addressed by general algorithms.

Although exact methods solve many practical problems, NP-hard problems eventually reach sizes where exact methods are impractical, and inexact approaches are the only choice. Inexact approaches settle for suboptimal solutions. A basic requirement of an inexact method is efficiency, which means stopping the method in polynomially-bounded time.

Inexact (heuristic or approximation) methods have been increasingly applied to solve real-world discrete optimization problems. Examples of inexact search methods are local improvement [60,90,166,167,173,222,223,237,243], stochastic methods [41,95,121,144,164], and tabu search [92, 112, 223]. They work with both transformational and non-transformational approaches to handle constraints.

In local improvement (neighborhood search), a search proceeds by sequential improvement of problem solutions, advancing at each step from a current solution to a superior neighbor. For constrained problems with unknown feasible solutions, a two-phase approach is applied. Beginning with an infeasible solution, local improvement first minimizes the infeasibility of the solution. If a feasible solution is discovered, the search is switched to an optimizing phase starting from the feasible solution.

In local improvement, a search is trapped by a local minimum when all neighbors are worse than the current point. Global search strategies have been developed in order to get out of local minima. Examples are multi-start (or random restart), simulated annealing (SA), genetic algorithms (GAs), tabu search, and other variations of randomized adaptive search methods.

In multi-start, the search escapes from a local minimum by jumping to a new, randomly generated starting point. Local improvement is performed until a local minimum is reached. Then, a new initial point is generated from which local improvement starts again. The best solution found is remembered and reported when the search terminates.

SA and GA use more sophisticated probabilistic mechanisms to escape from local minima. SA adds a stochastic mechanism to local improvement [2,53,144,208,259]. Starting from an initial point, SA probes nearby points whose objective value, or possibly constraint values, is evaluated. When minimizing a function, any down-hill movement is accepted, and the process is repeated from this new point. An uphill movement may be accepted, and by doing so SA escapes from a local minimum. Uphill decisions are made according to the Metropolis criteria. As the minimization process proceeds, the distance between the probed point and the current point decreases, and the probability of accepting uphill movements decreases as well. The search converges to a local (sometimes global) minimum at the end.

GA is based on the computational model of evolution [92,98,121,164]. GAs maintain a population of individual points in the search space, and the performance of the population evolves to be better through selection, recombination, mutation, and reproduction. The fittest individual has the largest probability of survival. The population evolves to contain better and better individuals and eventually converges to a local (sometimes global) minimum at the end.

SA and GA have been widely used to solve NP-hard problems. SA and GA achieve more success in solving discrete problems than continuous problems. They perform a search based on the objective (or fitness) function values, which makes them work well for discrete problems that do not have gradient information.

Tabu search was first introduced by Glover [92]. In tabu search, a *tabu list* containing historical information is maintained during the search. In each iteration, a local improvement is performed. However, moves that lead to solutions on the tabu list are forbidden, or are tabu. If there are no better solutions in the neighborhood, the tabu search moves to the best neighboring solution. The tabu list prevents returning to the local optimum from which the search has recently escaped. Tabu search has obtained good results in solving some large discrete optimization problems [18,19,44,112,223].

2.2 Proposed Methods for Handling Nonlinear Constraints

In this section, we present a new Lagrangian formulation to handle inequality constraints and propose new methods to search for saddle points of Lagrangian functions more efficiently. Our method employs adaptive control of relative weights between the objective and the constraints to achieve faster and more robust convergence. We, then, extend Lagrangian theory to discrete problems, and present a new discrete Lagrangian method to solve discrete constrained problems. Our approach of solving continuous and discrete constrained problems is to transform them into Lagrangian formulations, and then search for the corresponding saddle points.

2.2.1 Handling Continuous Constraints

We use Lagrange multipliers to handle nonlinear constraints. Lagrange-multiplier theory works well on equality constraints, but cannot directly deal with inequality constraints. Often, inequality constraints are first converted to equality constraints before Lagrange-multiplier methods are applied.

We have applied two methods to convert inequality constraints to equality constraints: the *slack-variable method* and the *MaxQ method.* Recall the general constrained problem as follows:

$$\begin{aligned} \min_{\mathbf{x}\in R^n} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & h_i(\mathbf{x}) = 0 \quad i = 1, \cdots, m_1 \\ & g_j(\mathbf{x}) \le 0 \quad j = 1, \cdots, m_2 \end{aligned} \tag{2.13}$$

In the slack-variable method introduced in the last section, inequality constraints are transformed to equality constraints by introducing Lagrange multipliers

$$\lambda = (\lambda_1, \cdots, \lambda_{m_1}, \lambda_{m_1+1}, \cdots, \lambda_{m_1+m_2}).$$

The augmented Lagrangian function of (2.13) is

$$\begin{aligned} L_z(\mathbf{x},\lambda) &= f(\mathbf{x}) + (\lambda_1,\cdots,\lambda_{m_1})^T h(\mathbf{x}) + \frac{1}{2}||h(\mathbf{x})||_2^2 \\ &+\frac{1}{2}\sum_{j=1}^{m_2}[\max{}^2(0,\lambda_{m_1+j}+g_j(\mathbf{x})) - \lambda_{m_1+j}^2] \end{aligned} \tag{2.14}$$

The first-order derivatives of $L_z(\mathbf{x},\lambda)$ with respect to $\mathbf{x}$ and λ are

$$\begin{aligned} \frac{\partial L_z(\mathbf{x},\lambda)}{\partial x_i} &= \frac{\partial f(\mathbf{x})}{\partial x_i} + \sum_{k=1}^{m_1}(\lambda_k + h_k(\mathbf{x}))\frac{\partial h_k(\mathbf{x})}{\partial x_i} + \sum_{k=1}^{m_2}\left[\max(0,\lambda_{m_1+k}+g_k(\mathbf{x}))\frac{\partial g_k(\mathbf{x})}{\partial x_i}\right] \\ \frac{\partial L_z(\mathbf{x},\lambda)}{\partial \lambda_j} &= h_j(\mathbf{x}), \qquad j=1,\cdots,m_1 \\ \frac{\partial L_z(\mathbf{x},\lambda)}{\partial \lambda_{m_1+j}} &= \max(0,\lambda_{m_1+j}+g_j(\mathbf{x})) - \lambda_{m_1+j}, \qquad j=1,\cdots,m_2 \end{aligned}$$

The other method we have applied to convert inequality constraints is the *MaxQ method* developed by Mr. Tao Wang, another student in our group [271,272]. In the MaxQ method, the maximum function is used to convert an inequality constraint into an equality constraint, i.e.,

$$g_j(\mathbf{x}) \le 0 \iff \max(0, g_j(\mathbf{x}))^{q_j} = 0$$

where, $q_j > 1, (j = 1, 2, \ldots, m_2)$ are control parameters. Inside the feasible region, constraint $g_j(\mathbf{x}) \le 0$, hence, $\max(0, g_j(\mathbf{x}))^{q_j} = 0$. In contrast to that in the slack-variable method, constraints have no effect inside feasible regions.

In the MaxQ method, the augmented Lagrangian function of (2.13) is

$$\begin{aligned} L_q(\mathbf{x},\lambda) &= f(\mathbf{x}) + (\lambda_1,\cdots,\lambda_{m_1})^T h(\mathbf{x}) + \frac{1}{2}||h(\mathbf{x})||_2^2 \\ &+\sum_{j=1}^{m_2}\lambda_{m_1+j}\max{}^{q_j}(0,g_j(\mathbf{x})) + \frac{1}{2}\sum_{j=1}^{m_2}\max{}^{2q_j}(0,g_j(\mathbf{x})) \end{aligned} \tag{2.15}$$

When the q_js are constant, the first-order derivatives of $L_q(\mathbf{x},\lambda)$ with respect to $\mathbf{x}$ and λ are

$$\begin{aligned}
\frac{\partial L_q(\mathbf{x},\lambda)}{\partial x_i} &= \frac{\partial f(\mathbf{x})}{\partial x_i} + \sum_{j=1}^{m_2} \lambda_{m_1+j} q_j \frac{\partial g_j(\mathbf{x})}{\partial x_i} \max{}^{q_j-1}(0, g_j(\mathbf{x})) \\
&\quad + \sum_{j=1}^{m_2} q_j \frac{\partial g_j(\mathbf{x})}{\partial x_i} \max{}^{2q_j-1}(0, g_j(\mathbf{x})) + \sum_{k=1}^{m_1} (\lambda_k + h_k(\mathbf{x})) \frac{\partial h_k(\mathbf{x})}{\partial x_i} \\
\frac{\partial L_q(\mathbf{x},\lambda)}{\partial \lambda_j} &= h_j(\mathbf{x}), \qquad j = 1, \cdots, m_1 \\
\frac{\partial L_q(\mathbf{x},\lambda)}{\partial \lambda_{m_1+j}} &= \max{}^{q_j}(0, g_j(\mathbf{x})), \qquad j = 1, \cdots, m_2
\end{aligned}$$

q_j must be larger than 1 in order for the term $\max^{q_j-1}(0, g_j(\mathbf{x}))$ to exist.

Based on the Lagrangian function and its gradients, we search for saddle points by performing gradient descents in the original variable $\mathbf{x}$ space and ascents in the Lagrange-multiplier λ space. The search is modeled by the following system of ordinary differential equations (ODEs):

$$\begin{aligned}
\frac{d\mathbf{x}(t)}{dt} &= -\nabla_{\mathbf{x}} L_z(\mathbf{x}(t), \lambda(t)) \qquad (2.16) \\
\frac{d\lambda(t)}{dt} &= \nabla_{\lambda} L_z(\mathbf{x}(t), \lambda(t))
\end{aligned}$$

This system of ODEs is solved as an initial-value problem, which converges to a saddle point corresponding to a local minimum of the original constrained optimization problem.

The convergence behavior of the slack-variable method is different from that of the MaxQ method when the saddle point is on the boundary of a feasible region. When a saddle point is inside a feasible region, and the initial point of the search is also inside the feasible region and close to the saddle point, the search trajectories of both methods can be similar. However, when a saddle point is on the boundary, the search trajectories of these two methods are usually different, even when they start from the same initial point.

When a saddle point is on the boundary of a feasible region, the slack-variable method approaches the saddle point from both inside and outside the feasible region. The search

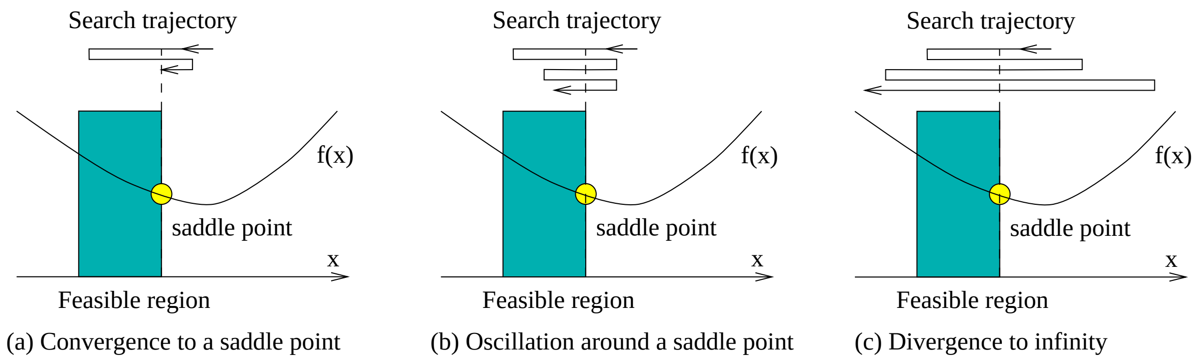


Figure 2.3: Illustration of the search trajectories using the slack-variable method when the saddle point is on the boundary of a feasible region. $f(x)$ is an objective function of variable x. The search can (a) converge to a saddle point, (b) oscillate around a saddle point, or (c) diverge to infinity.

trajectory oscillates around the saddle point and converges when the magnitude of oscillations goes to 0. In some situations, the magnitude of oscillations does not reduce, or even increases, causing the search to diverge eventually. Depending on the initial point and the relative magnitude of the objective function and the constraints, the search trajectory could end up in one of three different states as illustrated in Figure 2.3:

- convergence to a saddle point,
- oscillations around a saddle point, and
- divergence to large values and eventually to infinity.

The first state is desirable, while the other two states should be avoided. Oscillations and divergence happen when the accumulated force of the constraints becomes too large. The constraints accumulate momentum during the search in infeasible regions, which is represented as the values of corresponding Lagrange multipliers. When the search enters a feasible region, the momentum of the constraint force makes the search to go way deep into the feasible region. At this point, the objective and the corresponding gradient can become large. Eventually, the descent force on the objective dominates the constraint force, and

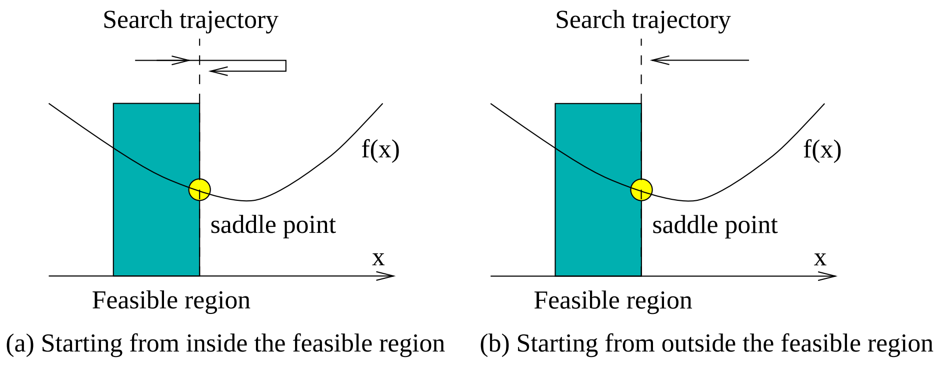


Figure 2.4: Illustration of the search trajectory using the MaxQ method when the saddle point is on the boundary of a feasible region. $f(x)$ is the objective function. The search starts from inside (left) or outside (right) the feasible region. In both cases, the search trajectory eventually approaches the saddle point from outside the feasible region.

the search trajectory follows gradient descent on the objective function and goes outside the feasible region again.

We have found that by appropriate scaling the relative weights between the objective function and the constraints, oscillations and divergence can be reduced or eliminated. Usually, when constraints have large weights, the search can successfully converge to a saddle point. Scaling can also speed up convergence.

Comparing to the slack-variable method, the MaxQ method does not have the problems of oscillations and divergence when the saddle point is on the boundary of a feasible region. The MaxQ method approaches the convergence point on the boundary asymptotically from the outside the feasible region. The closer it gets to the boundary, the slower it goes. Figure 2.4 illustrates the situations when the search starts from either inside or outside a feasible region. In both cases, the search trajectory eventually approaches the saddle point from outside the feasible region. The MaxQ method converges slowly when the search approaches a saddle point on the boundary of a feasible region.

There are two ways to speed up the convergence of the MaxQ method. The first one is to scale the weights between the objective function and the constraints in a way similar to that in the slack-variable method. Another way is to convert inequality constraints to equality constraints based on certain convergence conditions. After the conversion, convergence is usually much faster.

In both the slack-variable and the MaxQ methods, saddle points are searched by doing descents in the original-variable space and ascents in the Lagrange-multiplier space. For continuous problems whose first-order derivatives of the objective and the constraints exist, gradient descents and ascents are modeled by a system of ODEs and solved by an ODE solver. For discrete problems, derivatives of the objective function and the constraints do not exist. Therefore, descents are performed by some heuristic local search, e.g., hill-climbing. Ascents do not need derivatives and are only based on the values of constraint functions.

2.2.2 Convergence Speed of Lagrangian Methods

Lagrangian methods rely on two counteracting forces to resolve constraints and find high-quality solutions. When constraints are satisfied, Lagrangian methods rely on gradient descents in the objective space to find high-quality solutions. On the other hand, when constraints are not satisfied, they rely on gradient ascents in the Lagrange-multiplier space in order to increase the penalties on unsatisfied constraints and to force the constraints into satisfaction. The balance between gradient descents and gradient ascents depends on the relative magnitudes of the Lagrange multipliers λ, which play a role in balancing objective $f(\mathbf{x})$ and constraints $h(\mathbf{x})$ and $g(\mathbf{x})$ and in controlling indirectly the convergence speed and solution quality of the Lagrangian method. At a saddle point, the forces due to descent and ascent reach a balance through appropriate Lagrange-multiplier values.

We show in this section that the convergence speed and/or the solution quality can be affected by an additional weight w in the objective part of the Lagrangian function. This results in a new Lagrangian function of the slack-variable method as follows:

$$\begin{aligned} L_o(\mathbf{x}, \lambda) &= w\, f(\mathbf{x}) + (\lambda_1, \cdots, \lambda_{m_1})^T h(\mathbf{x}) + \frac{1}{2}||h(\mathbf{x})||_2^2 \\ &\quad + \frac{1}{2}\sum_{j=1}^{m_2}\left[\max{}^2(0, \lambda_{m_1+j} + g_j(\mathbf{x})) - \lambda_{m_1+j}^2\right] \end{aligned} \tag{2.17}$$

where $w > 0$ is a static weight on the objective. When $w = 1$, $L_o(\mathbf{x}, \lambda) = L_z(\mathbf{x}, \lambda)$, which is the original Lagrangian function.

In general, adding a weight to the objective changes the Lagrangian function, which in turn may cause the dynamic system to settle at a different saddle point with different solution quality. This is especially true when the saddle point is on the boundary of a feasible region. In this section we only show that the convergence time can be influenced greatly by the choice of the initial weight when the Lagrangian method converges to the same saddle points.

Starting from an initial point $(\mathbf{x}(0), \lambda(0))$, the dynamic system to find saddle points is based on (2.16) in which L_z is replaced by L_o:

$$\begin{aligned} \frac{d\mathbf{x}(t)}{dt} &= -\nabla_{\mathbf{x}} L_o(\mathbf{x}(t), \lambda(t)) \\ \frac{d\lambda(t)}{dt} &= \nabla_{\lambda} L_o(\mathbf{x}(t), \lambda(t)) \end{aligned} \tag{2.18}$$

We solve this dynamic system using an ordinary differential equation solver *LSODE*[1] and observe a search trajectory $(\mathbf{x}(t), \lambda(t))$. When a saddle point is on the boundary of the feasible region (which is true for most problems studied), the dynamic equation approaches it from both inside and outside the feasible region.

Depending on the weights between the objective and the constraints, the convergence speeds of Lagrangian methods can vary significantly. There is no good method in selecting the appropriate weights for a given problem. Our proposed dynamic weight-adaptation algorithm adjusts the weights between the objective and the constraints dynamically based on the search profile. It avoids selecting appropriate weights in the beginning, and achieves faster and robust convergence than using static weights.

We use the 48e QMF filter-bank design problem [133] to illustrate the convergence speed of Lagrangian methods using static weights. Depending on the weight, w, of the objective function, the convergence time of Lagrangian methods in solving the QMF design problem

[1]*LSODE* is a solver for first-order ordinary differential equations, a public-domain package available from http://www.netlib.org.

vary greatly, ranging from minutes to hours, even days. We use Johnston's solution as our starting point. This point is feasible as we use its performance measures as our constraints. However, it is not a local minimum of the objective function. Experimentally, using static w of $10^{-4}, 10^{-5}$, and 10^{-6} represents the three convergence behaviors: objective over-weighted, balanced objective and constraints, and constraint over-weighted.

For static weight $w = 10^{-4}$, Figure 2.5 shows the dynamic changes of the objective, the Lagrangian-function value, and the maximum violation as the search progresses. Note that the trajectories of the objective and the Lagrangian-function values are overlapped because constraint violations are small. As the starting point is not a local minimum of the objective, the search descends in the original-variable $\mathbf{x}$ space as the objective value decreases. In the meantime, the constraints are getting violated. As constraint violations become large, the Lagrangian part slowly gains ground and pushes the search back toward the feasible region, leading to increases in the objective value and decreases in the constraint violation. Eventually, the constraint violation becomes 0, and the objective value stabilizes. The overall convergence speed to the saddle point is slow (949.0 CPU minutes at $t = 2.819$). Note that fluctuations of the maximum violation as well as the objective is due to inaccuracy in numerical estimation of the function values and gradients.

Figure 2.6 shows the search profile using static weight $w = 10^{-5}$. The objective and the constraints are more balanced in this case, and the convergence time to the saddle point is shorter (125.7 CPU minutes at $t = 1.577$ time units).

Figure 2.7 shows the search profile using static weight $w = 10^{-6}$. The constraints are over-weighted in this case, and constraint satisfaction dominates the search process. The trajectory is kept inside or very close to the feasible region. However, due to the small weight on the objective, improvements of the objective are slow, causing slow overall convergence to the saddle point (293.8 CPU minutes at $t = 7.608$).

2.2.3 Dynamic Weight Adaptation

In this section, we propose a strategy to adapt weight w based on the behavior of the dynamic system (2.18) in order to obtain high-quality solutions with short convergence

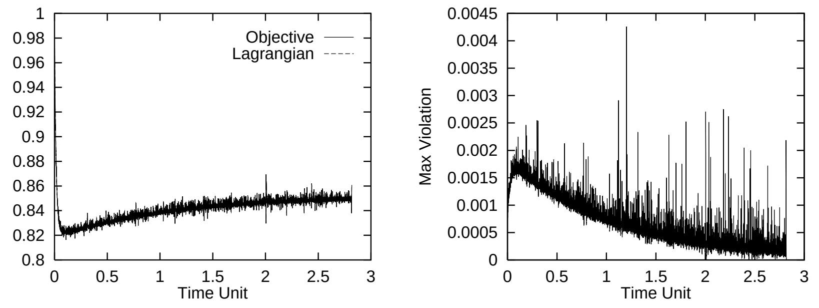


Figure 2.5: Search profile with static weight $w = 10^{-4}$ in designing a 48e QMF filter bank. The objective is over-weighted, and its values are reduced quickly in the beginning. Later, the Lagrangian part slowly gains ground and pushes the search back into the feasible region. The convergence time to the saddle point is long (949.0 CPU minutes at $t = 2.819$).

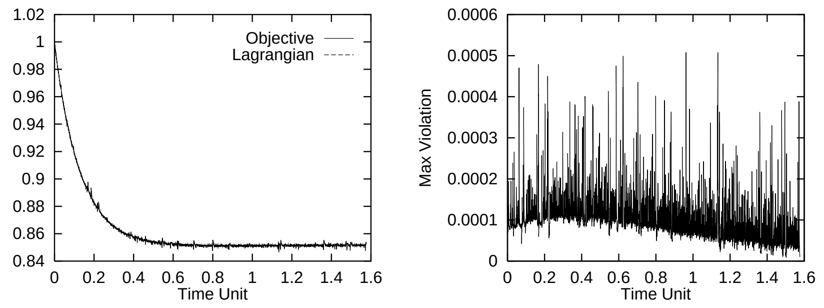


Figure 2.6: Search profile with static weight $w = 10^{-5}$ in designing a 48e QMF filter bank. The objective and the constraints are balanced, and the convergence speed to the saddle point is faster (125.7 CPU minutes at $t = 1.577$).

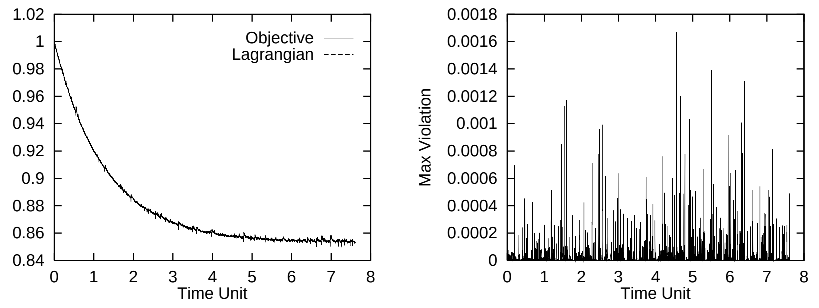


Figure 2.7: Search profile with static weight $w = 10^{-6}$ in designing a 48e QMF filter bank. The constraints are over-weighted, and constraint satisfaction dominates the search process. The trajectory is kept inside or very close to the feasible region. However, the improvement of the objective value is slow, causing slow overall convergence to the saddle point (293.8 CPU minutes at $t = 7.608$).

```
1.  Select control parameters:
        time interval $\triangle t$
        initial weight $w(t = 0)$
        maximum number of iterations $i_{max}$
2.  Set window size $N_w = 100 \triangle t$ or $10 \triangle t$
3.  $j := 1$        /* $j$ is the iteration number */
4.  **while** $j \leq i_{max}$ **and** stopping condition is not satisfied **do**
5.      advance search trajectory by $\triangle t$ time unit to get to $(\mathbf{x}^j, \lambda^j)$
6.      **if** trajectory diverges **then**
            reduce $w$; restart the algorithm by going to Step 2
        **end if**
7.      record useful information for calculating performance metrics
8.      **if** $(mod(j, N_w) == 0)$ **then**
            /* Test whether $w$ should be modified at the end of a window */
9.          compute performance metrics based on data collected
10.         change $w$ if some conditions hold (see Figure 2.9)
        **end if**
11. **end while**
```

Figure 2.8: Framework of the dynamic weight-adaptation algorithm

times [268]. In general, before a trajectory reaches a saddle point, changing the weight of the objective may speed up or delay convergence. Further, when the trajectory is at a saddle point, changing the weight of the objective may bring the trajectory out of the saddle point into another. In this section, we exploit the first property by designing weight-adaptation algorithms so that we can speed up convergence without affecting solution quality. We plan to exploit the second property to bring a trajectory out from a saddle point in our future work. We show in Chapter 3 a trace-based approach to bring the search trajectory out of saddle points.

Figure 2.8 outlines the general strategy. Its basic idea is to first estimate the initial weight $w(t = 0)$ (Step 1), measure the performance metrics of the search trajectory $(\mathbf{x}(t), \lambda(t))$ periodically, and adapt $w(t)$ to improve convergence time or solution quality.

Let t_{max} be the total (logical) time for the search, and t_{max} be divided into small units of time $\triangle t$ so that the maximum number of iterations is $t_{max} / \triangle t$. Further, assume a stopping condition if the search were to stop before t_{max} (Step 4). Given a starting point $\mathbf{x}(t=0)$, we set the initial Lagrange multipliers to be zero; *i.e.*, $\lambda(t=0)=0$. Let $(\mathbf{x}^i, \lambda^i)$ be the point of the i^{th} iteration.

To monitor the progress of the search trajectory, we divide time into non-overlapping windows of size N_w iterations each (Step 2). In each window, we compute some metrics to measure the progress of the search relative to that of previous windows. Let $v_{max,i}$ be the maximum constraint violation and $f_i(\mathbf{x})$ be the objective function value at the i^{th} iteration. For the m^{th} window ($m = 1, 2, \cdots$), we calculate the average (or median) of $v_{max,i}$ over all the iterations in the window,

$$\bar{v}_m = \frac{1}{N_w} \sum_{j=(m-1)N_w+1}^{mN_w} v_{max,j} \quad \text{or} \quad \bar{v}_m = median_{(m-1)N_w+1 \,\leq\, j \,\leq\, mN_w}\{v_{max,j}\} \tag{2.19}$$

and the average (or median) of the objective $f_i(\mathbf{x})$:

$$\bar{f}_m = \frac{1}{N_w} \sum_{j=(m-1)N_w+1}^{mN_w} f_j(\mathbf{x}) \quad \text{or} \quad \bar{f}_m = median_{(m-1)N_w+1 \,\leq\, j \,\leq\, mN_w}\{f_j(\mathbf{x})\} \tag{2.20}$$

During the search, we apply LSODE to solve the dynamic system (2.18) and advance the trajectory by time interval $\triangle t$ in each iteration in order to arrive at point $(\mathbf{x}^j, \lambda^j)$ (Step 5).

At this point, we test whether the trajectory diverges or not (Step 6). Divergence happens when the maximum violation $v_{max,j}$ is larger than an extremely large value (e.g. 10^{20}). If it happens, we reduce w by a large amount, say $w \Longleftarrow w/10$, and restart the algorithm. In each iteration, we also record some statistics, such as $v_{max,j}$ and $f_j(\mathbf{x})$, that will be used to calculate the performance metrics for each window (Step 7).

At the end of each window or every N_w iterations (Step 8), we decide whether to update w based on the performance metrics (2.19) and (2.20) (Step 9). In our current implementation, we use the averages (or medians) of maximum violation $v_{max,i}$ and objective $f_j(\mathbf{x})$. In general, other application-specific metrics can be used, such as the number of oscillations of the trajectory in nonlinear continuous problems. Based on these measurements, we adjust w accordingly (Step 10).

```
Given performance measurements for the $m^{th}$ window
    average (or median) of the maximum violation: $\bar{v}_m$
    average (or median) of the objective: $\bar{f}_m$
    number of oscillations: $NO_m$
and application-specific constants $\alpha_0, \alpha_1, \beta_0, \beta_1, \delta, \epsilon$
1. Increase weight $w \Longleftarrow w/\alpha_0$ **if**
    $(c_{1,1})$ $\bar{v}_{m-1}, \bar{v}_m < \delta$
    $(c_{1,2})$ $\beta_0|\bar{f}_{m-1}| > \bar{f}_{m-1} - \bar{f}_m > \beta_1|\bar{f}_{m-1}|$
2. Decrease weight $w \Longleftarrow \alpha_1 w$ **if**
    $(c_{2,1})$ $\bar{v}_m \geq \delta$
    $(c_{2,2})$ $\bar{v}_{m-1} - \bar{v}_m \leq \beta_0 \bar{v}_{m-1}$
    $(c_{2,3})$ $\begin{cases} NO_m < \epsilon & \text{(for continuous QMF design problems)} \\ NO_m \geq \epsilon & \text{(for other continuous and discrete problems)} \end{cases}$
```

Figure 2.9: Weight-adaptation rules for continuous and discrete problems (Step 10 of Figure 2.8).

Figure 2.9 shows a comprehensive weight-adaptation algorithm (Step 10). Scaling factors $0 < \alpha_0, \alpha_1 < 1$ represent how fast w is updated. Because we use numerical methods to solve the dynamic system defined in (2.18), a trajectory in window m is said to satisfy all the constraints when $v_m < \delta$, where δ is related to the convergence condition and the required precision. Parameters $0 < \beta_0, \beta_1 < 1$ control, respectively, the degrees of improvement over the objective and the reduction of the maximum violation. Note that when comparing values between two successive windows $m-1$ and m, both must use the same weight w; otherwise, the comparison is not meaningful because the terrain may be totally different. Hence, after adapting w, we should wait at least two windows before changing it again.

Weight w should be increased (Rule 1) when we observe the third convergence behavior. In this case, the trajectory is within a feasible region, and the objective is improved in successive windows. Weight w will be increased when the improvement of the objective in a feasible region is not fast enough, but will not be increased when the improvement is beyond an upper bound.

Weight w should be decreased (Rule 2) when we observe the second convergence behavior (the trajectory oscillating around a saddle point) or the fourth convergence behavior (the trajectory moving slowly back to the feasible region). Weight w will be decreased when the trajectory oscillates, and the trend of the maximum violation does not decrease. We have also defined two cases in condition $c_{2,3}$ of Rule 2: the first case applies to the continuous QMF design problems that do not exhibit oscillatory behavior, whereas the second case applies to the rest of the problems.

We use the 48e QMF filter-bank design problem [133] to illustrate the improvement of our dynamic weight-adaptation method as compared to that of using static weights. We use Johnston's solution as our starting point. To avoid the difficulty in choosing the initial weight, we choose the starting weight $w(t=0)$ using a two-step method:

1. Set the initial weight based on the maximum-gradient component of the starting point and the number of variables, $w(t=0) = 1/(n\ \max_{1 \le i \le n}\{\frac{d}{dx_i} L_z(\mathbf{x}(t=0), \lambda = 0)\})$;
2. Perform the Lagrangian search for a small amount of time, e.g., $\delta t = 10^{-5}$. Adjust the weight $w(t=0)$ based on the decrement $\triangle f$ of the objective-function value: $w(t=0) \longleftarrow \frac{w(t=0)\,\delta t}{\triangle f}$.

Figure 2.10 shows the search profile of our Lagrangian method with adaptive weight control in solving the 48e QMF filter bank problem. We have used a time interval $\triangle t = 10^{-4}$, window size $N_w = 10$, $\alpha_0 = \alpha_1 = 0.9$, $\delta = 10^{-5}$, $\beta_0 = 10^{-3}$, and $\beta_1 = 10^{-2}$. As shown in the search profiles, the objective value was improved fairly quickly in the beginning, and the trajectory was then pushed into a feasible region quickly. Our dynamic method converges at $t = 0.142$ (35.1 CPU minutes), much faster than the static-weight Lagrangian method.

2.2.4 Discrete Constraint Handling Method

Discrete constraints are handled based on the Lagrange multipliers in a way similar to how we handle continuous constraints. We extend the traditional Lagrangian theory to discrete problems, and develop a new discrete optimization method based on the Lagrangian formulation, called the *discrete Lagrangian method* (DLM).

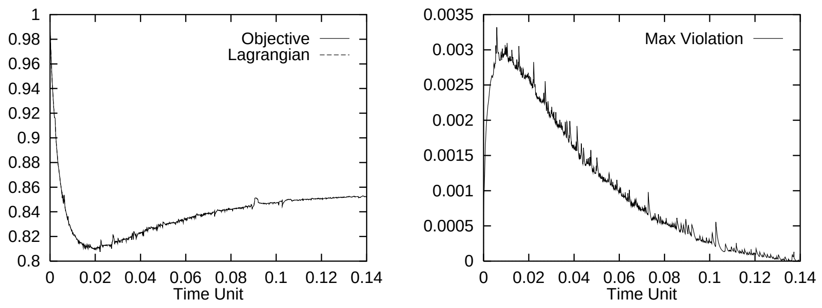


Figure 2.10: Profile with adaptive changes of w in designing a 48e QMF filter bank. The search converges much faster (35.1 CPU minutes in $t = 0.142$) than those using static weights.

Lagrangian methods have been used to solve discrete optimization problems whose discrete variables have been relaxed into continuous ones. However, little work has been done in applying Lagrangian methods to work on discrete problems directly [87,91,116]. The difficulty in traditional Lagrangian methods lies in the requirement of a differentiable continuous space. In order to apply Lagrangian methods to discrete optimization problems, we need to develop a new Lagrange-multiplier-based search method that works in discrete space.

Consider the following equality-constrained optimization problem in discrete space, which is similar to the continuous version in (2.1),

$$\begin{aligned} \min_{\mathbf{x}\in D^n} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h(\mathbf{x}) = 0 \end{aligned} \tag{2.21}$$

where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, $h(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_m(\mathbf{x}))$ and D is a subset of integers, i.e. $D^n \subset I^n$.

Definition 2.2.1 *A local minimum $\mathbf{x}^*$ to problem (2.21) is defined as $h(\mathbf{x}^*) = 0$ and $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for any feasible $\mathbf{x}$, and $\mathbf{x}^*$ and $\mathbf{x}$ differ in only one dimension by a magnitude of 1.*

For example, if $\mathbf{x}$ differs from $\mathbf{x}^*$ in the k^{th} dimension by 1, then $|x_k^* - x_k| = 1$. A local minimum is defined with respect to a certain *neighborhood*. In this definition, the neighborhood is within 1 of one dimension. Note that this definition can be extended to the case in which two points differ by more than one dimension.

Discrete Lagrangian function F has the same form as in the continuous case and is defined as follows:

$$F(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i h_i(\mathbf{x}) \tag{2.22}$$

where $\lambda = (\lambda_1, \lambda_2, \cdots, \lambda_m)$ are Lagrange multipliers that have real values.

A saddle point $(\mathbf{x}^*, \lambda^*)$ of $F(\mathbf{x}, \lambda)$ is reached when the following condition is satisfied:

$$F(\mathbf{x}^*, \lambda) \leq F(\mathbf{x}^*, \lambda^*) \leq F(\mathbf{x}, \lambda^*) \tag{2.23}$$

for all λ sufficiently close to λ^* and all $\mathbf{x}$ that differ from $\mathbf{x}^*$ in only one dimension by a magnitude 1.

In a way similar to the continuous case, we derive the following theorem specifying the relation between local minima and saddle points.

Discrete Saddle-Point Theorem for Discrete Problems. $\mathbf{x}^*$ is a local minimum solution to the discrete constrained problem in (2.21) if there exists λ^* such that $(\mathbf{x}^*, \lambda^*)$ constitutes a saddle point of the associated Lagrangian function $F(\mathbf{x}, \lambda)$.

Proof. The proof is similar to that of the continuous case. ■

In the continuous case, methods that look for saddle points utilize gradient information. In order for these methods to work in the discrete variable space $\mathbf{x}$ of discrete problems, we need to define the counterpart of the gradient operator. Note that λ can remain to be continuous even in the discrete case. In the following, we define a discrete *difference gradient descent operator* $\Delta_{\mathbf{x}}$. (Note that this operator is not unique, and other operators can be defined to work in a similar way.)

Definition 2.2.2 *Difference gradient descent operator* $\Delta_{\mathbf{x}}$ *is defined with respect to* $\mathbf{x}$ *in such a way that* $\Delta_{\mathbf{x}} F(\mathbf{x}, \lambda) = (\delta_1, \delta_2, \cdots, \delta_n) \in \{-1, 0, 1\}^n$, $\sum_{i=1}^{n} |\delta_i| = 1$, *and* $(\mathbf{x} - \Delta_{\mathbf{x}} F(\mathbf{x}, \lambda)) \in D^n$. *For any* $\mathbf{x}'$ *such that* $\sum_{i=1}^{n} |x'_i - x_i| = 1$,

$$F(\mathbf{x} - \Delta_{\mathbf{x}} F(\mathbf{x}, \lambda), \lambda) \leq F(\mathbf{x}', \lambda).$$

Further, if $\forall \mathbf{x}', F(\mathbf{x}, \lambda) \le F(\mathbf{x}', \lambda)$, *then* $\Delta_{\mathbf{x}} F(\mathbf{x}, \lambda) = 0$.

A more general discrete descent operator $\Gamma_{\mathbf{x}}$ is defined as follows:

Definition 2.2.3 *Discrete descent operator* $\Gamma_{\mathbf{x}}$ *is defined with respect to* $\mathbf{x}$ *in such a way that* $\Gamma_{\mathbf{x}} F(\mathbf{x}, \lambda) = (\delta_1, \delta_2, \cdots, \delta_n) \in \{-1, 0, 1\}^n$, $\sum_{i=1}^{n} |\delta_i| = 1$, $(\mathbf{x} - \Gamma_{\mathbf{x}} F(\mathbf{x}, \lambda)) \in D^n$, *and*

$$F(\mathbf{x} - \Gamma_{\mathbf{x}} F(\mathbf{x}, \lambda), \lambda) \le F(\mathbf{x}, \lambda).$$

Further, if $\forall \mathbf{x}'$ *such that* $\sum_{i=1}^{n} |x_i' - x_i| = 1$, $F(\mathbf{x}, \lambda) \le F(\mathbf{x}', \lambda)$, *then* $\Gamma_{\mathbf{x}} F(\mathbf{x}, \lambda) = 0$.

Based on this definition, Lagrangian methods for continuous problems can be extended to discrete problems. The basic idea is to descend in the original discrete variable space of $\mathbf{x}$ and ascend in the Lagrange-multiplier space of λ. We propose a generic discrete Lagrangian method as follows:

Generic Discrete Lagrangian Method (DLM)

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \Gamma_{\mathbf{x}}(\mathbf{x}^k, \lambda^k) \tag{2.24}$$

$$\lambda^{k+1} = \lambda^k + c\, h(\mathbf{x}^k) \tag{2.25}$$

where $\Gamma_{\mathbf{x}}(\mathbf{x}^k, \lambda^k)$ is a discrete descent operator for updating $\mathbf{x}$ based on the Lagrangian function $F(\mathbf{x}, \lambda)$, c is a positive real number controlling how fast the Lagrange multipliers change, and k is the iteration index. ■

Fixed-Point Theorem of DLM. A saddle point $(\mathbf{x}^*, \lambda^*)$ of the Lagrangian function $F(\mathbf{x}, \lambda)$ is found if and only if (2.24) and (2.25) terminate.

Proof. "$\Rightarrow$" part: If (2.24) and (2.25) terminate at $(\mathbf{x}^*, \lambda^*)$, then $h(\mathbf{x}^*) = 0$ and $\Gamma_{\mathbf{x}} F(\mathbf{x}^*, \lambda^*) = 0$. $h(\mathbf{x}^*) = 0$ implies that $F(\mathbf{x}^*, \lambda) = F(\mathbf{x}^*, \lambda^*)$ for any λ. $\Gamma_{\mathbf{x}} F(\mathbf{x}^*, \lambda^*) = 0$ implies that $F(\mathbf{x}^*, \lambda^*) \le F(\mathbf{x}, \lambda^*)$ for all $\mathbf{x}$ that differ from $\mathbf{x}^*$ in only one dimension by a magnitude 1. Therefore, $(\mathbf{x}^*, \lambda^*)$ is a saddle point.

"$\Leftarrow$" part: If $(\mathbf{x}^*, \lambda^*)$ is a saddle point, by the Discrete Saddle Point Theorem, $\mathbf{x}^*$ is a local minimum and satisfies the constraints, i.e., $h(\mathbf{x}^*) = 0$. Also, because $(\mathbf{x}^*, \lambda^*)$ is a saddle point, we have $F(\mathbf{x}^*, \lambda^*) \leq F(\mathbf{x}, \lambda^*)$ for all $\mathbf{x}$ that differ from $\mathbf{x}^*$ in only one dimension by a magnitude 1. By the definition of $\Gamma_{\mathbf{x}}$, $\Gamma_{\mathbf{x}} F(\mathbf{x}^*, \lambda^*) = 0$. Hence, (2.24) and (2.25) terminates at $(\mathbf{x}^*, \lambda^*)$. ■

$\Gamma_{\mathbf{x}}(\mathbf{x}, \lambda)$ is not unique and can be defined by either steepest descent or hill climbing. In steepest-descent, $\Gamma_{\mathbf{x}}(\mathbf{x}, \lambda) = \Delta_{\mathbf{x}} F(\mathbf{x}, \lambda)$, is the direction with the maximum improvement. A hill-climbing approach, on the other hand, chooses the first point in the neighborhood of the current $\mathbf{x}$ that reduces F. Thus, the descent direction $\Gamma_{\mathbf{x}}(\mathbf{x}, \lambda) \neq \Delta_{\mathbf{x}} F(\mathbf{x}, \lambda)$ in this case. Although both approaches have different descent trajectories, they can both reach equilibrium that satisfies the saddle-point condition. Consequently, they can be regarded as alternative approaches to calculate $\Delta_{\mathbf{x}} F(\mathbf{x}, \lambda)$.

In solving discrete constrained problems with inequality constraints, we first transform the inequality constraints to equality constraints using the maximum function. Consider the following discrete constrained problem:

$$\begin{aligned} \min_{\mathbf{x} \in D^n} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h(\mathbf{x}) = 0 \\ & g(\mathbf{x}) \leq 0 \end{aligned} \tag{2.26}$$

where $h(\mathbf{x}) = (h_1(\mathbf{x}), \cdots, h_{m_1}(\mathbf{x}))$, $g(\mathbf{x}) = (g_1(\mathbf{x}), \cdots, g_{m_2}(\mathbf{x}))$, and D is a subset of integers, i.e. $D^n \subset I^n$. This problem with inequality constraints is transformed into an optimization problem with equality constraints as follows:

$$\begin{aligned} \min_{\mathbf{x} \in D^n} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h(\mathbf{x}) = 0 \\ & \max(g(\mathbf{x}), 0) = 0 \end{aligned} \tag{2.27}$$

Obviously, the local minima of this equality constrained problem are one-to-one correspondent with those of the original inequality constrained problem. Then, we apply our discrete Lagrangian method (DLM) to find the local minima of the transformed problem, which are actually the local minima of the original problem.

As in applying Lagrangian method to solve continuous constrained optimization problems, the relative weights between the objective and the constraints affect the performance of the discrete Lagrangian method (DLM), too. Domination of either the objective or the constraints can hinder the search of saddle points by DLM.

We have developed mechanisms to dynamically adjust the relative weights between the objective and the constraints in DLM. In solving satisfiability problems using DLM (to be discussed in Chapter 6), we have applied a mechanism of periodic reduction of the values of the Lagrange multipliers to dynamically adjust the weights and to obtain improved performance.

Consider the effect of dynamic weight scaling using a profile obtained from solving a difficult satisfiability benchmark problem "g125-17.cnf." In our formulation of satisfiability problems in Chapter 6, the constraints have values of 0 or 1. Accordingly, Lagrange multipliers λ in DLM are associated with the constraints and are always nondecreasing. For difficult problems that require millions of iterations, λ values can become very large as the search progresses. Large λ's are generally undesirable because they put too much weight on the constraints, causing large swings in the Lagrangian-function value, and making the search of saddle points more difficult.

Figure 2.11 shows the profiles of DLM without dynamic scaling of Lagrange multipliers on problem "*g125-17*" starting from a random initial point. Lagrange multipliers and Lagrangian-function values can grow very large. The objective-part and constraint-part (Lagrangian-part) values of the Lagrangian function are shown in the lower two graphs. As time goes on, the constraint-part becomes much larger than the objective-part. It takes a very long time for DLM to find a solution from a randomly generated initial point, or it may not find one in a reasonable amount of time.

We have introduced periodic scale-downs of Lagrange multipliers to control the growth of Lagrange multipliers as well as Lagrangian-function values. Figure 2.12 shows the result when all the Lagrange multipliers are scaled down by a factor of 1.5 every 10,000 iterations. These graphs indicate that periodic scaling leads to bounded values of Lagrange multipliers and Lagrangian-function values. The objective-part and constraint-part values of the

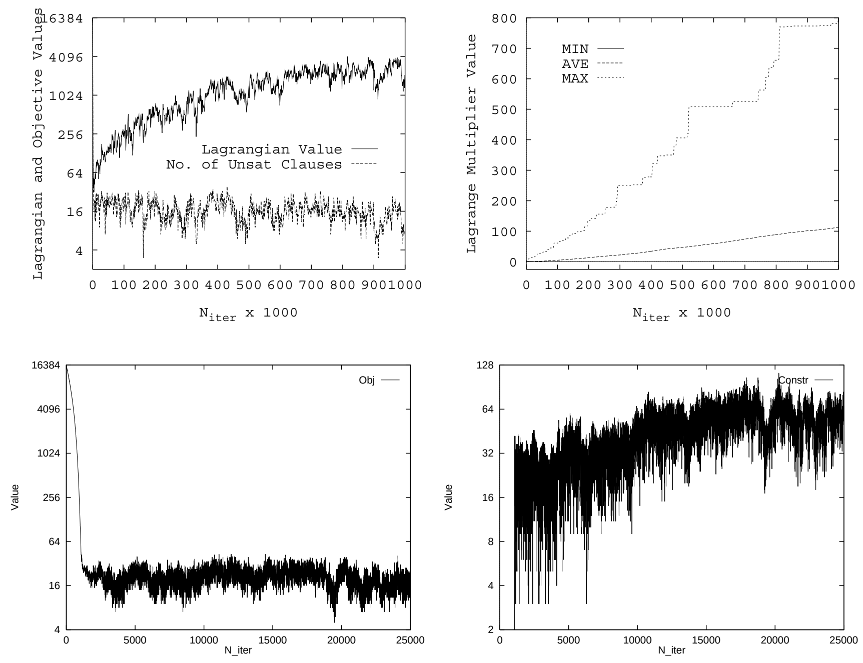


Figure 2.11: Execution profiles of DLM $\mathcal{A}_3$ without dynamic scaling of Lagrange multipliers λ on problem "*g125-17*" starting from a random initial point. The top left graph plots the Lagrangian-function values and the number of unsatisfied clauses against the number of iterations sampled every 1000 iterations. The top right graph plots the minimum, average and maximum values of the Lagrange multipliers sampled every 1000 iterations. The bottom two graphs plot the iteration-by-iteration objective and Lagrangian-part values for the first 25,000 iterations.

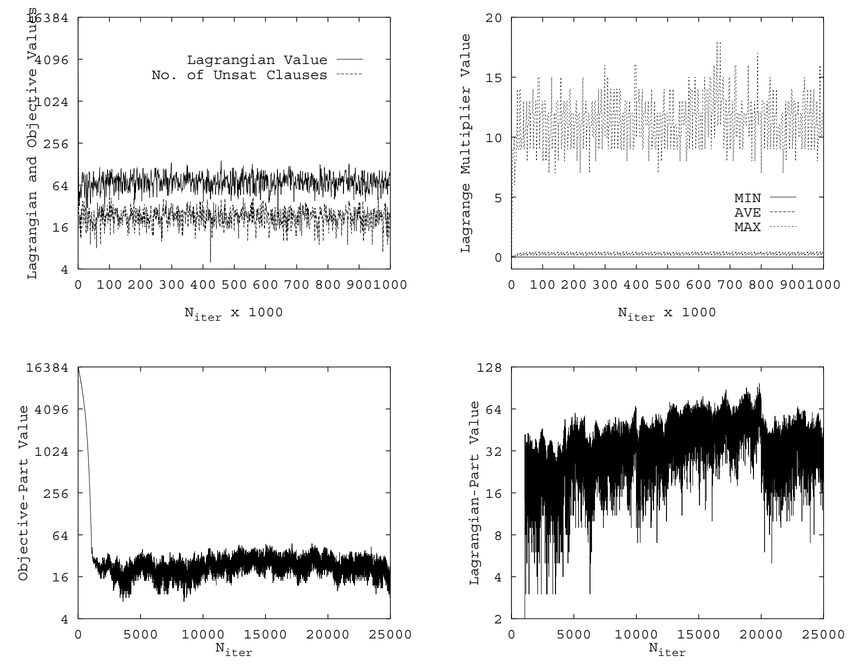


Figure 2.12: Execution profiles of $\mathcal{A}_3$ with periodic scaling of λ by a factor of 1.5 every 10,000 iterations on problem "*g125-17*" starting from a random initial point. Note that the average Lagrange-multiplier values are very close to 1. See Figure 2.11 for further explanation.

Lagrangian function are more balanced and have about the same magnitude. Using this dynamic scaling method, we can solve some of the more difficult satisfiability benchmark problems.

2.3 Summary

In this chapter, we have addressed issues in handling nonlinear constraints. Existing methods that handle nonlinear constraints are either transformational or non-transformational. In non-transformational approaches, the search works on the constraints directly, and tries to stay within feasible regions. Such methods have difficulty when the nonlinear constraints form feasible regions that are irregular and are difficult to find.

Transformational approaches, on the other hand, are more popular in dealing with nonlinear constraints. Constrained optimization problems are transformed into unconstrained problems to be solved. Widely used methods include penalty methods, barrier methods, sequential quadratic methods, and Lagrangian methods. Among these methods, Lagrangian methods are general, robust, and can achieve high precision of solutions. Due to these advantages, we use a variation of Lagrangian methods to handle nonlinear constraints in this thesis.

In our Lagrangian methods, we first convert inequality constraints to equality constraints, and then search for saddle points of the Lagrangian function by performing descents in the original variable space and ascents in the Lagrange-multiplier space. There are two methods to convert inequalities, the slack variable methods and the MaxQ method. They derive different Lagrangian formulations. The convergence speed of the slack-variable method is usually faster than that of the MaxQ method. However, the slack-variable method may oscillate or diverge under certain conditions, and may not converge to a saddle point. The MaxQ method does not have this problem, but its convergence may be slow.

We have discovered that through dynamic control of the relative weights of the objective and the constraints, we can reduce or eliminate oscillations and divergence of the slack-variable method. We have developed various mechanisms to control the relative weight adaptively during a search. In applying the Lagrangian method with adaptive control to application problems, we are able to achieve faster and more robust convergence.

Discrete optimization problems can be solved by either exact or inexact methods. For NP-hard problems, exact methods such as branch-and-bound are computationally intractable for large discrete problems. Inexact methods use heuristics to search for solutions and offer no optimality guarantee. However, inexact methods have been applied to much larger problems than what exact methods can handle, and have found good solutions.

We have taken the inexact approach in solving large problems, and have developed the discrete Lagrangian method (DLM), a Lagrange multiplier-based optimization method, to solve discrete constrained optimization problems. A Lagrangian function of discrete variables are formed by combining the objectives and the constraints using Lagrange multipliers. Then, saddle points are searched by doing descents in the original variable space and ascents in the Lagrange multiplier space. In contrast to solving continuous problems in which descents are based on continuous gradients, heuristic discrete descents are performed by using hill-climbing.

The relative magnitude of the objective and the constraints affects the performance of the discrete Lagrangian method. We control their relative magnitude by associating weight coefficients with them, and adjust the weights dynamically during the search. In our experiments of solving difficult discrete problems, we have found that appropriate dynamic control of the relative weights can improve the performance of the discrete Lagrangian method by significantly reducing the time to find saddle points.

3. OVERCOMING LOCAL MINIMA

General nonlinear optimization problems are difficult to solve due to the large number of local minima in the search space. Good local minima are difficult to be found by local search methods because they stop at every local minimum. Hence, to obtain globally optimal solutions, global search methods have been developed to escape from local minima once the search gets there, and continue the search.

In this chapter, we address an important issue in solving nonlinear optimization problems – overcoming local minima. We first review existing nonlinear local and global optimization methods, and identify their advantages and disadvantages. Then, we propose a new multi-level global search method that combines global and local searches. We present the overall framework of this method and discuss each component in details. Our method employs an innovative trace-based global-search method that searches the space continuously and escapes from local minima without restarts. In the end, we present Novel, a prototype implementing our proposed methods for nonlinear continuous and discrete optimization.

3.1 Previous Work on Global Optimization

Due to the importance of optimal solutions for engineering, economical and social sciences applications, many optimization methods have been developed. In recent decades, as computers become more powerful, numerical optimization algorithms have been developed for many applications.

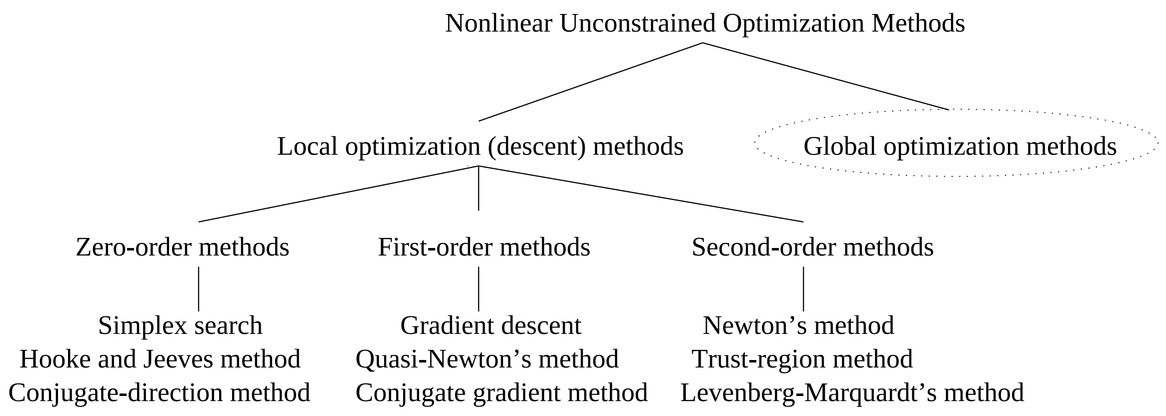

Nonlinear Unconstrained Optimization Methods
Local optimization (descent) methods
Global optimization methods
Zero-order methods
First-order methods
Second-order methods
Simplex search
Hooke and Jeeves method
Conjugate-direction method
Gradient descent
Quasi-Newton's method
Conjugate gradient method
Newton's method
Trust-region method
Levenberg-Marquardt's method

Figure 3.1: Classification of local optimization methods for unconstrained nonlinear optimization problems.

Solution methods for nonlinear optimization problems can be classified into local and global methods. Local optimization methods, such as gradient-descent and Newton's methods, use local information (gradient or Hessian) to perform descents and converge to a local minimum. They can find local minima efficiently and work best in uni-modal problems. Global methods, in contrast, employ heuristic strategies to look for global minima and do not stop after finding a local minimum [187]. A taxonomy on global optimization methods can be found in [83,111,125,188,253]. Note that gradients and Hessians can be used in both local and global methods.

Figure 3.1 shows a classification of local optimization methods for unconstrained nonlinear optimization problems. The methods can be broadly classified as zero-order, first-order, and second-order methods based on the derivative information used during the search. Generally, higher-order methods converge to local minima faster.

Zero-order methods do not use derivatives of objective functions during optimization. Examples are the simplex search method, the Hooke and Jeeves method, the Rosenbrock method, and the conjugate direction method [46,137,178,200,248,270].

First-order methods use first-order derivatives of the objective function during the search. Examples are the gradient-descent method, the discrete Newton's method, the quasi-Newton methods, and the conjugate gradient methods. The gradient-descent method performs a linear search along the direction of the negative gradient of the minimized function [6, 7, 13, 182, 247, 277]. The discrete Newton's method approximates the Hessian matrix by the finite difference of the gradient. Quasi-Newton methods approximate the curvature of the nonlinear function using information of the function and its gradient only, and avoid the explicit evaluation of the Hessian matrix [8, 34, 62, 161]. Conjugate gradient methods combine the current gradient with the gradients of previous iterations and the previous search direction to form the new search direction. They generate search directions without storing a matrix [73, 97, 117, 127, 143, 160, 170].

Second-order methods make use of second-order derivatives. They include Newton's method, Levenberg-Marquardt's method, and trust region methods [8, 64, 161, 232, 276]. In Newton's method, the inverse of the Hessian matrix multiplies the gradient, and a suitable search direction is found based on a quadratic approximation of the function. Newton's method converge quadratically if the initial point is close to a local minimum. Levenberg-Marquardt's method and trust region methods are modifications of Newton's method. By using either a line-search or a trust-region approach, these algorithms converge when their starting point is not close to a minimum. Line-search and trust-region techniques are suitable if the number of variables is not too large. Truncated Newton's methods are more suitable for problems with a large number of variables. They use iterative techniques to obtain a direction in a line-search method or a step in a trust-region method. The iteration is stopped (truncated) as soon as a termination criterion is satisfied.

Local optimization methods converge to local minima. For some applications, local optima may be good enough, particularly when the user can draw on his/her own experience and provide a good starting point for local optimization algorithms. However, for many applications, globally optimal or near-optimal solutions are desired.

In nonlinear optimization, objective functions are multi-modal with many local minima. Local search methods converge to local minima close to the initial points. Therefore, the solution quality depends heavily on the initial point picked. When the objective function is highly nonlinear, local-search methods may return solutions much worse than the global optima when starting from a random initial point.

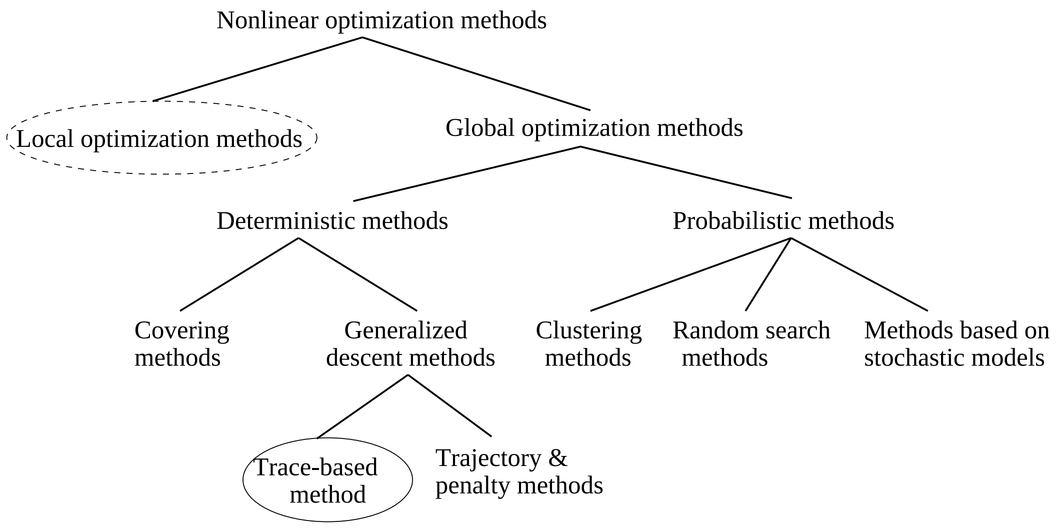

Nonlinear optimization methods
Local optimization methods
Global optimization methods
Deterministic methods
Probabilistic methods
Covering methods
Generalized descent methods
Clustering methods
Random search methods
Methods based on stochastic models
Trace-based method
Trajectory & penalty methods

Figure 3.2: Classification of unconstrained nonlinear continuous global minimization methods.

To overcome the deficiencies in local search methods, global search methods have been developed with global search mechanisms. Global search methods use local search to determine local minima, and focus on bringing the search out of a local minimum once it gets there.

Figure 3.2 shows a classification of nonlinear global optimization methods. Methods to solve global optimization problems have been classified as either probabilistic or deterministic. Probabilistic (stochastic) methods evaluate the objective function at randomly sampled points from the solution space. Deterministic methods, on the other hand, involve no element of randomness.

Alternatively, global optimization algorithms can also be classified as reliable and unreliable. Reliable methods guarantee solution quality while unreliable methods do not. Probabilistic methods, including simulated annealing, clustering, and random searching, fall into the unreliable category. Unreliable methods usually have the strength of efficiency and better performance in solving large-scale problems.

Table 3.1: Global and local search components used in existing global optimization methods.

Method	Global Component	Local Component
Random Search	Uniform Sampling	Any Available Local Method
Genetic Algorithm	Selective Recombination	Optional
Simulated Annealing	Boltzmann Motion	Optional
Clustering Method	Cluster Analysis	Any Available Local Method
Bayesian Method	Bayesian Decision	Optional
Interval Method	Interval Calculation	Rarely Used
Covering Method	Informed Search with Bound Approximation	Rarely Used
Generalized Gradient	Traveling Trajectory	Rarely Used

The survey that follows focuses on features of methods for solving continuous, constrained problems. Whenever it is possible, we analyze the balance that each algorithm strikes between global search and local refinement, and relate this balance to its performance. Table 3.1 summarizes this balance for a number of popular global optimization methods [262].

3.1.1 Deterministic Global Optimization Methods

Many deterministic methods have been developed in the past. Some of them apply deterministic heuristics, such as modifying the search trajectory in trajectory methods and adding penalties in penalty-based methods, to bring a search out of a local minimum. Other methods, like branch-and-bound and interval methods, partition a search space recursively into smaller subspaces and exclude regions containing no optimal solution. These methods do not work well when the search space is too large for deterministic methods to cover adequately.

(a) **Covering methods**. Covering methods detect subregions not containing the global minimum, and exclude them from further consideration. Covering methods provide guarantee of solution quality, and approximate the global solution by iteratively using tighter bounds [72, 111, 125, 172]. Obtaining a solution with guaranteed accuracy implies an exhaustive search of the solution space for the global minimum. Hence, these methods can be computationally expensive, with a computation time that increases dramatically as the problem size increases [14,15].

Branch-and-bound methods, including interval methods, are examples of covering methods. Branch-and-bound methods evaluate lower bounds on the objective function of subspaces. They allow an assessment of the quality of the local minima obtained. Combining with computationally verifiable sufficient conditions for global optimality, they allow one to actually prove global optimality of the best solution obtained.

Covering methods are reliable since, to the extent they work, they have built-in guarantees of solution quality. However, they require some global properties of the optimization problem, such as the Lipschitz condition. In the worst case, covering methods take an exponential amount of work. Many of the heuristic techniques used for searching global solutions can be adapted to or combined with branch-and-bound approach to take advantage of structural insights of specific applications.

(b) **Generalized descent methods**. These methods continue the search trajectory every time a local solution is found. There are two approaches. First, trajectory methods modify the differential equations describing the local-descent trajectory so that they can escape from local minima [5,65,215,236,247,260]. Their disadvantage is the large number of function evaluations spent in unpromising regions. Second, penalty methods prevent multiple determination of the same local minima by modifying the objective function, namely, by introducing a penalty term on each local minimum [42,88]. Their problem is that as more local minima are found, the modified objective function becomes more difficult to minimize. In existing generalized descent methods, the descent trajectory is modified using internal function information, e.g. local minima along the search.

We propose a new global search method, the trace-based method, in this chapter. Our trace-based method belongs to the class of generalized descent methods because it modifies the descent trajectory. The difference between it and existing trajectory methods is that our trace-based method uses external information to modify the search trajectory. A function-independent trace function is used to guide the search throughout the search space, and pull the search out of local minima.

Alternatively, deterministic methods can be classified into two categories: (a) point-based methods – methods that compute function values at sampled points, such as generalized descent methods, and (b) region-based methods – methods that compute function bounds over compact sets, such as covering methods. Point-based methods are unreliable, but usually have less computational complexity. Region-based methods are expensive, but can produce rigorous global optimization solutions when they are applicable.

3.1.2 Probabilistic Global Optimization Methods

Probabilistic global minimization methods rely on probability to make decisions. The simplest probabilistic algorithm uses restarts to bring a search out of a local minimum when little improvement can be made locally. Advanced methods use more elaborate techniques. We classify probabilistic methods into clustering methods, random-search methods, and methods based on stochastic models.

(a) **Clustering methods**. Clustering analysis [30, 251, 252] is used to prevent the re-determination of already known local minima. There are two strategies for grouping points around a local minimum: (i) retain only points with relatively low function values [152]; (ii) push each point toward a local minimum by performing a few steps of a local search [250]. They do not work well when the function terrain is very rugged or when the search gets trapped in a deep but suboptimal valley [187,218].

(b) **Random search methods**. These include pure random search, single-start [192,281, 282], multi-start [115,212,218,244], random line search, adaptive random search, partitioning into subsets, replacing the worst point, evolutionary algorithms [85, 95, 164], and simulated annealing [1, 41, 144, 159, 195, 259].

Simulated annealing and genetic algorithms are two popular stochastic global optimization methods. Simulated annealing takes its intuition from the fact that heating and slowly cooling (annealing) a piece of metal brings it into a more uniformly crystalline state, which is believed to be the state where the free energy of bulk matter takes its global minimum. The role of temperature is to allow the configurations to reach lower energy states with a probability given by Boltzmann's exponential law, so that they can overcome energy barriers that would otherwise force them into local minima. Simulated annealing is provably convergent asymptotically in a probabilistic sense, but may be exceedingly slow. Various ad hoc enhancements make it much faster. Simulated annealing has been successfully applied to solve many nonlinear optimization problems [2,53,144,208,259].

Genetic algorithms make use of analogies to biological evolution by allowing mutations and crossovers between candidates of good local optima in the hope to derive even better ones. At each stage, a whole population of configurations are stored. Mutations are performed as local search, whereas crossover operators provide the ability to leave regions of attraction of local minimizers. With high probability, the crossover rules produce offsprings of similar or even better fitness. The efficiency of a genetic algorithm depends on the proper selection of crossover rules. The effect of interchanging coordinates is beneficial mainly when these coordinates have a nearly independent influence on the fitness, whereas if their influence is highly correlated, such as for functions with deep and narrow valleys not parallel to the coordinate axes, genetic algorithms have more difficulties. Successful tuning of genetic algorithms requires a considerable amount of insight into the nature of the problem at hand. Genetic algorithms have shown promising results in solving nonlinear optimization problems [92,98,121,164].

Random search methods are easy to understand and simple to realize. The simplest random algorithm uses restarts to bring a search out of a local minimum. Others, like simulated annealing, rely on probability to indicate whether a search should ascend from a local minimum. Other stochastic methods rely on probability to decide which intermediate points to interpolate as new starting points, like in random recombinations and mutations in genetic algorithms. These algorithms are weak in either their local or their global search. For instance, gradient information useful in local search is not used well in simulated annealing and genetic algorithms. In contrast, gradient-descent algorithms with multi-starts are weak in global search. These methods perform well for some applications. However, they usually have many problem-specific parameters, leading to low efficiency when improperly applied [24,25,28,29].

(c) **Methods based on stochastic models**. Most of these methods use random variables to model unknown values of an objective function. One example is the Bayesian method, which is based on a stochastic function and minimizes the expected deviation of the estimate from the real global minimum [168, 169, 261]. Bayesian methods do not work well because most of the samples they collect randomly from the error surface are close to the average error value, and these samples are inadequate to model the behavior at minimal points. Other methods based on stochastic models include methods that approximate the level sets. Although very attractive theoretically, this class of methods are too expensive to be applied to problems with more than twenty variables [253].

Up to today, general nonlinear optimization algorithms can at best find good local minima of a multi-modal optimization problem. Only in cases with restrictive assumptions, such as Lipschitz condition, algorithms with guaranteed accuracy can be constructed.

Existing global optimization methods are weak in either their local or their global search. To address this problem, we have developed a new global search method consisting of a combination of global and local searches. Our method has the following key features of a good global search algorithm:

- It identifies promising regions efficiently.
- It uses gradient information to descend into local minima, and is able to adjust to changing gradients.
- It escapes from a local minimum once the search gets there. It leaves regions of attraction of local minimizers in a continuous fashion without restarting from one local minimum to another.
- It avoids redetermination of the same local minima.

Our method is based on a new trace-based global-search method, in which a deterministic external force guides the search throughout the space and pulls the search trajectory out of local minima once it gets there. This method is presented in Section 3.4 of this chapter.

3.1.3 Global Search in Constrained Optimization

At the level of global search, strategies for solving constrained problems are similar to those for solving unconstrained problems except in the handling of constraints. Deterministic and probabilistic global search methods have been applied to solve constrained optimization problems in conjunction with both non-transformational and transformational constraint-handling approaches.

When constraints are handled using transformational approaches, constrained optimization problems are transformed into one or a series of unconstrained optimization problems. Global search methods, either deterministic or probabilistic, can then be applied to solve the transformed problems. For example, multi-start of gradient descent, genetic algorithms (GAs), and simulated annealing (SA) have been applied to solve penalty or barrier functions [53, 129, 195, 199]. In Lagrangian methods, multi-starts of local searches find saddle points from randomly generated initial points.

Many global search methods also work with non-transformational approaches of handling constraints. For example, in pure random search with rejecting/discarding of infeasible points, points in the search space are randomly generated according to some probability distribution, and the best feasible point found is remembered. This method performs a blind search through the search space with no local refinement. In multi-starts, points (feasible points if possible) in the search space are randomly generated and are used as initial points for local improvement. When working directly on constraints, GAs maintain a population of sample points, and generate new sample points based on both constraint satisfaction and objective values of existing sample points [164]. SA performs an adaptive stochastic search, and the acceptance rates of search points are determined by a combination of constraint satisfaction and objective value [32, 208, 259].

Our new global search method handles constraints using the transformational approach. Constrained optimization problems are transformed into Lagrangian functions using Lagrange multipliers. Saddle points of Lagrangian functions are, then, searched using local and global search methods.

3.2 Problem Specification and Transformation

In the rest of this chapter, we present our new nonlinear global search method. This method solves general nonlinear continuous and discrete, constrained and unconstrained optimization problems. By first transforming nonlinear constrained optimization problems using Lagrange multipliers into unconstrained optimization formulations, both constrained and unconstrained problems are solved in a unified framework.

As defined in Chapter 1, an n-dimensional unconstrained optimization problem is

$$\min_{\mathbf{x} \in R^n} f(\mathbf{x}), \qquad \mathbf{x} = (x_1, x_2, \cdots, x_n)^T \tag{3.1}$$

while $\mathbf{x}$ is an n-dimensional vector, $f(\mathbf{x})$ is the objective function, and T stands for transpose. The gradient of function $f(\mathbf{x})$, $\nabla f(\mathbf{x})$, is an n-dimensional vector as follows:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \cdots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right)^T \tag{3.2}$$

An n-dimensional constrained optimization problem is defined as

$$\begin{aligned} \min_{\mathbf{x} \in R^n} \quad & f(\mathbf{x}) && (3.3) \\ \text{s.t.} \quad & h_i(\mathbf{x}) = 0 \quad i = 1, \cdots, m_1 && (3.4) \\ & g_j(\mathbf{x}) \le 0 \quad j = 1, \cdots, m_2 && (3.5) \end{aligned}$$

where there are m_1 equality constraints and m_2 inequality constraints.

Constrained optimization problems are transformed into Lagrangian formulations. Inequality constraints are handled using the slack-variable method or the MaxQ method as discussed in Chapter 2. Using the slack-variable method, the corresponding augmented Lagrangian function is

$$\begin{aligned} L(\mathbf{x}, \lambda) \quad = \quad & f(\mathbf{x}) + (\lambda_1, \cdots, \lambda_{m_1})^T h(\mathbf{x}) + \frac{1}{2} ||h(\mathbf{x})||_2^2 && (3.6) \\ & + \frac{1}{2} \sum_{j=1}^{m_2} [\max{}^2(0, \lambda_{m_1+j} + g_j(\mathbf{x})) - \lambda_{m_1+j}^2] \end{aligned}$$

where $\lambda \in R^m, m = m_1 + m_2$, are Lagrange multipliers. Using the MaxQ method, the augmented Lagrangian function has the following form:

$$\begin{aligned} L(\mathbf{x}, \lambda) &= f(\mathbf{x}) + (\lambda_1, \cdots, \lambda_{m_1})^T h(\mathbf{x}) + \frac{1}{2}||h(\mathbf{x})||_2^2 \\ &\quad + \sum_{j=1}^{m_2} \lambda_{m_1+j} \max^{q_j}(0, g_j(\mathbf{x})) + \frac{1}{2}\sum_{j=1}^{m_2} \max^{2q_j}(0, g_j(\mathbf{x})) \end{aligned} \tag{3.7}$$

Lagrangian function $L(\mathbf{x}, \lambda)$ is a scalar function with gradient $\nabla L(\mathbf{x}, \lambda)$:

$$\begin{aligned} \nabla L(\mathbf{x}, \lambda) &= (\nabla_{\mathbf{x}} L(\mathbf{x}, \lambda), \nabla_{\lambda} L(\mathbf{x}, \lambda)) \\ &= \left(\frac{\partial L(\mathbf{x}, \lambda)}{\partial x_i}, \frac{\partial L(\mathbf{x}, \lambda)}{\partial \lambda_j} \right) \qquad i = 1, \cdots, n \text{ and } j = 1, \cdots, m \end{aligned} \tag{3.8}$$

Our method is a first-order method because only first-order derivatives as well as the objective and the Lagrangian-function value are used during the search.

In Figure 1.4, we have shown how various forms of nonlinear optimization problems are solved in a unified framework. Unconstrained optimization problems are solved directly by our global search method without any transformation.

Constrained optimization problems, whether discrete or continuous, are first transformed into Lagrangian formulations, in which the constraints are combined with the objective function using Lagrange multipliers. For a decision problem without an objective function, an artificial objective is introduced. Hence, our global search method works on the Lagrangian function and finds saddle points that correspond to globally optimal or near-optimal solutions of the original constrained problems.

3.3 A New Multi-level Global Search Method

In this section, we propose a new global search method with the focus of overcoming local minima in a unique, deterministic, and continuous fashion.

Our method employs a combination of global and local searches. A global search is applied to locate promising regions in which local searches are invoked to find the local

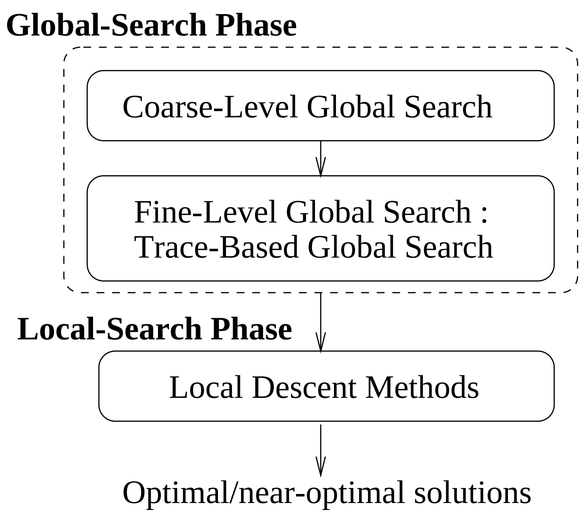

Global-Search Phase
Coarse-Level Global Search
Fine-Level Global Search : Trace-Based Global Search
Local-Search Phase
Local Descent Methods
Optimal/near-optimal solutions

Figure 3.3: Framework of a new multi-level global search method. This method has a combined global and local search phases.

minima. The global search is further divided into coarse-level and fine-level global search. A coarse-level global search quickly identifies promising sub-spaces in a large high-dimensional search space. Then, a fine-level global search applies a trace-based method to perform more detailed search. The trace-based global-search method uses a problem-independent trace function to guide the search throughout the search space and pulls a search out of a local optimum (a local saddle point in a constrained optimization and a local minimum in an unconstrained optimization) without having to restart it from a new starting point. Promising local minimal regions are located by the fine-level global search. Starting from initial points provided by the fine-level global search, a local search finds local minima that contain globally optimal or near-optimal solutions. Our method strives a balance between global and local searches.

Figure 3.3 shows the overall framework of our global search method. It consists of a global-search phase and a local-search phase. In the global-search phase, a coarse-level global search is followed by a fine-level global search. The search space of a high-dimensional optimization problem can be enormous, making it impossible to examine the whole space exhaustively. Usually, a small portion of the search space can be searched, given a reasonable amount of time. The task of coarse-level global search is to quickly identify promising sub-regions in a large high-dimensional search space. Outputs of the coarse-level search are used as initial points in the fine-level global search.

Existing global search methods, such as simulated annealing (SA), genetic algorithms, and multi-start of descents, are good at finding promising sub-regions given a limited amount of time. We apply them in the coarse-level search. In applying SA, we impose a fast annealing scheme and a limited number of iterations in order for SA to converge quickly. In applying GA, we set a limit on the number of generations and select the best solutions in the final population as outputs. In applying multi-start of descents, we perform a limited number of descent steps from randomly generated sample points and select the best descent results as outputs.

As an alternative to random multi-starts, we have developed a trajectory-based method to perform a coarse-level search. It is based on a 1-D parametric function that traverses inside the search space. Sample points along the trajectory with small objective values are used as starting points to perform limited steps of descents. The best solutions after descents are output.

Besides search methods, domain knowledge is another alternative to help identify promising search regions. For well understood problems, users may provide good initial points directly for the fine-level global search. Heuristic methods, such as greedy search methods, can also be applied to generate initial points for the fine-level global search.

Starting from the initial points provided by the coarse-level search, the fine-level global search examines sub-regions in more details. We apply an innovative trace-based global search method (to be presented next) that is a variation of the generalized descent method. In contrast to existing trajectory-based global search methods, our trace-based method modifies the search trajectory using external information. The search trajectory is formed by a combined force of a problem-independent trace function and local gradient. The trace function is used to guide the search throughout the search space and pulls the search out of a local optimum without having to restart it from a new starting point. Thus, our trace-based search method overcomes local minima in a continuous and deterministic manner. Along the search trajectory, points corresponding to promising local minimal regions are fed to the local-search phase.

The local-search phase performs local descents starting from initial points provided by the global-search phase. Existing local-descent methods, including gradient descent, quasi-Newton's methods, and conjugate gradient methods, are applied. The best solutions after local descents are reported as the final solutions.

3.4 Trace-based Global Search

Our trace-based global search method relies on an *external* force to pull the search out of a local minimum, and employs local descents to locate local minima. It has three features: exploring the solution space, locating promising regions, and finding local minima. In exploring the solution space, the search is guided by a continuous terrain-independent trace function that does not get trapped by local minima. This trace function is usually an aperiodic continuous function that runs in a bounded space. In locating promising regions, the trace-based method uses local gradient to attract the search to a local minimum but relies on the trace to pull it out of the local minimum once little improvement can be found. Finally, the trace-based method selects one initial point for each promising local region and uses them as initial points for a descent algorithm to find local minima.

3.4.1 General Framework

Figure 3.4 illustrates the process of our trace-based global search. The search process uncovers promising local optimal regions without going deep into each one. A problem-independent trace function leads the search throughout the search space and is responsible for pulling the search out of local minima once it gets there. Local minima unveil themselves through gradients. The global search trajectory is formed by the combination of two forces: a problem-independent force provided by the trace function and a problem-dependent force provided by the local gradient. These two counteracting forces (descents into local minima and attraction exerted by the trace) form a composite vector that represents the route taken

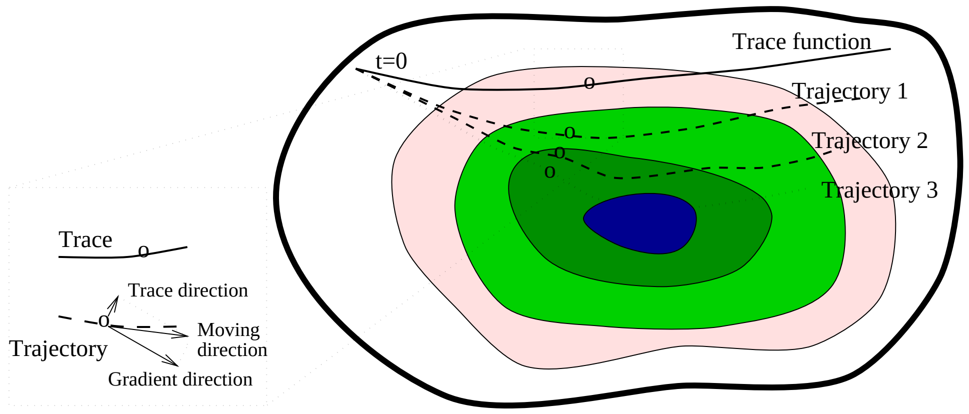


Figure 3.4: In the trace-based global search, the search trajectory shows a combined effect of gradient descents and pull exerted by the moving trace. Three cascaded global search stages are shown in this figure that generate trajectory 1, 2, and 3, respectively.

by the trajectory. Figure 3.4 shows on the left side how these two forces are combined to form the search trajectory.

Generally, there could be a number of bootstrapping stages in a trace-based global search. One stage is coupled to the next stage by feeding its output trajectory as the trace of the next stage. The problem-independent trace is fed into the first stage. Three cascaded global-search stages are shown in Figure 3.4. The trace and the local gradient generate trajectory 1. Then trajectory 1 and the local gradient generate trajectory 2, and so on.

In the trace-based global search, the dynamics of each stage in the global-search phase is characterized by a system of ordinary differential equations (ODEs), which contain both problem-dependent information of local gradient and problem-independent information of a trace function.

Consider the optimization problem specified in (3.1). The system of ODEs that characterize each stage of the trace-based global search has the following general form:

$$\dot{\mathbf{x}}(t) = P(\nabla_{\mathbf{x}} f(\mathbf{x}(t))) + Q(T(t), \mathbf{x}(t)) \tag{3.9}$$

where t is the only independent variable in the systems of ODEs; $T(t)$, the trace function, is a function of t; and P and Q are some nonlinear functions. $P(\nabla_{\mathbf{x}} f(\mathbf{x}))$ enables the gradient to attract the trajectory to a local minimum. $Q(T(t), \mathbf{x}(t))$ allows the trace function to lead the trajectory. Solving the system of ODEs as an initial-value problem from a given set of initial values of $\mathbf{x}$ generates a search *trajectory* $\mathbf{x}(t) = (x_1(t), \cdots, x_n(t))$.

P and Q can have various forms. A simple form used in our experiments is

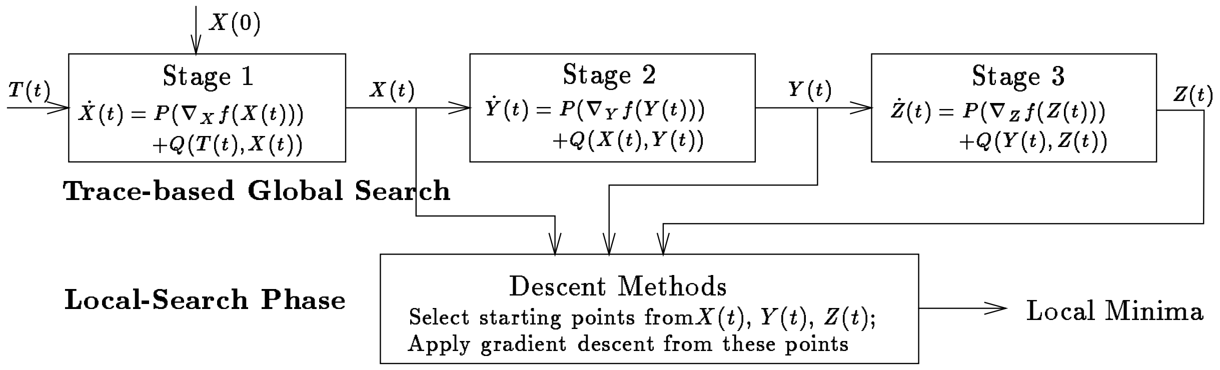
$$\frac{d\mathbf{x}(t)}{dt} = -\mu_g \, \nabla_{\mathbf{x}} f(\mathbf{x}(t)) - \mu_t \, (\mathbf{x}(t) - T(t)) \tag{3.10}$$

where μ_g and μ_t are coefficients specifying the weights for descents in local minimum regions and for exploration of broader space. Note that the first component on the right-hand side of (3.10) is actually gradient descent, and the second component forces $\mathbf{x}(t)$ to follow the trace function $T(t)$.

Generally, weights μ_g and μ_t can have different values in different global stages. For example, μ_t can be set to have large values relative to μ_g in the first stage so that the global search is emphasized. In later stages, μ_g can have larger values, and the search is focused on a local region. In the simplest case, μ_g and μ_t are set to constants and are the same in all stages of the global-search phase.

When there are several stages in the trace-based global search, one stage is coupled to the next stage by feeding its output trajectory as the trace function of the next stage, with a predefined problem-independent function as the input trace function of the first stage. The stages are cascaded together. Each stage is characterized by a system of ODEs in the form of (3.9), but with different $T(t)$. $T(t)$ in the first stage is an external trace function. $T(t)$ of later stages is the output trajectory $\mathbf{x}(t)$ of the previous stage. In our implementation, when the output trajectory is a set of discrete sample points of the continuous trajectory, interpolation is performed to form a continuous $T(t)$ for each stage.

Figure 3.5 shows a three-stage trace-based global-search method. In general, the functions P and Q in each stage of the trace-based method can be different. In earlier stages, more weight can be placed on the trace function, allowing the resulting trajectory to explore more regions. In later stages, more weight can be placed on local descents, allowing the trajectory to descend deeper into local basins. Note that all the equations in the trace-based global



$X(0)$
$T(t)$
Stage 1
$\dot{X}(t) = P(\nabla_X f(X(t))) + Q(T(t), X(t))$
$X(t)$
Stage 2
$\dot{Y}(t) = P(\nabla_Y f(Y(t))) + Q(X(t), Y(t))$
$Y(t)$
Stage 3
$\dot{Z}(t) = P(\nabla_Z f(Z(t))) + Q(Y(t), Z(t))$
$Z(t)$
Trace-based Global Search
Local-Search Phase
Descent Methods
Select starting points from $X(t)$, $Y(t)$, $Z(t)$;
Apply gradient descent from these points
Local Minima

Figure 3.5: Framework of a three-stage trace-based global search method.

search can be combined into a single equation before being solved. We did not do so because each trajectory may identify new starting points that lead to better local minima.

In the local-search phase, a traditional descent method, such as gradient descent, conjugate gradient, or Quasi-Newton's methods, is applied to find local minima. Initial points for the local search are selected based on trajectories output by the trace-based global search. Two heuristics can be applied in selecting initial points: use the best solutions in periodic time intervals as initial points, or use the local minima in the trajectory in each stage as initial points. As shown in Figure 3.5, starting points for the local search are selected from search trajectories $\mathbf{x}(t)$, $\mathbf{y}(t)$, and $\mathbf{z}(t)$ of the global-search phase. Descent methods are applied to find local minima starting from these initial points.

We have presented the formulation of the our trace-based search method for unconstrained problems. Its formulation for constrained problems are similar. Our trace-based method solves constrained problems based on Lagrangian functions. Traditional Lagrangian methods perform local search by doing descents in the original-variable space and ascents in the Lagrange-multiplier space. Local search converges to a saddle point that corresponds to a local minimum of the original constrained problem. Our trace-based search modifies the descent trajectory in the original-variable space and uses a trace function to pull the search out of local minima.

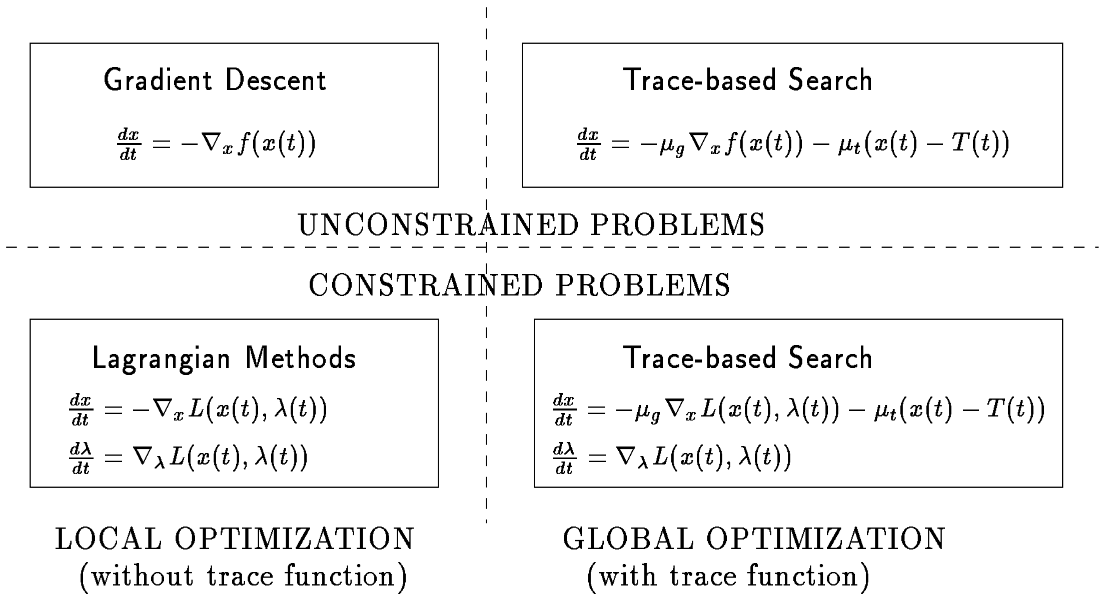

Gradient Descent
$\frac{dx}{dt} = -\nabla_x f(x(t))$
Trace-based Search
$\frac{dx}{dt} = -\mu_g \nabla_x f(x(t)) - \mu_t (x(t) - T(t))$
UNCONSTRAINED PROBLEMS
CONSTRAINED PROBLEMS
Lagrangian Methods
$\frac{dx}{dt} = -\nabla_x L(x(t), \lambda(t))$
$\frac{d\lambda}{dt} = \nabla_\lambda L(x(t), \lambda(t))$
Trace-based Search
$\frac{dx}{dt} = -\mu_g \nabla_x L(x(t), \lambda(t)) - \mu_t (x(t) - T(t))$
$\frac{d\lambda}{dt} = \nabla_\lambda L(x(t), \lambda(t))$
LOCAL OPTIMIZATION
(without trace function)
GLOBAL OPTIMIZATION
(with trace function)

Figure 3.6: Relationship between the trace-based global search and the local optimization method.

In comparison to the formulation (3.10) for an unconstrained problem, the system of ODEs specifying a global-search stage for a constrained problem is as follows:

$$\frac{d\mathbf{x}(t)}{dt} = -\mu_g \nabla_{\mathbf{x}} L(\mathbf{x}(t), \lambda(t)) - \mu_t (\mathbf{x}(t) - T(t)) \tag{3.11}$$

$$\frac{d\lambda(t)}{dt} = \nabla_\lambda L(\mathbf{x}(t), \lambda(t)) \tag{3.12}$$

where μ_g and μ_t are weight coefficients, $L(\mathbf{x}(t), \lambda(t))$ is the Lagrangian function, and $T(t)$ is the trace function. A trace function is used in the original variable space to overcome the local minima of the nonlinear objective. No trace function is necessary in the Lagrange-multiplier space, because the values of Lagrange multipliers change adaptively to reflect the degree and duration of constraint violation during the search. A trace function in the Lagrange-multiplier space breaks the relationship between a search point in the original-variable space and its corresponding Lagrange-multiplier values, which makes Lagrangian methods not work well.

In Figure 3.6, we summarize the mathematical formulations of the trace-based search method for unconstrained and constrained continuous problems, and show its relationship to local optimization methods. This figure indicates that our trace-based search is an extension of existing local search methods, and performs global search.

3.4.2 An Illustrative Example

Let's illustrate the trace-based global search using a simple example based on Levy's No. 3 problem [156]. This example involves finding the global minimum of a function of two variables x_1, x_2 in a bounded region. The function is

$$f_{l_3}(x_1, x_2) = \sum_{i=1}^{5} i\, cos[(i-1)x_1 + i] \sum_{j=1}^{5} j\, cos[(j+1)x_2 + j]. \tag{3.13}$$

Figure 3.7 shows the 3-D plots and 2-D contour plots of this function and the search trajectories of the trace-based global search. The bounded region of interest is $[-1, 1]$ in each dimension. The function has three local minima in this region, one of which is the global minimum. Using a search range of $[-1, 1]$ in each dimension, we start Novel from (0, 0) and run it until logical time $t = 5$.

Figure 3.7 shows the trace function and search trajectories of a three-stage trace-based global search. The plots in the first column show the trace function mapped on the 3-dimensional terrain and the 2-dimensional contour. Although the trace function visits all three basins, it only touches the edge of the basin containing the global minimum. The plots in the second column show the trajectory generated from stage 1. The search trajectory is pulled down toward the bottom of each local minimal basin. Likewise, the plots in the third and fourth columns show, respectively, that the trajectories are further pulled down to the local basins after stages 2 and 3. By following the trajectories, three basins with local minima are identified, and a set of minimal points in each trajectory can be used as initial points in the local-search phase. Clearly, all the local minima are identified in the global-search phase.

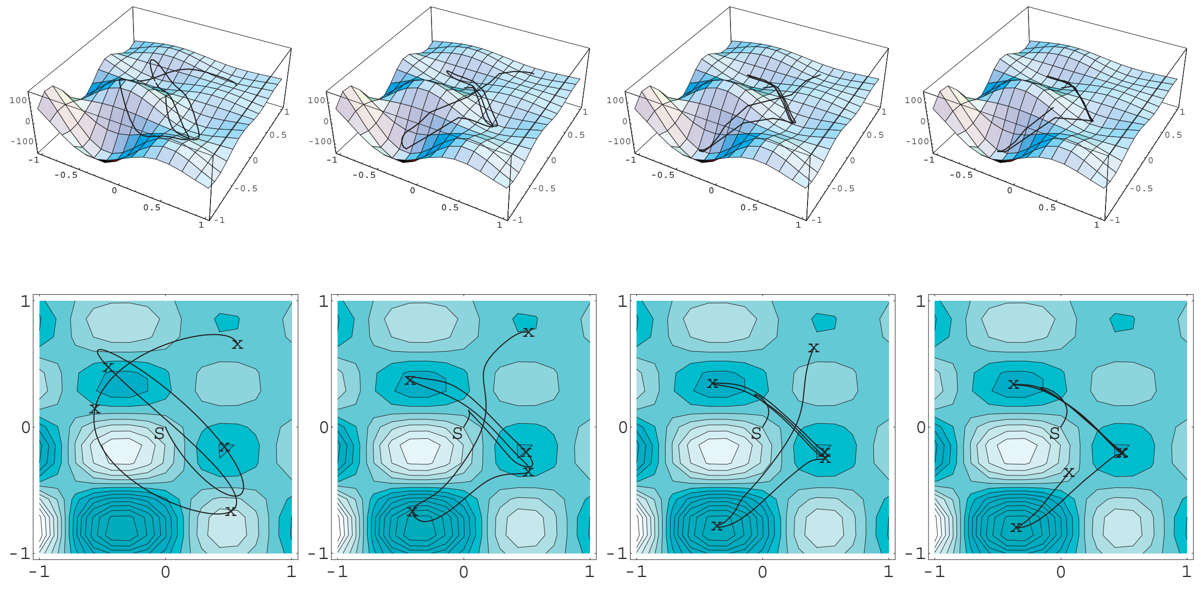


Figure 3.7: Illustration of the trace-based global search. 3-D plots (top row) and 2-D contour plots (bottom row) of Levy's No. 3 function with superimposed search trajectories: trace (first column), trajectory after stage 1 (second column), trajectory after stage 2 (third column), and trajectory after stage 3 (fourth column). Darker regions in the contour plot mean smaller function values. The trajectories are solved by LSODE, a differential-equation solver.

3.5 Trace Function Design

In a trace-based global search, a problem-independent trace function leads the search throughout the search space. To find the global minima efficiently without any knowledge of the terrain, a trace function should cover the search space well and guides the search in an efficient way.

A trace function should be continuous and cover the search space uniformly. In real-world applications, some physical time constraints are often imposed in solving a problem. Hence, it is desirable to continuously improve the best solution found when given more time.

Given the desired behavior of the trace function, we have developed some quantitative performance measures of the quality of a trace function.

3.5.1 Evaluation of Coverage

Quantitative measurements of the coverage of a trace function in a high-dimensional space are necessary in evaluating its performance. A trace function is a one-dimensional curve. We want to find a one-dimensional curve that covers a high-dimensional space well. In this section, we present the coverage measurement we have studied, and compare various trace functions based on it.

One possible distance measurement of a point i in the high-dimensional space S to a curve C is

$$\min_{j \in C} D_{ij}$$

where j is some point on the curve C, and D_{ij} is the Euclidean distance between j and i.

Coverage is evaluated based on the distances of all points in the space to the curve. One measurement is the distance distribution. Another one is the maximum distance using the following max-min function,

$$\xi = \max_{i \in S} \min_{j \in C} (D_{ij}) \tag{3.14}$$

Here, ξ is the radius of the largest sphere inside the high-dimensional space that curve C does not pass through. Accordingly, designing a curve that has the maximal coverage of a high-dimensional space is to minimize ξ.

Often there is no closed-form formula to evaluate the distance distribution or the maximum distance ξ for a given curve C. Numerical evaluation is possible, but usually expensive. Since the curve can have any shape, if we evaluate the distance of every point in the high-dimensional space to the curve, the computation would become intractable very quickly. To find the point that has the largest distance from a curve is also not trivial.

To make the evaluation of coverage tractable, we turn to approximate measurements. Instead of evaluating the distance of every points in the space to the curve, we divide the space into sub-spaces and measure the distance of each sub-space to the curve in terms of the number of sub-spaces in between. The Euclidean distance from the center of a sub-space to the curve is approximated by the multiplication of the number of sub-spaces in between and the size of a sub-space.

Using approximate measurement, the procedure to evaluate the coverage of a curve in a space is as follows. First, the high-dimensional space is uniformly partitioned into smaller sub-spaces. A sub-space is said to be covered if the curve passes through the sub-space, and has distance 0 from the curve. Sub-spaces contiguous to sub-space of distance 0 have distance 1. Sub-spaces contiguous to sub-space of distance 1 have distance 2, and so on. The evaluation proceeds like a waveform, starting from sub-spaces intersected with the curve and propagating to sub-spaces further away. Based on the distances of all sub-spaces, we can evaluate the coverage of a curve by using either the maximal distance or the distribution.

Figure 3.8 illustrates the process of finding the distances of sub-spaces to a given curve. The space is a 2-D square and is divided into 8x8 sub-squares. Then, we mark the sub-spaces intersecting with the curve as having distance 0. The sub-spaces contiguous to those with 0 distance are marked with distance 1, and so on. Eventually, we find the distances of all sub-spaces, which is the least number of sub-spaces between this sub-space and the nearest point of the curve. The distance is called *unit distance.*

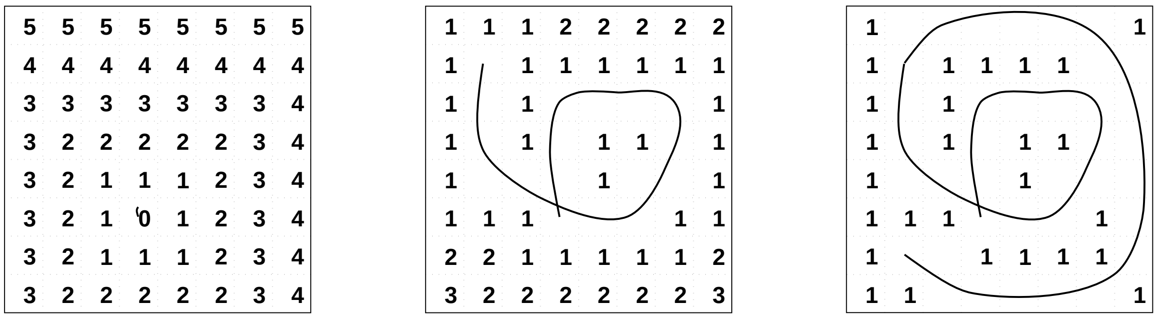


Figure 3.8: Illustration of the distance evaluation in a 2-D square and several curves with different lengths. The left graph contains a very short curve. The curve is extended in the middle graph. The right graph have the longest curve and the best coverage. The number in each sub-square is the shortest unit distance from the sub-square to the curve.

The left graph in Figure 3.8 shows the unit distance of each sub-square to a very short curve inside the sub-square with distance of 0. Sub-squares contiguous to 0-distance sub-squares have distance 1. Sub-squares on the top row have the largest distance of 5. In the middle graph, a longer curve covers more regions of the space. Sub-squares are marked by their unit distance to the curve as shown in the graph. Distances of 0 are omitted from the graph. The maximum unit distance of 3 is possessed by the two sub-squares at the lower left and right corners. In the graph on the right, the curve is further extended and covers even more regions of the space. Now, the maximum unit distance is 1. This curve achieves better coverage than the one in the middle.

After getting the distance information of sub-spaces, we can then evaluate the coverage of a curve. One metric is the maximal distance, which corresponds to the value ξ in formula (3.14). However, the single value may not tell the whole story of the coverage. Two curves with the same maximal distance could have very different coverage. A better evaluation method is to look at the distribution of sub-space distances, and use a set of points from the distribution, such as the 10, 50, or 90 percentile, as evaluation metrics.

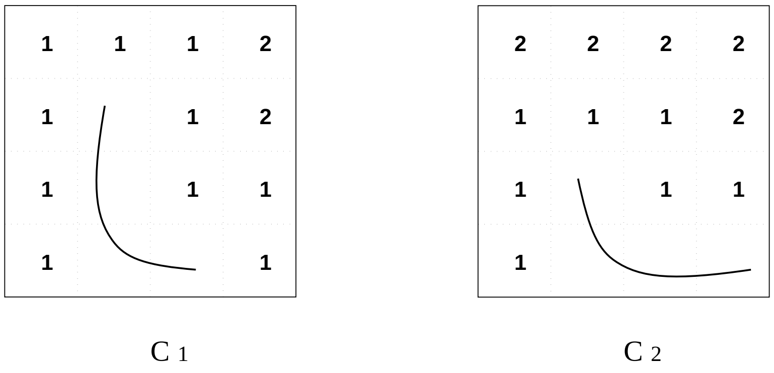


Figure 3.9: Two equal-length curves C_1 and C_2 with the same maximal distance of sub-spaces, but with different distance distributions.

For example, Figure 3.9 shows two curves C_1 and C_2 of the same length in a 2-D space. The space is partitioned into 16 sub-squares. For both curves, the maximal distance of sub-squares is 2. Their distance distributions of sub-squares are listed in the following table.

Distance to curve		0	1	2
Number of sub-squares	C_1	4	10	2
	C_2	4	7	5

From the distribution, we know that curve C_1 has better coverage.

The accuracy of the approximate measurement based on sub-spaces depends on the size of a sub-space. When the space is divided more finely and each sub-space is smaller, the approximation can have high accuracy, but at the expense of more computations. For an n-dimensional space, if the space is divided into half along each dimension, there will be 2^n sub-spaces. If each dimension is divided into k pieces, then there will be k^n sub-spaces. The computational cost of evaluating distances of all sub-spaces is proportional to the number of sub-spaces. Therefore, the computational cost increases rapidly as the space is divided into smaller pieces. For a high-dimensional space, the computational complexity is exponential and prohibitive even for a moderate division of the space. Hence, to evaluate coverage in high-dimensional spaces, we apply sampling techniques. A large number of sub-spaces are sampled to give a good estimation of the actual distance distribution of sub-spaces. We found out that 1000 samples are usually good enough.

Based on the coverage evaluation method we have described, we then study the coverage of various trace functions.

3.5.2 Space Filling Curves

There are two alternative ways to search a space: (a) divide the space into sub-spaces and search sub-spaces extensively one by one; and (b) search the space from coarse to fine.

The first is a divide-and-conquer approach. One example is to use space filling curves. Space filling curves have been studied by mathematicians since Peano introduced them in 1890 [193]. A space filling curve is any continuous and non-overlapping set of line segments which completely fills a subset of high-dimensional space. In the 2-dimensional case, a space filling curve is a one-dimensional curve that fills a plane and leaves no space free. One example is the so-called "snake curve" shown on the left of Figure 3.10. The first two Peano curves, P_1 and P_2, are also plotted in Figure 3.10. Peano curves are generated recursively. Curve P_i is generated from curve P_{i-1} by duplicating P_{i-1} nine times, translating and rotating each duplicate appropriately, and joining the ends.

Peano curves are generated based on the base-3 number system. Hilbert [118] produced space filling curves based on the even-number system. The first three Hilbert curves of degree 2 are shown in Figure 3.11. Curve H_1 has four vertices at the center of each quarter of the unit square. Curve H_2 has 16 vertices, each at the center of a sixteenth of the unit square. The Hilbert space filling curve is the limit of the sequence of curves and visits every point within a two-dimensional space. Generally, for an n-dimensional space, the jth Hilbert curve is constructed from n^2 copies of the *j-1*st Hilbert curve. The true space filling curve is the one that the order goes to the limit of infinity.

A fundamental property of space filling curves is that a space filling curve never intersects itself. To completely fill a plane, space filling curves have to turn over themselves a great number of times. They often provide a convenient means of exploring the space around a

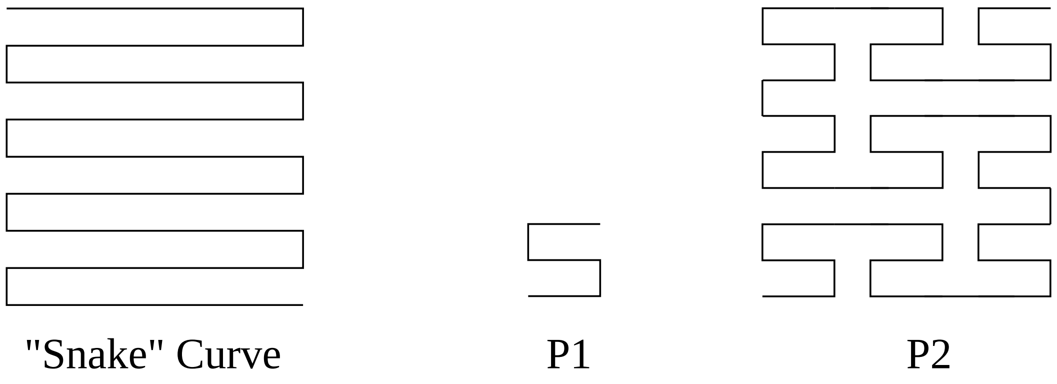


Figure 3.10: Space filling curves: snake curve (left) and the first two Peano curves, P_1 and P_2, (right).

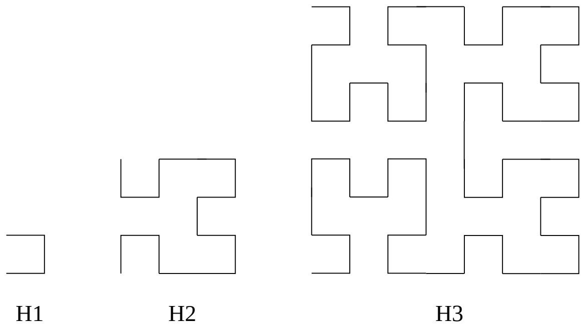


Figure 3.11: The first three Hilbert space filling curves of degree 2.

specific point. Space filling curves have been applied in computer graphics and databases as a basis for multi-attribute indices.

One disadvantage of using a space filling curve as a trace to guide the search is that the appropriate degree of the space filling curve needs to be decided before the search starts. That gives the granularity in which the search covers the space. Then, the search traverses subregions one by one. This type of search doesn't provide a continuous improvement of solution quality as time goes by. Without prior knowledge of the search terrain, the search process may spend a lot of time in some unpromising subregions before finally going to regions with good solutions. In that situation, the quality of the solution is poor and will not be improved for a long period of time.

3.5.3 Searching from Coarse to Fine

The second approach is to search the space from coarse to fine. The more time there is available, the finer details the search discovers. The advantage of this approach is the continuous improvement of solution quality. The trace function of this approach may intersect itself and does not cover the space as uniformly as space filling curves do. However, this issue is not serious when the curve is used to search a high-dimensional space and the time available is limited.

We have designed a family of trace functions that use this approach. These trace functions are aperiodic and do not repeat themselves. They are parametric functions of one independent variable, are designed based on the sinusoidal function, have a bounded range, and move in a cyclic mode. To make a sinusoidal function aperiodic, we put a non-integer power on the independent variable. The basic function looks like the following:

$$\sin(2\pi t^{\alpha}), \tag{3.15}$$

where $\alpha, 0 < \alpha < 1$, controls the cycle frequency. The larger the α is, the higher the frequency is. With an aperiodic sinusoidal function of different α in each dimension, a trace function traverses the space in an aperiodic fashion and does not repeat itself. By having an appropriate α for each dimension, the curve can cover the space approximately in a uniform way.

In our trace-based global-search method, a dynamic system containing the trace function and the optimized function information evolves on a pseudo-time scale. A parametric trace function with one automatic variable can be incooperated into the dynamic system naturally.

In an n-dimensional space, the trace function we have designed has the following form:

$$T_i(t) = \rho_i \sin\left[2\pi\left(\frac{t}{s}\right)^{1-a_0-(1-a_0-d_0)(i-1)/k}\right], \qquad i = 1, \cdots, n \tag{3.16}$$

where i represents the ith dimension, a_0, d_0, s, ρ, and k are parameters that control the speed and shape of the function. a_0 and d_0 are in the range of $[0,1]$ and $a_0 + d_0 < 1$. They specify the difference that the function moves in different dimensions. $s > 0$ controls the speed of the function. The larger the s is, the slower the function progresses with respect to t. ρ_i specifies the range that the function moves in the ith dimension. $k \geq n$ is a scaling factor. The larger the value of k is, the smaller the function changes from one dimension to another.

When $\rho_i = 1$, the trace function is bounded in space $[-1,1]^n$. If the search space of an application problem is not in $[-1,1]^n$, we can either map the search space to $[-1,1]^n$ or enlarge the range of the trace function by increasing ρ_i.

A further extension of function (3.16) is to add an initial phase shift to the trace function in each dimension as follows:

$$T_i(t) = \rho_i \sin\left[2\pi\left(\frac{t}{s}\right)^{1-a_0-(1-a_0-d_0)(i-1)/n} + 2\pi\frac{i-1}{n}\right], \qquad i = 1, \cdots, n \tag{3.17}$$

The second term inside the sine function is the initial dimension-dependent phase angle.

We compare trace functions with different parameter values using the coverage-evaluation method presented in the last section. We use trace function (3.16) as an example to illustrate the evaluation of coverage. In this example, the set of parameters are $a_0 = 0.05$, $s = 2$,

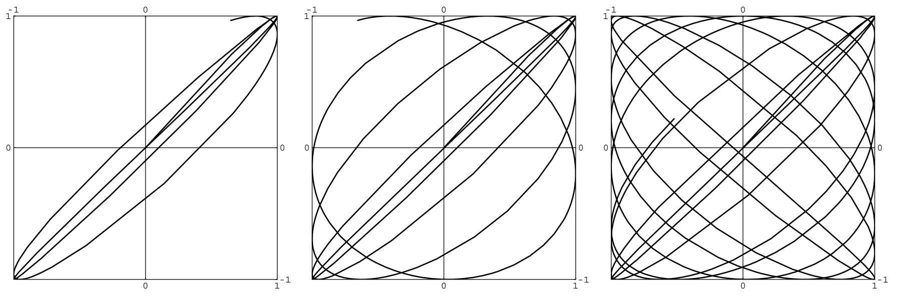


Figure 3.12: 2-D plots of trace function at $t = 5, 10$ and 20, respectively. The parameter values of the trace function are $a_0 = 0.05$, $s = 2$, $d_0 = 0.5$, $\rho_i = 1$ and $k = 10$.

$d_0 = 0.5$, $\rho_i = 1$, and $k = 10$. Figure 3.12 plots the trace function at $t = 5, 10$, and 20, respectively, in a 2-dimensional space.

As shown in Figure 3.12, the trace function moves in a band in the first 5 time units. By $t = 10$, the trace function has pretty much traversed the whole space once. By $t = 20$, the trace function continues to cover more vacant regions, and approximately in a uniform way.

Figure 3.13 shows the coverage measurement of this same trace in Figure 3.12 on a 2-D space based on our algorithm. The 2-D square is divided into 20x20 sub-squares. The number in each sub-square is the unit distance of the sub-square to the trace segment. The three graphs show the coverage of the trace when it runs for 5, 10, and 20 units of time, respectively. For $t = 5$, the maximal distance is 9. After 10 time units, the maximal distance is reduced to 3. After 20 time units, the maximal distance is 1.

The coverage depends on how long the trace runs and the length of the trace segment. Therefore, the coverage of trace functions should be compared based on same length of trace segments.

Parameters a_0 and d_0 are two important parameters in trace function (3.16). They determine the difference that the trace moves in each dimension: a_0 specifies the fastest movement and d_0 specifies the slowest movement. When $a_0 = 0$, the function is periodic with a cycle of 2π in the first dimension, and moves the fastest among all dimensions. In

$t = 5$

9	8	8	7	7	6	6	5	5	4	4	3	3	2	2	1	0	0	0	0
8	8	7	7	6	6	5	5	4	4	3	3	2	2	1	1	1	0	0	0
8	7	7	6	6	5	5	4	4	3	3	2	2	1	1	0	0	0	0	0
7	7	6	6	5	5	4	4	3	3	2	2	1	1	0	0	0	0	1	0
7	6	6	5	5	4	4	3	3	2	2	1	1	0	0	0	0	1	0	0
6	6	5	5	4	4	3	3	2	2	1	1	0	0	0	0	1	0	0	1
6	5	5	4	4	3	3	2	2	1	1	0	0	0	0	1	0	0	1	1
5	5	4	4	3	3	2	2	1	1	0	0	0	0	0	1	0	1	1	2
5	4	4	3	3	2	2	1	1	0	0	0	0	0	1	0	0	1	2	2
4	4	3	3	2	2	1	1	0	0	0	0	0	1	0	0	1	1	2	3
4	3	3	2	2	1	1	0	0	0	0	0	1	0	0	1	1	2	2	3
3	3	2	2	1	1	0	0	0	0	0	1	0	0	1	1	2	2	3	3
3	2	2	1	1	0	0	0	0	0	1	0	0	1	1	2	2	3	3	4
2	2	1	1	0	0	0	0	1	0	0	0	1	1	2	2	3	3	4	4
2	1	1	0	0	0	0	1	0	0	1	1	1	2	2	3	3	4	4	5
1	1	0	0	0	0	1	0	0	1	1	2	2	2	3	3	4	4	5	5
1	0	0	0	0	1	0	0	1	1	2	2	3	3	3	4	4	5	5	6
0	0	0	0	0	0	0	1	1	2	2	3	3	4	4	4	5	5	6	6
0	0	0	0	0	1	1	1	2	2	3	3	4	4	5	5	5	6	6	7
0	0	0	0	1	1	2	2	2	3	3	4	4	5	5	6	6	6	7	8

$t = 10$

3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	1	1	1	1	0	0	0	1	0	0	0	0	0	1	1	0	0	0
2	2	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	1	1	0	0	0	1	1	0	0	0	0	1	0
1	1	0	0	1	1	1	1	0	0	1	1	1	0	0	0	0	1	0	0
1	0	0	1	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	1	1	1	0	0	0	0	1	0	0	0	0
0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	1	0	1	0	0
0	1	1	1	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	0	0
0	1	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	1	0	0
0	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	1	0	0	1	0
0	0	1	1	0	0	0	0	1	0	0	0	1	1	1	0	0	1	1	0
0	1	1	0	0	0	0	1	0	0	1	1	1	1	0	0	1	1	1	0
0	0	0	0	0	0	1	0	0	1	1	1	1	0	0	1	1	1	0	0
0	0	0	0	0	1	0	0	1	1	1	1	0	0	1	1	1	0	0	1
0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	1	0	0	1	1
0	0	0	0	0	0	1	1	0	0	0	1	1	1	1	0	0	1	1	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2	3

$t = 20$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0
0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.13: Coverage measurements of a trace function in a 2-D space. Each dimension has 20 sample points. The three graphs correspond to the coverage when the trace runs for 5, 10, and 20 time units, respectively. Each number is the unit distance of the corresponding sub-square to the trace function.

the highest dimension, the function is close to $\sin(2\pi(t/s)^{d_0})$. Parameter s in trace function (3.16) controls the speed of the trace with respect to t, and does not affect the shape of the trace.

3.5.4 Numerical Evaluation of Trace-Function Coverage

In this section, we evaluate the coverage of different trace functions in the form of (3.16) by varying the values of a_0 and d_0. The search space is assumed to be a hypercube in the range of $[-1, 1]$ in each dimension. Each dimension of the high-dimensional space is divided into 20 equal-length components. In this way, an n-dimensional space is decomposed into 20^n sub-spaces of equal size. The coverage of a trace is evaluated based on the distribution of distances from sub-spaces to the trace.

Figure 3.14 shows the coverage of 4 trace functions with $a_0 = 0.05$ and $d_0 = 0.3, 0.5, 0.7$, and 0.9, respectively. Each of the four graphs shows a 3-D plot and the corresponding contour plot of the cumulative distribution of distances from sub-spaces to the trace. The three axes are the length of the trace, the distance of a sub-space to the trace (in the range of [0, 20]), and the cumulative distribution of distances from all sub-spaces to the trace. For example, the upper-left graph shows the coverage distribution of a trace function with $d_0 = 0.3$. When the trace reaches a length of 100, 1.6% of the sub-spaces are passed by the trace, 28.0% within one unit distance of the trace, 72.3% within a distance of 2, 95.8% within a distance of 3, 99.8% within a distance of 4, and all sub-spaces are within a distance of 5. The contour plot shows the 10, 50, and 90 percentile of the cumulative distribution.

The distance distributions of these trace functions with various d_0 are very similar to each other. For all of them, the 50-percentile of distance distribution are less than 1 after the curve traverses longer than 300. Moreover, the distance distribution is not sensitive to values of d_0: large d_0 values, e.g. 0.9, give worse coverage in the beginning for shorter curves, but are slightly better for longer curves. All the trace functions show the general trend that an exponentially longer curve is needed to improve the coverage linearly.

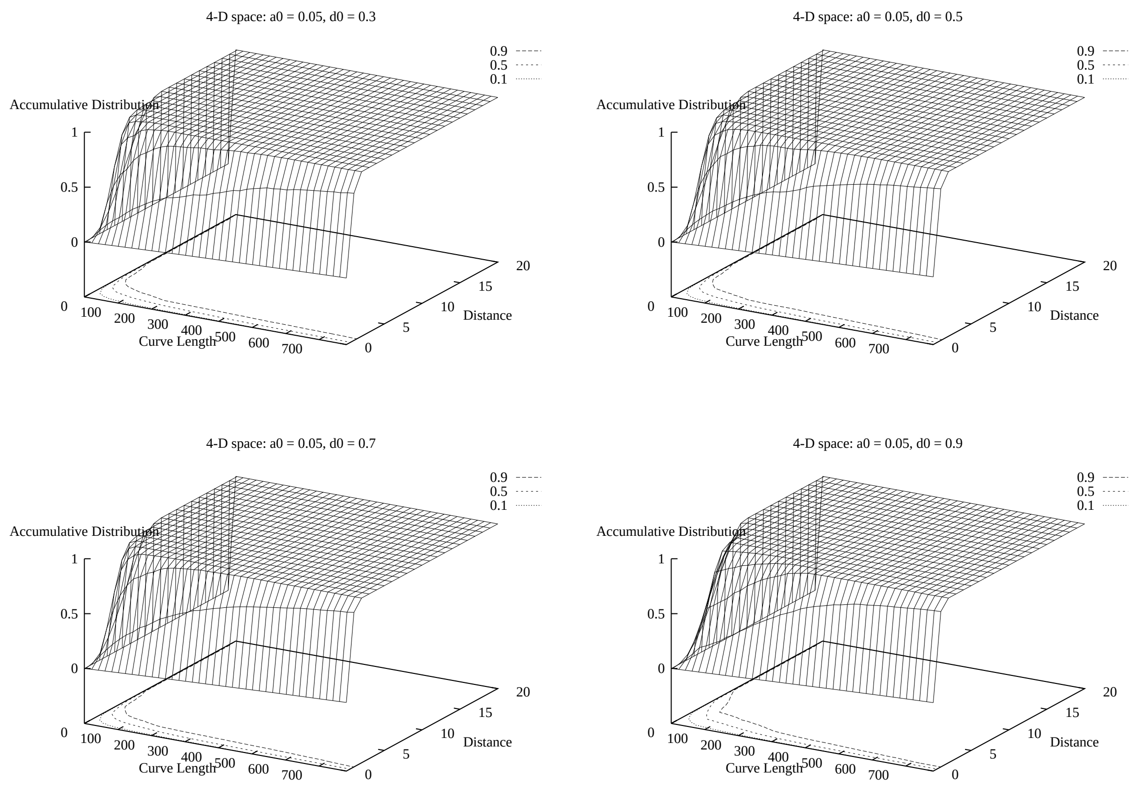


Figure 3.14: The coverage of 4 trace functions in a 4-D space with $a_0 = 0.05$ and $d_0 = 0.3, 0.5, 0.7$, and 0.9, respectively. Each graph shows a 3-D plot and a contour plot of the cumulative distribution of distances from sub-spaces to the trace function. The contour plots show the 10, 50, and 90 percentile of the distribution.

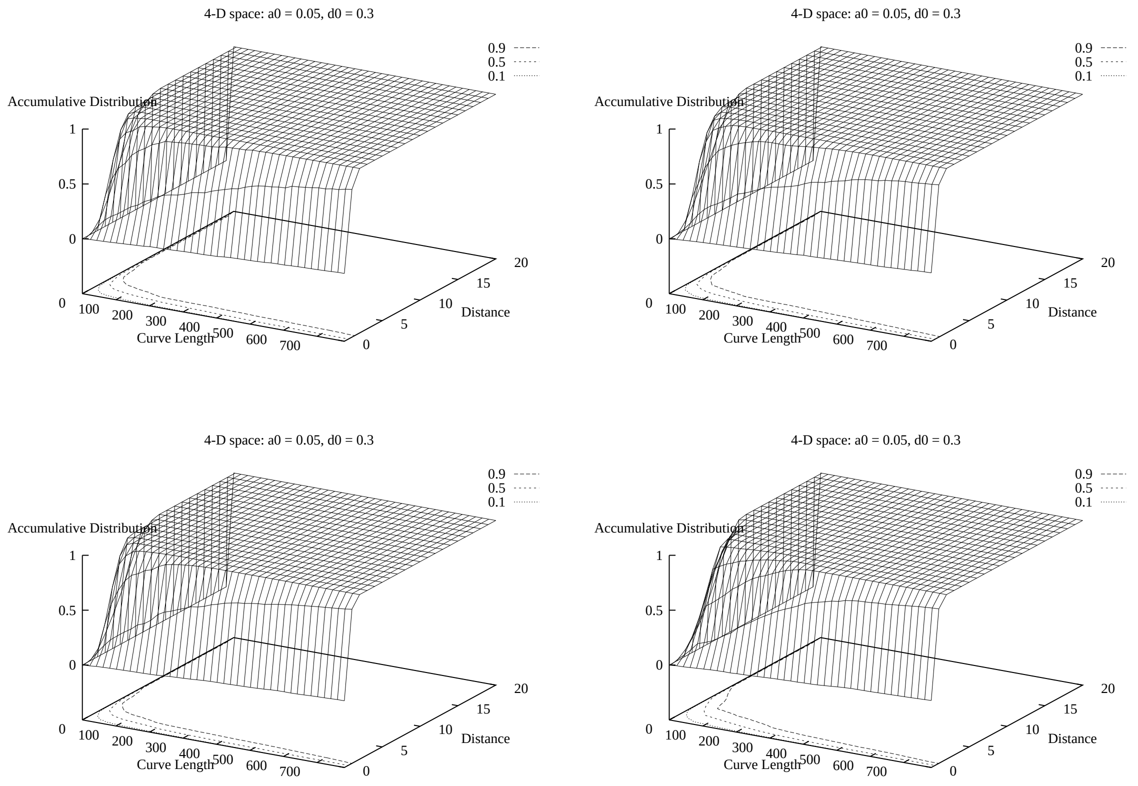


Figure 3.15: Sampling is used to evaluate the coverage of a set of trace functions in a 4-D space with $a_0 = 0.05$ and $d_0 = 0.3, 0.5, 0.7$, and 0.9, respectively. Each graph shows a 3-d plot and a contour plot of the cumulative distribution of distances between sub-spaces and the trace function. The contour plots show the 10, 50, and 90 percentiles of the distribution.

We have also varied a_0 while fixing $d_0 = 0.5$. Our results are similar to the situation of varying a_0. The distance distribution is not very sensitive to the values of a_0 in the range of [0.05, 0.3].

We are interested in solving high-dimensional optimization problems. Hence, we want to find out the coverage of the trace function for high-dimensional space. The computational complexity of complete coverage evaluation based on sub-spaces increases exponentially with respect to the number of dimensions, because the number of sub-spaces increases exponentially. Therefore, we use a sampling method to evaluate the coverage.

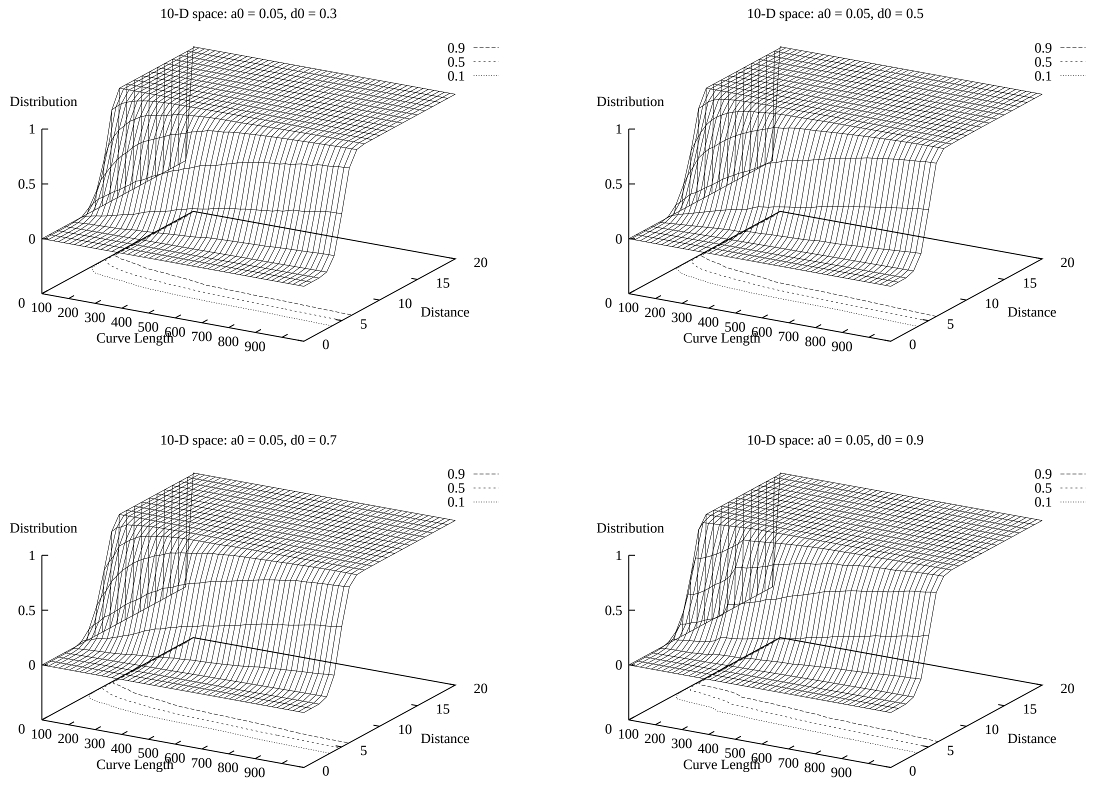


Figure 3.16: The coverage of a set of trace functions in a 10-dimensional space with $a_0 = 0.05$ and $d_0 = 0.3, 0.5, 0.7$, and 0.9, respectively. Each graph shows a 3-D plot and a contour plot of the cumulative distribution of distances between sub-spaces and the trace function. The contour plots show the 10, 50, and 90 percentiles of the distribution.

To check the accuracy of the sampling method, we compare the distance distribution obtained by the sampling method to that by the exact method. Figure 3.15 shows the coverage of the same trace functions as in Figure 3.14 in a 4-D space with $a_0 = 0.05$ and $d_0 = 0.3, 0.5, 0.7$, and 0.9, respectively. 1000 sub-spaces were sampled randomly for a given trace segment to obtain the sampling distance distribution. As shown in Figures 3.14 and 3.15, the distribution obtained by sampling is very close to that obtained by evaluating all the sub-spaces.

Next, we use sampling to evaluate the coverage of trace functions in high-dimensional space. Figures 3.16, 3.17, and 3.18 show the coverage of trace functions in 10-, 20-, and 40-dimensional spaces. As before, each dimension of the high-dimensional spaces is divided into 20 equal-length parts. We fix $a_0 = 0.05$ and vary d_0 as $0.3, 0.5, 0.7$, and 0.9.

In each of these three figures, various values of d_0 generate similar distributions. In a 10-dimensional space, the coverage of trace functions is significantly worse than that in a 4-dimensional space. For example, the 50-percentile contour line flattens out at a distance of 5. In contrast, in a 4-dimensional space, the 50-percentile contour line decreases to less than 1 and approaches 0. The coverage is even worse in higher-dimensional spaces. In 20- and 40-dimensional space, the 50-percentile line only reaches a distance of 8 and 9, respectively.

Our conclusions of trace-function coverage are as follows: The coverage is insensitive to values of a_0 and d_0 in a large range. The coverage improves significantly in the beginning of the search and shows little improvement afterwards. The behavior is worse for higher-dimensional space. In high-dimensional space, the coverage is low when the curve length is short.

3.5.5 Summary of Trace-Function Design

In this section, we have studied the coverage of trace functions. We introduce the coverage of trace function in a high-dimensional space, and present coverage measurements based on sub-spaces. A high-dimensional space is divided into smaller sub-spaces, and each sub-space is treated as an entity. The distance from each sub-space to a trace is calculated. The distribution of distances is used as the basis for evaluating coverage. In a higher-dimensional space, the number of sub-spaces is very large, and calculating distances of all the sub-spaces is prohibitively expensive. In this situation, sampling is used to get an approximate distribution. A large number of sub-spaces are sampled randomly to give a good approximation.

We have identified two ways to traverse in a search space. One is to divide the space into sub-spaces and search one sub-space extensively after another. The other one is to search the space from coarse to fine. We prefer the second approach due to its advantage

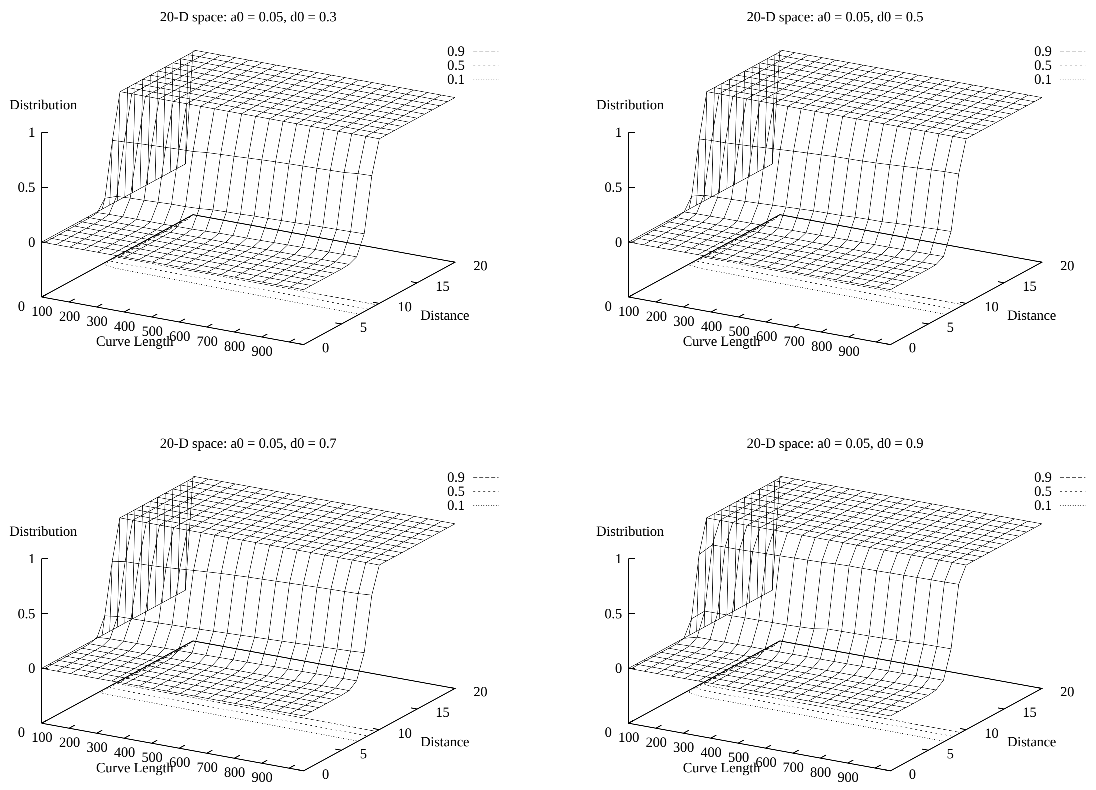


Figure 3.17: The coverage of a set of trace functions in a 20-dimensional space with $a_0 = 0.05$ and $d_0 = 0.3, 0.5, 0.7$, and 0.9, respectively. Each graph shows a 3-D plot and a contour plot of the cumulative distribution of distances between sub-spaces and the trace function. The contour plots show the 10, 50, and 90 percentiles of the distribution.

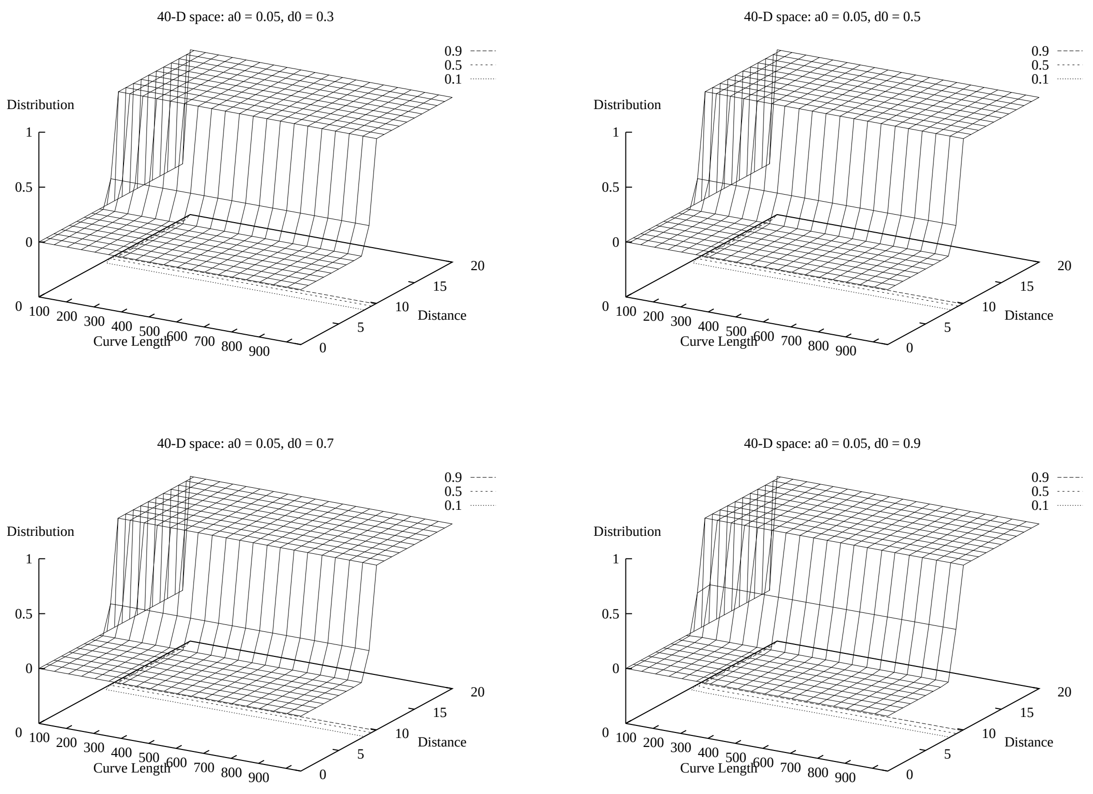


Figure 3.18: The coverage of a set of trace functions in a 40-dimensional space with $a_0 = 0.05$ and $d_0 = 0.3, 0.5, 0.7$, and 0.9, respectively. Each graph shows a 3-D plot and a contour plot of the cumulative distribution of distances between sub-spaces and the trace function. The contour plots show the 10, 50, and 90 percentiles of the distribution.

of continuous improvement of solution quality as time goes by. We have designed a family of trace functions (3.16) using the second approach, and have studied the coverage of these trace functions with different parameter values in 4-, 10-, 20-, and 40-dimensional spaces. Our results are summarized as follows:

- The coverage of trace functions is not sensitive to their parameter values of a_0 and d_0 in fairly large ranges.
- The coverage improves quickly as a trace starts to traverse a search space and then flatten out.
- An exponentially longer trace is needed to improve the coverage linearly.
- Trace functions can obtain dense coverage for low-dimensional space, but can only achieve sparse coverage for high-dimensional space when the length of the trace is short.

Using trace functions to cover a high-dimensional space densely is expensive. Therefore, in our trace-based global search method, good initial points are important to bring the trace close to good solutions, reducing the space that the trace function tries to cover, and shortening the time to find global optima.

In the rest of this thesis, we use the trace function in (3.16) with parameters $a_0 = 0.05$ and $d_0 = 0.5$ in our experiments.

3.6 Speed of Trace Function

The search trajectory of our trace-based global search is affected by the speed of the trace function. The system of ODEs as specified in (3.10) has a limit on how fast the system can change. If the trace moves too fast, the system of ODE is not able to keep up with the trace. On the other hand, a slow trace is also not good because the search trajectory cannot cover a large space given the time available. Also, the trace may have little pulling force, resulting in gradient descent being the dominant force.

In trace function (3.16), we introduce a parameter, s, to adjust the speed of the trace. Smaller s makes the trace move faster in the same amount of time. We want the trace to move fast, yet not over the limit of the ODE system.

To check whether trace $T(t)$ moves too fast, we pass the trace through the following ODE system:

$$\dot{\mathbf{x}}(t) = -(\mathbf{x}(t) - T(t)) \tag{3.18}$$

This system generates a trajectory that tries to follow the trace function. If the trace moves too fast, the generated trajectory only traverses a much smaller region as compared to the space the trace passes.

We have tested different values of parameter s of the trace function in (3.16). We want s to be small in order to give faster speed, while keeping the output trajectory close to the trace. Empirically, we have found that $s = 2$ is good and use it in the rest of this thesis.

3.7 Variable Scaling During Global Search

In our trace-based global-search method, the search trajectory is generated by the combined force of the trace and the local gradient. These two forces have to strike a balance for the trace-based global search to work well.

When the global-search stages are modeled as a system of ODEs in (3.10), a pair of coefficients, μ_t and μ_t, specify the relative weights put on the trace and the local descent. Appropriate values of the coefficients are necessary to generate a good search trajectory. If the weight on the trace is too large, then the trace becomes dominant, and the search trajectory follows the trace closely, but ignores terrain information. On the other hand, if the weight on the gradient is too large, then local descents become dominant, and the search trajectory is trapped in local minima and cannot get out.

When we use fixed weights, gradient descents dominate the trace in some places where gradients are large. To overcome the difficulty caused by large gradients, we have developed an adaptive variable scaling technique. The idea of variable scaling is to stretch out the

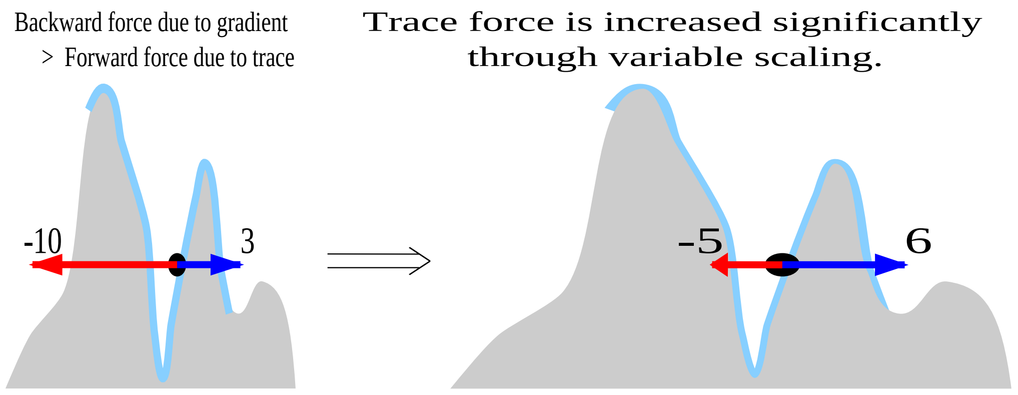

Backward force due to gradient
> Forward force due to trace
-10
3
Trace force is increased significantly through variable scaling.
-5
6

Figure 3.19: Variable scaling stretches out the variable, which reduces the backward force due to the gradient and increases the forward force due to the trace.

objective function along the variable dimensions. Figure 3.19 illustrates the effect of variable scaling that changes the relative value of the gradient and the trace.

In the left graph of Figure 3.19, the gradient on the steep slope contributes a -10 backward force, while a forward force of 3 is provided by the trace. Since the forward force is less than the backward force, the search cannot move forward. The right graph shows the situation after scaling variable values by 2. After the variable is stretched out, the backward force is reduced by half and the forward force is increased by 2. Now the search overcomes the slope and can move forward.

Figure 3.20 provides more detailed analysis on what happens when local gradients dominate the trace, and before and after variable scaling. In the left graph, a trace function is going from left to right along one variable dimension x. In this example, x has values in the range of 1 and 3. The search trajectory follows the trace function and hits a local minimum. Function values along the search trajectory are plotted below the search trajectory. On the left side of the local minimum basin, the gradient and the trace point in the same direction, causing the search trajectory to go down the slope. The problem happens when the trace tries to pull the search trajectory out of the local minimum from the right side of the basin. The gradient force counteracts and dominates the trace force. The result is that the search trajectory cannot get out and is trapped in the local minimum.

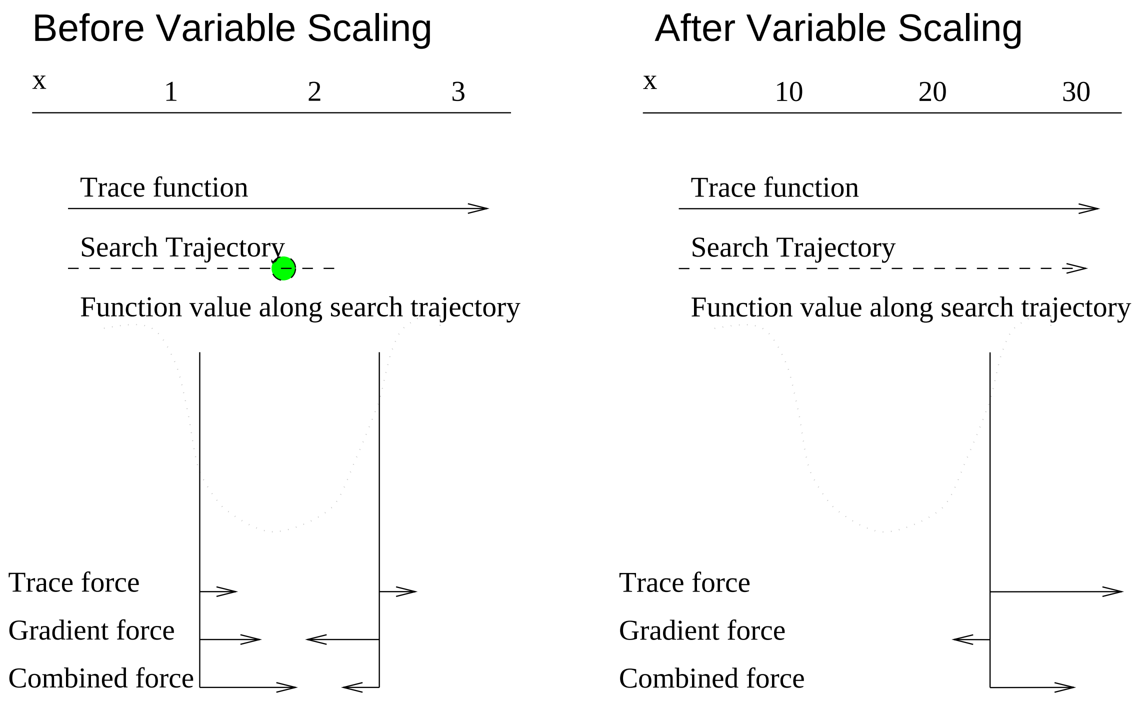


Figure 3.20: Illustration of gradient and trace force around a local minimum along dimension x. The left graph shows the situation before variable scaling, in which the search trajectory is trapped in the local minimum. The right graph shows the situation after variable scaling where variable x is scaled up 10 times and the search trajectory overcomes the local minimum. The graphs show the trace function going from left to right, and the resulting search trajectory.

In the right graph of Figure 3.20, variable x is scaled up 10 times and has values from 10 to 30. The trace force is increased by 10 times because the difference between the trace position and the search trajectory position is increased by 10 times. At the same time, the gradient magnitude is reduced by 10 times. When the trace tries to pull the search trajectory out from the right side of the local minimum basin, the trace force is larger than the gradient force, and the search trajectory successfully overcomes the local minimum.

Scaling up variables reduces the gradient force and increases the trace force. Conversely, scaling down variables increases the gradient and reduces the trace force. The next problem is whether we should scale up or down variables at a certain time during the search, and by how much.

The trace force is proportional to the distance of the current trace position and the search position. Usually global optimization problems have an explicitly specified region of interests that corresponds to the search space. The trace function traverses the inside of the search space. So we have a good idea of how large the trace force can be. On the other hand, the magnitude of the gradient of the optimization function is more difficult to estimate. Its value can be in a very large range, varying from one problem to another, and changing significantly from one search region to another.

In our adaptive variable-scaling method, variables are scaled adaptively based on the local gradients and the values of the trace function. When the gradient is much larger than the trace force, variables are scaled up to make the gradient smaller than the trace force. When variables are scaled up, the trace force is increased proportionally because the trace now is in a larger region. At the same time, the local gradient is reduce proportionally because the function value is not changed and the variables have larger values. For example, suppose the function to be minimized is $f(x)$, and x is in the range of $[-1, 1]$. Scaling x up by 10 times means that x is mapped to the range of $[-10, 10]$, and the function becomes $f(x/10)$. By doing variable scaling adaptively during the search, the problem caused by deep local minima and large gradients can be effectively solved.

Let us illustrate the adaptive variable-scaling method using a simple example based on minimizing a 2-dimensional Grewank's function [253]. Figure 3.21 shows a trace and the search trajectories with and without adaptive variable scaling on a 2-D contour plot of a portion of this objective function. A darker color in the contour plot represents a smaller function value. The left graph shows the super-imposed trace function starting from the position marked as "x" near the center of the graph. The starting point is inside a local minimal basin. The trace function is evaluated for 10 time units, from $t = 0$ to 10.

The middle graph shows the search trajectory after one stage of the trace-based global search without adaptive variable scaling. The trajectory is obtained by evaluating the system of ODEs (3.10) from $t = 0$ to 10 using an ordinary differential equation solver, LSODE. Because the gradient force is much larger than the trace force, the search trajectory is trapped inside a local minimal region.

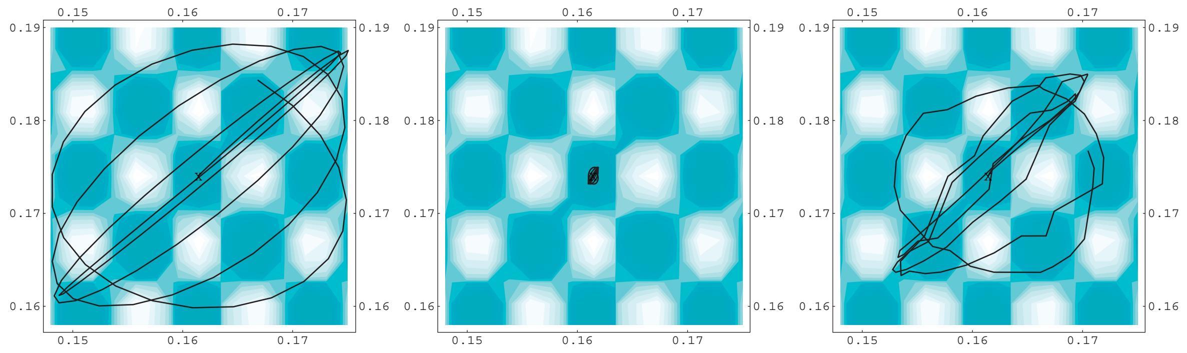


Figure 3.21: The trace function and search trajectories after one stage of the trace-based global search with and without adaptive variable scaling on the 2-D Grewank's function. The 2-D contour plots of Grewank's function are shown in all three graphs. A darker color in the contour plot represents a smaller function value. The left graph shows the trace function starting from the position marked "x". The middle graph shows search trajectory without variable scaling. The right graph shows search trajectory with adaptive variable scaling.

The right graph of Figure 3.21 shows the search trajectory generated using adaptive variable scaling. In this example, the trajectory is not very smooth because variables are not scaled continuously. Instead, variable scaling is performed every $\delta t = 0.1$ time intervals. After every δt time unit, the gradient of the current search point is evaluated. If the gradient along some variable dimensions are too large, then these variables are scaled up. The variables are scaled separately with different scaling factors. As shown in the right graph, due to the effect that adaptive variable scaling reduces large gradients, the search trajectory overcomes the local minima, and follows the trace function nicely.

Note that besides variable scaling, function scaling is also helpful in modifying function and gradient values to overcome local minima. For example, when the objective function takes on values in a large range and has large gradients, function transformations, such as logarithmic or sigmoidal transformation, can be applied to compress the function values.

3.8 Numerical Methods for Solving the Dynamic Systems of Equations

Each stage of the trace-based global search is modeled by a system of ordinary differential equation (ODEs). The search trajectory of each stage is generated by solving the system of ODEs as an initial value problem (IVP).

Not all initial value problems can be solved explicitly. Oftentimes, it is impossible to find a closed-form solution. Many numerical methods have been developed to approximate the solution. Widely used methods include Euler's method, Runge-Kutta methods, predictor-corrector methods, and implicit methods. Next, we briefly introduce IVP and present some popular numerical methods [23, 27, 36, 55, 119, 240].

A general IVP has the following form:

$$y'(t) = f(t, y) \qquad \text{with} \qquad y(0) = y_0. \tag{3.19}$$

In numerical methods, a set of points $\{(t_k, y_k)\}$ is generated as an approximation of the solution, i.e. $y(t_k) \approx y_k$. Because the solution is approximated at a set of discrete points, these methods are called *difference methods* or *discrete variable methods.*

A simple single-step method has the form $y_{k+1} = y_k + h\Phi(t_k, y_k)$ for some function, where $\Phi(t_k, y_k)$ is the *increment function.* The value h is called the *step size.* Single-step methods use only the information from one previous point to compute the next point.

The Euler method belongs to the class of single-step methods, in which the increment function is $f(t_k, y_k)$. The Euler method is simple and easy to understand. However, it has limited usage because larger errors are accumulated as the search proceeds.

Another family of single-step methods are Runge-Kutta methods, which are widely used in solving IVPs. The following Runge-Kutta method of order 4 is the most popular.

$$y_{k+1} = y_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3.20}$$

where

$$\begin{aligned}
k_1 &= hf(t_k, y_k) \\
k_2 &= hf(t_k + h/2, y_k + k_1/2) \\
k_3 &= hf(t_k + h/2, y_k + k_2/2) \\
k_4 &= hf(t_k + h, y_k + k_3)
\end{aligned}$$

Runge-Kutta methods are good general purpose methods because they are quite accurate, stable, and easy to program.

Single-step methods are self-starting in the sense that points can be computed one by one starting from the initial value. *Multi-step methods* use several prior points in the calculation of one point. They are not self-starting. A desirable feature of a multi-step method is that the local error can be determined and a correction term can be included, which improves the accuracy of the answer in each step. Also, adaptive step-size adjustment can be accomplished. It is possible to determine if the step size is small enough to obtain an accurate value for the current point, yet large enough that unnecessary and time-consuming calculations can be eliminated. An example of a multi-step method is the Adams-Bashforth-Moulton method.

The methods we describes above are *explicit methods*. They use one or multiple previous points to calculate the current point. Explicit methods have a limited stability region from which the step size is chosen. A larger stability region can be obtained by using information at the current time, which makes the method *implicit*. The simplest examples is the backward Euler method,

$$y_{k+1} = y_k + hf(t_{k+1}, y_{k+1}) \tag{3.21}$$

This method is implicit because f is evaluated with the argument y_{k+1} before the value of y_{k+1} is known. The value of y_{k+1} is often determined iteratively by methods such as Newton's method or fixed-point iteration.

Implicit methods have a significantly larger stability region. Much larger steps may be taken, and thus attain much higher overall efficiency than explicit methods, despite requiring more computations per step. Also, certain types of implicit methods are much more effective than explicit methods in solving stiff ODEs.

A system of ODEs is *stiff* if its Jacobian matrix $J = (\partial f_i / \partial y_j)$ has at least one eigenvalue with a large negative real part and the solution is slowly varying on most of the interval of integration. Stiffness is shown as the convergence of solution curves is too rapid.

All commonly used formulas for solving stiff IVPs are implicit in some sense. Implicit methods solve a linear or nonlinear system in each step of the integration. A good starting guess for the iteration in solving y_{k+1} can be obtained from an explicit formula, using the explicit and implicit formulae as a *predictor-corrector* pair.

One of the most popular pairs of multi-step methods is the Adams-Bashforth-Moulton method. It uses the explicit fourth-order Adams-Bashforth predictor

$$y_{k+1} = y_k + \frac{h}{24}(55y'_k - 59y'_{k-1} + 37y'_{k-2} - 9y'_{k-3}) \tag{3.22}$$

and the implicit fourth-order Adams-Moulton corrector

$$y_{k+1} = y_k + \frac{h}{24}(9y'_{k+1} + 19y'_k - 5y'_{k-1} + y'_{k-2}) \tag{3.23}$$

Another family of implicit multi-step methods are the *backward differentiation formulas.* One example is

$$y_{k+1} = \frac{1}{11}(18y_k - 9y_{k-1} + 2y_{k-2}) + \frac{6}{11}hy'_{k+1} \tag{3.24}$$

This type of methods are particularly good for solving stiff ODEs.

After briefly reviewing existing numerical methods for solving IVPs, we come back to solving the IVPs modeling the global-search stages of our trace-based search. We have used the explicit Euler method, the Adams-Bashforth-Moulton method, and a method based on backward differentiation formulas.

In applying the Euler method to solve a system of ODEs defined in (3.10), we have the following iterative equation:

$$\mathbf{x}(t + \delta t) = \mathbf{x}(t) + \delta t[-\mu_g \nabla_{\mathbf{x}} f(\mathbf{x}(t)) - \mu_t(\mathbf{x}(t) - T(t))] \tag{3.25}$$

where δt is the step size. A large δt causes a large stride of variable modification, possibly resulting in instability. On the other hand, a small δt means a longer computation time for traversing the same distance. When the exact search trajectory of the global-search phase is not critical, the Euler method can obtain good results. The Euler method runs faster and is applied to solve larger problems.

The more sophisticated numerical methods we have used are implemented in a software package, LSODE, which is the Livermore Solver for Ordinary Differential Equations [120]. LSODE solves the system of ODEs to within a prescribed degree of accuracy efficiently using adaptive step-size adjustments. LSODE implements both the Adams method and the method based on backward differentiation formulas. Most of the time, we have used the method based on backward differentiation formulas in case the ODE system is stiff. We have also used the Adams method for designing QMF filter banks.

LSODE obtains high precision solutions in solving IVPs. When the function to be optimized is smooth, LSODE is accurate and usually runs fast as well. However, when the optimized function is rugged, or when there is noise in function and gradient evaluations, LSODE may progress slowly. When the number of variables is large, LSODE may also become computationally expensive.

Therefore, the Euler method has advantages in situations in which the function to be optimized is rugged, has noisy values, and has large number of dimensions. The execution time of the Euler method can be much faster than more sophisticated methods, and is relatively insensitive to noise. Its disadvantage is that it has larger errors, which may cause the search trajectory to diverge from the theoretical trajectory. Also, it is not easy to select the appropriate step size in the Euler method.

3.9 Novel: A Prototype Implementing Proposed Methods

In the last chapter, we have presented Lagrangian methods to handle nonlinear constraints in continuous and discrete optimization problems. In this chapter, we have presented a multi-level global search method to overcome local minima. In this section, we present a prototype that implements the methods proposed in this and the last chapter. In

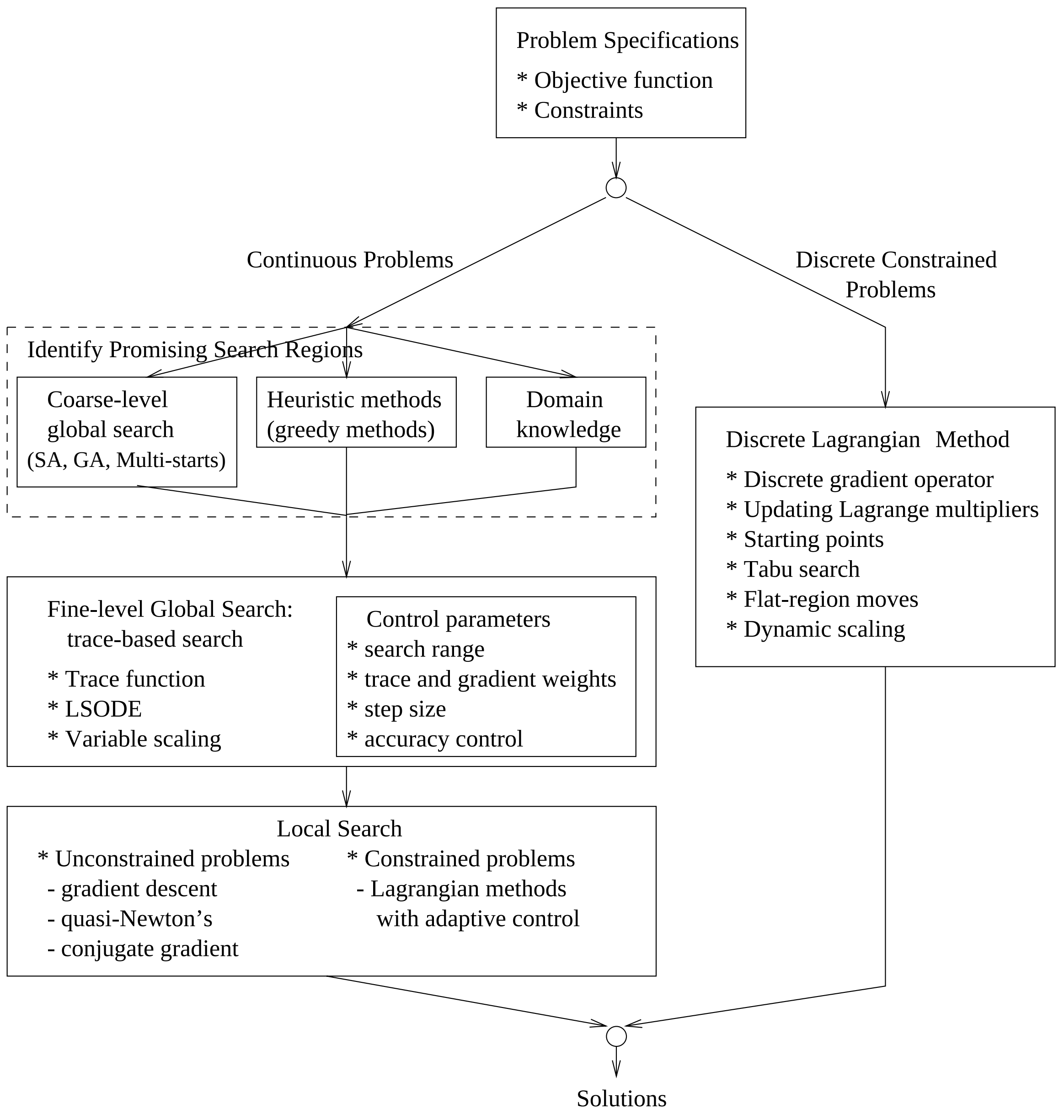


Figure 3.22: Organization of the Novel global search method.

the prototype, our proposed methods are put together to solve nonlinear unconstrained and constrained, continuous and discrete problems. The prototype is called Novel, which stands for *Nonlinear Optimization Via External Lead*.

Figure 3.22 shows the organization of the Novel prototype. Users specify an objective function and a constraint set as well as some control parameters. Novel solves discrete constrained problems using discrete Lagrangian methods. Continuous problems are solved using a combination of global and local searches. We present each component of Novel in detail next.

3.9.1 Continuous Methods

For nonlinear continuous unconstrained optimization problems, the optimal solutions of an objective function are solved by global- and local-search methods. For constrained problems, constraints are handled by Lagrangian methods. Inequality constraints are first converted to equality constraints using the slack-variable or the MaxQ method. Then, the combination of the objective function and the constraints forms a Lagrangian function. Finally, global and local searches are performed on the Lagrangian function to find good solutions.

The search space of a continuous optimization problem is usually large. Identifying promising search regions and concentrating the search in a smaller sub-space can effectively reduce the computation time in finding good solutions. Promising regions are represented as initial points that are fed into the fine-level global search of Novel.

In our optimization method, promising regions are identified through three ways: (1) domain knowledge, (2) heuristic method, and (3) coarse-level global search. For knowledge-rich applications, domain knowledge provides reliable information about promising search regions. For example, in feedforward neural-network learning problems, optimal weights usually have small values and optimal solutions are not far from the origin. Therefore, the sub-space around the origin is a promising search region. In designing filter banks, we try to improve the best existing solutions. Thus, the sub-spaces around existing solutions are promising search regions.

In applications that have little or no domain knowledge, some types of search methods are applied. These include heuristic search methods and general-purpose search methods. Heuristic search methods, including various greedy methods, use heuristic strategies to find promising regions. For instance, a greedy method can first find the optimal value for each dimension while keeping the values of other variables constant. Then, it combines the optimal value for each dimension to form a sub-optimal solution of the original multi-dimensional problem.

General-purpose global-search methods can be used in the coarse-level global search to find promising regions. In Novel, we have used existing global-search methods that include simulated annealing, genetic algorithms, and multi-start of local descents. In addition, we have also developed a trajectory-based method as a variation of multi-start of descents.

Novel consists of a unique fine-level global search, the trace-based global search method. Our trace-based search uses a problem-independent trace to lead the search and achieves a balance between global and local searches. During the search, our trace-based search relies on two counteracting forces, the local gradient force that drives the search to a local optimum and a deterministic trace force that leads the search out of local minima. Good starting points identified in the global-search phase are used in the local search to find good local minima.

In Novel, the trace function for an n-dimensional space is defined in (3.16), and the default parameter values are $a_0 = 0.05$, $d_0 = 0.5$, $s = 2$, $\rho = 1$, and $k = n$. The global-search stage of our trace-based search is modeled as a system of ordinary differential equations (ODEs). For an unconstrained problem with objective function $f(\mathbf{x})$, the system of ODEs is

$$\frac{d\mathbf{x}(t)}{dt} = -\mu_g \, \nabla_{\mathbf{x}} f(\mathbf{x}(t)) - \mu_t \, (\mathbf{x}(t) - T(t))$$

where μ_g and μ_t are coefficients specifying the weights for descents in local minimum regions and for exploration of broader space. The default value of each is 1. The default number of global-search stages is 3. An adaptive variable scaling technique is implemented to overcome deep local minima with large gradients.

For a constrained optimization problem with Lagrangian function $L(\mathbf{x}, \lambda)$, the system of ODEs is as follows:

$$\begin{aligned}
\frac{d\mathbf{x}(t)}{dt} &= -\mu_g \nabla_{\mathbf{x}} L(\mathbf{x}(t), \lambda(t)) - \mu_t (\mathbf{x}(t) - T(t)) \\
\frac{d\lambda(t)}{dt} &= \nabla_{\lambda} L(\mathbf{x}(t), \lambda(t))
\end{aligned}$$

where μ_g and μ_t are coefficients. The search performs descents in the original-variable $\mathbf{x}$ space and ascents in the Lagrange-multiplier λ space. A trace function $T(t)$ brings the search out of local minima in the original-variable space.

The system of ODEs is solved by the Euler method or LSODE [120], which is the Livermore Solver for Ordinary Differential Equations. LSODE solves the system of ODEs to within a prescribed degree of accuracy using adaptive step size adjustments. LSODE implements both Adams method and the method based on backward differentiation formulas.

Fine-level global searches discover local-minima basins, and local searches find local minima. Fine-level global searches provide initial points for local searches. Initial points for local searches are selected based on search trajectories generated by each stage of the trace-based global search. Novel implements two heuristics to select the initial points: the best solutions in periodic time intervals or the local minima in each trajectory.

Existing methods of local descents are applied in local searches. For unconstrained optimization problems, they are gradient descent, quasi-Newton's and conjugate gradient methods. Problem-specific methods, such as back-propagation in neural-network training application, can also be applied.

For constrained optimization problems, Novel applies the Lagrangian method with adaptive control to perform local search. For constrained optimization problems with Lagrangian function $L(\mathbf{x}, \lambda)$, the system of ODEs is

$$\begin{aligned}
\frac{d\mathbf{x}(t)}{dt} &= -\nabla_{\mathbf{x}} L(\mathbf{x}(t), \lambda(t)) \\
\frac{d\lambda(t)}{dt} &= \nabla_{\lambda} L(\mathbf{x}(t), \lambda(t))
\end{aligned}$$

where μ_g and μ_t are coefficients. A combination of the objective function and the constraints forms the Lagrangian function $L(\mathbf{x}, \lambda)$. By doing descents in the original variable $\mathbf{x}$ space and ascents in the Lagrange-multiplier λ space, the system converges to a saddle point corresponding to a local minimum of the original constrained optimization problem. Based on the search profile, the Lagrangian method adjusts the weights of the objective and the constraints to achieve faster and more robust convergence.

In the appendix, we demonstrate Novel's ability of finding good solutions by solving some nonlinear simple-bounded optimization problems. We compare the experimental results obtained by Novel with those obtained by multi-start of gradient descent.

3.9.2 Discrete Methods

Novel applies discrete Lagrangian methods to solve discrete constrained problems. For the following equality constrained problem:

$$\begin{array}{ll} \min_{\mathbf{x}\in D^n} & f(\mathbf{x}) \\ \text{subject to} & h(\mathbf{x}) = 0 \end{array}$$

where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, $h(\mathbf{x}) = (h_1(x), h_2(x), \cdots, h_m(x))$, and D is a subset of integers, i.e., $D^n \subset I^n$, the Lagrangian function is

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda^T h(\mathbf{x}).$$

The general structure of DLM to solve this problem is shown in Figure 3.23. In discrete space, $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$ is used in place of the gradient function in continuous space. DLM performs descents in the original variable space of $\mathbf{x}$ and ascents in the Lagrange-multiplier space of λ. We call one *iteration* as one pass through the while loop. In the following, we describe the features of our implementations of DLM.

(a) *Descent Strategies.* There are two ways to calculate $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$: greedy and hill-climbing, each involving a search in the neighborhood of the current value of $\mathbf{x}$.

In a *greedy strategy*, the value of $\mathbf{x}$ leading to the maximum decrease in the Lagrangian-function value is selected to update the current value. Therefore, all values in the vicinity

```
Set initial $\mathbf{x}$ and $\lambda$
while $\mathbf{x}$ is not a saddle point or stopping condition has not been reached
    update $\mathbf{x}$: $\mathbf{x} \longleftarrow \mathbf{x} - \Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$
    if condition for updating $\lambda$ is satisfied then
        update $\lambda$: $\lambda \longleftarrow \lambda + c \times h(\mathbf{x})$
    end if
end while
```

Figure 3.23: A generic discrete Lagrangian method.

need to be searched every time, leading to high computation complexity. In *hill-climbing*, the first value of $\mathbf{x}$ leading to a decrease in the Lagrangian-function value is selected to update the current $\mathbf{x}$. Depending on the order of search and the number of points that can improve, hill-climbing strategies are generally less computationally expensive than greedy strategies. Hence, we usually use the hill-climbing strategy in our experiments.

(b) *Updating* λ. The frequency in which λ is updated affects the performance of a search. The considerations here are different from those of continuous problems. In a discrete problem, descents based on discrete gradients usually make small changes in $L(\mathbf{x}, \lambda)$ in each update of $\mathbf{x}$ because only one variable changes. Hence, λ should not be updated in each iteration to avoid biasing the search in the Lagrange-multiplier space of λ over the original variable space of $\mathbf{x}$.

In our implementation, λ is updated in two situations. One is when the search reaches a local minimum; another is decided by parameter T that specifies the number of iterations before λ is updated. T can be changed dynamically according to the value of $L(\mathbf{x}, \lambda)$ or when $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$. When $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$, a local minimum in the original variable space is reached, and the search can only escape from it by updating λ.

A parameter c in the term $c \times h(x)$ in Figure 3.23 controls the magnitude of changes in λ. c is the weight put on constraints, while the weight of the objective is always 1. In general, c can be a vector of real numbers, allowing non-uniform updates of λ across different dimensions and possibly across time. For simplicity, we have used a constant c in our implementation for all λ's.

For some problems, the constraints $h(\mathbf{x})$ are only evaluated to non-negative values, such as in satisfiability problems. This makes λ in Figure 3.23 nondecreasing. For difficult problems that require millions of iterations, λ values can become very large as the search progresses. Large λ's are generally undesirable because they cause large swings in the Lagrangian-function value, and make the search space rugged and difficult to find good solutions.

To overcome this problem, we have developed strategies to reduce λ dynamically. One method is to reduce λ periodically every certain number of iterations. More sophisticated methods reduce λ adaptively based on the profiles of the objective, constraints, and Lagrange-multiplier values.

(c) *Starting Points and Restarts.* In contrast to random-start methods relying on restarting from randomly generated initial points to bring a search out of a local minimum, DLM continue to evolve without restarts until a saddle point is found. This avoids restarting from a new starting point when a search is already in the proximity of a good local minimum. Another major advantage of DLM is that there are very few parameters to be selected or tuned by users, including the initial starting point. This makes it possible for DLM to always start from the origin or from a random starting point generated by a fixed random seed, and find a saddle point if one exists. The initial of λ is always 0.

(d) *Plateaus in the Search Space.* In discrete problems, plateaus with equal Lagrangian values exist in the search space. Discrete gradient operator may have difficulties in plateaus because it only examines adjacent points of $L(\mathbf{x}, \lambda)$ that differ in one dimension. Hence, it may not be able to distinguish a plateau from a local minimum. We have implemented two strategies to allow a plateau to be searched.

First, we need to determine when to change λ when the search reaches a plateau. As indicated earlier, λ should be updated when the search reaches a local minimum in the Lagrangian space. However, updating λ when the search is in a plateau changes the surface of the plateau and may make it more difficult for the search to find a local minimum somewhere inside the plateau. To avoid updating λ immediately when the search reaches a plateau, we have developed a strategy called *flat move*. This allows the search to continue for some time in the plateau without changing λ, so that the search can traverse states with the same Lagrangian-function value. How long flat moves should be allowed is heuristic and possibly problem-dependent.

Our second search strategy is to avoid revisiting the same set of states in a plateau. In general, it is impractical to remember every state the search visits in a plateau due to the large storage and computational overheads. In our implementation, we have kept a tabu list to maintain the set of variables flipped in the recent past [92, 112] and to avoid flipping a variable if it is in the tabu list.

3.10 Potential for Parallel Processing

Parallel processing is essential in solving large nonlinear optimization problems. These problems are computationally intensive because the number of local minima and the number of variables can be large; the objectives, constraints, and derivatives can be expensive to evaluate; and many evaluations of the objectives, constraints, and derivatives may be needed. Also, the most powerful computers in the world are parallel computers, and the trend will continue in the future.

In our global search method shown in Figure 3.3, many components can be executed in parallel. Generally, parallelism is found at three levels: (1) global search in parallel, (2) local search in parallel, and (3) evaluation of objective, constraints, and derivatives in parallel.

Our coarse-level global search in Novel provides initial points for our fine-level global search. Parallel search methods can be used in the coarse-level global search, such as parallel genetic algorithms, parallel simulated annealing, and parallel multi-starts of descents. Other heuristic methods for coarse-level global search can also be parallelized. For instance, in the trajectory-based method, sample points are selected along a 1-dimensional trajectory function over the search space. Then, descents from these sample points, which take much more time than selecting the sample points, can be done in parallel.

Trace-based global searches starting from different initial points generated by the coarse-level search can be executed in parallel. They are independent and have minimal communication and synchronization overheads. In each trace-based global search, multiple stages can be executed in parallel. These stages are cascaded, and can be executed in a pipelined fashion. However, since the number of stages is not large, the amount of parallelism is limited.

Local searches of our optimization method start from initial points generated by fine-level global searches. They are independent and can be executed in parallel. For many application problems, local descents are much more expensive. In contrast, the global-search phase is much faster and can quickly generate many initial points for local descents. In this case, many descents can be performed in parallel since there is very little communication overhead among them.

There are a large number of function and gradient evaluations during the execution of our global search method. They make up the majority of the work. In many applications, the function or gradients are expensive to evaluate. Thus, evaluating functions and gradients at different points in parallel can significantly reduce execution time.

Overall, many aspects of parallelism can be explored in our global search method. Our method can effectively run on parallel computers and achieve near linear speedups.

3.11 Summary

In this chapter, we have presented a global search method that overcomes local minima in nonlinear optimization problems. This method consists of a global-search phase and a local-search phase. The global-search phase is further divided into a coarse-level search and a fine-level search. The coarse-level search identifies promising search regions to be searched more thoroughly by the fine-level search.

We have applied existing global-search methods, including simulated annealing, genetic algorithms, and multi-start of descents, in the coarse-level global search. These methods are executed for a limited amount of time to coarsely identify promising search regions. We have also developed a variation of the trajectory methods, which selects sample points based on a user-provided trajectory in the search space. The outputs of the coarse-level global search are used as initial points in the fine-level global search.

For the fine-level global search, we have proposed a new global-search method for continuous optimization problems, called trace-based global search. The trace-based search uses a problem-independent continuous trace to lead the search. It relies on two counteracting forces, the local gradient force that drives the search to a local optimum and a deterministic trace force that leads the search out of local minima. The trace force is expressed in terms of the distance between the current trajectory position and the position of the trace. It produces an information-bearing trajectory from which sample points can be identified as starting points for the local search.

We have studied the design of new trace functions, and have proposed trace functions that search a space from coarse to fine. Trace-based search is modeled by systems of ordinary differential equations (ODEs). We have developed adaptive variable scaling technique to overcome the difficulty caused by large gradients during global search, and enable the search trajectory to follow the trace closely.

Our trace-based method is efficient in the sense that it tries to first identify good starting points before applying a local search. Due to informed decisions in selecting good starting points, our trace-based search avoids many unnecessary efforts in re-determining already known regions of attraction, and spends more time in finding new unvisited ones. Its complexity is related to the number of "regions of attraction" rather than the number of dimensions.

Our trace-based method is also efficient because it provides a continuous means of going from one local region to another. This avoids problems in methods that determine new starting points heuristically as in evolutionary algorithms. When starting points are determined randomly, the search may miss many local minima between two starting points or may spend too much time in one region.

Our trace-based method is better than multi-start methods that perform blind sampling, as well as random search algorithms (such as genetic algorithms and simulated annealing) that do not exploit gradient information in their global search strategies. It incurs less overhead as compared to existing generalized gradient methods and other deterministic global search methods. It can be integrated into any existing local-search method.

In the local-search phase, existing local-descent methods have been applied. Gradient descent, quasi-Newton's, and conjugate gradient methods are efficient for continuous unconstrained problems. For continuous constrained problems, we apply Lagrangian methods with adaptive control to achieve faster and more robust convergence.

Finally, we have presented a prototype that implements our proposed methods for solving nonlinear optimization problems. The prototype is called Novel, an acronym for Nonlinear Optimization Via External Lead.

Novel combines methods for handling nonlinear constraints (proposed in Chapter 2) and for overcoming local minima (proposed in Chapter 3), and solves constrained and unconstrained problems in a unified framework. Constrained problems are first transformed into Lagrangian functions. Then, Novel performs global searches based on the objective functions of unconstrained problems and the Lagrangian functions of constrained problems, overcoming local minima in a deterministic and continuous fashion.

In the next part of this thesis, we show some very promising results in applying Novel to solve real-world applications. These applications are (a) neural network learning problems formulated as unconstrained optimization problems, (b) digital filter-bank designs formulated as nonlinear constrained optimization problems, (c) satisfiability and maximum satisfiability problems, and (d) multiplierless filter-bank designs formulated as discrete optimization problems.

4. UNCONSTRAINED OPTIMIZATION IN NEURAL-NETWORK LEARNING

In this chapter, we apply Novel in feed-forward neural-network learning. In general, such learning can be considered as a nonlinear global optimization problem in which the goal is to minimize a nonlinear error function that spans the space of weights and to look for global optima (in contrast to local optima). First, we introduce artificial neural networks, present the formulation of neural-network learning as an optimization problem, and survey existing learning algorithms. Then, using five benchmark problems, we compare Novel against some of the best global optimization algorithms and demonstrate its superior improvement in performance.

4.1 Artificial Neural Networks

The concept of artificial neural networks (ANNs) was inspired by biological neural networks. Biological neurons, believed to be the structural constituents of the brain, are much slower than silicon logic gates. The brain compensates for the relatively slower operation of biological neurons by having an enormous number of them that are massively interconnected. The result is that inferencing in biological neural networks is much faster than the fastest computer. In a word, a biological neural network is a nonlinear, highly parallel device characterized by robustness and fault tolerance. It learns by adapting its synaptic weights to changes in the surrounding environment.

Table 4.1: Von Neumann computer versus biological neural systems.

	Von Neumann Computer	Biological Neural System
Processor	Complex	Simple
	High speed	Low speed
	One or a few	A large number
Memory	Separate from a processor	Integrated into processor
	Localized	Distributed
	Not content addressable	Content addressable
Computing	Centralized	Distributed
	Sequential	Parallel
	Stored programs	Self-learning
Reliability	Very vulnerable	Robust
Expertise	Numerical computation and symbolic manipulations	Perceptual problems
Operating environment	Well-defined	Poorly defined
	Well-constrained	Unconstrained

Modern digital computers outperform humans in the domain of numeric computation and symbol manipulation. However, humans can solve complex perceptual problems at a much higher speed and broader extent than the world's fastest computer. The biological brain is completely different from the von Neumann architecture implemented in modern computers. Table 4.1 from [131] summarizes the difference.

ANNs attempt to mimic some characteristics of biological neural networks. In ANNs, information is stored in the synaptic connections. Each neuron is an elementary processor with primitive operations, like summing the weighted inputs and then amplifying or thresholding the sum.

Figure 4.1 shows a binary threshold unit proposed by McCulloch and Pitts as an artificial neuron. This neuron computes a weighted sum of n input signals, x_i, $i = 1, \cdots, n$, and passes it through an *activation function*, a threshold function in this example, to produce output y:

$$y = \theta\left(\sum_{i=1}^{n} w_i x_i - \sigma\right) \tag{4.1}$$

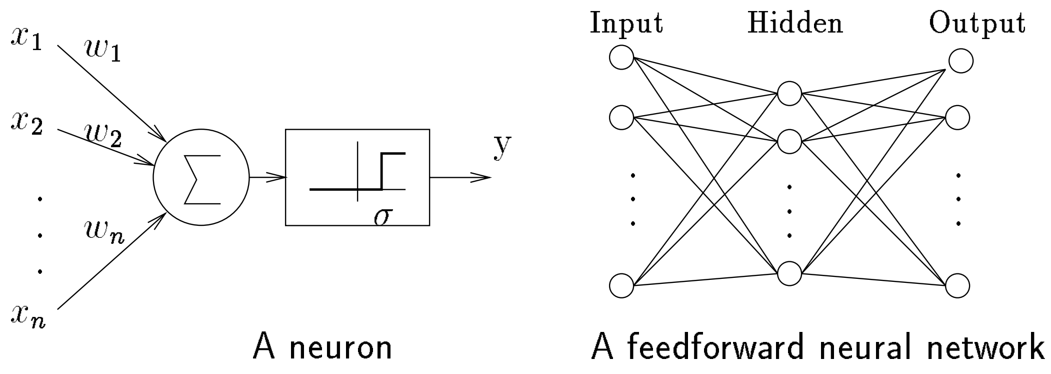


Figure 4.1: McCulloch-Pitts model of a neuron (left) and a three-layer feedforward neural network (right).

where $\theta(\cdot)$ is a step function and w_i is the synapse weight associated with the ith input. McCulloch and Pitts proved that suitably chosen weights let a synchronous arrangement of such neurons perform universal computations [163]. The McCulloch-Pitts neuron has been generalized in many ways. For example, other than threshold functions, activation functions have taken the forms of piecewise linear, sigmoid, or Gaussian functions.

A neural network is characterized by the network topology, the connection strength between pairs of neurons (weights), the node properties, and the state-updating rules. The updating rules control the states of the processing elements (neurons).

According to their architectures, neural networks can be classified into feedforward and feedback networks. In feedforward neural networks, connections between nodes (neurons) are uni-directional – from the input nodes to the output nodes through possibly several layers of hidden nodes. The most popular family of feed-forward networks are the *multilayer perceptrons*, in which neurons are organized into layers that have uni-directional connections between them. Figure 4.1 shows a typical structure of this type of network. On the other hand, in feedback neural networks, there are connections in the reverse direction, i.e., from the output nodes to the input nodes. These feedback connections feed information back from output nodes to input or hidden nodes.

An alternative classification of neural networks is based on learning paradigms, by which neural networks are classified into supervised-, reinforcement-, and unsupervised-learning networks.

In supervised learning, the target outputs are known ahead of time, and the error between the desired outputs and the actual outputs during learning is fed back to the network to adjust its weights. Reinforcement learning differs in that its feedback only distinguishes between correct and incorrect outputs but not the amount of error. In unsupervised learning, no target output is supplied during training, and the network self-organizes and classifies the training data on its own.

In supervised learning of both feedforward and feedback neural networks, the learning can be formulated as an unconstrained nonlinear optimization problem. Despite various forms of error functions, such as mean-square error, arctangent error, and cross-entropy error, the function to be optimized is usually highly nonlinear with many local minima.

In reinforcement learning, the gradients of the optimization criterion with respect to network weights is estimated instead of computed. Since this information is inaccurate and noisy, learning is usually carried out by stochastic learning methods and does not involve traditional numerical optimization.

Unsupervised learning methods use a pre-specified internal optimization criterion to capture the quality of a neural network's internal representation. For certain problems whose outputs are known ahead of time, for instance, encoding and clustering, unsupervised learning can be formulated as optimization problems. For feedback networks such as Hopfield networks, the network designed is for a specific function it implements and does not involve optimization.

4.2 Feedforward Neural-Network Learning Problems

Artificial neural nets have found many successful applications ranging from finger-print recognition to intelligent cameras. Neural-network design (learning) is a difficult task, and current design methods are far from optimal.

Learning in neural networks – whether supervised, reinforcement, or unsupervised – is accomplished by adjusting weights between connections in response to the inputs of training patterns. In feed-forward neural networks, this involves mapping from an input space to an output space. That is, output O of a neural network can be defined as a function of inputs I and connection weights W: $O = \phi(I, W)$, where ϕ represents a mapping function.

Supervised learning involves finding a good mapping function that maps training patterns correctly as well as to generalize the mapping found to test patterns not seen in training [227]. This is usually done by adjusting weights W while fixing the topology and the activation function. In other words, given a set of training patterns of input-output pairs $\{(I_1, D_1), (I_2, D_2), \cdots, (I_m, D_m)\}$ and an error function $\epsilon(W, I, D)$, learning strives to minimize learning error $E(W)$:

$$\min_W E(W) = \min_W \sum_{i=1}^{m} \epsilon(W, I_i, D_i). \tag{4.2}$$

One popular error function is the squared-error function in which $\epsilon(W, I_i, D_i) = (\phi(I_i, W) - D_i)^2$. Since this error function is non-negative, i.e. $E(W) \geq 0$, if there exists W' such that $E(W') = 0$, then W' is a global minimizer; otherwise, the W that gives the smallest $E(W)$ is the global minimizer. The quality of a learned network is measured by its error on a given set of training patterns and its (generalization) error on a set of test patterns.

Supervised learning can be considered as an unconstrained nonlinear minimization problem in which the objective function is defined by the error function and the search space is defined by the weight space. Unfortunately, the terrain modeled by the error function in its weight space can be extremely rugged and has many local minima. This phenomenon is illustrated in Figure 1.3, which shows two graphs of a 33-D error surface projected to two different pairs of dimensions around a global minimum. The left graph shows a rugged terrain with a large number of small local minima in the weight space, whereas the right one shows a distinctive terrain with large flat regions and steep slopes. This 33-D error surface corresponds to a five hidden-unit network that has been trained to solve the two-spiral problem (to be discussed in Section 4.4.2). Obviously, a search method that cannot escape from a local minimum will have difficulty in finding the global minimum in the rugged weight space.

To see how the network structure can affect the solution quality, consider the fact that one hidden-layer perceptron can perform arbitrary mappings as long as the number of hidden units is large enough. The capacity of a neural network and the number of weights increase as the number of hidden units increases. This means more global minimizers exist in the error surface, and local-search methods such as gradient descent can find them more readily. On the other hand, when the number of hidden units is small, there are fewer global minimizers. Moreover, the error surface can be extremely rugged, making it difficult for a descent method from a random starting point to find a good solution. To overcome this problem, more powerful global search methods are needed.

There are many benefits in using smaller neural networks. First, they are less costly to implement and are faster, both in hardware and in software implementations. Second, they generalize better because they avoid over-fitting the weights to the training patterns. This happens because we are using a lower-dimensional function to fit the training patterns.

4.3 Existing Methods for Feedforward Neural Network Learning

As we know, the supervised learning of feedforward neural networks is an unconstrained continuous optimization problem. The task is to find the appropriate values of connection weights so that a given error function is minimized. Optimization problems are classified into uni-modal and multi-modal, depending on whether there is only one or there are more than one local minima in the search space. In general, neural network learning is a multi-modal nonlinear minimization problem with many local minima.

Our study of error functions in the supervised learning of neural networks reveals the following features:

- Flat regions may mislead gradient-based methods.
- There may be many local minima that trap gradient-based methods.

- Deep but suboptimal valleys may trap any search method.
- Gradients may differ by many orders of magnitude, making it difficult to use gradients during a search.

A good search method should, therefore, have mechanisms (a) to use gradient information to perform local search (and be able to adjust to changing gradients) and (b) to escape from a local minimum after getting there.

Learning algorithms of neural networks find their roots in optimization methods, which are classified into local or global optimization. Local optimization algorithms, such as gradient-descent and Newton's method, find local minima efficiently and work best in unimodal problems. They execute fast, but converge to local minima that could be much worse than the global minimum. Global optimization algorithms, in contrast, employ heuristic strategies to look for global minima and do not stop after finding a local minimum [161,254]. Next, we briefly review local- and global-optimization algorithms in neural-network learning.

4.3.1 Learning Using Local Optimization

Many local-optimization algorithms, such as gradient descent, second-order gradient descent, quasi-Newton, and conjugate gradient methods, have been adapted to the learning of neural networks. The popular back-propagation (BP) algorithm is a variation of gradient descent algorithms [155, 210]. Other gradient-descent type algorithms include BP through time [209], recurrent BP [196], steepest descent, and other variations [13, 113, 191, 194, 275]. Second-order gradient descents, including BP with momentum and BP with dynamic learning rate, have been developed to improve the convergence time of BP [58, 74, 130, 189, 280]. Also, quasi-Newton's methods have shown good speedup over BP [22, 49], whereas conjugate gradient methods are among the fastest [16, 17, 67, 143, 170]. Other heuristic methods that have fast learning speed include methods that learn layer by layer [70], iterative methods [9], hybrid learning algorithms [97], and methods developed from the field of optimal filtering [213, 233]. Recurrent neural networks have also been trained by gradient-based methods [26, 179, 197, 274].

Local minimization algorithms have difficulties when the surface is flat (gradient close to zero), when gradients can be in a large range, or when the surface is very rugged. When gradients can vary greatly, local search may progress too slowly when the gradient is small and may over-shoot when the gradient is large. When the error surface is rugged, a local search from a randomly chosen starting point will likely converge to a local minimum close to the initial point and a solution worse than the global minimum. Moreover, these algorithms usually require choosing some parameters, as incorrectly chosen parameters may result in slow convergence and poor quality solutions.

4.3.2 Learning Using Global Search

In order to overcome local minima in the search space and find better solutions, global minimization methods have been developed. They use local search to determine local minima, and focus on bringing the search out of a local minimum once it gets there. Many global optimization algorithms have been applied in neural-network learning. Examples of deterministic methods are trajectory methods [43] and branch-and-bound methods [124,249]. The deficiency of these methods is that they can only solve small networks.

Stochastic global optimization methods are more successful in solving the learning problem of neural networks. These include random search algorithms [10, 35, 198], simulated annealing [68, 230], and genetic algorithms [95, 114, 171, 273]. They have shown improved performance with respect to local optimization algorithms. However, their execution time can be significantly longer.

One thing that needs to be mentioned is that neural network learning has also been formulated as a combinatorial optimization problem and solved by combinatorial optimization methods, such as reactive tabu search [20, 92].

4.4 Experimental Results of Novel

In this section, we present the experimental results of Novel in solving five neural network benchmark problems and show its improvement in performance against some of the

Table 4.2: Benchmark problems studied in our experiments. The numbers of inputs, outputs, and training and test patterns are listed in the table.

Problems	# inputs	# outputs	# training patterns	# test patterns
Two-spiral	2	1	194	194
Sonar	60	1	104	104
Vowel	10	11	528	462
10-parity	10	1	1024	0
NetTalk	203	26	5438	146762

best global-optimization algorithms [227, 264]. First, we present the neural network problems used in our experiments. Using the two-spiral problem as an example, we study the implementation issues of Novel and compare Novel with other global optimization methods. Finally we present the summary of experimental results on the other benchmark problems. In general, Novel is able to find better results as compared to other global optimization algorithms in the same amount of time.

4.4.1 Benchmark Problems Studied in Our Experiments

Table 4.2 summarizes the benchmark problems studied in our experiments. They represent neural network learning problems of different size and complexity. These problems can be obtained from ftp.cs.cmu.edu in directory /afs/cs/project/connect/bench.

(1) **Two-spiral problem.** This problem discriminates between two sets of training points that lie on two distinct spirals in the x-y plane. Two spirals coil three times around the origin and around one another. Problems like this one, whose inputs are points on the 2-D plane, enable us to display the output of the network in a 2-D graph. The two-spiral problem appears to be a very difficult task for back-propagation networks.

Figure 4.2 shows the training and test sets. Each data set has two spirals that consist of 97 input-output pairs, respectively. Each training or testing pattern represents a point on the corresponding spiral. The neural network that solves the two-spiral problem has

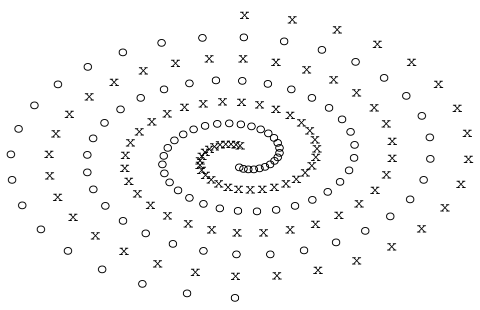
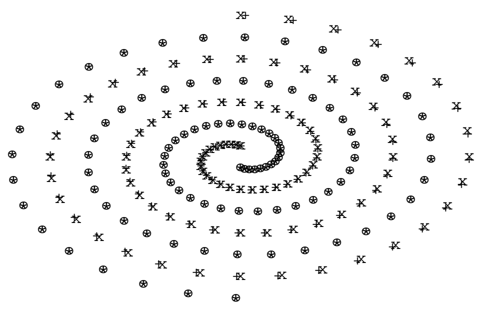



Figure 4.2: The training set of the two-spiral problem (left) and the test set of the two-spiral problem on top of the training set (right).

two real-valued inputs presenting the x and y coordinates of a point, and one binary target output that classifies the point to one of the two spirals.

In general, we determine the correctness of a binary output using the 40-20-40 criterion: an output is considered to be 0 if it is in the lower 40% of the real-valued output range, 1 if it is in the upper 40%, and incorrect if it is in the middle 20% of the range.

(2) **Sonar problem.** This problem discriminates between sonar signals bounced off a metallic cylinder and those bounced off a roughly cylindrical rock. This is the data set used by Gorman and Sejnowski in their study of the classification of sonar signals using a neural network [96]. We used the training and test samples in their "aspect angle dependent" experiments. The problem has sixty real-valued inputs and one binary output.

(3) **Vowel recognition problem.** This problem trains a network to have speaker-independent recognition of the eleven steady-state vowels of British English. The problem was used by Tony Robinson in his Ph.D. thesis to study problems that have no exact solution and maximize a less than perfect performance of neural networks [206].

This problem has ten real-valued inputs and eleven binary outputs. Vowels are classified correctly when the distance of the correct output to the actual output is the smallest of the set of distances from the actual output to all possible target outputs.

(4) **10-parity problem.** This problem trains a network that computes the modulo-two sum of ten binary digits. It is a special case of the general parity problem, in which the task is to train a network to produce the sum, modulo 2, of N binary inputs. The parity problem has N binary inputs and one binary output. The output is 1 if there are an odd number of 1 bits in the input, and 0 if an even number. In the 10-parity problem, there are 1,024 training patterns and no test patterns.

(5) **NetTalk problem.** This problem trains a network to produce proper phonemes, given a string of letters as input. This is an example of an input/output mapping task that exhibits strong global regularities, but also a large number of more specialized rules and exceptional cases. The data set of NetTalk was contributed by Terry Sejnowski. It contains 20,008 English words along with a phonetic transcription for each word.

We have used (a) the same network settings and unary encoding as in Sejnowski and Rosenberg's experiments [220]. There are $29 \times 7 = 203$ inputs and 26 outputs; (b) 1,000 most common English words as the training set and the entire data set as the test set; and (c) the "best-guess" criterion: an output is treated as correct if it is closer with the smallest angle to the correct output vector than to any other phoneme output vector. There are a total of 5,438 training patterns.

4.4.2 Experimental Results on the Two-spiral Problem

The two-spiral problem has been used extensively in testing different types of neural networks and their corresponding learning algorithms. Published results include the learning of feed-forward networks using BP, CASCOR [75], and projection pursuit learning [128]. The smallest network is believed to have nine hidden units with 75 weights trained by CASCOR.

In our experiments, we have used feed-forward networks with shortcuts (see Figure 4.3.) Each hidden unit is ordered and labeled by an index, and has incoming connections from all input nodes and from all hidden units with smaller indexes. The activation function is an

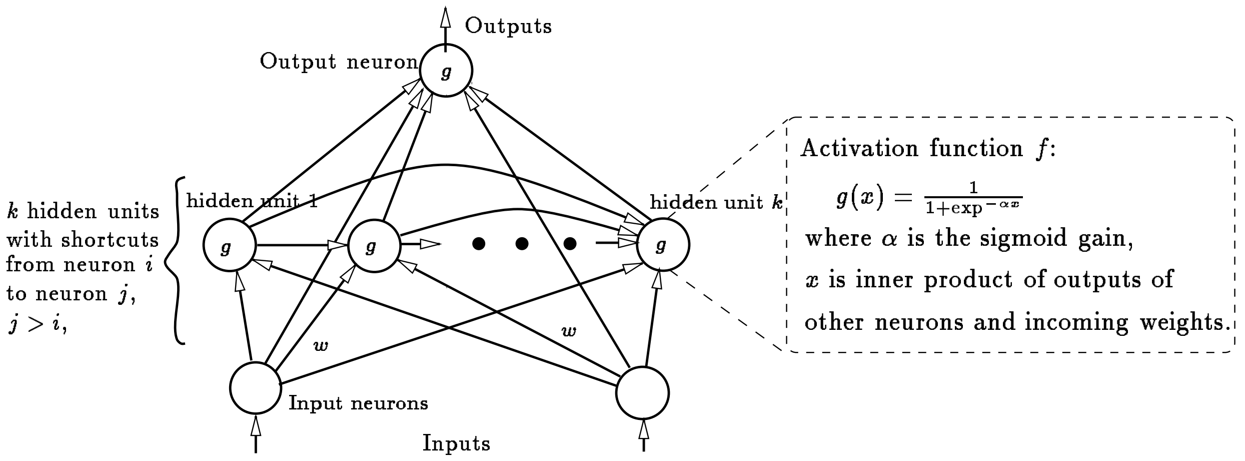

Outputs
Output neuron
g
k hidden units with shortcuts from neuron i to neuron j, $j > i$,
hidden unit 1
hidden unit k
g
g
g
w
w
Input neurons
Inputs
Activation function f:
$g(x) = \frac{1}{1+\exp^{-\alpha x}}$
where α is the sigmoid gain,
x is inner product of outputs of
other neurons and incoming weights.

Figure 4.3: Structure of the neural network for the two-spiral problem.

asymmetric sigmoidal function $g(x) = 1/(1 + e^{-\alpha x})$, where α is the sigmoid gain, and x is the inner product of the outputs of other neurons and the incoming weights.

We have fixed the search range as $[-1, 1]$ in each dimension, and have varied α from 1 to 150. A larger α has the effect of compressing the search regions.

The objective function to be minimized is the total sum of squared error (TSSE), $f(\mathbf{x}) = E(\mathbf{x})$, defined in (4.2), where the variable vector $\mathbf{x}$ represents the weights W. The error function $f(\mathbf{x})$ is computed based on the training patterns during learning. Test patterns are not used in the objective functions in learning, and are only used to check the results obtained.

The global-search phase of Novel consists of three cascaded stages. Each stage is modeled as a system of ODEs in Eq. (3.10), which is re-stated as follows:

$$\frac{d\mathbf{x}(t)}{dt} = -\mu_g \nabla_{\mathbf{x}} f(\mathbf{x}(t)) - \mu_t \left(\mathbf{x}(t) - T(t)\right)$$

where μ_g and μ_t are constant coefficients and are the same for all stages. The systems of ODEs is solved by LSODE using the method based on a backward differentiation formula. The local-search phase of Novel performs gradient descents. The gradient descent is also

Table 4.3: Results of using NOVEL with the LSODE solver to train four and five hidden-unit neural networks for solving the two-spiral problem.

Sigmoid Gain	4 Hidden-Unit Networks (25 weights)				
	Coefficients		Training		Testing
α	μ_g	μ_t	TSSE	Correct %	Correct %
100	**1**	**20**	**6.0**	**96.9**	**91.8**
	0.1	2	8.2	95.7	90.7
100	0.5	10	17.0	91.8	84.5
	2	40	18.1	90.7	83.5
50	1	20	6.2	96.9	92.3
30	1	20	15.8	89.2	87.6
	Maximum time units executed: 200 Avg. CPU time/time unit: 2 min.				
	5 Hidden-Unit Networks (33 weights)				
	Coefficients		Training		Testing
	μ_g	μ_t	TSSE	Correct %	Correct %
100	**1**	**20**	**0.0**	**100**	**95.9**
	0.1	2	2.0	98.5	96.4
100	0.2	4	8.9	95.4	89.7
	2	40	0.0	100.0	95.4
50	1	20	13.3	93.3	88.1
30	1	20	0.1	100.0	96.9
	Maximum time units executed: 100 Avg. CPU time/time unit: 5 min.				

modeled as a system of ODEs:

$$\frac{d\mathbf{x}(t)}{dt} = -\nabla_{\mathbf{x}} f(\mathbf{x}(t)) \tag{4.3}$$

and solved by solver LSODE. All our experiments were carried out on Sun SparcStation 20 model 71 (75 MHz) workstations.

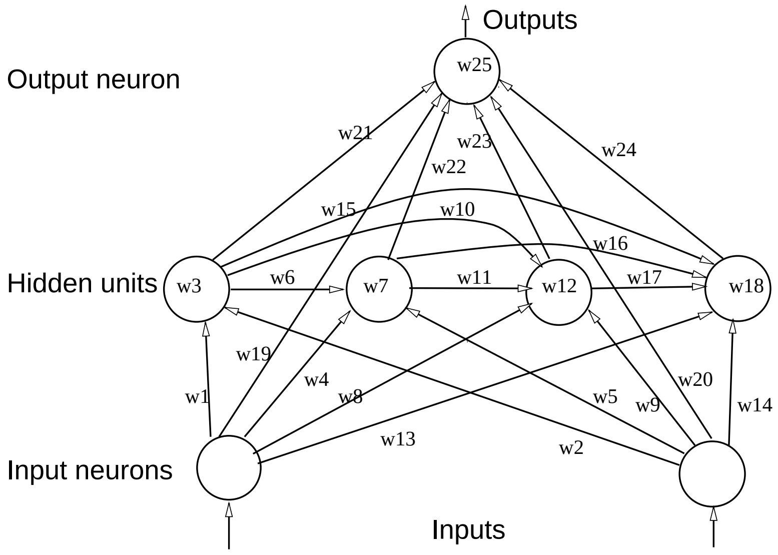
We have tried various combinations of algorithmic parameters α, μ_t, and μ_g in Novel. In Table 4.3, we show the results in training 4 and 5 hidden-unit networks to solve the two-spiral problem. The sigmoid gain α takes on values 30, 50, and 100, while the weight coefficients μ_t and μ_g are scaled up and down around baseline values of 20 and 1, respectively.

When μ_g and μ_t are both scaled down from the baseline values by the same magnitude, for instance $\mu_g = 0.1$ and $\mu_t = 2$, the speed of trace movement is slowed down. A slower trace takes more logical time and usually more CPU time to traverse the same distance than a faster one. One the other hand, when μ_g and μ_t are both scaled up, for instance $\mu_g = 2$ and $\mu_t = 40$, the trace moves faster. The danger is that if the trace moves too fast, the search trajectory generated from the ODE dynamic system will not be able to keep up with the trace. Therefore, μ_g and μ_t should be chosen appropriately so that the search trajectory moves at a fast speed, but under the speed limit of the systems of ODEs.

After trying various values of α, μ_g, and μ_t for the 4 and 5 hidden-unit networks, we found that the combination of $\mu_g = 1$, $\mu_t = 20$, and $\alpha = 100$ works well. These parameter values were then used in our experiments to solve the learning problems of 3 and 6 hidden-unit neural networks.

Table 4.3 shows that multiple solutions with 100% correctness on the training patterns are found for the 5 hidden-unit network using different parameter values. These solutions have different correct percentages on testing patterns. For the 4 hidden-unit network, the solutions do not reach 100% correctness on the training set. Generally, larger networks are easier to be trained to solve the training patterns 100% correctly because there are more global minimizers in the search space. From a function fitting point of view, a neural-network learning problem is to fit a set of training data by finding the appropriate weights of a neural network. A function with more variables, corresponding to a neural network with more weights, has more ways to fit a given training set than a function with fewer variables.

Novel successfully trained five hidden-unit networks in less than 100 time units. Training four hidden-unit networks is more difficult. After running Novel for 800 time units, which was 77.48 hours of CPU time on a SparcStation 20/71, we found a solution with TSSE of 2.1 and 99% correct on the training set. Using this solution as a new starting point, we executed Novel for another 89.44 hours and found a solution that is 100% correct. Figure 4.4 shows the four hidden-unit neural network trained by Novel that solves the two-spiral problem. The second figure in the first row of Figure 4.5 shows how the best four hidden-unit network classifies the 2-D space.



Synapse weights $\mathbf{w} = (w_1, w_2, \cdots, w_{25})$:

$$\left(\begin{array}{rrr}
0.22421415277833 & -9.9727673403983\text{E-}3 & -1.1806183150415\text{E-}2 \\
-3.0436258375969\text{E-}4 & -5.3835766352018\text{E-}3 & -3.5552190056235\text{E-}2 \\
1.7724121526051\text{E-}2 & 2.5025067498459\text{E-}2 & 2.7964728810376\text{E-}2 \\
0.12502732130251 & 0.47111370688417 & -0.29785571990963 \\
-0.13453471734405 & -0.12901696725105 & -0.64470248181304 \\
-2.2045004942827 & 0.70324080842905 & 1.0717580696294 \\
-0.30974130440544 & -0.32215585454022 & -1.5130472274848 \\
-5.4716749250999 & 4.1243107101010 & -2.9432867012179 \\
2.9004386193913 & &
\end{array}\right)$$

Figure 4.4: A four hidden-unit neural network trained by Novel that solves the two-spiral problem.

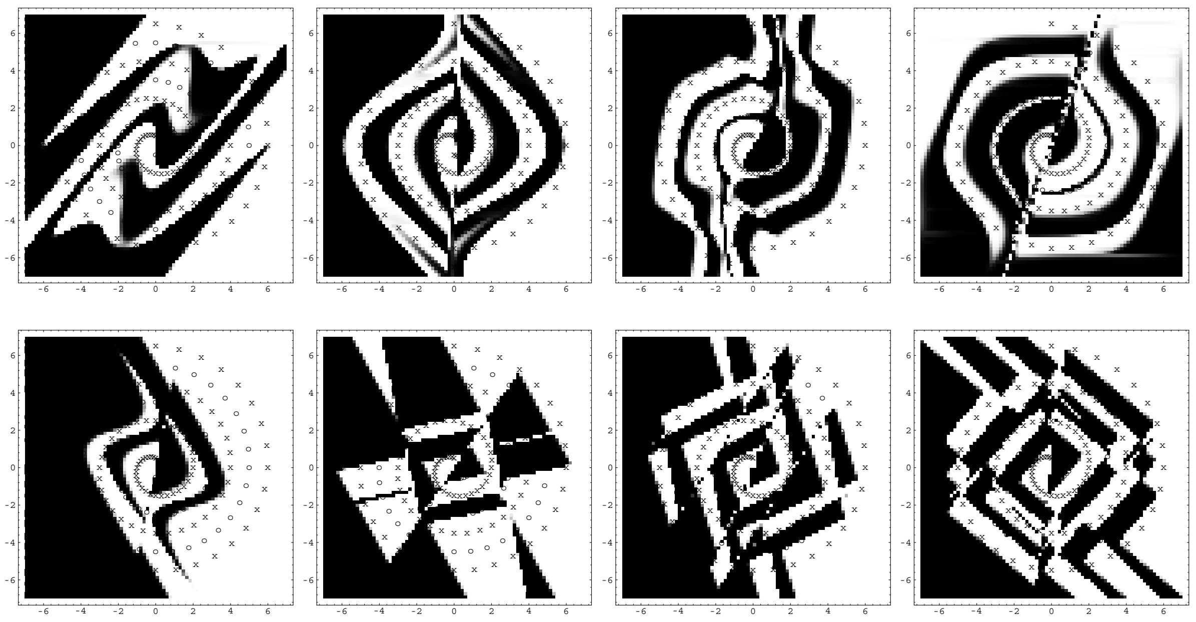


Figure 4.5: 2-D classification graphs for the two-spiral problem by 3 (first column), 4 (second column), 5 (third column), and 6 (fourth column) hidden-unit neural networks trained by Novel (upper row) and SIMANN (lower row). Parameters for Novel are $\mu_g = 1, \mu_t = 20$, and $\alpha = 100$, and search range $[-1.0, 1.0]$. Parameters for SIMANN are $RT = 0.99, NT = 5n$, and search range $[-2.0, 2.0]$. The crosses and circles represent the two spirals of training set. Note that the 4 hidden-unit network was trained by Novel differently from the other networks.

Novel has found optimal designs for the two-spiral problem, which correspond to 100% correct on the training set, for the 4, 5 and 6 hidden-unit neural networks. The best design for the 3 hidden-unit network is 85.5% correct on the training set. The top four graphs in Figure 4.5 show the 2-D classifications by the best 3, 4, 5, and 6 hidden-unit networks found by Novel.

Besides solving the system of ODEs in the global-search phase using LSODE, we have also tried the simpler, less accurate, but faster the Euler method as specified in (3.25). By using fixed step sizes, the Euler method advances in discrete points quickly without worrying about the stiffness of the ODE system. The drawback of the Euler method is that it has no error estimation and accuracy guarantee. In our experiments, we study the tradeoff of the solution quality with respect to the accuracy in generating the search trajectory.

An advantage of the Euler method is that it can work with the pattern-wise learning mode as well as the epoch-wise learning mode. In pattern-wise learning, variables (weights) are updated after every training pattern is presented, while in epoch-wise learning, variables are updated after all the training patterns are presented. Accordingly, the gradients of the objective function are calculated based on every training pattern in pattern-wise learning. By updating the weights more frequently, pattern-wise learning algorithms, such as BP with pattern-wise learning, have been able to converge faster than epoch-wise learning algorithms.

The pattern-wise learning mode introduces large variations of gradient values because they are calculated based on a single training pattern. In Novel, when we use pattern-wise learning and use LSODE to solve the system of ODEs in generating the search trajectories, the rapid change of gradients makes LSODE fail. In contrast, the Euler method uses fixed step sizes, and tolerates the changes of gradients well.

Hence, in our experiments, LSODE is only used with epoch-wise learning mode, where all the training patterns are presented before the gradient is calculated and variables are updated. The Euler method can be used with both epoch-wise and pattern-wise learning modes.

To measure the performance of both the epoch- and the pattern-wise learning, we use the mean-squared error (MSE) as the objective function to be minimized. MSE is the total sum-squared error (TSSE) divided by the number of patterns used in calculating the values of the error function and its gradient.

We have fixed the step size of 0.01 in the Euler method, and varied the values of parameters α, μ_t, μ_g. In epoch-wise training, we have tested four pairs of coefficients: $\mu_g = 0.001$ and $\mu_t = 0.01$; $\mu_g = 0.001$ and $\mu_t = 0.1$; $\mu_g = 0.01$ and $\mu_t = 0.1$; and $\mu_g = 0.01$ and $\mu_t = 0.1$. We have tried the following five values of sigmoid gain α: 1, 10, 30, 50 and 100. Table 4.4 presents the combination of parameters leading to the best results of Novel using the Euler method and epoch-wise training. Our results show that the Euler method is about ten

Table 4.4: Summary results of Novel using the Euler method to solve the two-spiral problem. The limit of time unit in each run is 400.

No. of hidden units	No. of weights	Sigmoid gain α	Coefficients		Best Solution			CPU time /time unit (minutes)
					Training		Testing	
			μ_g	μ_t	SSE	Correct %	Correct %	
4	25	50	0.01	0.1	14.0	92.8	85.6	0.20
5	33	50	0.001	0.01	6.0	96.9	94.8	0.36
6	42	10	0.01	0.1	0.0	100	95.4	0.55

times faster than LSODE in solving the system of ODEs. However, the quality of solutions obtained by using the Euler method is worse.

In using the Euler method together with pattern-wise training, we need to set the step size to be smaller, e.g. 0.001 and 0.0001, to get good solutions. Results obtained by using pattern-wise training are worse than those obtained by using epoch-wise training. For example, the best results for 5 hidden-unit networks are 93.8% and 96.9% for pattern-wise and epoch-wise training, respectively.

4.4.3 Comparison with Other Global Optimization Methods

In this subsection, we compare the performance of Novel with that of other global optimization methods for solving the two-spiral problem. These algorithms include simulated annealing, evolutionary algorithms, cascade correlation with multi-starts (CASCOR-MS), gradient descent with multi-starts (GRAD-MS), and truncated Newton's method with multi-starts (TN-MS). We first describe these methods as follows:

(a) **Simulated annealing** (SA) — SA is a stochastic global optimization method. Starting from an initial point, the algorithm takes a step and evaluates the function. When minimizing a function, any down-hill movement is accepted, and the process repeats from this new point. An uphill movement may be accepted, and by doing so it can escape from local minima. This uphill decision is made by the Metropolis criteria. As the minimization process proceeds, the length of the steps decreases and the probability of accepting uphill movements decreases as well. The search converges to a local (sometimes global) minimum at the end. The software package we used is SIMANN from netlib at http://www.netlib.att.com.

(b) **Evolutionary algorithm** (EA) — EA is based on the computational model of evolution. A variety of EAs have been proposed in the past, among which are genetic algorithms, evolutionary programming, and evolutionary strategies. EAs maintain a population of individual points in the search space, and the performance of the population evolves to be better through selection, recombination, mutation, and reproduction. The fittest individual has the largest probability of survival. EAs have been applied to solve complex, multimodal minimization problems with both discrete and continuous variables. The packages we used in our experiments are GENOCOP (GEnetic algorithm for Numerical Optimization for COnstrained Problems) [164] and LICE (LInear Cellular Evolution) [245].

(c) **Cascade correlation with multi-starts** (CASCOR-MS) — The cascade correlation learning algorithm is a constructive method that starts from a small network and gradually builds a larger network to solve the problem. This algorithm was originally proposed by Fahlman and Lebiere [75] and has been applied successfully to some neural-network learning problems. In CASCOR-MS, multiple runs of CASCOR are executed from randomly selected initial points.

(d) **Gradient descent with multi-starts** (GRAD-MS) — The gradient-descent algorithm is simple and popular, and its variants have been applied in many engineering applications. An example is the back-propagation algorithm in neural network learning. For the two-spiral problem, gradient descent is performed by solving a system of ODEs modeling gradient descent.

(e) **Truncated Newton's method with multi-starts** (TN-MS) — The truncated Newton's method uses second-order information that may help convergence in descents. They are faster than gradient descents solved by LSODE. The software package we used is TN from netlib.

Next, we compare the performance of Novel with that of these methods. To allow a fair comparison, we ran all these methods for the same amount of time using the same network structure. We used sigmoid gain $\alpha = 100$ for all algorithms except CASCOR-MS and TN-MS, which have $\alpha = 1$. The CPU time allowed for each experiment was 20 hours on a Sun 20/71.

The simulated annealing program used in our experiments is SIMANN from netlib [53] with some modifications by Goffe, Ferrier, and Rogers presented in [94]. We experimented with various temperature scheduling factors RT, function evaluation factors NT, and search ranges. The best results were achieved when $RT = 0.99$, $NT = 5n$ (n is the number of variables), and the search range is $[-2.0, 2.0]$.

We have studied two evolutionary algorithms (EAs): GENOCOP by Michalewicz and LICE by Sprave. GENOCOP aims at finding a global minimum of an objective function under linear constraints. We have tried various search ranges and population sizes. Search range $[-0.5, 0.5]$ and population size $100n$ give the best results. LICE is a parameter optimization program based on evolutionary strategies. In applying LICE, we have tried various initial search ranges and population sizes. Range $[-0.1, 0.1]$ and population size $100n$ give the best results.

In applying CASCOR-MS, we ran Fahlman's CASCOR program [75] from random initial weights in the range $[-1, 1]$. We started from a new starting point when the current run did not result in a converged network for a maximum of 3, 4, 5 and 6 hidden units, respectively.

In GRAD-MS, we generated multiple random initial points in the range $[-0.2, 0.2]$. Gradient descent is modeled by a system of ODEs and is solved using LSODE. One descent is usually fast. The CPU time limit determines how many descents are executed.

Finally, we have used truncated Newton's method obtained from netlib with multi-starts (TN-MS). We generated random initial points in the range $[-1, 1]$, and set the sigmoid gain to 1. Since one run of TN-MS is very fast, a large number of runs were done within the time limit.

The best performance of these algorithms is shown in Figure 4.6, in which the four graphs show the progress of all the learning algorithms for the 3, 4, 5, and 6 hidden-unit networks, respectively. The plots for Novel, SA, and EAs show the progress of the best run. The plots for other multi-start algorithms show the improvement of the incumbent, the best solution

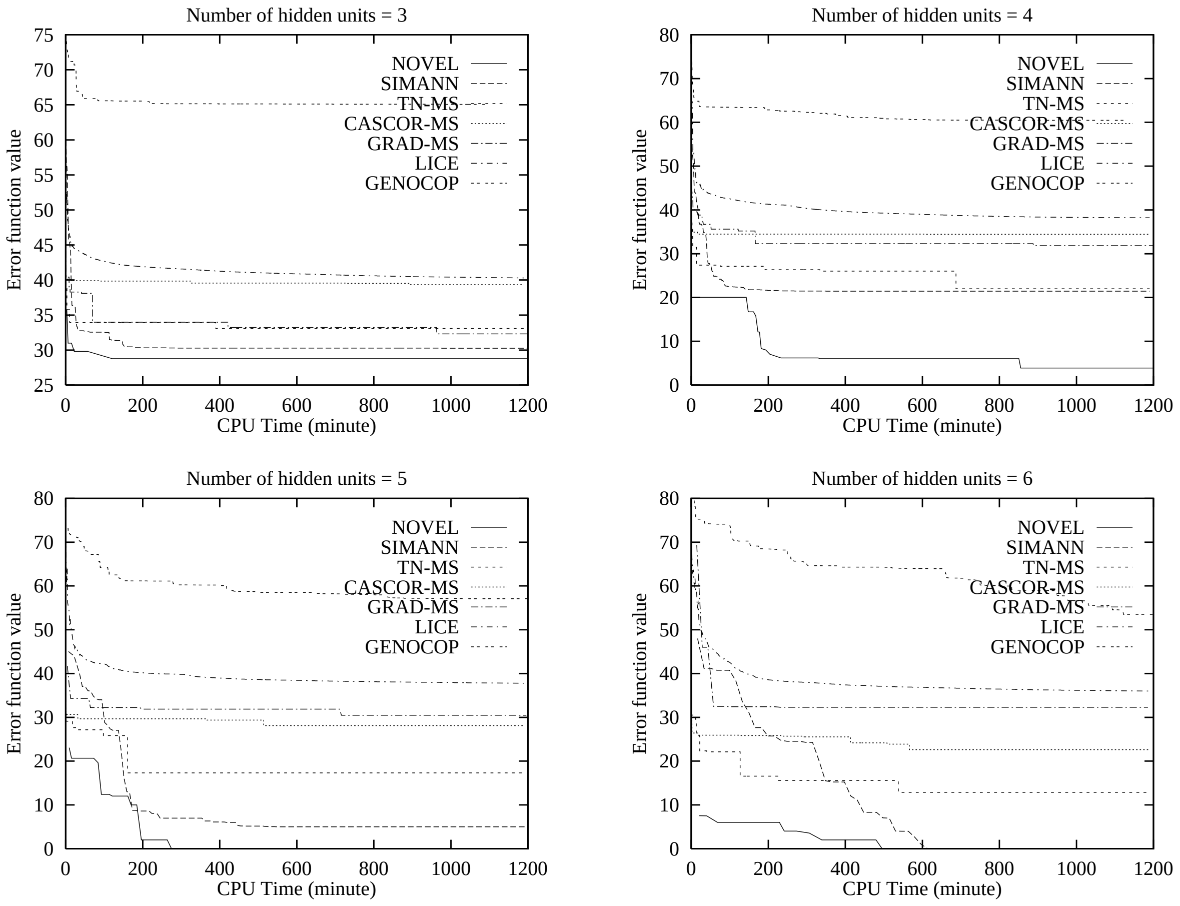


Figure 4.6: The best performance of various global minimization algorithms for learning the weights of neural networks with 3, 4, 5, and 6 hidden units for solving the two-spiral problem.

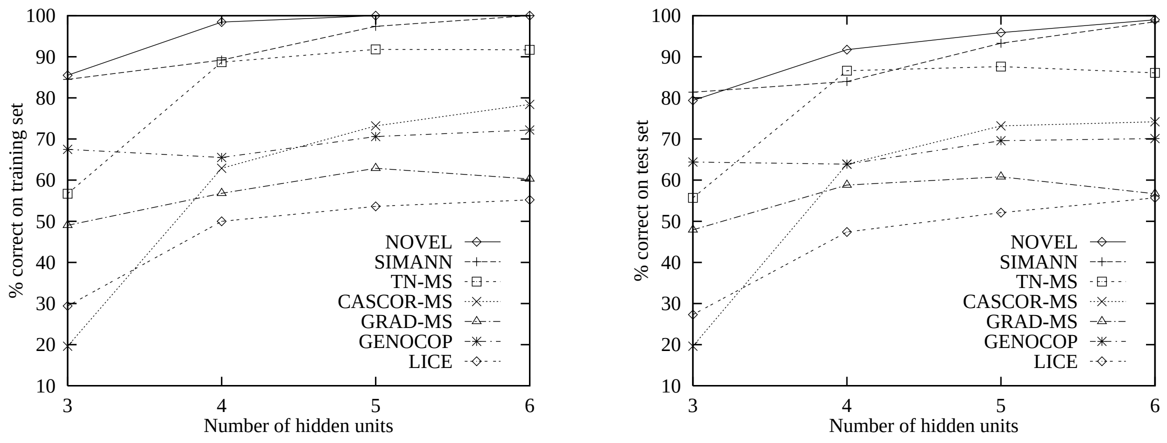


Figure 4.7: Training and testing errors of the best designs obtained by various algorithms for solving the two-spiral problem. There are 18, 25, 33, and 42 weights (including biases in neurons) in the neural networks with, respectively, 3, 4, 5, and 6 hidden units.

so far, until the CPU time limit is reached. The general trend of all these algorithms is that the solutions improve quickly in the beginning. Then, the improvement is slowed down as time goes on. Later, the values of the objective functions obtained by these algorithms stabilize at different levels.

Figure 4.7 summarizes the training and test results of the best solutions found by each of these algorithms when ran under 20 hours of CPU time on a Sun SparcStation 20/71. The graphs show that Novel has the best training and test results for the neural networks found, followed by SIMANN, TN-MS, CASCOR-MS, and the two evolutionary algorithms. The two evolutionary algorithms do not work well because genetic operators like mutation and cross-over do not utilize local gradient information in deciding where to search.

Simulated annealing algorithm has the best results among the algorithms we have tested except Novel. Table 4.5 shows the best solutions obtained by Novel and SA. For 3 hidden-unit networks, Novel has smaller training errors, and SA has smaller test error. For 4 and 5 hidden-unit networks, Novel is much better than SA. Both found the global minima of the training set for 6 hidden-unit networks.

Table 4.5: The performance of the best networks with 3, 4, 5, and 6 hidden units, which were trainned by NOVEL and simulated annealing (SA).

No. of hidden units	SSE of Training		Correct % of Training		Correct % of Testing	
	NOVEL	SA	NOVEL	SA	NOVEL	SA
3	28.8	30.2	85.5%	84.5%	79.4%	81.4%
4	0.2	28.0	100%	85.5%	94.8%	84.0%
5	0.0	5.0	100%	97.4%	95.9%	93.3%
6	0.0	0.0	100%	100%	99.0%	98.5%

Figure 4.5 shows the 2-D classification graphs for the two-spiral problem by the best solutions of Novel and SIMANN. Novel's results are in the upper row and SIMANN's, the lower row. The two graphs in the first column are classification graphs by 3 hidden-unit networks. The graphs in the second, third, and fourth columns are by 4, 5, and 6 hidden-unit networks, respectively. Training patterns are super-imposed in these graphs, where circles belong to one spiral and crosses belong to the other one. When all the circles are in the black region and all the crosses are in the white region, the classification is 100% percent correct. As shown in the graphs, the solutions found by Novel have smooth and curved classification region, whereas results by SIMANN show ragged classifications.

Novel has found one optimal solution for 4 and 5 hidden-unit networks, and have found many optimal solutions for 6 hidden-unit networks with different test performance. Their correct percentages on the test set range from 88.6% to 99.0%. This means that as network size increases, more solutions exist to solve the training set optimally. Also, when network size is increased from 4 hidden units to 6 hidden units, the best generalization correctness increase from 94.8% to 95.9%, and then 99.0%.

The experimental results show that a learning algorithm's performance depends on the complexity of the error function. When the error function is not complex, as in optimizing the weights of a 3-hidden-unit network, Novel as well as other algorithms like SA can find a good minimum in a small amount time. When the error function is complex and good solutions are few, Novel performs much better than other algorithms. When the terrain of the optimized function is rugged, Novel has the ability to search continuously and can identify deep and narrow valleys. Further, Novel is able to descend to the bottom of local minima precisely because it formulates gradient descent as a system of ODEs and solves them accurately.

4.5 Experimental Results on Other Benchmarks

In this section, we show our results in applying Novel on four more benchmark problems. These are sonar, vowel-recognition, 10-parity, and NetTalk problems. They represent classification problems of different complexity and characteristics. The network topologies used in these experiments are multilayer perceptrons – layered feed-forward networks without shortcuts. The objective function is the mean-squared error (MSE). Other setups are similar to those for the two-spiral problem.

Neural networks for solving these problems are larger than the ones used for the two-spiral problem and have more weights (variables). When the number of variables is large, it becomes slow for LSODE to solve the system of ODEs in Novel. Therefore, we apply the Euler method to solve the ODEs of global search. Local search is performed by local optimization methods with fast convergence speed, such as truncated Newton's method (TN) and the back propagation method (BP).

For the sonar problem, we have applied Novel with the Euler method, TN-MS, SIMANN, and BP. Our results comply with what was found by Dixon [67]: TN ran much faster than epoch-wise BP and achieved comparable solutions. SIMANN is one order of magnitude slower than TN-MS and Novel, and the results are not better. For these reasons, we describe only the results for TN-MS and Novel using the Euler method. TN is used in the local-search phase of Novel.

We have used 2 and 3 hidden-unit networks to solve the sonar problem. Table 4.6 shows the best solutions of TN-MS and Novel that achieve the highest percentage of correctness on the sonar problem's test patterns. Our results show that Novel improves test accuracy by 1%-4%.

All the results in Table 4.6 were run under similar conditions and time limits. In particular, Novel always started from the origin and searched in the range $[-1, 1]$ for each variable, using some combinations of sigmoid gains from the set {1, 10, 30, 50, 100, 300} and (μ_g, μ_t) from the set {(10, 1), (1, 1), (1, 0.1), (0.1, 0.1), (0.1, 1), (0.1, 0.01), (0.01, 0.1)}. TN-MS was run using different combinations of random initial points in search ranges from the set $\{[-0.1, 0.1], [-0.2, 0.2], [-0.5, 0.5], [-1, 1]\}$ and the same sigmoid gains as in Novel. In TN-MS+Novel, Novel always started from the best result of TN-MS using the same sigmoid gain when TN-MS was run. In solving the NetTalk problem, the sigmoid gain is set to 1. Novel used learning rates of 1 and 2 and a momentum of 0.1. Back propagation generated its initial point in the range $[-0.5, 0.5]$, using a momentum of 0.1 and learning rates from the set {0.5, 1, 2, 4, 8, 16}.

We attributed Novel's superiority in finding better local minima to its global-search stage. Since the function searched is rugged and the regions containing good solutions are not large, it is important to avoid probing from many random starting points and to identify good basins before committing expensive local descents. In contrast, many descents performed by multi-start algorithms are performed in unpromising regions. This is exemplified by the behavior of TN-MS, which gave no improvement when we extended the number of restarts from 200 to 400. However, multi-start algorithms may provide good starting points for Novel.

For the vowel-recognition problem, we have used networks with 2 and 4 hidden units. Table 4.6 shows that Novel improves training compared to TN-MS, but performs slightly worse in testing when there are two hidden units. TN-MS+Novel also improved training when compared with TN-MS. The sonar and vowel-recognition problems are not as hard as the two-spiral problem studied in the last section. TN-MS, for instance, can find good solutions readily. Therefore, it is more difficult for global-search methods to find better solutions. In spite of this, Novel can still improve the designs in a small amount of time.

On the 10-parity problem, we use neural networks with 5 and 6 hidden units to compare the solution quality of both algorithms. Using a similar setup as described for the sonar problem, Novel improves the learning results obtained by TN-MS.

Table 4.6: Comparison of the best results obtained by NOVEL and truncated Newton's algorithm with multi-starts (TN-MS) for solving four benchmark problems, where the parameters in one method that obtains the best result may be different from those of another method. Results in bold font are better than or equal to results obtained by TN-MS.

Problems	# of	# of	TN-MS			NOVEL			CPU
			Correct %		# of	Correct %		# time	time
	H.U.	Wts.	training	test	restarts	training	test	units	limits
Sonar	2	125	98.1	90.4	454	**98.1**	**94.2**	191	1000 s
	3	187	100	91.3	485	**100**	**92.3**	291	2000 s
Vowel	2	55	72.2	50.9	298	**72.5**	49.1	131	2 h
	4	99	80.7	56.5	152	**82.6**	**57.8**	41	2 h
10-parity	5	61	97.2	N/A	148	**98.9**	N/A	51	2000 s
	6	73	97.6	N/A	108	**99.8**	N/A	62	3000 s
			BP			NOVEL			
NetTalk	15	3,476	86.3	70.5	13	**87.4**	**72.7**	11	3 h
	30	6,926	92.9	73.1	9	**93.2**	72.5	4	4 h
			TN-MS+NOVEL						
Sonar	2	125	**98.1**	**92.3**	226				1000 s
	3	187	**100**	**92.3**	315				2000 s
Vowel	2	55	**73.5**	50.6	203				2 h
	4	99	**81.2**	**57.1**	168				2 h
10-parity	5	61	**97.2**	N/A	49				2000 s
	6	73	**97.6**	N/A	44				3000 s
			BP + NOVEL						
NetTalk	15	3,476	**89.0**	70.4	11				3 h
	30	6,926	**94.7**	72.3	7				4 h

In the last application, we have studied the NetTalk problem. Since the number of weights and training patterns are very large, we have used pattern-wise learning when applying BP (as in the original experiments by Sejnowski and Rosenberg [220]). By experimenting with different parameter settings of BP, the best learning result by BP is 86.3% for a 15 hidden-unit network.

NetTalk's large number of weights precluded using any method other than the pattern-wise mode in the global-search phase and pattern-wise back propagation in the local-search phase. Even so, very few (logical) time units could be simulated, and our designs perform better in training but sometimes worse in testing. To find better designs, we took the best designs obtained by pattern-wise BP and applied Novel. Table 4.6 shows improved training results but slightly worse testing results. The poor testing results are probably due to the small number of time units that Novel was run.

In short, Novel's training results are always better than or equal to those by TN-MS but are occasionally slightly worse in testing. This is attributed to the time constraint and the excellence of solutions already found by existing methods. In general, improving solutions that are close to the global minima is difficult, often requiring an exponential amount of time unless a better search method is used.

4.6 Summary

In this chapter, we have applied Novel to the supervised learning of feed-forward neural networks. The learning of weights in such networks can be treated as a nonlinear continuous minimization problem with rugged terrains. Our goal is to find neural networks with small number of weights, while avoiding the over-fitting of weights in large networks. Our reasoning is that there are many good local minima in the error space of large networks, hence increasing the chance to find a good local minimum that does not generalize well to test patterns.

We have identified two crucial features of suitable algorithms for solving these problems:

- Use gradient information to descend into local minima. Many algorithms have difficulties when the surface is flat, or when gradients can vary in a large range, or when the terrain is rugged. In the second case, the search may progress too slowly when the gradient is small and may over-shoot when the gradient is large.

- Escape from a local minimum once the search gets there. Such mechanisms can be classified into probabilistic and deterministic. The suitability of a specific strategy is usually problem dependent.

Different algorithms performs different trade-offs between local search and global search. Algorithms that focus on either extreme do not work well. These include gradient descent with multi-starts (such as back-propagation and cascade correlation that focus too much on local search) and covering methods (that focus too much on global search). A good algorithm generally combines global and local searches, switching from one to another dynamically depending on run-time information obtained. These include algorithms based on simulated annealing, evolution, and clustering.

Novel has a global-search phase that relies on two counteracting forces: local gradient information that drives the search to a local minimum, and a deterministic trace that leads the search out of a local minimum once it gets there. The result is an efficient method that identifies good basins without spending a lot of time in them. Good starting points identified in the global-search phase are used in the local-search phase in which pure gradient descents are applied.

We have shown improved performance of Novel for five neural-network benchmark problems as compared to that of existing minimization and neural-network learning algorithms. For the two-spiral problem, we have shown a design with near-perfect classification using four hidden units and 25 weights, while the best design known today requires nine hidden units and 75 weights.

5. CONSTRAINED OPTIMIZATION IN QMF FILTER-BANK DESIGN

In this chapter, we apply Novel to design quadrature-mirror-filter (QMF) filter banks. We formulate the design problem as a nonlinear constrained optimization problem, using the reconstruction error as the objective, and other performance metrics as constraints. This formulation allows us to search for designs that improve over the best existing designs. We derive closed-form formulas for some performance metrics and apply numerical methods to evaluate the values of other performance metrics that do not have closed-form formulas. Novel uses the adaptive Lagrangian method to find saddle points of the constrained optimization problems and relies on the trace-based global search to escape from local minima. In our experiments, we show that Novel finds better designs than the best existing solutions and improves the performance of other global search methods, including simulated annealing and genetic algorithms. We also study the tradeoffs among multiple design objectives, and show that relaxing the constraints on transition bandwidth and stopband energy leads to significant improvements in the other performance measures.

5.1 Two-Channel Filter Banks

The design of digital filter banks is important because improvements can have significant impact in many engineering fields. For example, filter banks have been applied in modems, data transmission, digital audio broadcasting, speech and audio coding, and image and video coding [81, 257, 278].

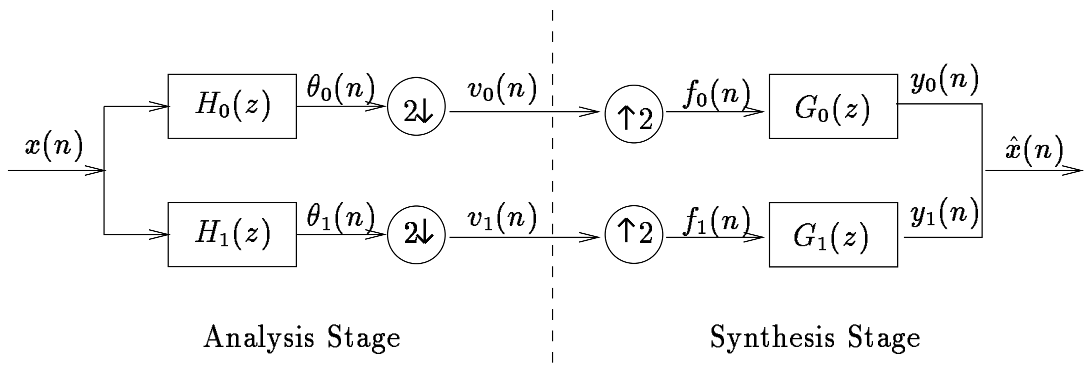


Figure 5.1: The structure of a two-channel filter bank.

Digital filter banks divide an input signal into multiple subbands to be processed. The simplest filter bank is the two-channel filter bank that divides an input signal into two subbands. It is the starting point in studying more complex, multi-channel filter banks. Figure 5.1 shows the typical structure of a two-channel filter bank, where $x(n)$ is the input signal in time domain and $\hat{x}(n)$ is the output signal. The filter bank consists of four filters: $H_0(z)$, $H_1(z)$, $G_0(z)$, and $G_1(z)$. The two filters in the upper channel, $H_0(z)$ and $G_0(z)$, are low-pass filters, while the two filters in the lower channel, $H_1(z)$ and $G_1(z)$, are high-pass filters.

As shown in Figure 5.1, the filter bank is divided into two parts: an analysis stage and a synthesis stage. In the analysis stage, the input spectrum $X(z)$ (or $X(\omega)$, $0 \leq \omega \leq \pi$), which is the Z-transform (or Fourier transform) of $x(n)$, is divided into two subbands, one high frequency band and one low frequency band, by a pair of analysis filters $H_0(z)$ and $H_1(z)$. Then, according to Nyquist theorem, the subband signals are down sampled by 2 to produce the outputs of the analysis stage, $v_0(n)$ and $v_1(n)$. These output signals can be analyzed and processed in various ways according to the applications. For example, in a subband coder for audio or image compression, these signals are quantized and encoded before being transmitted to the receiver, which is the synthesis part.

When no error occurs between the analysis stage and the synthesis stage, the input signals of the synthesis stage are the same as the outputs of the analysis stage. In the synthesis stage, the subband signals are up-sampled by 2 to produce $f_0(n)$ and $f_1(n)$, which are then passed through the synthesis filters $G_0(z)$ and $G_1(z)$ that perform interpolation. In the end, the subband signals are added together to produce the reconstructed signal $\hat{x}(n)$ at the output.

In short, output signal $\hat{x}(n)$ is a function of input signal $x(n)$ and the filters in the filter bank, $H_0(z)$, $H_1(z)$, $G_0(z)$, and $G_1(z)$ [4, 181]. Following the top branch in Figure 5.1, we have the following equations for the analysis and synthesis filters:

$$\theta_0(z) = H_0(z)X(z) \tag{5.1}$$

$$Y_0(z) = G_0(z)F_0(z) \tag{5.2}$$

We have the following equations for the down-sampling and up-sampling:

$$V_0(z) = \frac{1}{2}[\theta_0(z^{1/2}) + \theta_0(-z^{1/2})] \tag{5.3}$$

$$F_0(z) = V_0(z^2) \tag{5.4}$$

Substituting Eq. (5.1), (5.3), and (5.4) into (5.2), we obtain the output of the top branch as

$$Y_0(z) = \frac{1}{2}G_0(z)[H_0(z)X(z) + H_0(-z)X(-z)]. \tag{5.5}$$

Similarly, for the low branch in Figure 5.1, we can derive

$$Y_1(z) = \frac{1}{2}G_1(z)[H_1(z)X(z) + H_1(-z)X(-z)]. \tag{5.6}$$

The outputs of the upper and lower branches are added together to produce the reconstructed output signal:

$$\begin{aligned}\hat{X}(z) &= \frac{1}{2}[G_0(z)H_0(z) + G_1(z)H_1(z)]X(z) + \frac{1}{2}[G_0(z)H_0(-z) + G_1(z)H_1(-z)]X(-z) \\ &= T(z)X(z) + S(z)X(-z) \qquad (5.7)\end{aligned}$$

where $T(z) = T(e^{j\omega}) = |T(e^{j\omega})|e^{j\phi(\omega)}$ and $S(z) = S(e^{j\omega}) = |S(e^{j\omega})|e^{j\psi(\omega)}$.

In the reconstructed signal $\hat{X}(z)$, the filter bank may introduce three types of distortions: (a) *Aliasing distortion*: The sub-sampling in the filter bank generates aliasing, and the up-sampling produces images. The term $S(z)X(-z)$ in (5.7) is called the aliasing term; (b) *Amplitude distortion*: The deviation of $|T(z)|$ in (5.7) from unity constitutes the amplitude distortion; (c) *Phase distortion*: The deviation of $\phi(\omega)$ from the desired phase properties, such as linear phase, is called the phase distortion. These distortions are undesirable in many filter-bank applications. For example, in digital image/video compression applications, amplitude distortions change image color and brightness, whereas phase distortions introduce unpleasant visual effects.

Many methods have been developed to minimize the undesired distortions of filter banks. For instance, aliasing distortions can be removed by enforcing appropriate relationships between the synthesis filters and the analysis filters; magnitude distortions can be removed by using infinite impulse response (IIR) filters; and the linear phase property can be achieved by using linear-phase finite impulse response (FIR) filters [4, 181].

When all the distortions are removed, the original signal is said to be reconstructed perfectly. Based on (5.7), the perfect reconstruction of the original signal requires $S(z) = 0$, for all z, and $T(z) = z^{-n_0}$, where n_0 is a constant. In this case, the transfer function of the filter bank is just a pure delay.

Let's re-write the transfer function in (5.7) using the Fourier transformation as follows:

$$\begin{aligned}\hat{X}(\omega) &= \frac{1}{2}[G_0(\omega)H_0(\omega) + G_1(\omega)H_1(\omega)]X(\omega) \\ &\quad + \frac{1}{2}[G_0(\omega)H_0(\pi+\omega) + G_1(\omega)H_1(\pi+\omega)]X(\pi+\omega) \\ &= T(\omega)X(\omega) + S(\omega)X(\pi+\omega) \qquad (5.8)\end{aligned}$$

In quadrature-mirror-filter (QMF) filter banks, the filters are chosen to satisfy the following conditions [56, 71, 278]:

$$G_0(\omega) = H_0(-\omega) \qquad (5.9)$$

$$G_1(\omega) = H_1(-\omega) \qquad (5.10)$$

$$H_1(\omega) = e^{j\omega}H_0(\pi-\omega) \qquad (5.11)$$

Thus, the pairs of filters in the analysis and the synthesis stages, $H_0(z)$ and $H_1(z)$, $G_0(z)$ and $G_1(z)$, are mirror filters, respectively. With the particular selection of filters, (5.8) becomes

$$\begin{aligned}\hat{X}(\omega) &= \frac{1}{2}[H_0(-\omega)H_0(\omega) + H_0(\pi-\omega)H_0(\pi+\omega)]X(\omega) \\ &\quad +\frac{1}{2}[H_0(\pi+\omega)H_0(-\omega) + e^{j\pi}H_0(-\omega)H_0(\pi+\omega)]X(\pi+\omega) \\ &= \frac{1}{2}[H_0(-\omega)H_0(\omega) + H_0(\pi-\omega)H_0(\pi+\omega)]X(\omega) \\ &= T(\omega)X(\omega) \qquad (5.12)\end{aligned}$$

Note that the aliasing term disappears, and the aliasing cancellation is exact, independent of the choice of function $H_0(\omega)$. Now, the function $T(\omega)$ is only a function of the filter $H_0(\omega)$, which is called the ***prototype filter*** in the QMF filter bank. The design problem is now reduced to finding a prototype filter $H_0(\omega) = H(\omega)$ such that $T(\omega)$ is a pure delay.

In an FIR QMF filter bank, when the prototype filter $H(\omega)$ is a symmetric low-pass FIR filter, $H(\omega)$, as well as the function $T(\omega)$, has linear phase. In this case, the problem to design a perfect-reconstruction linear-phase filter bank entails making the amplitude of $T(\omega)$ to be 1, e.g.,

$$|H(\omega)|^2 + |H(\omega+\pi)|^2 = 1 \qquad (5.13)$$

where we have ignored the constant coefficient 1/2 in (5.12).

Figure 5.2 shows how the frequency responses of the filters in a QMF filter bank are added together to form the overall amplitude response.

It can be shown that once the filters are chosen as (5.9) to (5.11), it is impossible to obtain perfect reconstruction of the original signal using FIR filters except for the trivial, two-tap filter case. This means that, in general, the reconstruction error (the shaded area in the right graph of Figure 5.2), $E_r = \int_0^\pi (|H(\omega)|^2 + |H(\omega+\pi)|^2 - 1)d\omega$, is not exactly zero. However, by numerically minimizing the reconstruction error, filter banks of high quality can be designed [133].

To summarize, FIR QMF filter banks use mirror filters in the analysis and the synthesis stages to obtain the nice properties of linear phase and no aliasing distortion. The design

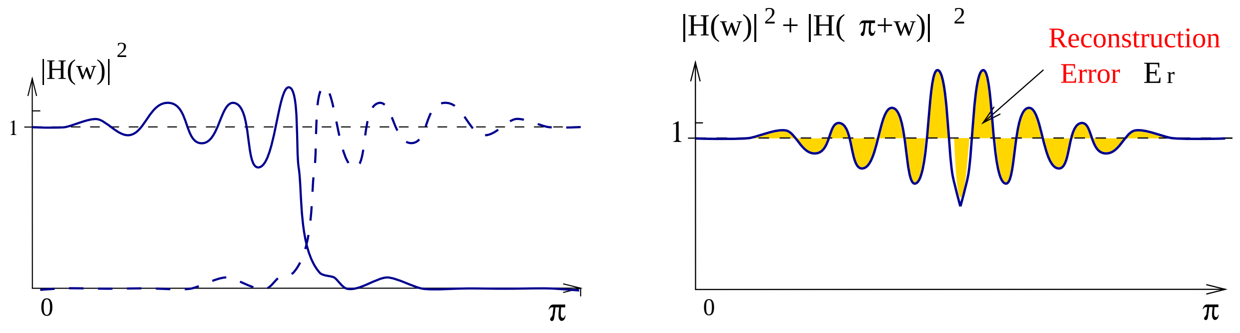


Figure 5.2: The prototype low-pass filter and its mirror filter form the reconstruction error, E_r, of the QMF filter bank.

of FIR QMF filter banks becomes designing a prototype FIR low-pass filter to minimize the reconstruction error, i.e., the amplitude distortion. Although, in general, these QMF filter banks cannot achieve perfect reconstruction, high-quality filter banks with very small reconstruction errors can be found numerically.

5.2 Existing Methods in QMF Filter-Bank Design

Generally speaking, the design objectives of filter banks consist of two parts: the response of the overall filter bank and the response of each individual filter. The performance metrics of the overall filter bank include amplitude distortion, aliasing distortion, and phase distortion. The performance metrics of a single filter vary from one type of filters to another. Filter banks can be composed of either finite impulse response filters (FIR) or infinite impulse response filters (IIR). Figure 5.3 shows an illustration of the performance metrics of a single low-pass filter. The five performance metrics, passband ripple δ_p, stopband ripple δ_s, passband flatness E_p, stopband energy E_s, and transition bandwidth T_t of a filter are calculated based on the difference of the frequency response of the filter and that of the ideal low-pass filter, which is a step function.

Table 5.1 summarizes the possible design objectives of a filter bank. The performance metrics of a single filter are for a low-pass filter. Because the design objectives are generally

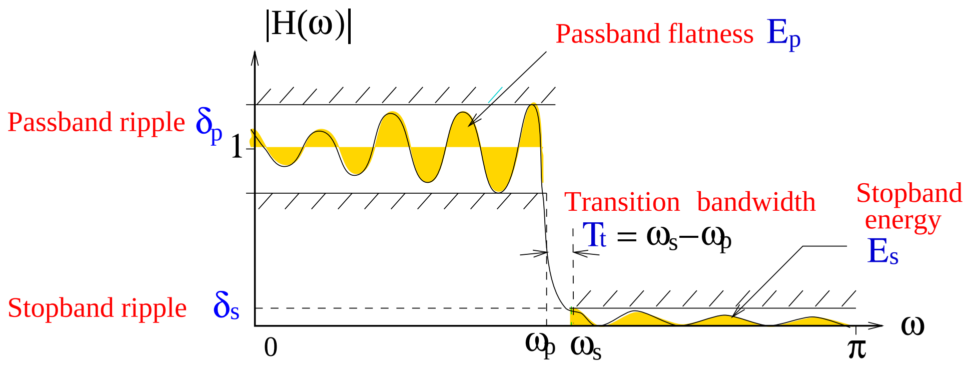


Figure 5.3: An illustration of the performance metrics of a single low-pass filter.

Table 5.1: Possible design objectives of filter banks. Refer to Eq. (5.7), (5.8), and Figure 5.3 for explanation.

	Design Objectives	Metrics
Overall Filter Bank	Minimize amplitude distortion E_r	$E_r = \int_0^{\pi}(\lvert T(\omega)\rvert^2 - 1)^2 d\omega$
	Minimize aliasing distortion ρ_a	$\rho_a = \int_0^{\pi} \lvert S(\omega)\rvert^2 d\omega$
	Minimize phase distortion ρ_p	$\rho_p = \int_0^{\pi} \lvert\phi(\omega) - \phi_0(\omega)\rvert d\omega$
Single Filter	Minimize stopband ripple (δ_s)	$\delta_s = \max(\lvert H(\omega)\rvert,\ \omega \in [\omega_s, \pi])$
	Minimize passband ripple (δ_p)	$\delta_p = \max(\lvert H(\omega) - 1\rvert,\ \omega \in [0, \omega_p])$
	Minimize transition bandwidth (T_t)	$T_t = \omega_s - \omega_p$
	Minimize stopband energy (E_s)	$E_s = \int_{\omega_s}^{\pi} \lvert H(\omega)\rvert^2 d\omega$
	Maximize passband flatness (E_p)	$E_p = \int_0^{\omega_p}(\lvert H(\omega)\rvert - 1)^2 d\omega$
$[0, \omega_p]$ — pass band; $[\omega_s, \pi]$ — stop band; $[\omega_p, \omega_s]$ — transition band. $\phi_0(\omega)$ — the desired linear phase.		

nonlinear continuous functions of filter coefficients, filter-bank design problems are multi-objective, continuous, nonlinear optimization problems.

Algorithms for designing filter banks can be classified into optimization-based and non-optimization-based. In optimization-based methods, a design problem is formulated as a multi-objective nonlinear optimization problem whose form may be application- and filter-dependent [256]. 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, simulated-annealing, and genetics-based methods [123, 133]. Filter-bank design problems have also been solved by non-optimization-based algorithms, which include spectral factorization [148, 258] and heuristic methods. These methods generally do not continue to find better designs once a suboptimal design has been found [258].

The design of filter banks involves complex nonlinear optimization with the following features.

- The objectives are not unique and may be conflicting, leading to designs with different tradeoffs. To design filter banks systematically, the multiple objectives are integrated into a single formulation.
- Some design objectives and their derivatives are not in closed forms and need to be evaluated using numerical methods.
- The design objectives are nonlinear.

5.3 Optimization Formulations of QMF Filter-bank Design

There are two approaches in optimization design of QMF filter banks: multi-objective unconstrained optimization and single-objective constrained optimization. In the multi-objective approach, the goals are

- to minimize the amplitude distortion (reconstruction error) of the overall filter bank, and
- to maximize the performance of the individual prototype filter $H(\omega)$.

A possible formulation is to optimize the design with respect to a subset of the measures defined in Table 5.1. For example, if we want to minimize the reconstruction error of the filter bank and the stopband energy of an n-tap FIR prototype filter with coefficients $\mathbf{x} =$

$(x_1, x_2, \cdots, x_n)$, the optimization problem is

$$\begin{aligned} \min_{\mathbf{x}\in R^n} \quad & E_r(\mathbf{x}) \quad \text{and} \quad E_s(\mathbf{x}) \qquad (5.14) \\ \text{where} \quad & E_r(\mathbf{x}) = \int_{\omega=0}^{\frac{\pi}{2}} (|H(\omega,\mathbf{x})|^2 + |H(\omega+\pi,\mathbf{x})|^2 - 1)^2 \, d\omega \\ & E_s(\mathbf{x}) = \int_{\omega=\omega_s}^{\pi} |H(\omega,\mathbf{x})|^2 d\omega \end{aligned}$$

Unfortunately, optimal solutions to the simplified optimization problem are not necessarily optimal solutions to the original problem. Oftentimes, performance measures not included in the formulation are compromised. For example, when E_r and E_s are the objectives to be minimized, the solution that gives the smallest E_r and E_s will probably have a large transition bandwidth.

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 Pareto frontier is to optimize a weighted sum of all the objectives [47, 69, 133, 180, 256]. This approach has difficulty when Pareto 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 filter is found. When the desired characteristics are difficult to satisfy, trial and error is not effective in finding feasible designs. In this case, a constrained formulation should be used instead.

Another approach to solve a multi-objective problem is to turn all the objectives except one into constraints. In this formulation, the constraints are defined with respect to a reference design. The specific measures constrained may be application- and filter-dependent [256].

Constraint-based methods have been applied to design QMF filter banks in both the frequency [47,54,133,148,234,239] and the time domains [177,238]. In the frequency domain, the most often considered objectives are the reconstruction error, E_r, and the stopband ripple, δ_s. As stopband ripples cannot be formulated in closed form, stopband attenuation, E_s, is usually used instead. In the time domain, Nayebi [177] gave a time-domain formulation with constraints in the frequency domain and designed filter banks using an iterative time-domain design algorithm.

In this thesis, we formulate the design of a QMF filter bank in the most general form as a constrained nonlinear optimization problem [265, 266].

$$\begin{aligned} \textit{minimize} \quad & E_r(\mathbf{x})/\theta_{E_r} & & \qquad (5.15) \\ \textit{subject to} \quad & E_p(\mathbf{x})/\theta_{E_p} \le 1 & \quad & E_s(\mathbf{x})/\theta_{E_s} \le 1 \\ & \delta_p(\mathbf{x})/\theta_{\delta_p} \le 1 & & \delta_s(\mathbf{x})/\theta_{\delta_s} \le 1 \\ & T_t(\mathbf{x})/\theta_{T_t} \le 1 & & \end{aligned}$$

where θ_{E_r}, θ_{E_p}, θ_{E_s}, θ_{δ_p}, θ_{δ_s}, and θ_{T_t} are the performance values of the baseline design (with possibly some constraint values relaxed or tightened in order to obtain designs of different trade-offs). Reconstruction error $E_r(\mathbf{x})$ is the objective to be minimized, and all other metrics of a single filter are used as constraints. This formulation allows us to improve on the best existing design (such as designs reported by Johnston [133]) with respect to all the performance metrics. In contrast, existing methods generally optimize a subset of the performance metrics in constrained form or a weighted sum of the metrics. Since the objective and the constraints in (5.15) are nonlinear, the optimization problem is generally multi-modal with many local minima.

We have applied our Novel global search method to design better QMF filter banks. In the rest of this chapter, we first present our method in evaluating the performance metrics and their corresponding derivatives. Then, in the experimental results, we show the improvement of Novel on existing designs, and compare the performance of Novel with those of other global search methods.

5.4 Evaluations of Performance Metrics

In QMF filter-bank design problems, some functions, such as reconstruction error E_r, stopband energy E_s, and passband energy E_p, have closed-form formulas. However, other functions, including stopband ripple δ_s, passband ripple δ_p, and transition bandwidth T_t, do not have closed-form formulas, and have to be evaluated numerically.

Let $\mathbf{x} = (x_1, x_2, \cdots, x_N)$ be the parameters of an N-tap prototype filter, and $\mathbf{x}$ is symmetric, i.e., $x_i = x_{N+1-i}$. The frequency response of the prototype filter is obtained by taking a Fourier transform of $\mathbf{x}$:

$$\begin{aligned} \mathcal{F}(\mathbf{x}) &= \sum_{n=1}^{N} x_n e^{-jn\omega} \\ &= e^{-j\frac{N-1}{2}\omega} \sum_{n=1}^{N/2} x_n \mathrm{Cos}\left(\frac{N+1}{2} - n\right)\omega \end{aligned} \tag{5.16}$$

From Eq. (5.16), the phase response, $-\frac{N-1}{2}\omega$, is a linear function of ω, and the amplitude response is

$$h(\mathbf{x}, \omega) = \sum_{n=1}^{N/2} x_n \mathrm{Cos}\left(\frac{N+1}{2} - n\right)\omega \tag{5.17}$$

Hence, the performance metrics listed in Table 5.1 are functions of $h(\mathbf{x}, \omega)$ as follows:

$$E_r(\mathbf{x}) = \int_0^{\pi} (h^2(\mathbf{x}, \omega) + h^2(\mathbf{x}, \pi - \omega) - 1)^2 d\omega \tag{5.18}$$

$$E_s(\mathbf{x}) = \int_{\omega_s(\mathbf{x})}^{\pi} h^2(\mathbf{x}, \omega) d\omega \tag{5.19}$$

$$E_p(\mathbf{x}) = \int_0^{\omega_p(\mathbf{x})} (h(\mathbf{x}, \omega) - 1)^2 d\omega \tag{5.20}$$

$$\delta_s(\mathbf{x}) = \max(|h(\mathbf{x}, \omega)|,\ \omega \in (\omega_s(\mathbf{x}), \pi)) \tag{5.21}$$

$$\delta_p(\mathbf{x}) = \max(|h(\mathbf{x}, \omega) - 1|,\ \omega \in (0, \omega_p(\mathbf{x}))) \tag{5.22}$$

$$T_t(\mathbf{x}) = \omega_s(\mathbf{x}) - \omega_p(\mathbf{x}) \tag{5.23}$$

The calculation of $h(\mathbf{x}, \omega)$ involves a series of summations. When the summation of large numbers results in a small number, the numerical cancellation errors can be significant. These happen in computing $h(\mathbf{x}, \omega)$ when the prototype filter is close to the ideal low-pass filter. It is desirable to make the cancellation error small.

```
$S = a[1]$
$C = 0$
for $j = 2$ to $k$ {
    $Y = a[j] - C$
    $T = S + Y$
    $C = (T - S) - Y$
    $S = T$
}
```

Figure 5.4: Kahan's formula for summation $\sum_{j=1}^{k} a_j$ [146].

We have used two numerical methods to do the summation. One is to add numbers from small to large. For summation $\sum_{j=1}^{k} a_j$, the numerical solution of this method is

$$\sum_{j=1}^{k} a_j(1 + \delta_j) \tag{5.24}$$

where $|\delta_j| < (k - j)\epsilon$, and ϵ is the machine precision. The numerical error is proportional to the number of terms in the summation.

A better way is to use Kahan's summation formula as shown in Figure 5.4, in which extra variables are introduced to maintain higher precision. Its solution of $\sum_{j=1}^{k} a_j$ is

$$\sum_{j=1}^{k} a_j(1 + \delta'_j) + O(N\epsilon^2) \sum |a_j| \tag{5.25}$$

where $|\delta'_j| < 2\epsilon$ [146]. Since machine precision ϵ is very small for double precision floating point representations, the second term in the sum is usually small. Further, since $|\delta'_j a_j|$ is smaller than $|\delta_j a_j|$ when k is large, the numerical error in (5.25) can be much smaller than that in (5.24) when the summation series is long.

Table 5.2: Lower and upper bounds for calculating the reconstruction error E_r

Index	Lower Bound	Upper Bound
1	Max(1, $-N/2+n+i$)	Min(N/2, $-1+n+i$)
2	Max(1, $1+n-i$)	Min(N/2, $N/2+n-i$)
3	Max(1, $N/2+1-n+i$)	Min(N/2, $N-n+i$)
4	Max(1, $N+2-n-i$)	Min(N/2, $3N/2+1-n-i$)
5	Max(1, $-N+n+i$)	Min(N/2, $-N/2-1+n+i$)
6	Max(1, $N/2+1+n-i$)	Min(N/2, $N+n-i$)
7	Max(1, $1-n+i$)	Min(N/2, $N/2-n+i$)

5.4.1 Closed-form Formulas of $E_r(\mathbf{x})$, $E_s(\mathbf{x})$, and $E_p(\mathbf{x})$

As shown in Eq. (5.18), (5.19), and (5.20), $E_r(\mathbf{x})$, $E_s(\mathbf{x})$, and $E_p(\mathbf{x})$ are expressed as integrations of $h(\mathbf{x}, \omega)$. In this section, we present their closed-form formulas. Please refer to [279] for more detailed derivations.

By substituting (5.17) into (5.18), we obtain the following formula for reconstruction error $E_r(\mathbf{x})$:

$$\begin{aligned} E_r(\mathbf{x}) &= \frac{1}{2}\pi \sum_{n=1}^{n=N/2} \sum_{i=1}^{i=N/2} x_n x_i \Bigg[2x_n x_i + \sum_{lb_1 \le m \le ub_1, m-n=2k, m \ne n} x_m x_{n-m+i} \\ &+ \sum_{lb_2 \le m \le ub_2, m-n=2k, m \ne n} x_m x_{-n+m+i} + \sum_{lb_3 \le m \le ub_3, m+n=2k+1} x_m x_{N+1-n-m+i} \\ &+ \sum_{lb_4 \le m \le ub_4, m+n=2k+1} x_m x_{-N-1+n+m+i} + \sum_{lb_5 \le m \le ub_5, m-n=2k} x_m x_{N+1-n+m-i} \\ &+ \sum_{lb_6 \le m \le ub_6, m-n=2k} x_m x_{N+1+n-m-i} + \sum_{lb_7 \le m \le ub_7, m+n=2k+1} x_m x_{n+m-i} \Bigg] \\ &-2\pi \sum_{n=1}^{N/2} x_n^2 \end{aligned} \tag{5.26}$$

where k is an integer and $1 \le k \le N/2$. The lower and upper bounds for the seven innermost summations are shown in Table 5.2.

The formula for the stopband energy $E_s(\mathbf{x})$ is obtained by substituting (5.17) into (5.19):

$$\begin{aligned} E_s(\mathbf{x}) &= \frac{1}{2}\sum_{n=1}^{N/2} x_n^2 \left[(\pi - \omega_s(\mathbf{x})) - \frac{\mathrm{Sin}(N+1-2n)\omega_s(\mathbf{x})}{N+1-2n}\right] - \\ & \frac{1}{2}\sum_{n=1}^{N/2} \sum_{m=1, m\neq n}^{N/2} x_n x_m \left[\frac{\mathrm{Sin}(n-m)\omega_s(\mathbf{x})}{n-m} + \frac{\mathrm{Sin}(N+1-n-m)\omega_s(\mathbf{x})}{N+1-n-m}\right] \quad (5.27) \end{aligned}$$

Similarly, by substituting (5.17) into (5.20), we obtain the formula for the passband energy $E_p(\mathbf{x})$ as follows:

$$\begin{aligned} E_p(\mathbf{x}) &= \frac{1}{2}\sum_{n=1}^{N/2} x_n^2 \left[\omega_p(\mathbf{x}) + \frac{\mathrm{Sin}(N+1-2n)\omega_p(\mathbf{x})}{N+1-2n}\right] \\ & + \frac{1}{2}\sum_{n=1}^{N/2} \sum_{m\neq n, m=1}^{N/2} x_n x_m \left[\frac{\mathrm{Sin}(n-m)\omega_p(\mathbf{x})}{n-m} + \frac{\mathrm{Sin}(N+1-n-m)\omega_p(\mathbf{x})}{N+1-n-m}\right] \\ & -2\sum_{n=1}^{N/2} x_n \frac{\mathrm{Sin}(\frac{N+1}{2}-n)\omega_p(\mathbf{x})}{\frac{N+1}{2}-n} + \omega_p(\mathbf{x}) \quad (5.28) \end{aligned}$$

5.4.2 Numerical Evaluation of $\delta_s(\mathbf{x})$, $\delta_p(\mathbf{x})$, $T_t(\mathbf{x})$

Although we do not have closed-form formulas for the other three performance metrics of a low-pass prototype filter (stopband ripple, passband ripple, and transition bandwidth), we can evaluate their values using numerical methods.

We find the stopband ripple, $\delta_s(\mathbf{x})$, in the frequency range $[\pi/2, \pi]$ based on the frequency response of prototype filter $\mathbf{x}$. First, we uniformly sample the frequency range from $\pi/2$ to π. Frequency brackets that contain all the ripples are found based on the sample points. Within each bracket, we apply Newton's method to precisely locate the ripple. When Newton's method does not converge after a fixed number of iterations, we use Golden search to reduce the bracket and re-start Newton's method again. Among all the ripples, the largest one is the stopband ripple $\delta_s(\mathbf{x})$. In a similar way, the passband ripple $\delta_p(\mathbf{x})$ is found in the frequency range of $[0, \pi/2]$.

It is desirable to compute the ripples fast. Newton's method has super-linear convergence speed when the initial point is close to the convergence point. However, Newton's method may have slow convergence, or even diverge in other cases. Golden search uses a bracket to safeguard the search. It guarantees to find the local minimum inside a bracket, but has only linear convergence speed. By combining Newton's method and Golden search, we get fast and guaranteed convergence.

The transition bandwidth $T_t(\mathbf{x}) = \omega_s(\mathbf{x}) - \omega_p(\mathbf{x})$ is found based on the passband cut-off frequency $\omega_p(\mathbf{x})$, and the stopband cut-off frequency $\omega_s(\mathbf{x})$. $\omega_p(\mathbf{x})$ and $\omega_s(\mathbf{x})$ are calculated based on passband and stopband ripples $\delta_p(\mathbf{x})$ and $\delta_s(\mathbf{x})$, respectively. Based on the frequency response of the prototype filter $\mathbf{x}$, $\omega_p(\mathbf{x})$ is the first ω value in the range from $\pi/2$ to 0 that makes $h(\mathbf{x}, \omega) = 1 - \delta_p(\mathbf{x})$. We find $\omega_p(\mathbf{x})$ in two steps. First, a bracket containing $\omega_p(\mathbf{x})$ is obtained based on sampling. Then, a combination of Newton's method and bisection search is used to find $\omega_p(\mathbf{x})$ within a predefined numerical error. A similar method is used to find the stopband cut-off frequency $\omega_s(\mathbf{x})$, which is the first $\omega(\mathbf{x})$ value in the range from $\pi/2$ to π that makes $h(\mathbf{x}, \omega) = \delta_s(\mathbf{x})$.

5.4.3 Evaluations of the First-order Derivatives of Performance Metrics

For performance metrics with closed-form formulas, which include reconstruction error $E_r(\mathbf{x})$, stopband energy $E_s(\mathbf{x})$, and passband energy $E_p(\mathbf{x})$, please refer to [279] for the analytical forms of their corresponding first-order derivatives:

$$\begin{aligned}
\frac{\partial E_s(\mathbf{x})}{\partial x_i} &= x_i \left[(\pi - \omega_s(\mathbf{x})) - \frac{\mathrm{Sin}(N+1-2i)\omega_s(\mathbf{x})}{N+1-2i} \right] \\
&- \sum_{n=1, n\neq i}^{N/2} x_n \left[\frac{\mathrm{Sin}(n-i)\omega_s(\mathbf{x})}{n-i} + \frac{\mathrm{Sin}(N+1-n-i)\omega_s(\mathbf{x})}{N+1-n-i} \right] \\
&- \frac{\partial \omega_s(\mathbf{x})}{\partial x_i} h^2(\mathbf{x}, \omega_s(\mathbf{x})). \qquad (5.29)
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial E_p(\mathbf{x})}{\partial x_i} &= x_i \left[\omega_p(\mathbf{x}) + \frac{\mathrm{Sin}(N+1-2i)\omega_p(\mathbf{x})}{N+1-2i}\right] \\
&+ \sum_{n=1,n\neq i}^{N/2} x_n \left[\frac{\mathrm{Sin}(n-i)\omega_p(\mathbf{x})}{n-i} + \frac{\mathrm{Sin}(N+1-n-i)\omega_p(\mathbf{x})}{N+1-n-i}\right] \\
&-2\, \frac{\mathrm{Sin}(\frac{N+1}{2}-i)\omega_p(\mathbf{x})}{\frac{N+1}{2}-i} \\
&+ \frac{\partial \omega_p(\mathbf{x})}{\partial x_i}(h(\mathbf{x},\omega_p(\mathbf{x}))-1)^2. \qquad (5.30)
\end{aligned}
$$

and

$$
\begin{aligned}
\frac{\partial E_r(\mathbf{x})}{\partial x_i} &= 2\pi \sum_{n=1}^{n=N/2} x_n \Bigg[2x_n x_i + \sum_{lb_1\le m\le ub_1, m-n=2k, m\neq n} x_m x_{n-m+i} \\
&+ \sum_{lb_2\le m\le ub_2, m-n=2k, m\neq n} x_m x_{-n+m+i} + \sum_{lb_3\le m\le ub_3, m+n=2k+1} x_m x_{N+1-n-m+i} \\
&+ \sum_{lb_4\le m\le ub_4, m+n=2k+1} x_m x_{-N-1+n+m+i} + \sum_{lb_5\le m\le ub_5, m-n=2k} x_m x_{N+1-n+m-i} \\
&+ \sum_{lb_6\le m\le ub_6, m-n=2k} x_m x_{N+1+n-m-i} + \sum_{lb_7\le m\le ub_7, m+n=2k+1} x_m x_{n+m-i}\Bigg] \\
&-4\pi x_i. \qquad (5.31)
\end{aligned}
$$

where, $i = 1, \cdots, N/2$ and $\mathbf{x}$ is symmetric.

Note that $\frac{\partial E_s(\mathbf{x})}{\partial x_i}$ and $\frac{\partial E_p(\mathbf{x})}{\partial x_i}$ are partly in closed form because $\frac{\partial \omega_s(\mathbf{x})}{\partial x_i}$ and $\frac{\partial \omega_p(\mathbf{x})}{\partial x_i}$ are estimated using finite difference methods, e.g.,

$$\frac{\partial \omega_s(\mathbf{x})}{\partial x_i} = \frac{\omega_s(x_1,\cdots,x_i+\Delta x,\cdots,x_{N/2}) - \omega_s(x_1,\cdots,x_i-\Delta x,\cdots,x_{N/2})}{2\Delta x} \qquad (5.32)$$

$$\frac{\partial \omega_p(\mathbf{x})}{\partial x_i} = \frac{\omega_p(x_1,\cdots,x_i+\Delta x,\cdots,x_{N/2}) - \omega_p(x_1,\cdots,x_i-\Delta x,\cdots,x_{N/2})}{2\Delta x} \qquad (5.33)$$

For all the performance metrics without closed-form formulas, which include stopband ripple, passband ripple, and transition bandwidth, we use finite difference methods to approximate their derivatives, $\partial\delta_s(\mathbf{x})/\partial x_i$, $\partial\delta_p(\mathbf{x})/\partial x_i$, and $\partial T_t(\mathbf{x})/\partial x_i$.

5.5 Experimental Results

In our experiments, we have applied Novel to solve the 14 QMF filter-bank design problems formulated by Johnston [133]. These include 16a, 16b, 16c, 24b, 24c, 24d, 32c, 32d, 32e, 48c, 48d, 48e, 64d, and 64e, where the integer in each identifier represents the number of filter taps, and the types "a" to "e" represent prototype filters with increasingly sharper (shorter) transition bands.

Our goal is to find designs that are better than Johnston's results across all six performance measures, including reconstruction error, stopband energy, passband energy, stopband ripple, passband ripple, and transition bandwidth. Hence, we use the constrained optimization formulation (5.15) with the constraint bounds defined by those of Johnston's designs. In our experiments, we improve existing Johnston's results using Novel, compare the performance of Novel with those of simulated annealing and genetic algorithms, and study the performance tradeoffs of design objectives [265, 266].

5.5.1 Performance of Novel

Novel uses the adaptive Lagrangian method presented in Section 2.2.3 to handle nonlinear constraints. Figure 5.5 compares the execution time of the adaptive Lagrangian method with that of the static-weight Lagrangian method when both methods achieve the same quality of solutions. It shows the convergence times of the static-weight Lagrangian method normalized with respect to that of the adaptive method in solving the 24d and 48e design problems. For the method with static weights, we have used, respectively, weight values of $10^{-4}, 5\times10^{-5}, 10^{-5}, 5\times10^{-6}$, and 10^{-6} in our experiments. For the method with dynamic weights, we have used the weight-initialization algorithm described in Section 2.2.3 to set the initial weights. Our program was run on a 200MHz Intel Pentinm Pro Computer.

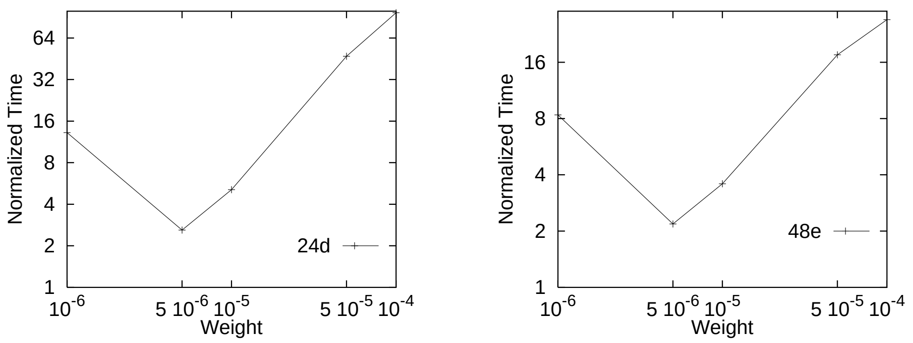


Figure 5.5: Comparison of convergence speed of Lagrangian methods with and without dynamic weight control for 24d (left) and 48e (right) QMF filter-bank design problems. The convergence times of the Lagrangian method using different static weights are normalized with respect to the time spent by the adaptive Lagrangian method.

In solving the 24d problem, our adaptive Lagrangian method takes 6.6 minutes to converge to a saddle point with an objective value of 0.789, whereas the Lagrangian method with static weights obtains the same quality of solution with vastly different convergence speed, ranging from 2.6 times to 97.5 times as long as the adaptive Lagrangian method. Similarly, the dynamic method takes 35.1 minutes in solving the 48e design problem, but the static method takes vastly different and longer times for different initial weights, ranging from 2.2 times to 27.0 times longer.

Table 5.3 shows the experimental results for all the design problems found using the adaptive Lagrangian method normalized with respect to Johnston's solutions. For each problem, the adaptive Lagrangian method starts from Johnston's solution and converges to a saddle point. We use LSODE to generate the search trajectory. In Table 5.3, a value less than 1 for a performance metric means that our design is better on this metric as compared to Johnston's design. The table shows that the dynamic Lagrangian method improves the objective value (second column), while satisfying all the design constraints (Columns 3-7). The execution times to find the solutions differ significantly, varying from a few minutes to several days.

Table 5.3: Experimental results of the dynamic Lagrangian method in designing QMF filter banks. We use Johnston's solutions as starting points.

Filter-type	E_r	δ_p	E_p	δ_s	E_s	T_r	CPU time (minutes)	Time unit
16a	0.990	0.997	0.785	0.999	1.000	1.000	89.8	0.156
16b	0.995	0.962	0.806	1.000	1.000	1.000	16.5	0.278
16c	0.826	1.000	0.916	1.000	1.000	1.000	16.3	0.213
24b	0.966	0.994	0.820	0.999	1.000	1.000	119.0	0.107
24c	0.910	1.000	0.768	1.000	1.000	1.000	11.4	0.148
24d	0.789	1.000	0.835	1.000	1.000	1.000	6.6	0.175
32c	0.959	0.999	0.735	0.999	1.000	1.000	111.8	0.115
32d	0.870	0.999	0.800	1.000	1.000	1.000	15.4	0.136
32e	0.735	1.000	0.924	1.000	1.000	1.000	17.5	0.194
48c	0.793	1.000	0.810	0.945	0.999	1.000	2899.1	0.123
48d	0.948	0.999	0.752	0.999	1.000	1.000	250.5	0.106
48e	0.852	1.000	0.840	1.000	1.000	1.000	35.1	0.142
64d	0.784	0.991	0.787	0.591	0.997	1.000	3143.5	0.021
64e	0.845	1.000	0.764	1.000	1.000	1.000	118.9	0.138

Due to the nonlinear function in the objective and the constraints, there exist local minima in the search space of QMF filter-bank design problems. Lagrangian methods only perform local searches. Global-search methods are needed to overcome local minima. Our Novel relies on the trace-based global search to escape from local minima, and to identify good starting points for local searches. Table 5.4 shows the experimental results of Novel that uses the trace-based global search together with the adaptive Lagrangian method in solving QMF filter-bank design problems. The trace-based global search consists of one global-search stage that starts from Johnston's solution and searches in the range ± 0.01 in each dimension around Johnston's solution. The global-search stage is run for one time unit, i.e., from $t = 0$ to $t = 1$. In every 0.1 time units, a starting point for the local search is selected, so there are a total 10 descents performed. We use LSODE to generate both the global- and the local-search trajectories.

Table 5.4 shows the results of Novel in solving the QMF design problems. A value less than 1 for a performance metric means that the design is better on this metric as compared to Johnston's design. Due to the excessive time of the Lagrangian method in solving "48c"

Table 5.4: Experimental results of Novel using trace-based global search and the dynamic Lagrangian method as local search in solving the QMF filter-bank design problems. (A 200-MHz Pentium Pro running Linux.)

Filter-type	E_r	δ_p	E_p	δ_s	E_s	T_r	CPU time (hrs)	# Descents
16a	0.986	1.000	0.858	1.000	1.000	1.000	35.7	10
16b	0.985	1.000	0.893	1.000	1.000	1.000	6.9	10
16c	0.822	1.000	0.919	1.000	1.000	1.000	6.2	10
24b	0.964	1.000	0.778	1.000	1.000	1.000	33.1	3
24c	0.910	1.000	0.768	1.000	1.000	1.000	10.9	10
24d	0.753	1.000	0.770	1.000	1.000	1.000	7.6	10
32c	0.959	1.000	0.738	1.000	1.000	1.000	40.2	4
32d	0.870	1.000	0.800	1.000	1.000	1.000	15.2	10
32e	0.716	1.000	0.889	1.000	1.000	1.000	14.1	10
48d	0.947	0.999	0.756	0.999	1.000	1.000	63.6	2
48e	0.852	1.000	0.838	1.000	1.000	1.000	43.4	10
64e	0.842	1.000	0.737	1.000	1.000	1.000	36.4	2

and "64d" problems, we did not run Novel on them. The last column in Table 5.4 shows the number of local descents based on our Lagrangian method performed in the corresponding CPU time.

Comparing to Table 5.3, Novel with global search finds better solutions than Novel with only local search for problems "16a", "16b", "16c", "24d", and "32e". The reason that no better solutions are found for the other problems is attributed to the good quality of solutions found by the Lagrangian method starting from Johnston's solutions and to the execution time available.

Johnston used sampling in computing energy values whereas Novel used closed-form integration. Hence, designs found by Johnston are not necessarily at the local minima in a continuous formulation. 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 0.789 of Johnston's result. By applying global search, Novel can further improve the design to result in a reconstruction error of 0.753 of Johnston's result.

Figure 5.6 depicts our 24D QMF and Johnston's designs. It indicates that our design has smoother passband response and lower reconstruction error.

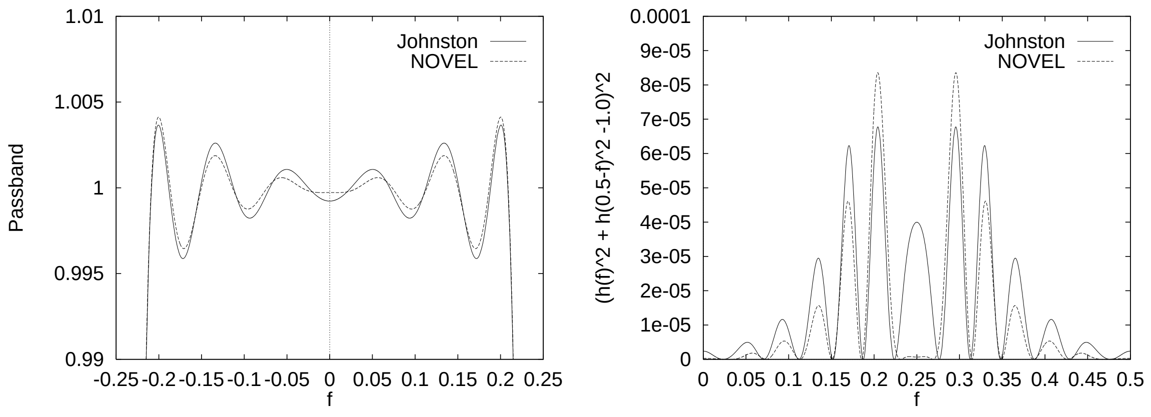


Figure 5.6: A comparison of our 24D QMF design and Johnston's design. The left graph shows the passband frequency response, and the right graph shows the reconstruction error.

To summarize, Novel has improved over Johnston's results over all these filter-bank design problems. The objective, reconstruction error, has been improved for all the problems. Other performance measures are at least as good as Johnston's with a few better. Note that other design methods generally perform trade-offs, resulting in designs that are better in one or more measures but worse in others.

5.5.2 Comparison with Other Global Optimization Methods

We have applied simulated annealing (SA) and evolutionary algorithms (EA) in QMF filter-bank design. The SA we have used is SIMANN from netlib that works on the following weighted-sum formulation:

$$\begin{aligned}\min_{\mathbf{x}} f(\mathbf{x}) &= w_1 \frac{E_r(\mathbf{x})}{\theta_{E_r}} + w_2 \max\left(\frac{E_p(\mathbf{x})}{\theta_{E_p}} - 1, 0\right) \\ &\quad + w_3 \max\left(\frac{E_s(\mathbf{x})}{\theta_{E_s}} - 1, 0\right) + w_4 \max\left(\frac{\delta_p(\mathbf{x})}{\theta_{\delta_p}} - 1, 0\right) \\ &\quad + w_5 \max\left(\frac{\delta_s(\mathbf{x})}{\theta_{\delta_s}} - 1, 0\right) + w_6 \max\left(\frac{T_t(\mathbf{x})}{\theta_{T_t}} - 1, 0\right) \qquad (5.34)\end{aligned}$$

Table 5.5: Experimental results of simulated annealing (SA) for solving QMF filter-bank problems. The cooling rates of SA for 16-, 24-, 32-, 48-, and 64-tap filter banks are 0.98, 0.98, 0.95, 0.85, and 0.80, respectively. The total number of evaluations for 16-, 24-, 32-, 48-, and 64-tap filter banks are 20, 10, 10, 3, and 2 millions, respectively. (A 200-MHz Pentium Pro running Linux.)

Filter-type	E_r	δ_p	E_p	δ_s	E_s	T_r	CPU time (hrs)
16a	0.986	1.000	0.862	1.000	1.000	1.000	12.4
16b	0.985	1.000	0.895	1.000	1.000	1.000	12.8
16c	0.419	0.994	0.423	1.000	1.000	1.015	12.0
24b	0.964	1.000	0.801	1.000	1.000	1.000	12.9
24c	0.853	0.903	0.551	1.000	1.000	1.003	13.4
24d	0.399	0.909	0.404	1.000	1.000	1.018	13.1
32c	0.959	1.000	0.748	1.000	1.000	1.000	16.5
32d	0.617	0.769	0.570	1.000	1.000	1.015	17.0
32e	0.501	1.000	0.583	1.000	1.000	1.013	16.3
48c	0.753	0.999	0.802	0.984	1.000	1.000	13.1
48d	0.943	0.999	0.757	0.999	1.000	1.001	14.4
48e	0.618	0.889	0.585	1.000	1.000	1.015	14.4
64d	0.867	0.926	0.794	0.992	0.959	1.000	30.1
64e	0.541	0.829	0.514	0.976	0.994	1.032	31.2

where θ_{E_r}, θ_{E_p}, θ_{E_s}, θ_{δ_p}, θ_{δ_s}, and θ_{T_t} are performance values of the baseline design. In our experiments, we assign weight 1 for reconstruction error ($w_1 = 1$) and weight 10 for the other performance measures ($w_i = 10, i = 2, \cdots, 6$). SA uses Johnston's solutions as initial points. We have tried various parameter settings and report the best solutions in Table 5.5. Like Novel, SA searches in the range $[-0.01, 0.01]$ in each dimension around Johnston's solution, uses cooling rates of 0.98, 0.98, 0.95, 0.85, and 0.80 for 16-, 24-, 32-, 48-, and 64-tap QMF filter banks, respectively, and runs for a total of 20, 10, 10, 3, and 2 millions evaluations, respectively.

For some problems, SA finds solutions that improve Johnston's solutions on all six performance measures. However, for others, SA's solutions have a larger transition band. This is because in weighted-sum formulation, there is no way to force all the objectives to be smaller than certain values.

Table 5.6: Experimental results of the evolutionary algorithm for solving a constrained formulation (EA-Constr) of QMF filter-bank problems. The population size of EA is $10n$, where n is the number of variables, and the generations for 16-, 24-, 32-, 48-, and 64-tap filter banks are 2000, 2000, 2000, 1000, and 1000, respectively. (A Sun SparcStation 10/51)

Filter-type	E_r	δ_p	E_p	δ_s	E_s	T_r	CPU time (hrs)
16a	937.0	1.000	0.479	1.000	0.769	1.604	2.7
16b	154.2	1.000	1.000	1.000	0.852	1.176	2.7
16c	1.056	1.000	0.854	1.000	1.000	1.000	2.7
24b	889.2	1.000	1.000	1.000	1.000	1.564	7.8
24c	421.9	0.999	1.000	1.000	0.948	1.287	7.5
24d	6.075	1.000	1.000	1.000	1.000	1.003	7.6
32c	2283.8	1.000	1.000	1.000	1.000	1.647	22.0
32d	4.914	1.000	1.000	1.000	0.999	1.011	17.9
32e	0.724	1.000	0.905	1.000	1.000	1.000	24.5
48c	1656.9	1.000	0.999	1.000	0.937	1.166	25.2
48d	1472.6	1.000	1.000	0.994	1.000	1.438	27.7
48e	2.350	1.000	1.000	1.000	1.000	1.001	29.2
64d	3862.7	0.910	0.987	0.942	0.963	0.974	62.9
64e	0.940	1.000	0.854	0.975	1.000	1.000	68.0

The EA used in our experiments is Sprave's Lice (Linear Cellular Evolution) program that can solve both constrained and weighted-sum formulations. As in Novel and SA, we have set the search range to be $[-0.01, 0.01]$ in each dimension around Johnston's solution, and have tried various population size and number of generations.

Table 5.6 reports the experimental results of EA solving the constrained formulation 5.15 (EA-Constr). The fitness value of each individual is based on its feasibility and its objective value. Feasible individuals have higher fitness value than infeasible ones. Among infeasible individuals, the one with a smaller constraint violation has a higher fitness value. The best solutions we obtained are reported in Table 5.6. We have used a population size $10n$, where n is the number of variables, and ran for a total of 2000, 2000, 2000, 1000, and 1000 generations for 16-, 24-, 32-, 48-, and 64-tap QMF filter banks, respectively.

EA-Constr has difficulty in finding good feasible solutions. This is because the constraints based on Johnston's solutions form a tiny feasible region in the search space. Randomly generated points have little chance of being feasible.

Table 5.7 reports the experimental results of EA solving the weighted-sum formulation (EA-Wt). As in the experiments with SA, We assign weight 1 for the reconstruction error and weight 10 for the rest of the performance measures. The population size and the number of generations executed are the same as in the experiments of EA-Constr: the population size is $10n$, where n is the number of variables, and a total of 2000, 2000, 2000, 1000, and 1000 generations were executed for 16-, 24-, 32-, 48-, and 64-tap QMF filter banks, respectively.

Except "16b" and "64d", where EA-Wt improves Johnston's solutions in all six performance measures, EA-Wt only finds solutions with trade-offs. Usually, the solutions of EA-Wt have larger transition bands than Johnston's solutions and improve in other performance measures.

To summarize, the performance improvement of Novel comes from three sources:

- The closed-form formulation used in Novel is more accurate than the sampling method used in Johnston's approach. Local optima found by Novel are true local optima. Johnston's solution are local optima in a discrete approximation of the design problem.
- Novel uses a constrained formulation which allows it to find designs that are guaranteed to be better than or equal to Johnston's design with respect to all performance measures.
- The Lagrangian formulation is solved as a system of ordinary differential equations (ODEs). The system of ODEs precisely converges to the saddle point that corresponds to the local minimum of original constrained optimization problem. ODE solver LSODE solves the equations accurately.
- Novel uses the trace-based global search to overcome local minima, and to find better solutions than pure local-search methods do.

Table 5.7: Experimental results of the evolutionary algorithm for solving the weighted-sum formulation (EA-Wt) of QMF filter-bank problems. The population size of EA is $10n$, where n is the number of variables, and the generations for 16-, 24-, 32-, 48-, and 64-tap filter banks are 2000, 2000, 2000, 1000, and 1000, respectively. (A Sun SparcStation 10/51)

Filter-type	E_r	δ_p	E_p	δ_s	E_s	T_r	CPU time (hrs)
16a	1.520	1.000	0.622	1.000	0.796	1.322	4.1
16b	0.986	1.000	0.893	1.000	1.000	1.000	2.9
16c	0.418	0.994	0.422	1.000	1.000	1.015	2.8
24b	1.049	1.000	0.651	0.997	1.000	1.448	8.3
24c	0.825	0.882	0.511	1.000	1.000	1.011	8.8
24d	0.399	0.909	0.403	1.000	1.000	1.019	8.1
32c	0.842	1.000	0.671	1.000	0.986	1.025	17.8
32d	0.560	0.740	0.504	1.000	1.000	1.026	20.9
32e	0.501	1.000	0.582	1.000	1.000	1.013	19.2
48c	0.780	1.000	0.794	0.812	1.000	1.020	25.4
48d	0.757	0.998	0.606	1.000	1.000	1.049	26.9
48e	0.566	0.886	0.530	1.000	1.000	1.023	28.1
64d	0.803	1.000	0.816	1.000	1.000	1.000	58.7
64e	0.492	0.843	0.445	1.000	1.000	1.033	61.8

Novel improves Johnston's solutions constantly. SA obtains solutions with trade-offs for most of the problems. They improve most performance measures, but has worse transition bandwidth than Johnston's. This is due to the weighted-sum formulations. EA-Wt has the same difficulty in improving over Johnston's solution across all the performance measures. In particular, the solutions of EA-Wt also have larger transition bandwidths. EA-Constr uses a direct search method to probe the search space. When the feasible region is small, direct search methods have difficulty in generating good feasible solutions. Clearly, Novel achieves the best performance among these methods.

5.5.3 Performance Tradeoffs

By using our constrained formulation, we can further study trade-offs in designing QMF filter banks in a controlled environment. Loosening some constraints in (5.15) generally leads to smaller reconstruction errors.

Figure 5.7 demonstrates these trade-offs for 32D QMF filter banks. In our experiments, we have used Johnston's designs as our baselines. In the left graph, constraints on stopband ripple and energy are relaxed by 5% to 50% from Johnston's solution. In the right graph, the constraint on transition bandwidth is relaxed by 5% to 50% from Johnston's solution. The y-axis shows the solution ratio, which is the ratio between the measures found by Novel and that of Johnston's. All solutions obtained by Novel satisfy the constraints.

As shown in the left graph, when constraints on stopband ripple and energy are loosened, the reconstruction error, passband ripple and passband energy decrease, while transition bandwidth is the same with respect to the relaxation ratio. This means that for relaxed requirements of stopband ripple and energy, the solutions have better reconstruction error, passband ripple, and passband energy. But they do not have better transition bandwidth.

In the right graph, the constraint on transition bandwidth is loosened. The reconstruction error and passband energy decrease significantly. The stopband ripple is reduced slightly while the stopband energy is the same.

In the previous experiments, we have relaxed one or two performance requirements. In the next experiments, we study the change of reconstruction error when all the constraints

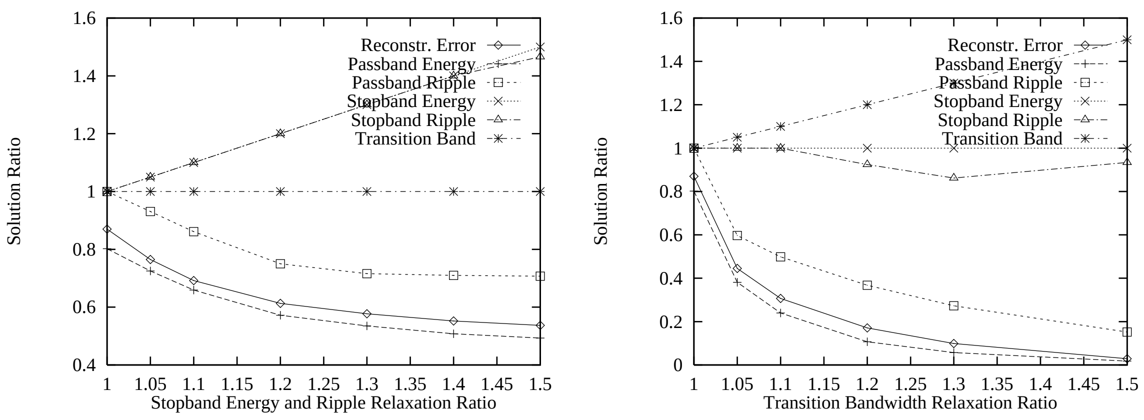
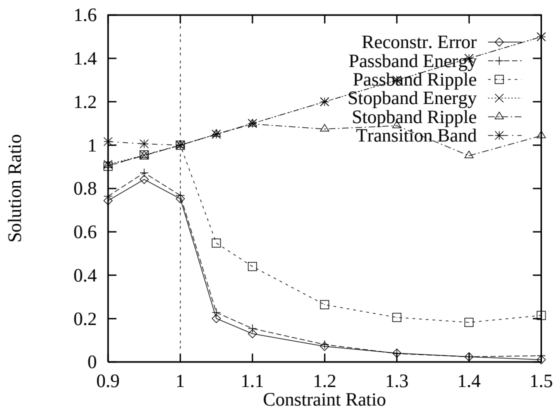


Figure 5.7: Experimental results in relaxing the constraints with respect to Johnston's designs for 32D QMFs. The x-axis shows the relaxation ratio of stopband energy and ripple (left) , or transition bandwidth (right) as constraints in Novel with respect to Johnston's value. The y-axis shows the ratio of the measure found by Novel with respect to Johnston's.

are either tightened or loosened at the same time with the same ratio. Figure 5.8 shows the experimental results in relaxing and in tightening the constraints with respect to Johnston's designs for 24D and 32D QMFs. Tightening all the constraints in (5.15) causes the reconstruction error to increase, whereas loosening them leads to smaller reconstruction error.

When the constraints are loosened, the reconstruction error, passband energy, passband ripple and stopband ripple decrease significantly with respect to the relaxed constraints. These improvements are at the expense of the transition bandwidth and stopband energy, which increase according to the relaxed constraints.

When the constraints are tightened, we have difficulty in finding solutions that satisfy all constraints. For both 24D and 32D QMF, the transition bandwidth is not satisfied according to the tightened constraint. This means that Johnston's solutions are at or very close to the Pareto frontier of optimal solutions. There is no solution that is 5% better than Johnston's solution on all six performance metrics.



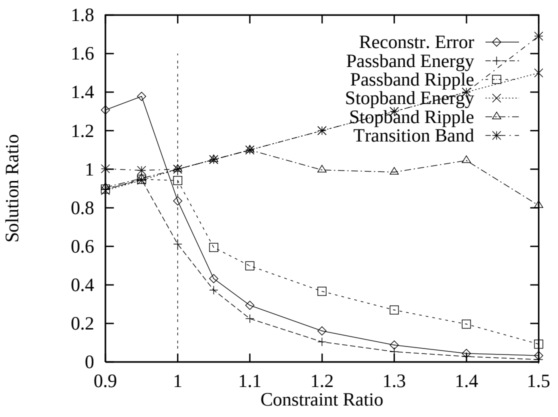


Figure 5.8: Experimental results in relaxing and in tightening the constraints with respect to Johnston's designs for 24D (left) and 32D (right) QMFs. The x-axis shows the ratio of the constraint in Novel with respect to Johnston's value. The y-axis shows the ratio of the performance measures of Novel's solutions with respect to Johnston's.

5.6 Summary

In this chapter, we have applied Novel to design QMF filter banks. We formulate the design problem as a single-objective constrained optimization problem that takes all the performance metrics into consideration. We use the reconstruction error of the overall filter bank as the objective function to be minimized, and five performance metrics of the prototype low-pass filter, including stopband ripple, stopband energy, passband ripple, passband flatness, and transition bandwidth, to form constraint functions constrained by the corresponding performance measures of the baseline solution. Our objective is to improve the reconstruction error while all other performance measures are at least as good as the baseline solution.

For three performance metrics, which include reconstruction error, stopband and passband energies, we derive their closed-form formulas and their corresponding derivatives. For the other three performance metrics, which include stopband ripple, passband ripple, and transition bandwidth, we apply numerical methods to find their values and the corresponding derivatives.

In our experiments, we use Johnston's solutions as the baselines and apply Novel to find better solutions. We compare the performance of our adaptive Lagrangian method with the static Lagrangian method. We show that the convergence time of the static Lagrangian method is generally longer than that of the dynamic Lagrangian method, and can vary significantly depending on the initial weights.

By using the dynamic Lagrangian method, we have successfully improved the solutions of all the test problems. We have found solutions with smaller reconstruction errors than Johnston's solutions, while all the other performance metrics are equal to or better than those of Johnston's solutions. The nonlinear functions in the objective and the constraints produce local minima that trap local-search methods. By using trace-based global search in Novel, we have further improved the solutions obtained by the Lagrangian methods.

We have compared the results obtained by Novel with those obtained by simulated annealing (SA) and evolutionary algorithm (EA). SA solves an unconstrained optimization problem with a weighted-sum formulation. EA works on both the weighted-sum formulation and the single-objective constrained formulation solved by Novel. It is inherently difficult to improve all the performance measures at the same time by using the weighted-sum formulation. SA and EA usually obtain trade-off solutions with some performance metrics improved and others getting worse. In solving the constrained formulation, EA fails to find good feasible solutions for all the test problems because the feasible regions are very small and difficult to find by random probing. Overall, Novel has the best performance and improves the baseline solutions consistently.

Using our constrained optimization formulation, we have studied performance trade-offs among the six performance metrics of a QMF filter bank. We have found that relaxing some constraints slightly can lead to a large reduction of the reconstruction error. When the stopband ripple and energy or the transition bandwidth is relaxed, the other three performance measures, reconstruction error, stopband energy, and stopband ripple can become significantly smaller. When we relax all the constraints, the stopband energy and transition bandwidth usually meet the relaxed constraints, which other metrics are significantly improved.

As a final note, the design approach and the Novel method we have applied to design QMF filter banks can be carried out to design other digital filters and filter banks, such as IIR filters and filter banks and multi-rate and multi-band filter banks.

6. DISCRETE OPTIMIZATION

In this chapter, we apply Novel, particularly the discrete Lagrangian method (DLM), to solve discrete optimization problems, including the satisfiability problems (SAT), the maximum satisfiability problems (MAX-SAT), and the design of multiplierless (power-of-two, PO2) QMF filter-banks.

SAT is a class of NP-complete problems that model a wide range of real-world applications. These problems are difficult to solve because they have many local minima in their search space, often trapping greedy search methods that utilize some form of descent. We formulate SAT problems as discrete constrained optimization problems and apply Novel to solve them. Instead of restarting from a new starting point when a search reaches a local trap, the Lagrange multipliers in our discrete Lagrangian method provide a force to lead the search out of a local minimum and move it in the direction provided by the Lagrange multipliers. One of the major advantages of our method is that it has very few algorithmic parameters to be tuned by users, and the search procedure can be made deterministic and the results, reproducible. We demonstrate our method by applying it to solve an extensive set of benchmark problems archived in DIMACS of Rutgers University. Our method usually performs better than the best existing methods and can achieve an order-of-magnitude speedup for many problems.

MAX-SAT is a general case of SAT. As in solving SAT, we formulate a MAX-SAT problem as a discrete constrained optimization problem and solve it using Novel. In our experiments, we have applied Novel to solve a large number of MAX-SAT test problems. Novel is usually two orders faster than other competing methods, and can also find better solutions.

Finally, we apply Novel to design multiplierless QMF filter banks. In multiplierless QMF filter banks, filter coefficients are represented as sums or differences of powers of two, and multiplications become additions, subtractions, and shifting. Without the complexity of multiplications, each filter takes less area to implement in VLSI, and more filter taps could be accommodated in a given area to achieve higher performance. We formulate the design problem as a discrete constrained optimization problem and solve it using Novel. Novel has obtained designs with similar quality to real-value designs, but less computation and hardware implementation costs.

This chapter is divided into 8 sections. In Section 6.1 to Section 6.5, we present the application of Novel to solve SAT problems. In Section 6.6, we show the application of Novel to solve MAX-SAT problems. In Section 6.7, we present the application of Novel in designing multiplierless QMF filter banks. Finally, in Section 6.8, we conclude this chapter.

6.1 Introduction to Satisfiability Problems

The satisfiability (SAT) problem is one of the most studied problems in contemporary complexity theory. It was the first NP-complete problem discovered by Stephen Cook in 1971.

Define a set of Boolean variables $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, $x_i \in \{0, 1\}$, and let the complement of any of these variables x_i be denoted as $\bar{x}_i$. In propositional logic, the variables are called *literals* . Using standard notations, where the symbol $\vee$ denotes *or* and $\wedge$ denotes *and*, we can write any Boolean expression in *conjunctive normal form* (CNF), i.e., a finite conjunction of disjunctions, in which each literal appears at most once. Each disjunctive grouping is called a *clause*. Given a set of m clauses $\{C_1, C_2, \cdots, C_m\}$ on the n variables $\mathbf{x}$, a Boolean formula in conjunctive normal form (CNF) is

$$C_1 \wedge C_2 \wedge \cdots \wedge C_m \tag{6.1}$$

A clause is satisfied if at least one of its member literals is true. A Boolean expression is satisfied if all of its clauses are simultaneously satisfied given a particular assignment to the literals (variables).

The satisfiability problem is defined as follows:

> *Given a set of literals and a conjunction of clauses defined over the literals, find an assignment of values to the literals so that the Boolean expression is true, or derive its infeasibility if the Boolean expression is infeasible.*

The SAT problem belongs to an important class of discrete constraint-satisfaction problems (CSP). Many problems in artificial intelligence, logic, computer aided design, database query and planning, etc., can be formulated as SAT problems. These problems are known to be NP-complete and require algorithms of exponential complexity in the worst case to obtain a satisfying assignment.

Many search methods have been developed in the past for solving SAT problems. These include resolution, constraint satisfaction, and backtracking. These methods are computationally expensive and are not suitable to apply to large problems.

In addition to the formulation in (6.1), SAT problems can be formulated as discrete or continuous, constrained or unconstrained, optimization problems. In Section 6.2, we present five formulations, show the objective and/or constraints for each formulation, and discuss approaches for solving each.

SAT algorithms can be classified as incomplete and complete, depending on whether they can find a random solution or find all the solutions. The advantage of complete algorithms is that they can detect infeasibility when a SAT problem is infeasible. However, they are generally computationally expensive and are suitable for relatively small problems. On the other hand, incomplete methods are much faster, but cannot conclude whether a SAT problem is feasible or infeasible when no solution is found within a limited amount of time.

Recently, a class of local (incomplete) search methods were proposed, solving a class of hard SAT problems with size of an order-of-magnitude larger than those solved by complete methods. A major disadvantage of these methods is that they require users to set some problem-specific parameters in order to find solutions efficiently. For this reason, one of our goals is to design a fast local search method whose results can be reproduced easily.

In our approach, we formulate a SAT problem as a discrete constrained optimization problem with a goal of minimizing $N(\mathbf{x})$ subject to a set of constraints:

$$\begin{aligned} \min_{\mathbf{x}\in\{0,1\}^n} \quad & N(\mathbf{x}) = \sum_{i=1}^{m} U_i(\mathbf{x}) \\ \text{subject to} \quad & U_i(\mathbf{x}) = 0 \quad \forall i \in \{1, 2, \ldots, m\}. \end{aligned} \tag{6.2}$$

where $U_i(\mathbf{x}) = 0$ if the logical assignment $\mathbf{x}$ satisfies C_i, and $U_i = 1$ if C_i is false. $N(\mathbf{x})$ measures the number of unsatisfied clauses, and $N(\mathbf{x}) = 0$ when all the clauses are satisfied. We then apply Novel with the discrete Lagrangian method to solve this problem.

Traditionally, Lagrangian methods have been developed to solve continuous constrained optimization problems. By doing descents in the original variable space and ascents in the Lagrange-multiplier space, equilibrium is reached when optimal solutions are found. To apply these methods to solve discrete SAT problems, we need to develop discrete Lagrangian operators that can work on discrete values. Novel uses the *Discrete Lagrangian method* (DLM) to solve discrete optimization problems, and searches for saddle points based on a discrete Lagrangian function. Equilibrium is reached when a feasible assignment to the original problem is found. Novel moves a search trajectory out of a local minimum in a direction provided by the Lagrange multipliers, without restarting the search.

In the next four sections of this chapter, we present our results in solving SAT problems. In Section 6.2, we first summarize previous formulations and algorithms to solve SAT problems. In Section 6.3, we apply our discrete Lagrangian method (DLM) to solve SAT problems, and derive related theoretical foundations. In Section 6.4, we address the issues and alternatives in implementing DLM. Finally, in Section 6.5, we show our experimental results in applying DLM to solve SAT problems from the DIMACS benchmark suite. Afterwards, we switch to present our results in applying Novel to solve other discrete problems.

6.2 Previous Work

In this section, we review the previous work in solving SAT problems, which include various discrete and continuous, constrained and unconstrained formulations, and the corresponding algorithms for solving them.

6.2.1 Discrete Formulations

Discrete formulations of SAT problems can be classified as unconstrained versus constrained as follows:

(a) *Discrete Constrained Feasibility Formulation.* This is the formulation defined in (6.1). Methods to solve it can be either complete or incomplete, depending on their ability to prove infeasibility. Complete methods for solving this formulation include resolution [89, 207], backtracking [203], and consistency testing [100, 107, 110]. An important resolution method is Davis-Putnam's algorithm [61]. These methods enumerate the search space systematically, and may rely on incomplete methods to find feasible solutions. Their disadvantage is that they are computationally expensive. For instance, Selman *et al.* [222] and Gu [105, 106, 108] have reported that Davis-Putnam's algorithm cannot handle SAT problems with more than 150 variables, and better algorithms today have difficulty in solving SAT problems with more than 400 variables.

(b) *Discrete Unconstrained Formulation.* In this formulation, the goal is to minimize $N(\mathbf{x})$, the number of unsatisfiable clauses. That is,

$$\min_{\mathbf{x} \in \{0,1\}^n} N(\mathbf{x}) = \sum_{i=1}^{m} U_i(\mathbf{x}) \tag{6.3}$$

Many local search algorithms were designed for this formulation. These algorithms can deal with large SAT problems of thousands of variables. However, they may be trapped by local minima in the search space, where no state in its local neighborhood is strictly better. Consequently, steepest-descent or hill-climbing methods will be trapped there. Restarts merely bring the search to a completely new region.

Methods designed for (6.3) are usually incomplete methods, although some mechanisms like backtracking can make them complete. Most incomplete methods are random methods, relying on ad hoc heuristics to find random solutions quickly. Those that have been applied include multi-start (restart) of local improvement (descent) methods, stochastic methods such as simulated annealing (SA) [41,144], and genetic algorithms (GA) [121,164]. They are discussed briefly as follows.

A pure descent method with multi-starts descends in the space of the objective function from an initial point, and generates a new starting point when no further improvement can be found locally. Examples include hill-climbing and steepest descent [100,105,166,167,222, 223,237,243]. For large SAT problems, hill-climbing methods are much faster than steepest descent because they descend in the first direction that leads to improvement, whereas steepest descent methods find the best direction. An example of an objective function suitable to be searched by descent or hill-climbing methods is (6.3). Pure descent methods are not suitable when there are constraints in the search space as formulated in (6.2).

Recently, some local search methods were proposed and applied to solve large SAT problems [60, 90, 173]. The most notable ones are those developed independently by Gu and Selman.

Gu developed a group of local search methods for solving SAT and CSP problems. In his Ph.D thesis [100], he first formulated conflicts in the objective function and proposed a discrete relaxation algorithm (a class of deterministic local search) to minimize the number of conflicts in these problems. The algorithms he developed subsequently focused on two components: methods to continue a search when it reaches a local minimum, and methods for variable selection and value assignment. In the first component, he first developed the so-called *min-conflicts* heuristic [100] and showed significant performance improvement in solving large size SAT, n-queen, and graph coloring problems [100,237,241–243]. His methods use various local handlers to escape from local traps when a greedy search stops progressing [101–103,108–110]. Here, a search can continue without improvement when it reaches a local minimum [108] and can escape from it by a combination of backtracking, restarts, and random swaps. In variable selection and value assignment, Gu and his colleagues have developed random and partial random heuristics [101,102,104–106,108–110,237,241,242].

These simple and effective heuristics significantly improve the performance of local search algorithms by many orders of magnitude.

Selman developed GSAT [219,222–226] that starts from a randomly generated assignment and performs local search iteratively by flipping variables. Such flipping is repeated until either a satisfiable assignment is found or a pre-set maximum number of flips is reached. When trapped in a local minimum, GSAT either moves up-hill or jumps to another random point. To avoid getting stuck on a *plateau*, which is not a local minimum, GSAT makes side-way moves.

In short, the objective function in (6.3) may have many local minima that trap local search methods. Consequently, a search in a seemingly good direction may get stuck in a small local minimum, and will rely on random restarts or hill climbing to bring the search out of the local minimum. However, both schemes do not explore the search space systematically, and random restarts may bring the search to a completely different search space.

Stochastic methods, such as GA and SA, have more mechanisms to bring a search out of a local minimum, but are more computationally expensive. Selman *et al.* [223] reported that annealing is not effective for solving SAT problems. To the best of our knowledge, there is no successful application of genetic algorithms to solve SAT problems. In general, these methods are much slower than descent methods and can only solve small problems.

(c) *Discrete Constrained Formulation.* There are various forms in this formulation. One approach is to formulate SAT problems as 0-1 integer linear programming (ILP) problems, and apply existing ILP algorithms to solve them [122, 138]. However, this approach is generally computationally expensive.

Another approach is to minimize an objective function $N(\mathbf{x})$, subject to a set of constraints, as defined in (6.2) and restated as follows:

$$\begin{aligned} &\min_{\mathbf{x}\in\{0,1\}^n} \quad N(\mathbf{x}) = \sum_{i=1}^{m} U_i(\mathbf{x}) \\ &\text{subject to} \quad U_i(\mathbf{x}) = 0 \quad \forall i \in \{1, 2, \ldots, m\}. \end{aligned}$$

This formulation is better than those in (6.3) and (6.1) because the constraints provide another mechanism to bring the search out of a local minimum. When a search is stuck in a local minimum, the objective value as formulated in (6.3) is a discrete integer, and the vicinity of the local minimum may either be worse or be the same. On the other hand, in formulating the problem as in (6.1), there is very little guidance given to the search algorithm as to which variable to flip when a clause is not satisfied. Novel solves SAT problems based on the discrete constrained formulation (6.2). We show efficient heuristic algorithms that search in discrete space, while satisfying the constraints.

6.2.2 Continuous Formulations

In formulating a discrete SAT problem in continuous space, we transform discrete variables in the original problem into continuous variables in such a way that solutions to the continuous problem are binary solutions to the original problem. This transformation is potentially beneficial because an objective in continuous space may smooth out some infeasible solutions, leading to smaller number of local minima explored. Unfortunately, continuous formulations require computationally expensive algorithms, rendering them applicable to only small problems. In the following, we show two such formulations:

(a) *Continuous Unconstrained Formulation.*

$$\min_{\mathbf{x}\in R^n} f(\mathbf{x}) = \sum_{i=1}^{m} c_i(\mathbf{x}), \tag{6.4}$$

where $c_i(\mathbf{x})$ is a transformation of clause C_i:

$$c_i(\mathbf{x}) \quad = \quad \prod_{j=1}^{n} a_{i,j}(x_j) \tag{6.5}$$

$$a_{i,j}(x_j) \quad = \quad \begin{cases} (1-x_j)^2 & \text{if } x_j \text{ in } C_i \\ x_j^2 & \text{if } \bar{x}_j \text{ in } C_i \\ 1 & \text{otherwise} \end{cases} \tag{6.6}$$

Values of $\mathbf{x}$ that make $f(\mathbf{x}) = 0$ are solutions to the original problem in (6.1).

Note that the objective is a nonlinear polynomial function. Hence, there may be many local minima in the search space, and descent methods, such as gradient descent, conjugate gradient, and quasi-Newton's methods, can be trapped by the local minima [103–106, 161]. Global search techniques, such as clustering methods, generalized gradient methods, Bayesian methods, and stochastic methods, can also be applied; however, they are usually much more computationally expensive than descent methods.

To overcome the inefficiency of continuous unconstrained optimization methods, Gu developed discrete bit-parallel optimization algorithms (SAT 14.5 and SAT 14.6) to evaluate continuous objective function [106] and have found significant performance improvements.

(b) *Continuous Constrained Formulation.* This generally involves a heuristic objective function that indicates the quality of the solution obtained (such as the number of clauses satisfied). One formulation similar to (6.4) is as follows:

$$\begin{aligned} \min_{\mathbf{x} \in R^n} \quad & f(\mathbf{x}) = \sum_{i=1}^{m} c_i(\mathbf{x}) \\ \text{subject to} \quad & c_i(\mathbf{x}) = 0 \quad \forall i \in \{1, 2, \ldots, m\} \end{aligned} \tag{6.7}$$

where $c_i(\mathbf{x})$ is defined in (6.5).

The key in the continuous approach lies in the transformation. When it does not smooth out local minima in the discrete space or when the solution density is low, continuous methods are much more computationally expensive to apply than discrete methods.

Since (6.7) is a continuous constrained optimization problem with a nonlinear objective function and nonlinear constraints, we can apply existing Lagrange-multiplier methods to solve it. Our experience is that a Lagrangian transformation does not reduce the number of local minima, and continuous Lagrangian methods are an order-of-magnitude more expensive to apply than the corresponding discrete algorithms [45].

6.3 Applying DLM to Solve SAT Problems

To apply our discrete Lagrangian method (DLM) introduced in Section 2.2.4 to solve a SAT problem, we introduce an artificial objective $H(\mathbf{x})$ and formulate the problem into a constrained optimization problem as follows:

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

where $U_i = 0$ if C_i is true, and $U_i = 1$ if C_i is false.

Let's show that a saddle point $(\mathbf{x}^*, \lambda^*)$ to the Lagrangian formulation of (6.8) consists of a feasible solution $\mathbf{x}^*$ to the SAT problem in (6.1). For a given saddle point $(\mathbf{x}^*, \lambda^*)$,

$$\begin{aligned} F(\mathbf{x}^*,\lambda) \le F(\mathbf{x}^*,\lambda^*) \quad &\Rightarrow \quad H(\mathbf{x}^*) + \lambda^T U(\mathbf{x}^*) \le H(\mathbf{x}^*) + \lambda^{*T} U(\mathbf{x}^*) \\ &\Rightarrow \quad \lambda^T U(\mathbf{x}^*) \le \lambda^{*T} U(\mathbf{x}^*) \end{aligned}$$

for any λ close to λ^*. The condition holds only when $U(\mathbf{x}^*) = 0$; *i.e.*, $\mathbf{x}^*$ is a feasible solution to the SAT problem in (6.1).

In order for all feasible solutions to the SAT problem in (6.1) to become saddle points of (6.8), the objective function $H(\mathbf{x})$ must satisfy the following condition:

Necessary Condition for Objective Function. If a feasible solution $\mathbf{x}^*$ to the SAT problem in (6.1) is a local minimum of the objective function $H(\mathbf{x})$, then $(\mathbf{x}^*, \lambda^*)$ is a saddle point of the Lagrangian formulation (6.8) for any positive λ^*.

Proof. The Lagrangian function of (6.8) is

$$F(\mathbf{x},\lambda) = H(\mathbf{x}) + \lambda^T U(\mathbf{x})$$

For a feasible solution $\mathbf{x}^*$ of (6.1), $U(\mathbf{x}^*) = 0$. Hence, for any positive λ^* and any λ,

$$F(\mathbf{x}^*,\lambda) = H(\mathbf{x}^*) + \lambda^T U(\mathbf{x}^*) = H(\mathbf{x}^*) = H(\mathbf{x}^*) + \lambda^{*T} U(\mathbf{x}^*) = F(\mathbf{x}^*,\lambda^*).$$

By the definition of $U(\mathbf{x})$, $U(\mathbf{x}) \ge 0$ for any $\mathbf{x}$. Since $\mathbf{x}^*$ is a local minima of $U(\mathbf{x})$ and λ^* is positive,

$$F(\mathbf{x}^*,\lambda^*) = H(\mathbf{x}^*) + \lambda^{*T} U(\mathbf{x}^*) = H(\mathbf{x}^*) \le H(\mathbf{x}) + \lambda^{*T} U(\mathbf{x}) = F(\mathbf{x},\lambda^*)$$

for $\mathbf{x}$ close to $\mathbf{x}^*$. Therefore, we have

$$F(\mathbf{x}^*, \lambda) = F(\mathbf{x}^*, \lambda^*) \le F(\mathbf{x}, \lambda^*)$$

and $(\mathbf{x}^*, \lambda^*)$ is a saddle point. ■

There are many functions that satisfy the necessary condition. Examples are

$$H(\mathbf{x}) = \sum_{i=1}^{m} U_i(\mathbf{x}) \tag{6.9}$$

$$H(\mathbf{x}) = \sum_{i=1}^{m} l_i U_i(\mathbf{x}) \tag{6.10}$$

$$H(\mathbf{x}) = \sum_{i=1}^{m} (l_{max} + 1 - l_i) U_i(\mathbf{x}) \tag{6.11}$$

where l_i is the number of variables in clause C_i, and $l_{max} = \max(l_i, i = 1, \cdots, m)$. The first function assigns uniform weights to all the clauses. The second function assigns more weights to longer clauses, whereas the third one is the opposite. A feasible solution $\mathbf{x}^*$ is a local minimum of these functions because $U_i(\mathbf{x}^*) = 0, i = 1, \cdots, m$, and $H(\mathbf{x}^*) = 0$. In our experiments, we use the first function as our objective function, and formulate SAT problems as constrained optimization problems in (6.2).

The SAT problem defined in (6.2) is a special case of the discrete constrained optimization problem defined in (2.21). An important property of the formulation in (6.2) is that all local minima are also the global minima. This is true because, based on (6.2), $\mathbf{x}^*$ is defined as a local minimum if $U(\mathbf{x}^*) = 0$, which implies that $N(\mathbf{x}^*) = 0$ (a global minimum). This condition is stated more formally as follows:

Necessary and Sufficient Condition for Optimality. $\mathbf{x}^*$ is a global minimum of the SAT formulation in (6.2) if and only if $U(\mathbf{x}^*) = 0$.

Proof. Straightforward. ■

Due to this property, a SAT problem formulated in (6.2) can be solved by the difference equations in (2.24) and (2.25) that find saddle points of a discrete Lagrangian function. In the rest of this section, we show how the general discrete Lagrangian method (DLM) can be applied to solve discrete SAT problems.

The discrete Lagrangian function for (6.2) is defined as follows:

$$L(\mathbf{x},\lambda) = N(\mathbf{x}) + \lambda^T U(\mathbf{x}) = \sum_{i=1}^{m} U_i(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i U_i(\mathbf{x}) = \sum_{i=1}^{m} (1+\lambda_i) U_i(\mathbf{x}) \tag{6.12}$$

where $\mathbf{x} \in \{0,1\}^n$, $U(\mathbf{x}) = (U_1(\mathbf{x}), \ldots, U_m(\mathbf{x})) \in \{0,1\}^m$, and λ^T is the transpose of $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ that denotes the Lagrange multipliers. Due to the binary domain of $\mathbf{x}$, the neighborhood of $\mathbf{x}$ is more restricted than in general discrete problems, which is reflected in the definition of the saddle point.

A saddle point $(\mathbf{x}^*, \lambda^*)$ of $L(\mathbf{x},\lambda)$ is defined as one that satisfies the following condition:

$$L(\mathbf{x}^*, \lambda) \leq L(\mathbf{x}^*, \lambda^*) \leq L(\mathbf{x}, \lambda^*) \tag{6.13}$$

for all λ sufficiently close to λ^* and for all $\mathbf{x}$ whose Hamming distance between $\mathbf{x}^*$ and $\mathbf{x}$ is 1.

Saddle-Point Theorem for SAT. $\mathbf{x}^*$ is a global minimum of (6.2) if and only if there exists some λ^* such that $(\mathbf{x}^*, \lambda^*)$ constitutes a saddle point of the associated Lagrangian function $L(\mathbf{x},\lambda)$.

Proof: The Saddle-Point Theorem discussed in Section 2.2.4 for discrete problems can be applied here. A simpler proof is as follows:

"$\Leftarrow$" part: Since $(\mathbf{x}^*, \lambda^*)$ is a saddle point, $L(\mathbf{x}^*, \lambda) \leq L(\mathbf{x}^*, \lambda^*)$ for λ sufficiently close to λ^*. From the definition of the Lagrangian function in (6.12), this implies

$$\sum_{i=1}^{m} \lambda_i U_i(\mathbf{x}^*) \leq \sum_{i=1}^{m} \lambda_i^* U_i(\mathbf{x}^*).$$

Suppose some $U_k(\mathbf{x}^*) \neq 0$, which means $U_k(\mathbf{x}^*) = 1$. Then $\lambda = (\lambda_1^*, \cdots, \lambda_k^* + \delta, \cdots, \lambda_n^*)$ would violate the inequality for a positive δ. Therefore, $U(\mathbf{x}^*) = 0$, and $\mathbf{x}^*$ is a global minimum.

"$\Rightarrow$" part: If $\mathbf{x}^*$ is a global minimum, then $U(\mathbf{x}^*) = 0$, and $L(\mathbf{x}^*, \lambda^*) = L(\mathbf{x}^*, \lambda) = 0$. The vector $\lambda^* \geq 0$ makes $L(\mathbf{x}^*, \lambda^*) \leq L(\mathbf{x}, \lambda^*)$. Therefore, $(\mathbf{x}^*, \lambda^*)$ is a saddle point. ■

Corollary. If a SAT problem formulated in (6.2) is feasible, then any algorithm that can find a saddle point of $L(\mathbf{x}, \lambda)$ defined in (6.12) from any starting point can find a feasible solution to the SAT problem.

Proof: If a SAT problem is feasible, then its solutions are global minima of (6.2). These correspond to saddle points of $L(\mathbf{x}, \lambda)$ defined in (6.12). If an algorithm can find a saddle point from any starting point, then it will find a solution to the problem. ■

Since a Lagrangian method only stops at saddle points, this corollary implies that the method will find a saddle point regardless of its starting point (including the origin) if the problem is feasible. Unfortunately, the corollary does not guarantee that it will find a saddle point in a finite amount of time.

To apply DLM to solve SAT problems, we define the *discrete gradient operator* $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$ with respect to $\mathbf{x}$ such that $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$ points to state $\mathbf{x}'$ in the Hamming distance-1 neighborhood of the current state $\mathbf{x}$ that gives the maximum improvement in $L(\mathbf{x}, \lambda)$. If no state in the 1-neighborhood of $\mathbf{x}$ improves $L(\mathbf{x}, \lambda)$, then $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$.

Figure 6.1 illustrates the discrete gradient operator. It is a 3-dimensional problem. The current point $\mathbf{x}$ has Lagrangian value $L = 10$. States within Hamming distance of 1 from $\mathbf{x}$ have Lagrangian values 7, 10, and 13 along dimensions 1, 2, and 3, respectively. The neighboring state along dimension 1 improves the Lagrangian value most. Therefore, $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = (1, 0, 0)$.

Next, we propose a method to update $(\mathbf{x}, \lambda)$ so that it will eventually satisfy the optimality condition defined in (6.13).

Discrete Lagrangian Method (DLM) $\mathcal{A}$ for SAT.

$$\mathbf{x}^{k+1} = \mathbf{x}^k \oplus \Delta_{\mathbf{x}} L(\mathbf{x}^k, \lambda^k) \tag{6.14}$$

$$\lambda^{k+1} = \lambda^k + U(\mathbf{x}^k) \tag{6.15}$$

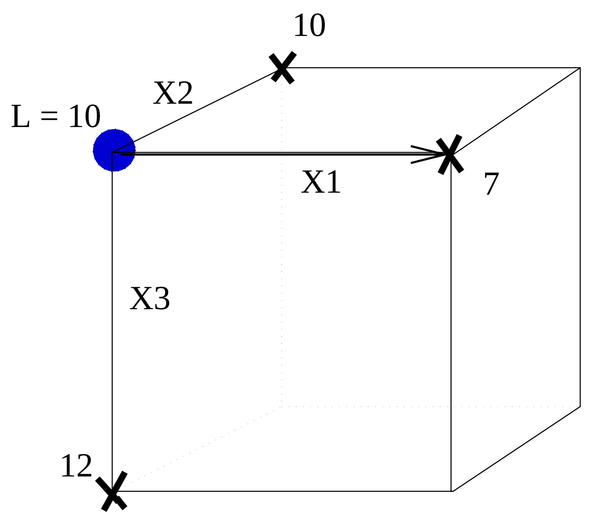


Figure 6.1: Illustration of the discrete gradient operator for SAT.

It is easy to see that the necessary condition for algorithm $\mathcal{A}$ to converge is when $U(\mathbf{x}) = 0$, implying that $\mathbf{x}$ is optimal. If any of the constraints in $U(\mathbf{x})$ is not satisfied, then λ will continue to evolve to handle the unsatisfied constraints.

The following theorem establishes the correctness of $\mathcal{A}$ and provides the conditions for termination.

Fixed-Point Theorem for SAT. An optimal solution $\mathbf{x}^*$ to the SAT problem defined in (6.2) is found if and only if $\mathcal{A}$ terminates.

Proof. "$\Rightarrow$" part: If $\mathcal{A}$ terminates, then $U(\mathbf{x}) = 0$, which makes $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$. Since this is a sufficient condition for the optimality of (6.2), the optimal solution is found.

"$\Leftarrow$" part: If an optimal solution $\mathbf{x}^*$ is found, then according to the necessary condition of optimality, $U(\mathbf{x}^*) = 0$, implying that $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$. Therefore, neither $\mathbf{x}$ nor λ will change, leading to the conclusion that $\mathcal{A}$ terminates. ■

Example. The following simple example illustrates the discrete Lagrangian algorithm. The problem has four variables, $\{x_1, x_2, x_3, x_4\}$, 7 clauses,

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

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

and 2 solutions, {(1, 0, 0, 0), (0, 0, 1, 1)},

Algorithm $\mathcal{A}$ works as follows:

(1) Initially, $\mathbf{x}^0 = \{(1, 1, 1, 1)\}$ and $\lambda^0 = \{(0, 0, 0, 0, 0, 0, 0)\}$, and $L(\mathbf{x}^0, \lambda^0) = 1$.

(2) The L values of neighboring points of the initial point are

$$L((1,1,1,0), \lambda^0) = L((1,1,0,1), \lambda^0) = L((1,0,1,1), \lambda^0) = L((0,1,1,1), \lambda^0) = 1.$$

Since $L(\mathbf{x}^0, \lambda^0)$ is less than or equal to the values of the neighboring points, $\Delta_{\mathbf{x}} L(\mathbf{x}^0, \lambda^0) = 0$. As the fourth clause is not satisfied, λ^1 is updated to {0, 0, 0, 1, 0, 0, 0}. Further, $\mathbf{x}$ is updated to be $\mathbf{x}^1 = \mathbf{x}^0$. Note that λ_4, the penalty for the fourth clause, is increased in order to provide a force to pull the search out of the local minimum.

(3) $L(\mathbf{x}^1, \lambda^1) = 2$. The L values of $\mathbf{x}^1$'s neighboring points are

$$L((1,1,1,0), \lambda^1) = L((1,1,0,1), \lambda^1) = L((0,1,1,1), \lambda^1) = 1$$
$$\text{and } L((1,0,1,1), \lambda^1) = 2.$$

There are three choices of $\Delta_{\mathbf{x}} L(\mathbf{x}^1, \lambda^1)$.

- If we choose $\Delta_{\mathbf{x}} L(\mathbf{x}^1, \lambda^1) = (0,0,0,1)$, then $\mathbf{x}^2 = (1,1,1,0)$ and $\lambda^2 = (0,0,0,2,0,0,0)$.
- If we choose $\Delta_{\mathbf{x}} L(\mathbf{x}^1, \lambda^1) = (0,0,1,0)$, then $\mathbf{x}^2 = (1,1,0,1)$ and $\lambda^2 = (0,0,0,2,0,0,0)$.
- If we choose $\Delta_{\mathbf{x}} L(\mathbf{x}^1, \lambda^1) = (1,0,0,0)$, then $\mathbf{x}^2 = (0,1,1,1)$ and $\lambda^2 = (0,0,0,2,0,0,0)$.

Assume that we choose $\Delta_{\mathbf{x}} L(\mathbf{x}^1, \lambda^1) = (1,0,0,0)$ in this example.

(4) $\mathbf{x}^2 = (0,1,1,1)$ and $L(\mathbf{x}^2, \lambda^2) = 1$. The L values of neighboring points are

$$L((1,1,1,1), \lambda^2) = 3,\ L((0,0,1,1), \lambda^2) = 0,$$
$$\text{and } L((0,1,0,1), \lambda^2) = L((0,1,1,0), \lambda^2) = 1.$$

Therefore, $\Delta_{\mathbf{x}} L(\mathbf{x}^2, \lambda^2) = (0,1,0,0)$. $\mathbf{x}$ and λ are updated to $\mathbf{x}^3 = (0,0,1,1)$ and $\lambda^3 = (0,0,0,2,0,1,0)$, respectively.

(5) $U(\mathbf{x}^3) = 0$ implies that $\Delta_{\mathbf{x}} L(\mathbf{x}^3, \lambda^3) = 0$. Hence, $\mathcal{A}$ terminates, and $\mathbf{x}^3$ is a solution.

Note that λ defined in (6.15) is non-decreasing. Ways to decrease λ are discussed in Section 6.4.2. ■

One of the important features of DLM is that it continues to search until a solution to the SAT problem is found, independent of its starting point. Therefore, DLM does not involve restarts that are generally used in other randomized search methods. However, as in other incomplete methods, DLM does not terminate if there is no feasible solution. Further, as in general Lagrangian methods, the time for DLM to find a saddle point can only be determined empirically.

DLM has a continuous search trajectory without any breaks. DLM descends in the original variable space, similar to what is used in other algorithms. When the search reaches a local minimum, DLM brings the search out of the local minimum using its Lagrange multipliers. As a result, we often see small fluctuations in the number of unsatisfied clauses as the search progresses, indicating that DLM bypasses small "dents" in the original variable space with the aid of Lagrange multipliers. In contrast, some local search algorithms rely on randomized mechanisms to bring the search out of local minima in the original variable space. When the search reaches a local minimum, it is restarted from a completely new starting point. Consequently, the search may fail to explore the vicinity of the local minimum it has just reached, and the number of unsatisfied clauses at the new starting point is unpredictable.

In contrast to other local search methods, such as Gu's local-search methods [104–106] and GSAT [219, 222–226], DLM descends in Lagrangian space. To get out of local minima that do not satisfy all the constraints, DLM increases the penalties on constraints that are violated, recording history information on constraint violation in the Lagrange multipliers. Eventually as time passes, the penalties on violated constraints will be very large, forcing these constraints to be satisfied. On the other hand, GSAT uses uphill movements and random restarts to get out of local minima, whereas Gu's local-minimum handler uses stochastic mechanisms to escape from local minima.

Although our strategy is similar to Morris' "break-out" strategy [173] and Selman and Kautz's GSAT [222, 223] that applies adaptive penalties to escape from local minima, DLM provides a theoretical framework for better understanding of these heuristic strategies. In addition, DLM can incorporate new techniques for controlling Lagrange multipliers in order to obtain improved performance. Some of these strategies are described in Section 6.4.2.

6.4 Implementations of DLM

In this section we discuss issues related to the implementation of DLM and present three implementations. There are three components in applying a Lagrangian method: evaluating the derivative of the Lagrangian function, updating the Lagrange multipliers, and evaluating the constraint functions. In the continuous domain, these operations are computationally expensive, especially when the number of variables is large and the function is complex. However, as we show in this and the next sections, implementation in the discrete domain is very efficient, and our method is faster than other local-search methods for solving SAT problems.

6.4.1 Algorithmic Design Considerations

The general algorithm of DLM is shown in Figure 6.2. It performs descents in the original variable space of $\mathbf{x}$ and ascents in the Lagrange-multiplier space of λ. In discrete space, $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$ is used in place of the gradient function in continuous space. We call one *iteration* as one pass through the while loop. In the following, we describe the features of our implementation of $\mathcal{A}$ in Figure 6.2.

(a) *Descent and Ascent Strategies.* There are two ways to calculate $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$: greedy and hill-climbing, each involving a search in the range of Hamming distance one from the current $\mathbf{x}$ (assignments with one variable flipped from the current assignment $\mathbf{x}$).

$\boxed{\textbf{Generic algorithm } \mathcal{A}}$

```
Set initial $\mathbf{x}$ and $\lambda$
while $\mathbf{x}$ is not a solution, i.e., $N(\mathbf{x}) > 0$
    update $\mathbf{x}$: $\mathbf{x} \longleftarrow \mathbf{x} \oplus \Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$
    if condition for updating $\lambda$ is satisfied then
        update $\lambda$: $\lambda \longleftarrow \lambda + c \times U(\mathbf{x})$
    end if
end while
```

Figure 6.2: Generic discrete Lagrangian algorithm $\mathcal{A}$ for solving SAT problems.

In a *greedy strategy*, the assignment leading to the maximum decrease in the Lagrangian-function value is selected to update the current assignment. Therefore, all assignments in the vicinity need to be searched every time, leading to computation complexity of $O(n)$, where n is number of variables in the SAT problem. In *hill-climbing*, the first assignment leading to a decrease in the Lagrangian-function value is selected to update the current assignment. Depending on the order of search and the number of assignments that can be improved, hill-climbing strategies are generally less computationally expensive than greedy strategies.

We have compared both strategies in solving SAT benchmark problems, and have found hill-climbing to be orders of magnitude faster with solutions of comparable quality. Hence, we have used hill-climbing in our experiments.

(b) *Updating* λ. The frequency in which λ is updated affects the performance of a search. The considerations here are different from those of continuous problems. In a discrete problem, descents based on discrete gradients usually make small changes in $L(\mathbf{x}, \lambda)$ in each update of $\mathbf{x}$ because only one variable changes. Hence, λ should not be updated in each iteration of the search to avoid biasing the search in the Lagrange-multiplier space of λ over the original variable space of $\mathbf{x}$.

In our implementation, λ is updated under two situations. One is when $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$; the other is every T iterations. In our experiments, we have found that DLM works better when T is large. Consequently, λ will be updated infrequently and most likely be updated only when $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$. When $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0$, a local minimum in the original variable space is reached, and the search can only escape from it by updating λ.

By setting T to infinity, the strategy amounts to pure descents in the original $\mathbf{x}$ variable space, while holding λ constant, until a local minimum is reached. This corresponds to Morris' "break out" strategy [173]. In the Breakout algorithm, a *state* is a complete set of assignments for variables. In the case of a SAT problem, each clause is associated a weight, and the cost of a state is the sum of weights of unsatisfied clauses of the state. Initially all weights are one. Iterative improvements of the state continues until a local minimum is reached. Then the weight of each currently unsatisfied clause is increased by unit increments until breakout from the local minimum occurs. Iterative improvements resume afterwards.

A parameter c in the term $c \times U(\mathbf{x})$ in Figure 6.2 controls the magnitude of changes in λ. In general, c can be a vector of real numbers, allowing non-uniform updates of λ across different dimensions and possibly across time. For simplicity, we have used a constant c in our implementation for all λ's. Empirically, $c = 1$ has been found to work well for most of the benchmark problems tested. However, for some larger and more difficult problems, we have used a smaller c in order to reduce the search time.

The last point on λ in Figure 6.2 is that it is always nondecreasing. This is not true in continuous problems with equality constraints. In applying Lagrangian methods to solve continuous problems, the Lagrange multiplier λ of a constraint $g(\mathbf{x}) = 0$ increases when $g(\mathbf{x}) > 0$ and decreases when $g(\mathbf{x}) < 0$. In $\mathcal{A}$ shown in Figure 6.2, λ is nondecreasing because $U(\mathbf{x})$ is either 0 or 1: when a clause is not satisfied, its corresponding λ is increased; and when a clause is satisfied, its corresponding λ is not changed. For most of the benchmark problems we have tested, this strategy does not worsen the search time as these problems are relatively easy to solve. However, for difficult problems that require millions of iterations, λ can become very large as time goes on. Large λ's are generally undesirable because they cause large swings in the Lagrangian-function value.

To overcome this problem, we develop a strategy to reduce λ periodically in DLM $\mathcal{A}_3$ in Section 6.4.2. Using this strategy, we can solve some of the more difficult benchmark problems.

(c) *Starting Points and Restarts.* In contrast to other SAT algorithms that rely on random restarts to bring a search out of a local minimum, DLM will continue to evolve without restarts until a satisfiable assignment is found. This avoids restarting to a new starting point when a search is already in the proximity of a good local minimum. Another major advantage of DLM is that there are very few parameters to be selected or tuned by users, including the initial starting point. This makes it possible for DLM to always start from the origin or from a random starting point generated by a fixed random seed, and find a feasible assignment if one exists.

(d) *Plateaus in the Search Space.* In discrete problems, plateaus with equal values exist in the Lagrangian-function space. Our proposed discrete gradient operator may have difficulties in plateaus because it only examines adjacent points of $L(\mathbf{x}, \lambda)$ that differ in one dimension. Hence, it may not be able to distinguish a plateau from a local minimum. We have implemented two strategies to allow a plateau to be searched.

First, we need to determine when to change λ when the search reaches a plateau. As indicated earlier, λ should be updated when the search reaches a local minimum. However, updating λ when the search is in a plateau changes the surface of the plateau and may make it more difficult for the search to find a local minimum somewhere inside the plateau. To avoid updating λ immediately when the search reaches a plateau, we have developed a strategy called *flat move*. This allows the search to continue for some time in the plateau without changing λ, so that the search can traverse states with the same Lagrangian-function value. How long flat moves should be allowed is heuristic and possibly problem dependent. Note that this strategy is similar to Selman's "sideway-move" strategy [223].

Our second search strategy is to avoid revisiting the same set of states in a plateau. In general, it is impractical to remember every state the search visits in a plateau due to the large storage and computational overheads. In our implementation, we have kept a tabu list to maintain the set of variables flipped in the recent past [92, 112] and to avoid flipping a variable if it is in the tabu list.

To summarize, we employ the following strategies and settings of parameters in our implementations of the generic DLM $\mathcal{A}$:

- A hill-climbing strategy is used for descents in the original variable space.
- T, the time interval between updating λ, is infinity.
- The initial λ is 0.
- The initial starting point is either the origin or a randomly-generated starting point obtained by calling random number generator *drand48()*.
- The random number generator uses a fixed initial seed of 101.
- To handle more difficult and complex problems, flat moves and tabu lists may be used in a search to traverse plateaus, and λ may be reduced systematically.

Without any specific statement, these strategies and parameters settings are used in our implementations in the following section.

6.4.2 Three Implementations of the Generic DLM

Figures 6.3, 6.4, and 6.5 show three implementations of the general algorithm $\mathcal{A}$ with increasing complexity.

$\mathcal{A}_1$, DLM Version 1 shown in Figure 6.3, is the simplest. It has two alternatives to find a variable in order to improve the Lagrangian-function value: flip variables one by one in a predefined order, or flip variables in unsatisfied clauses. Since only variables appearing in unsatisfied clauses can potentially improve the current Lagrangian-function value, it is not necessary to check variables that appear only in currently satisfied clauses. The first alternative is fast when the search starts. By starting from a randomly generated initial assignment, it usually takes just a few flips to find a variable that improves the current Lagrangian-function value. As the search progresses, there are fewer variables that can improve the Lagrangian-function value. At this point, the second alternative should be applied.

$\mathcal{A}_1$ uses parameter ϑ to control switching from the first alternative to the second. We found that $\vartheta = 10$ works well and used this value in all our experiments.

$\boxed{\textbf{DLM } \mathcal{A}_1}$

```
Set initial $\mathbf{x}$
Set $\lambda = 0$
Set $c = 1$
Set $\vartheta = 10$
while $\mathbf{x}$ is not a solution, i.e., $N(\mathbf{x}) > 0$
    if number of unsatisfied clauses $\leq \vartheta$, then
        Maintain a list of unsatisfied clauses
        if $\exists$ variable $v$ in one of the unsatisfied clauses such that
          $L(\mathbf{x}', \lambda) < L(\mathbf{x}, \lambda)$ when flipping $v$ in $\mathbf{x}$ to get $\mathbf{x}'$ then
          $\mathbf{x} \longleftarrow \mathbf{x}'$
        else
          Update $\lambda$: $\lambda \longleftarrow \lambda + c \cdot U(\mathbf{x})$
        end_if
    else
        if $\exists$ variable $v$ such that $L(\mathbf{x}', \lambda) < L(\mathbf{x}, \lambda)$ when flipping $v$
          in a predefined order in $\mathbf{x}$ to get $\mathbf{x}'$ then
          $\mathbf{x} \longleftarrow \mathbf{x}'$
        else
          Update $\lambda$: $\lambda \longleftarrow \lambda + c \cdot U(\mathbf{x})$
        end_if
    end_if
end_while
```

Figure 6.3: Discrete Lagrangian Algorithm Version 1, $\mathcal{A}_1$, an implementation of $\mathcal{A}$ for solving SAT problems.

$\boxed{\textbf{DLM } \mathcal{A}_2}$

```
Set initial $\mathbf{x}$
Set $\lambda = 0$
Set $c = 1$
Set $\kappa = n/3$, where $n$ is the number of variables
**while** $\mathbf{x}$ is not a solution, *i.e.*, $N(\mathbf{x}) > 0$
    **if** number of iterations $\geq \kappa$ **then**
        Maintain a list, $l$, of variables such that
          if one of them is flipped, the solution will improve.
        **if** $l$ is not empty **then**
          Update $\mathbf{x}$ by flipping the first element of $l$
        **else**
          Update $\lambda$: $\lambda \longleftarrow \lambda + c \cdot U(\mathbf{x})$
        **end_if**
    **else**
        **if** $\exists$ variable $v$ such that $L(\mathbf{x}', \lambda) < L(\mathbf{x}, \lambda)$ when flipping $v$
          in a predefined order in $\mathbf{x}$ to get $\mathbf{x}'$ **then**
          $\mathbf{x} \longleftarrow \mathbf{x}'$
        **else**
          Update $\lambda$: $\lambda \longleftarrow \lambda + c \cdot U(\mathbf{x})$
        **end_if**
    **end_if**
**end_while**
```

Figure 6.4: Discrete Lagrangian Algorithm Version 2, $\mathcal{A}_2$, for solving SAT problems.

$\mathcal{A}_2$, DLM Version 2 shown in Figure 6.4, employs another strategy to make local improvements. Initially, it is similar to $\mathcal{A}_1$. As the search progresses, the number of variables that can improve the current Lagrangian-function value are greatly reduced. At this point, a list is created and maintained to contain variables that can improve the current Lagrangian-function value. Consequently, local descent is as simple as flipping the first variable in the list.

$\mathcal{A}_2$ uses κ to control the switching from the first strategy to the second. We found that $\kappa = n/3$ works well and used this value in our experiments. $\mathcal{A}_2$ also has more efficient data structures to deal with larger problems.

$\mathcal{A}_3$, DLM Version 3 shown in Figure 6.5, has more complex control mechanisms and was introduced to solve some of the more difficult benchmark problems (such as the "*g*," "*f*" and large "*par*" problems in the DIMACS benchmarks) better than $\mathcal{A}_2$. It is based on $\mathcal{A}_2$ and uses all of A_2's parameters. We have applied strategies based on flat moves and tabu lists to handle plateaus [92, 112]. An important element of $\mathcal{A}_3$ is the periodic scaling down of the Lagrange multipliers in order to prevent them from growing to be very large. Further, to get better performance, we may have to tune c for each problem instance.

Program efficiency is critical when dealing with SAT problems with a large number of variables and clauses. Since DLM searches by constantly updating state information (current assignment of $\mathbf{x}$, Lagrange-multiplier values, and Lagrangian-function value), state updates have to be very efficient. In our implementation, we update state information incrementally in a way similar to that in GSAT. In large SAT problems, each variable usually appears in a small number of clauses. Therefore, state changes incurred by flipping a variable are very limited. When flipping a variable, some clauses become unsatisfied while some others become satisfied. The incremental update of the Lagrangian-function value is done by subtracting the part of improvement and adding the part of degradation. This leads to very efficient evaluation of $L(\mathbf{x}, \lambda)$. In a similar way, the computation of $\Delta_{\mathbf{x}} L(\mathbf{x}, \lambda)$ can also be done efficiently.

In general Lagrangian methods, Lagrange multipliers introduced in the formulation add extra overhead in computing the Lagrangian function as compared to the original objective

$\boxed{\textbf{DLM } \mathcal{A}_3}$

```
Set initial $\mathbf{x}$
Set $\lambda = 0$
Set $\kappa = n/3$, where $n$ is the number of variables
Set tabu length $L_t$, e.g., 50
Set flat-region limit $L_f$, e.g., 50
Set $\lambda$ reset interval $I_\lambda$, e.g., 10000
Set constant $c$, e.g., 1/2
Set constant $r$, e.g., 1.5
while $\mathbf{x}$ is not a solution, i.e., $N(\mathbf{x}) > 0$
   if number of iterations $\geq \kappa$ then
      Maintain a list, $l$, of variables such that
        if one of them is flipped, the solution will improve.
   end_if
   if number of iterations $\geq \kappa$ and $l$ is not empty then
      Update $\mathbf{x}$ by flipping the first element of $l$
   else if $\exists$ variable $v$ such that $L(\mathbf{x}', \lambda) < L(\mathbf{x}, \lambda)$ when flipping $v$
        in a predefined order in $\mathbf{x}$ to get $\mathbf{x}'$ then
        $\mathbf{x} \longleftarrow \mathbf{x}'$
   else if $\exists$ $v$ such that $L(\mathbf{x}', \lambda) = L(\mathbf{x}, \lambda)$ when flipping $v$ in $\mathbf{x}$ to get $\mathbf{x}'$
        and number of consecutive flat moves $\leq L_f$
        and $v$ has not been flipped in the last $L_t$ iterations then
      $\mathbf{x} \longleftarrow \mathbf{x}'$                /* flat move */
   else
      Update $\lambda$: $\lambda \longleftarrow \lambda + c \cdot U(\mathbf{x})$,
   end_if
   if iteration index mod $I_\lambda = 0$ then
      Reduce $\lambda$ for all clauses, e.g. $\lambda \longleftarrow \lambda / r$
   end_if
end_while
```

Figure 6.5: Discrete Lagrangian Algorithm Version 3, $\mathcal{A}_3$, for solving SAT problems.

function. This overhead in DLM is not significant because an update of λ requires $O(p)$ time, where p is the number of unsatisfied clauses, and $p \ll m$ when the number of clauses m is large.

6.4.3 Reasons behind the Development of $\mathcal{A}_3$

In our experiments, we found that $\mathcal{A}_2$ has difficulty in solving some of the hard benchmark problems. By looking into the execution profiles of $\mathcal{A}_2$ as illustrated in Figure 6.6, we have the following interesting observations.

- The sampled Lagrangian-function values decrease rapidly in the first few thousand iterations (the upper left graph of Figure 6.6), but keep increasing afterwards to become very large at the end. The iteration-by-iteration plot of the Lagrangian part (bottom right graph of Figure 6.6) shows the same increasing trend.
- Some Lagrange multipliers become very large as indicated in the spread of Lagrange multipliers (the upper right graph of Figure 6.6). This large spread leads to large Lagrangian-function values.
- The number of unsatisfied clauses is relatively stable, fluctuating at around 20 (the upper left graph).

As the Lagrangian-function space is very rugged and difficult to search when there is a large spread in Lagrange-multiplier values, we have developed three strategies in $\mathcal{A}_3$ to address this problem.

First, we introduce in $\mathcal{A}_3$ flat moves to handle flat regions in the Lagrangian-function space. In $\mathcal{A}_2$, Lagrange multipliers are increased whenever the search hits a flat region. Since flat regions in hard problems can be very large, Lagrange multipliers can increase very fast, making the search more difficult. Flat moves allow the search to explore flat regions without increasing the Lagrange multipliers and the Lagrangian-function value.

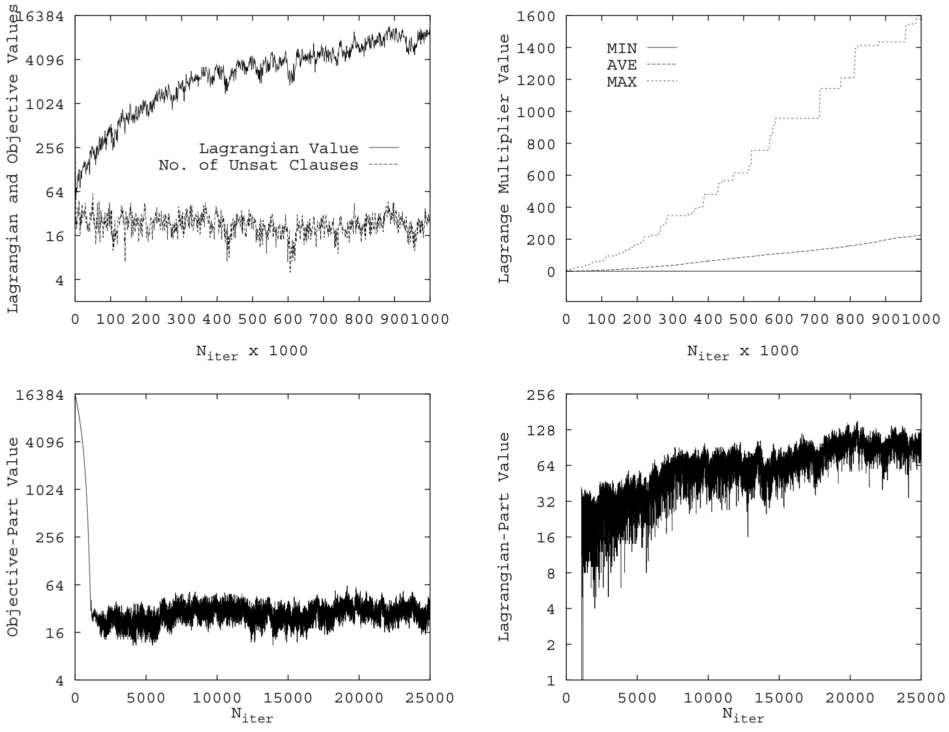


Figure 6.6: Execution profiles of $\mathcal{A}_2$ on problem "*g125-17*" starting from a random initial point. The top left graph plots the Lagrangian-function values and the number of unsatisfied clauses against the number of iterations sampled every 1000 iterations. The top right graph plots the minimum, average and maximum values of the Lagrange multipliers sampled every 1000 iterations. The bottom two graphs plot the iteration-by-iteration objective and Lagrangian-part values for the first 25,000 iterations.

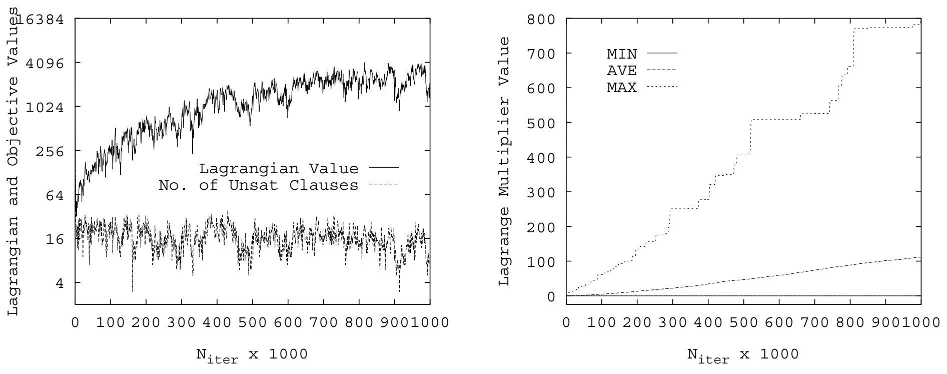


Figure 6.7: Execution profiles of $\mathcal{A}_3$ using a tabu list of size 30 and flat moves of limit of 50 on problem "*g125-17*" starting from a random initial point. See Figure 6.6 for further explanation.

Second, we introduce a tabu list [92, 112] to store variables that have been flipped in the recent past and to avoid flipping any of them in the near future. This strategy avoids flipping the same set of variables back and forth and revisiting the same state.

The result of applying these two strategies is shown in Figure 6.7. These graphs show significant reduction in the growth of Lagrangian-function values and Lagrange multipliers as compared to those of $\mathcal{A}_2$. However, these values still grow without bound.

Third, we introduce periodic scale-downs of Lagrange multipliers to control their growth as well as the growth of Lagrangian-function values. Figure 6.8 shows the result when all Lagrange multipliers are scaled down by a factor of 1.5 every 10,000 iterations. These graphs indicate that periodic scaling, when combined with a tabu list and flat moves, leads to bounded values of Lagrange multipliers and Lagrangian-function values. The bottom right graph in Figure 6.8 shows the reduction in the Lagrangian part after two reductions of its λ values at 10,000 and 20,000 iterations.

By using these three strategies, $\mathcal{A}_3$ can solve successfully most of the hard problems in the DIMACS benchmark suite.

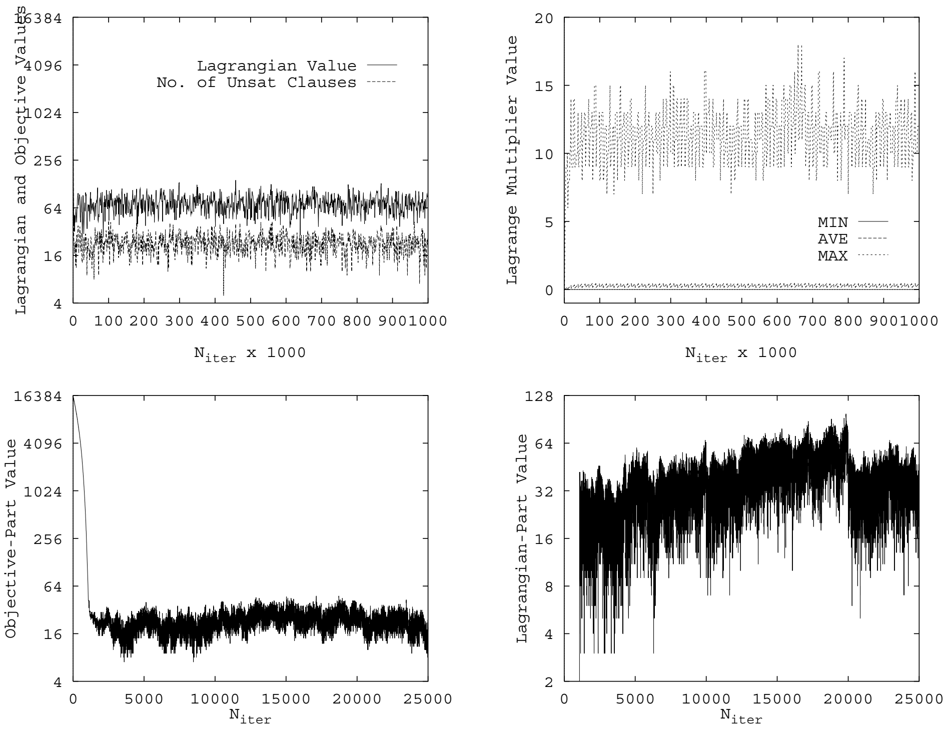


Figure 6.8: Execution profiles of $\mathcal{A}_3$ with tabu list of size 30, flat moves of limit 50, and periodic scaling of λ by a factor of 1.5 every 10,000 iterations on problem "*g125-17*" starting from a random initial point. Note that the average Lagrange multiplier values are very close to 1 and, therefore, overlap with the curve showing the minimum Lagrange multiplier values. See Figure 6.6 for further explanation.

6.5 Experimental Results

In this section, we evaluate DLM using SAT benchmarks in the DIMACS archive. The archive is made up of a repository of hundreds of easy and hard SAT problems with many variables and clauses.

Our DLM code was written in C. In our experiments, we have tested DLM on all satisfiable problems in the DIMACS archive. We have compared the performance of DLM with reported results in the literature on the following benchmark problems:

- Circuit synthesis problems (*ii*) by Kamath *et al.* [139] — a set of SAT encodings of Boolean circuit-synthesis problems;
- Circuit diagnosis problems (*ssa*) — a set of SAT formulas based on circuit fault analysis;
- Parity learning problems (*par*) — a collection of propositional versions of parity learning problems;
- Artificially generated 3-SAT instances (*aim*);
- Randomly generated SAT instances (*jnh*);
- Large random satisfiable 3-SAT instances (f);
- Hard graph coloring problems (*g*);
- An encoding of the Towers-of-Hanoi problems (*hanoi*); and
- Gu's asynchronous-circuit synthesis benchmarks (*as*) and technology mapping benchmarks (*tm*).

In this section, we show experimental results on the three versions of our DLM implementation described in Section 6.4.2.

$\mathcal{A}_1$ sets all parameters constant throughout the runs in order to avoid introducing random effects in the program and to allow easy reproduction of results. It works well on the "*aim*" problems, but not as well on others.

Table 6.1: Execution times of $\mathcal{A}_2$ in CPU seconds on a Sun SparcStation 10/51 for one run of $\mathcal{A}_2$ starting from $x = 0$ (origin) as the initial point on some DIMACS benchmark problems.

Problem	DLM $\mathcal{A}_2$		Problem	DLM $\mathcal{A}_2$	
Identifier	Time	# of Iter.	Identifier	Time	# of Iter.
ssa7552-038	0.200	4126	ii16a1	2.100	756
ssa7552-158	0.167	3279	ii16b1	2.100	1313
ssa7552-159	0.167	3802	ii16c1	1.217	916
ssa7552-160	0.167	3409	ii16d1	1.067	595
aim-100-2_0-yes1-1	0.033	982	ii16e1	1.317	1224
aim-100-2_0-yes1-2	0.067	1680	ii32b3	0.883	529
aim-100-2_0-yes1-3	0.000	513	ii32c3	0.533	735
aim-100-2_0-yes1-4	0.367	8510	ii32d3	3.817	1589
par8-2-c	0.317	7841	ii32e3	0.650	518
par8-4-c	0.233	5784			

$\mathcal{A}_2$ has no problem-dependent parameters to be tuned by users and generally works well for all the benchmark problems. However, it has difficulty in solving some larger and more difficult problems, including "*g*," "*f*," large "*par*," and "*hanoi*."

$\mathcal{A}_3$ has some parameters to be tuned in its complex control mechanisms. Although these parameters are problem dependent, we have tried a few sets of parameters and have found one set that generally works well. Results reported are based on the best set of parameters found. $\mathcal{A}_3$ solves some of the more difficult problems better than $\mathcal{A}_2$.

Table 6.1 shows the experimental results when $\mathcal{A}_2$ was always started from the origin. It shows execution times in CPU seconds and the number of iterations. In each iteration, either one variable was flipped or the λ values were updated. In each experiment, $\mathcal{A}_2$ succeeded in finding a feasible assignment. For most of the test problems, we have found the average time that $\mathcal{A}_2$ spent when started from the origin to be longer than the average time when started from randomly generated initial points. A possible explanation is that the distance between the origin and a local minimum is longer than the average distance between a randomly generated starting point and a nearby local minimum.

Table 6.2: Comparison of $\mathcal{A}_2$'s execution times in seconds averaged over 10 runs with respect to published results of WSAT, GSAT, and Davis-Putnam's algorithm [224] on some of the circuit diagnosis problems in the DIMACS archive. ($\mathcal{A}_2$: Sun SparcStation 10/51 and a 150-MHz SGI Challenge with MIPS R4400; GSAT, WSAT and DP: SGI Challenge with a 70 MHz MIPS R4400.)

Problem	No. of	No. of	DLM $\mathcal{A}_2$			WSAT	GSAT	DP
Identifier	Var.	Clauses	SUN	SGI	# Iter.	SGI	SGI	SGI
ssa7552-038	1501	3575	0.228	0.235	7970	2.3	129	7
ssa7552-158	1363	3034	0.088	0.102	2169	2	90	*
ssa7552-159	1363	3032	0.085	0.118	2154	0.8	14	*
ssa7552-160	1391	3126	0.097	0.113	3116	1.5	18	*

We have compared the performance of $\mathcal{A}_2$ with respect to the best known results on these benchmark problems. Most of our timing results were averaged over ten runs with randomly generated initial points, starting from a fixed seed of 101 in the random number generator *drand48()*. Consequently, our results can be reproduced deterministically.

In Table 6.2, we compare $\mathcal{A}_2$ with WSAT, GSAT, and Davis-Putnam's algorithm in solving the circuit diagnosis benchmark problems. We present the average execution times and the average number of iterations of $\mathcal{A}_2$ as well as the published average execution times of WSAT, GSAT and Davis-Putnam's method [224]. We did not attempt to reproduce the reported results of GSAT and WSAT, since the results may depend on initial conditions, such as the seeds of the random number generator and other program parameters. We ran $\mathcal{A}_2$ on an SGI Challenge[1] so that our timing results can be compared to those of GSAT and WSAT. Our results show that $\mathcal{A}_2$ is approximately one order of magnitude faster than WSAT.

[1]Based on a single-CPU 150-MHz SGI Challenge with MIPS R4400 at the University of Illinois National Center for Supercomputing Applications, we estimate empirically that it is 15.4% slower than a Sun SparcStation 10/51 for executing $\mathcal{A}_2$ to solve SAT benchmark problems. However, we did not evaluate the speed difference between a 150-MHz SGI Challenge and a 70-MHz SGI Challenge on which GSAT and WSAT were run.

Table 6.3: Comparison of $\mathcal{A}_2$'s execution times in seconds averaged over 10 runs with the published results on the circuit synthesis problems, including the best known results obtained by GSAT, integer programming (IP), and simulated annealing [224]. ($\mathcal{A}_2$: Sun SparcStation 10/51 and a 150-MHz SGI Challenge with MIPS R4400; GSAT and SA: SGI Challenge with a 70 MHz MIPS R4400; Integer Programming: VAX 8700.)

Problem	No. of	No. of	DLM $\mathcal{A}_2$			GSAT	IP	SA
Identifier	Var.	Clauses	SUN	SGI	# Iter.	SGI	Vax	SGI
ii16a1	1650	19368	0.122	0.128	819	2	2039	12
ii16b1	1728	24792	0.265	0.310	1546	12	78	11
ii16c1	1580	16467	0.163	0.173	797	1	758	5
ii16d1	1230	15901	0.188	0.233	908	3	1547	4
ii16e1	1245	14766	0.297	0.302	861	1	2156	3

In Table 6.3, we compare $\mathcal{A}_2$ with the published results of GSAT, integer programming and simulated annealing on the circuit synthesis problems [224]. Our results show that $\mathcal{A}_2$ performs several times faster than GSAT.

In Table 6.4, we compare the performance of the three versions of DLM with some of the best known results of GSAT on the circuit-synthesis, parity-learning, some artificially generated 3-SAT, and some of the hard graph coloring problems. The results on GSAT are from [221], which are better than other published results. Our results show that DLM is consistently faster than GSAT on the "*ii*" and "*par*" problems, and that $\mathcal{A}_1$ is an order-of-magnitude faster than GSAT on some "*aim*" problems.

Table 6.4 also shows the results of $\mathcal{A}_3$ on some "*g*" problems. Recall that $\mathcal{A}_3$ was developed to cope with large flat plateaus in the search space that confuse $\mathcal{A}_2$, which failed to find any solution within 5 million iterations. Hansen [112] and later Selman [223] addressed this problem by using the tabu search strategy. In a similar way, we have adopted this strategy in $\mathcal{A}_3$ by keeping a tabu list to prevent flipping the same variable back and forth. This led to better performance, although the performance is sensitive to the length of the tabu list. $\mathcal{A}_3$ performs comparably to GSAT on these "*g*" problems.

Table 6.4: Comparison of DLM's execution times in seconds averaged over 10 runs with the best known results obtained by GSAT [221] on the DIMACS circuit-synthesis, parity-learning, artificially generated 3-SAT instances, and graph coloring problems. Results on $\mathcal{A}_3$ were based on a tabu length of 50, flat region limit of 50, λ reset interval of 10,000, and λ reset to be $\lambda/1.5$ when the λ reset interval is reached. For "*g125-18*" and "*g250-15*," $c = 1/2$; For "*g125-17*" and "*g250-29*," $c = 1/16$. ($\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$: Sun SparcStation 10/51; GSAT: SGI Challenge (model unknown))

Problem Identifier	No. of Var.	No. of Clauses	SUN Sec.	Success Ratio	SGI Sec.	Success Ratio
			DLM $\mathcal{A}_1$		GSAT	
aim-100-2_0-yes1-1	100	200	0.19	10/10	1.96	9/10
aim-100-2_0-yes1-2	100	200	0.65	10/10	1.6	10/10
aim-100-2_0-yes1-3	100	200	0.19	10/10	1.09	10/10
aim-100-2_0-yes1-4	100	200	0.10	10/10	1.54	10/10
			DLM $\mathcal{A}_2$		GSAT	
ii32b3	348	5734	0.31	10/10	0.6	10/10
ii32c3	279	3272	0.12	10/10	0.27	10/10
ii32d3	824	19478	1.05	10/10	2.24	10/10
ii32e3	330	5020	0.16	10/10	0.49	10/10
par8-2-c	68	270	0.06	10/10	1.33	10/10
par8-4-c	67	266	0.09	10/10	0.2	10/10
			DLM $\mathcal{A}_3$		GSAT	
g125.17	2125	66272	1390.32	10/10	264.07	7/10
g125.18	2250	70163	3.197	10/10	1.9	10/10
g250.15	3750	233965	2.798	10/10	4.41	10/10
g250.29	7250	454622	1219.56	9/10	1219.88	9/10

In Tables 6.5 to 6.9, we compare the performance of DLM with the best results of Grasp on the DIMACS benchmark problems [204]. Grasp is a greedy randomized adaptive search procedure that can find good quality solutions for a wide variety of combinatorial optimization problems [76, 77, 145, 205].

In [204], four implementations of Grasp were applied to solve five classes of DIMACS SAT problems, "aim," "ii," "jnh," "ssa7552," and "par." Comparing to GSAT, Grasp did better on the "aim," "ssa7552," and "par" problems, and worse on the "ii" and "jnh" problems. The results of the most efficient implementation of Grasp, Grasp-A, are used in our comparison. Grasp was run on an SGI Challenge computer with 150 MHz MIPS R4400. The average CPU time of 10 random runs of Grasp are shown in the tables, in which Grasp succeeded in all 10 runs for each problem.

In Table 6.5 and 6.6, we compare $\mathcal{A}_1$, $\mathcal{A}_2$, and Grasp in solving the whole set of "*aim*" problems, which were randomly generated instances with a single satisfiable solution. For these problems, $\mathcal{A}_1$ performs better than $\mathcal{A}_2$ on the average, and is usually 1 to 3 orders of magnitude faster than Grasp. Grasp does not have results on some of the larger "aim-200" instances.

In Tables 6.7 and 6.8, we compare $\mathcal{A}_2$ with Grasp in solving the "ii," "jnh," "par", and "ssa7552" problems. Except for some small instances that both DLM and Grasp solve in no time, DLM is generally 1 to 3 orders of magnitude faster than Grasp in solving the "ii," "jnh" and "ssa7552" problems. For the "par" problems, DLM $\mathcal{A}_2$ obtains comparable results to Grasp in solving the "par8-x-c" instances, but is worse in solving the "par8-x" instances.

In addition to the "par8-x" instances, DLM $\mathcal{A}_1$ and $\mathcal{A}_2$ have difficulty in solving the " par16," " par32," " hanoi" and " f" problems. Tables 6.9 shows some preliminary but promising results of $\mathcal{A}_3$ on some of the more difficult but satisfiable DIMACS benchmark problems. Comparing with Grasp, $\mathcal{A}_3$ is better in solving the "par16-x-c" instances, but is worse in solving the "par8-x" and "par16-x" problems.

Finally, we show in Table 6.10 our results of $\mathcal{A}_2$ on the "as" and "tm" problems in the DIMACS archive. The average time over 10 runs is always under 1 second for these problems.

Table 6.5: Comparison of DLM's execution times in seconds over 10 runs with the published results of Grasp on the "aim" problems from the DIMACS archive [204]. (DLM $\mathcal{A}_1$, $\mathcal{A}_2$: Sun SparcStation 10/51; Grasp: SGI Challenge with a 150 MHz MIPS R4400; Success ratio for Grasp is always 10/10.)

		DLM $\mathcal{A}_2$				DLM $\mathcal{A}_1$			Grasp
Problem	Succ.	Sun CPU Seconds			Succ.	Sun CPU Seconds			Avg. SGI
Identifier	Ratio	Avg.	Min.	Max.	Ratio	Avg.	Min.	Max.	Seconds
aim-50-1_6-yes1-1	10/10	0.032	0.000	0.150	10/10	0.042	0.017	0.100	1.12
aim-50-1_6-yes1-2	6/10	0.008	0.000	0.017	10/10	0.020	0.000	0.050	0.12
aim-50-1_6-yes1-3	10/10	0.010	0.000	0.017	10/10	0.027	0.000	0.050	0.49
aim-50-1_6-yes1-4	5/10	0.210	0.017	0.917	10/10	0.030	0.000	0.067	0.46
aim-50-2_0-yes1-1	10/10	0.012	0.000	0.033	10/10	0.035	0.017	0.083	3.67
aim-50-2_0-yes1-2	8/10	0.010	0.000	0.017	10/10	0.035	0.017	0.100	2.82
aim-50-2_0-yes1-3	10/10	0.007	0.000	0.017	10/10	0.077	0.000	0.533	1.76
aim-50-2_0-yes1-4	10/10	0.048	0.000	0.250	10/10	12.60	0.000	125.7	0.62
aim-50-3_4-yes1-1	10/10	0.025	0.000	0.067	10/10	1.528	0.017	13.40	0.39
aim-50-3_4-yes1-2	10/10	0.015	0.000	0.033	10/10	0.162	0.017	0.350	0.58
aim-50-3_4-yes1-3	10/10	0.012	0.000	0.033	10/10	0.142	0.033	0.317	0.49
aim-50-3_4-yes1-4	9/10	0.013	0.000	0.033	10/10	0.057	0.033	0.150	0.91
aim-50-6_0-yes1-1	10/10	0.010	0.000	0.033	10/10	0.027	0.000	0.067	0.06
aim-50-6_0-yes1-2	10/10	0.007	0.000	0.017	10/10	0.028	0.017	0.050	0.14
aim-50-6_0-yes1-3	10/10	0.007	0.000	0.017	10/10	0.035	0.000	0.100	0.05
aim-50-6_0-yes1-4	10/10	0.007	0.000	0.017	10/10	0.027	0.000	0.067	0.05
aim-100-1_6-yes1-1	10/10	0.068	0.033	0.117	10/10	0.092	0.033	0.200	320.40
aim-100-1_6-yes1-2	10/10	0.053	0.017	0.100	10/10	0.098	0.050	0.167	122.27
aim-100-1_6-yes1-3	10/10	0.095	0.050	0.333	10/10	0.142	0.033	0.333	157.15
aim-100-1_6-yes1-4	10/10	0.052	0.000	0.117	10/10	0.095	0.017	0.317	50.10
aim-100-2_0-yes1-1	9/10	0.854	0.017	2.700	10/10	0.193	0.050	0.383	391.79
aim-100-2_0-yes1-2	10/10	0.287	0.050	0.900	10/10	0.652	0.117	1.650	208.40
aim-100-2_0-yes1-3	10/10	0.100	0.017	0.333	10/10	0.187	0.067	0.400	118.22
aim-100-2_0-yes1-4	10/10	0.357	0.033	1.917	10/10	0.097	0.050	0.167	1352.38
aim-100-3_4-yes1-1	10/10	0.450	0.000	2.950	10/10	0.795	0.133	4.417	10.27
aim-100-3_4-yes1-2	10/10	0.195	0.017	1.383	10/10	0.362	0.133	0.817	34.71
aim-100-3_4-yes1-3	10/10	0.050	0.017	0.100	10/10	0.858	0.067	3.800	49.69
aim-100-3_4-yes1-4	10/10	0.038	0.000	0.100	10/10	0.170	0.000	0.317	24.62
aim-100-6_0-yes1-1	10/10	0.020	0.000	0.033	10/10	0.075	0.017	0.133	0.52
aim-100-6_0-yes1-2	10/10	0.018	0.000	0.033	10/10	0.142	0.033	0.467	0.77
aim-100-6_0-yes1-3	10/10	0.017	0.000	0.050	10/10	0.082	0.017	0.233	0.38
aim-100-6_0-yes1-4	10/10	0.018	0.000	0.033	10/10	0.093	0.017	0.267	0.84

Table 6.6: Comparison of DLM's execution times in seconds over 10 runs with the published results of Grasp on the "aim-200" problems from the DIMACS archive [204]. (DLM $\mathcal{A}_1$, $\mathcal{A}_2$: Sun SparcStation 10/51; Grasp: SGI Challenge with a 150 MHz MIPS R4400; Success ratio for Grasp is always 10/10.)

		DLM $\mathcal{A}_2$				DLM $\mathcal{A}_1$			Grasp
Problem	Succ.	Sun CPU Seconds			Succ.	Sun CPU Seconds			Avg. SGI
Identifier	Ratio	Avg.	Min.	Max.	Ratio	Avg.	Min.	Max.	Seconds
aim-200-1_6-yes1-1	10/10	0.958	0.150	2.433	10/10	0.748	0.333	1.583	
aim-200-1_6-yes1-2	6/10	0.786	0.417	1.283	10/10	0.635	0.150	2.350	
aim-200-1_6-yes1-3	10/10	5.498	0.267	43.47	10/10	1.217	0.233	6.950	
aim-200-1_6-yes1-4	2/10	70.04	6.300	133.7	10/10	2.308	0.333	7.183	
aim-200-2_0-yes1-1	1/10	1.283	1.283	1.283	10/10	8.128	0.300	51.48	
aim-200-2_0-yes1-2	4/10	0.538	0.150	1.350	10/10	6.132	0.317	30.00	
aim-200-2_0-yes1-3	7/10	6.355	0.150	28.45	10/10	9.545	0.383	45.72	
aim-200-2_0-yes1-4	0/10				10/10	2.102	0.300	6.950	
aim-200-3_4-yes1-1	8/10	0.469	0.183	1.100	10/10	6.638	0.517	16.50	
aim-200-3_4-yes1-2	10/10	0.547	0.050	3.417	10/10	29.12	0.917	213.5	
aim-200-3_4-yes1-3	10/10	0.838	0.050	5.500	10/10	2.405	0.550	9.467	
aim-200-3_4-yes1-4	9/10	3.122	0.050	22.43	9/10	6.520	0.917	27.15	
aim-200-6_0-yes1-1	10/10	0.075	0.033	0.133	10/10	0.575	0.100	1.717	91.87
aim-200-6_0-yes1-2	9/10	0.209	0.050	0.583	10/10	0.350	0.117	0.933	108.2
aim-200-6_0-yes1-3	10/10	0.102	0.017	0.317	10/10	0.513	0.150	1.250	162.1
aim-200-6_0-yes1-4	10/10	0.218	0.017	0.717	10/10	0.415	0.050	1.000	134.0

Table 6.7: Comparison of DLM $\mathcal{A}_2$'s execution times in seconds over 10 runs with the published results of Grasp on the "ii" problems from the DIMACS archive [204]. (DLM $\mathcal{A}_2$: Sun SparcStation 10/51; Grasp: SGI Challenge with a 150 MHz MIPS R4400; The success ratios for DLM $\mathcal{A}_2$ and Grasp are always 10/10.)

	DLM $\mathcal{A}_2$			Grasp		DLM $\mathcal{A}_2$			Grasp
Problem	Sun CPU Seconds			Avg. SGI	Problem	Sun CPU Seconds			Avg. SGI
Id.	Avg.	Min.	Max.	Seconds	Id.	Avg.	Min.	Max.	Seconds
ii8a1	0.003	0.000	0.017	0.00	ii8a2	0.007	0.000	0.017	0.03
ii8a3	0.013	0.000	0.017	0.10	ii8a4	0.027	0.017	0.033	0.23
ii8b1	0.012	0.000	0.017	0.11	ii8b2	0.028	0.017	0.050	1.67
ii8b3	0.043	0.017	0.067	35.02	ii8b4	0.062	0.050	0.083	369.3
ii8c1	0.013	0.000	0.017	37.26	ii8c2	0.040	0.033	0.050	8.69
ii8d1	0.018	0.017	0.033	1.19	ii8d2	0.043	0.033	0.050	3.23
ii8e1	0.020	0.017	0.033	1.44	ii8e2	0.040	0.033	0.067	21.97
ii16a1	0.122	0.117	0.133	23.46	ii16a2	0.302	0.200	0.433	1970.6
ii16b1	0.265	0.217	0.350	449.9	ii16b2	0.377	0.183	0.717	58.41
ii16c1	0.163	0.133	0.200	20.83	ii16c2	0.667	0.133	1.350	43.30
ii16d1	0.188	0.167	0.217	36.09	ii16d2	0.618	0.250	1.333	56.32
ii16e1	0.297	0.267	0.367	74.62	ii16e2	1.273	0.183	3.350	28.06
ii32a1	0.337	0.133	1.000	68.36	ii32b1	0.028	0.017	0.033	0.98
ii32b2	0.130	0.050	0.517	8.08	ii32b3	0.305	0.150	0.767	8.21
ii32b4	0.460	0.167	1.033	28.21	ii32c1	0.022	0.000	0.033	1.79
ii32c2	0.050	0.033	0.083	0.41	ii32c3	0.118	0.083	0.233	2.01
ii32c4	2.940	0.567	6.217	200.9	ii32d1	0.065	0.017	0.167	2.04
ii32d2	0.202	0.083	0.833	17.92	ii32d3	1.047	0.333	2.750	666.7
ii32e1	0.022	0.017	0.033	17.67	ii32e2	0.097	0.050	0.183	1.89
ii32e3	0.160	0.100	0.450	6.04	ii32e4	0.190	0.150	0.233	11.53
ii32e5	0.402	0.250	1.450	16.47					

Table 6.8: Comparison of DLM $\mathcal{A}_2$'s execution times in seconds over 10 runs with the published results of Grasp on the "jnh," "par," and "ssa" problems from the DIMACS archive [204]. (DLM $\mathcal{A}_2$: Sun SparcStation 10/51; Grasp: SGI Challenge with a 150 MHz MIPS R4400, and success ratio for Grasp is always 10/10.)

Problem Identifier	Success Ratio	DLM $\mathcal{A}_2$ Sun CPU Seconds Avg.	Min.	Max.	Grasp Avg. SGI Seconds
jnh1	10/10	0.068	0.017	0.150	11.87
jnh7	10/10	0.043	0.017	0.083	3.61
jnh12	10/10	0.155	0.067	0.250	0.84
jnh17	10/10	0.082	0.033	0.167	1.66
jnh201	10/10	0.028	0.017	0.050	1.48
jnh204	10/10	0.172	0.017	0.667	14.64
jnh205	10/10	0.103	0.050	0.183	6.17
jnh207	10/10	0.337	0.050	1.817	3.61
jnh209	10/10	0.433	0.067	2.000	7.45
jnh210	10/10	0.033	0.017	0.067	2.35
jnh212	10/10	5.442	0.033	51.250	70.92
jnh213	10/10	0.083	0.017	0.183	9.43
jnh217	10/10	0.060	0.000	0.133	5.76
jnh218	10/10	0.063	0.017	0.183	1.45
jnh220	10/10	0.387	0.033	2.033	10.17
jnh301	10/10	0.333	0.117	0.950	46.23
par8-1	6/10	3.311	0.033	13.117	0.59
par8-2	6/10	3.981	0.400	7.667	1.78
par8-3	4/10	6.012	0.317	18.017	2.69
par8-4	5/10	6.440	1.083	11.017	0.57
par8-5	6/10	11.478	5.850	18.633	0.86
par8-1-c	10/10	0.075	0.000	0.400	0.07
par8-2-c	10/10	0.058	0.000	0.267	0.14
par8-3-c	10/10	1.998	0.000	9.233	0.35
par8-4-c	10/10	0.088	0.017	0.367	0.55
par8-5-c	10/10	0.477	0.017	2.633	0.17
ssa7552-038	10/10	0.228	0.083	0.933	7.05
ssa7552-158	10/10	0.088	0.050	0.167	3.42
ssa7552-159	10/10	0.085	0.067	0.150	1.72
ssa7552-160	10/10	0.097	0.050	0.183	22.13

Table 6.9: Comparison of DLM $\mathcal{A}_3$'s execution times in seconds over 10 runs with the published results of Grasp on some of the more difficult DIMACS benchmark problems from the DIMACS archive [204]. (Success ratio of Grasp is always 10/10.) Program parameters: For all the problems: flat region limit = 50; λ reset to $\lambda/1.5$ every 10,000 iterations. For the par-16-[1-5] problems: tabu length = 100, $\lambda = 1$. For the rest of the par problems: tabu length = 50, $\lambda = \frac{1}{2}$. For the f problems: tabu length = 50, $\lambda = \frac{1}{16}$. For the hanoi4 problem: tabu length = 50, $\lambda = \frac{1}{2}$. System configuration: DLM $\mathcal{A}_3$: Sun SparcStation 10/51; Grasp: SGI Challenge with a 150 MHz MIPS R4400.

		DLM $\mathcal{A}_3$			Grasp
Problem	Succ.	Sun CPU Seconds			Average
Identifier	Ratio	Avg.	Min.	Max.	SGI Sec.
par8-1	10/10	4.780	0.133	14.383	0.59
par8-2	10/10	5.058	0.100	13.067	1.78
par8-3	10/10	9.903	0.350	21.150	2.69
par8-4	10/10	5.842	0.850	16.433	0.57
par8-5	10/10	14.628	1.167	34.900	0.86
par16-1	5/10	11172.8	4630.6	20489.1	9643.38
par16-2	1/10	856.9	856.9	856.9	8993.89
par16-3	1/10	20281.6	20281.6	20281.6	
par16-4	3/10	3523.1	1015.0	7337.9	
par16-5	1/10	13023.4	13023.4	13023.4	
par16-1-c	10/10	398.1	11.7	1011.9	2235.83
par16-2-c	10/10	1324.3	191.0	4232.3	2179.04
par16-3-c	10/10	987.2	139.8	3705.2	2035.23
par16-4-c	10/10	316.7	5.7	692.66	2071.30
par16-5-c	10/10	1584.2	414.5	3313.2	2566.35
hanoi4	1/10	476.5	476.5	476.5	
f600	10/10	16.9	2.1	37.2	
f1000	10/10	126.8	4.4	280.7	
f2000	10/10	1808.6	174.3	8244.7	

Table 6.10: Execution times of DLM $\mathcal{A}_2$ in Sun SparcStation 10/51 CPU seconds over 10 runs on the "as" and "tm" problems from the DIMACS archive.

Problem	Success	Sun CPU Seconds			Problem	Success	Sun CPU Seconds		
Id.	Ratio	Avg.	Min.	Max.	Id.	Ratio	Avg.	Min.	Max.
as2	10/10	0.020	0.017	0.033	as10	10/10	0.103	0.067	0.150
as3	10/10	0.037	0.017	0.050	as11	10/10	0.047	0.017	0.067
as4	10/10	0.157	0.133	0.200	as12	10/10	0.038	0.017	0.067
as5	10/10	0.988	0.850	1.283	as13	10/10	0.085	0.067	0.117
as6	10/10	0.098	0.050	0.150	as14	10/10	0.017	0.017	0.033
as7	10/10	0.520	0.450	0.633	as15	10/10	0.157	0.117	0.200
as8	10/10	0.047	0.017	0.067					
tm1	10/10	0.238	0.217	0.250	tm2	10/10	0.013	0.000	0.033

To summarize, we have presented the application of Novel based on the discrete Lagrangian method (DLM) in solving SAT problems. DLM belongs to the class of incomplete methods that attempts to find a feasible assignment if one exists, but will not terminate if the problem is infeasible.

We formulate a SAT problem as a discrete constrained optimization problem in which local minima in the Lagrangian space correspond to feasible assignments of the SAT problem. Searching for saddle points, therefore, corresponds to solving the SAT problem.

We first extend the theory of Lagrange multipliers for continuous problems to discrete problems. With respect to SAT problems, we define the concept of saddle points, derive the Discrete Saddle Point Theorem, propose methods to compute discrete gradients, and apply DLM to look for saddle points. We show the Discrete Fixed Point theorem which guarantees that DLM will continue to search until a saddle point is found. We further investigate various heuristics in implementing DLM.

We have compared the performance of DLM with respect to the best existing methods for solving some SAT benchmark problems archived in DIMACS. Experimental results show that DLM can solve these benchmark problems often faster than other local-search methods. We have not been able to solve the remaining DIMACS benchmark problems that include

- *par32-1* thru *par32-5*,
- *par32-1-c* thru *par32-5-c*, and
- *hanoi5*.

6.6 Solving Maximum Satisfiability Problems

In this section, we apply DLM to solve a general class of satisfiability problem, the *weighted maximum satisfiability problem* (MAX-SAT). The MAX-SAT problem is a discrete optimization problem. They are difficult to solve due to the large amount of local minima in their search space.

We formulate a MAX-SAT problem as a discrete constrained optimization problem, and present the discrete Lagrangian method (DLM) for solving MAX-SAT problems. DLM provides an efficient way of searching in a discrete space. Instead of restarting from a new starting point when a search reaches a local minimum, the Lagrange multipliers in DLM provide a force to lead the search out of a local minimum and move it in the direction provided by the Lagrange multipliers. We further investigate issues in implementing DLM to solve MAX-SAT problems. Our method dynamically adjusts the magnitude of Lagrange multipliers and clause weights to find better solutions. Since our method has very few algorithmic parameters to be tuned, the search procedure can be made deterministic and the results, reproducible. In our experiments, we compare DLM with GRASP in solving a large set of test problems, and show that it finds better solutions and is substantially faster.

6.6.1 Previous Work

As presented in Section 6.1, the satisfiability (SAT) problem is defined as follows: Given a set of m clauses $\{C_1, C_2, \cdots, C_m\}$ on n variables $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, $x_i \in \{0, 1\}$, and a Boolean expression in conjunctive normal form (CNF),

$$C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

find an assignment to the variables so that the Boolean expression evaluates to be *true*, or derive infeasibility if it is infeasible.

MAX-SAT is a general case of SAT. In MAX-SAT, each clause C_i is associated with weight w_i. The objective is to find an assignment to the variables that maximizes the sum of the weights of satisfied clauses,

$$\max_{\mathbf{x}\in\{0,1\}^n} \sum_{i=1}^{m} w_i\, S_i(\mathbf{x}), \tag{6.16}$$

where $S_i(\mathbf{x})$ equals 1 if logical assignment $\mathbf{x}$ satisfies C_i, and 0 otherwise. This objective is equivalent to minimizing the sum of the weights of unsatisfied clauses. Based on this definition, SAT is a decision version of MAX-SAT in which all clauses have unit weights.

MAX-SAT problems are difficult to solve for the following reasons:

- They have a large number of local minima in their search space, where a local minimum is a state whose local neighborhood does not include any state that is strictly better.
- The weights in the objective function of a MAX-SAT problem can lead to a much more rugged search space than the corresponding SAT problem. When a MAX-SAT problem is satisfiable, existing SAT algorithms [100, 222] developed to find a satisfiable assignment to the corresponding SAT problem can be applied, and the resulting assignment is also optimal for (6.16). However, this approach does not work when the MAX-SAT problem is not satisfiable. In this case, existing SAT local-search methods have difficulties in overcoming the rugged search space.

Methods for solving MAX-SAT can be classified as incomplete and complete, depending on whether they can find the optimal assignment. Complete algorithms can determine optimal assignments. They include mixed integer linear programming methods [205] and various heuristics [93, 112, 136]. They are generally computationally expensive and have difficulties in solving large problems.

On the other hand, incomplete methods are usually faster and can solve some large problems that complete methods cannot handle. Many incomplete local-search methods have been designed to solve large SAT problems of thousands of variables [100, 222, 228, 263] and have obtained promising results in solving MAX-SAT problems [132, 205].

Another approach to solve MAX-SAT is to transform it into continuous formulations. Discrete variables in the original problem are relaxed into continuous variables in such a way that solutions to the continuous problem are binary solutions to the original problem. This transformation is potentially beneficial because an objective in continuous space may smooth out some local minima. Unfortunately, continuous formulations require computationally expensive algorithms, rendering them applicable only to small problems.

In this section we apply DLM to solve MAX-SAT problems. We formulate a MAX-SAT problem in (6.16) as a discrete constrained optimization problem [229]:

$$\begin{aligned} &\min_{\mathbf{x}\in\{0,1\}^n} \quad N(\mathbf{x}) = \sum_{i=1}^{m} w_i \, U_i(\mathbf{x}) \\ &\text{subject to} \quad U_i(\mathbf{x}) = 0 \quad \forall i \in \{1, 2, \ldots, m\}. \end{aligned} \tag{6.17}$$

where $w_i > 0$, and $U_i(\mathbf{x})$ equals to 0 if the logical assignment $\mathbf{x}$ satisfies C_i, and 1 otherwise. We first present DLM for solving MAX-SAT, and investigate the issues and alternatives in implementing DLM. Then, we present experimental results of DLM in solving a large set of MAX-SAT test problems.

6.6.2 DLM for Solving MAX-SAT Problems

The Lagrangian function for (6.17) is defined as follows:

$$L_{maxsat}(\mathbf{x}, \lambda) = N(\mathbf{x}) + \lambda^T U(\mathbf{x}) = \sum_{i=1}^{m} (w_i + \lambda_i) U(\mathbf{x}) \tag{6.18}$$

where $\mathbf{x} \in \{0,1\}^n$, $U(\mathbf{x}) \in \{0,1\}^m$, and λ^T (the transpose of $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$) denotes the Lagrange multipliers.

A saddle point $(\mathbf{x}^*, \lambda^*)$ of $L_{maxsat}(\mathbf{x}, \lambda)$ is defined as one that satisfies the following condition:

$$L_{maxsat}(\mathbf{x}^*, \lambda) \le L_{maxsat}(\mathbf{x}^*, \lambda^*) \le L_{maxsat}(\mathbf{x}, \lambda^*) \tag{6.19}$$

for all λ sufficiently close to λ^* and for all $\mathbf{x}$ whose Hamming distance between $\mathbf{x}^*$ and $\mathbf{x}$ is 1. The following version of DLM finds discrete saddle points of Lagrangian function $L_{maxsat}(\mathbf{x}^*, \lambda)$.

Discrete Lagrangian Method (DLM) $\mathcal{A}_{maxsat}$.

$$\mathbf{x}^{k+1} = \mathbf{x}^k \oplus \Delta_{\mathbf{x}} L_{maxsat}(\mathbf{x}^k, \lambda^k) \tag{6.20}$$

$$\lambda^{k+1} = \lambda^k + c \times U(\mathbf{x}^k) \tag{6.21}$$

where c is a positive real number and $\oplus$ represents exclusive OR (XOR). We define the *discrete gradient operator* $\Delta_{\mathbf{x}} L_{maxsat}(\mathbf{x}, \lambda)$ with respect to $\mathbf{x}$ such that $\Delta_{\mathbf{x}} L_{maxsat}(\mathbf{x}, \lambda) = (\delta_1, \cdots, \delta_n) \in \{0,1\}^n$ with at most one δ_i not zero, and it gives $L_{maxsat}(\mathbf{x}, \lambda)$ the greatest reduction in the neighborhood of $\mathbf{x}$ with Hamming distance 1. Note that $\Delta_{\mathbf{x}} L_{maxsat}(\mathbf{x}, \lambda)$ is similar to the discrete gradient operator that we have defined for SAT problems.

There are two important properties about DLM $\mathcal{A}_{maxsat}$:

- It is easy to see that the necessary condition for algorithm $\mathcal{A}_{maxsat}$ to converge is when $U(\mathbf{x}) = 0$, implying that $\mathbf{x}$ is optimal. If any of the constraints in $U(\mathbf{x})$ is not satisfied, then λ will continue to evolve to handle the unsatisfied constraints, and the search continues. In that case, a stopping criterion based on a time limit will be needed.

- DLM $\mathcal{A}_{maxsat}$ modeled by (6.20) and (6.21) minimizes $L_{maxsat}(\mathbf{x}, \lambda)$ in (6.18) while λ is changing. Since $U_i(\mathbf{x})$ is weighted by $w_i + \lambda_i$ in (6.18), changing λ is equivalent to perturbing the weights in the original objective function defined in (6.17). In finding better assignments while trying to satisfy all the clauses, DLM effectively changes the weights of unsatisfied clauses and modifies $L_{maxsat}(\mathbf{x}, \lambda)$ to be searched. Since the optimal assignment depends on the relative weights in (6.17), the change in weights in DLM is done in such a way that maintains their relative magnitudes. In contrast, existing MAX-SAT methods explore the search space without modifying the weights. This feature is the major difference between existing methods and DLM.

Next, we discuss various considerations in implementing DLM. Figure 6.9 shows the pseudo code of $\mathcal{A}_{maxsat1}$, a generic DLM implementing (6.20) and (6.21), where c is a positive

```
1. Set initial x randomly by a fixed random seed
2. Set initial $\lambda$ to be zero
3. while x is not a feasible solution, i.e. $N(\mathbf{x}) > 0$
4.     update x: $\mathbf{x} \longleftarrow \mathbf{x} \oplus \Delta_{\mathbf{x}} L_{maxsat}(\mathbf{x}, \lambda)$
5.     update incumbent if $N(\mathbf{x})$ is better than current incumbent
6.     if condition for updating $\lambda$ is satisfied then
7.         update $\lambda$: $\lambda \longleftarrow \lambda + c \times U(\mathbf{x})$
8.     end if
9. end while
```

Figure 6.9: Generic DLM $\mathcal{A}_{maxsat1}$ for solving SAT problems.

constant that regulates the magnitude of updates of λ. We define one *iteration* as one pass through the while loop. In the following, we describe some of our design considerations.

(a) *Initial Points* (Lines 1-2). DLM is started from either the origin or from a random initial point generated using a fixed random seed. Further, λ is always set to zero. The fixed initial points allow the results to be reproduced easily.

(b) *Descent and Ascent Strategies* (Line 4). There are two ways to calculate $\Delta_{\mathbf{x}} L_{maxsat}(\mathbf{x}, \lambda)$: greedy and hill-climbing, each involving a search in the range of Hamming distance one from the current $\mathbf{x}$. Depending on the order of search and the number of assignments that can be improved, hill-climbing strategies are generally much faster than greedy strategies.

Among various ways of hill-climbing, we used two alternatives in our implementation: flip the variables one by one in a predefined order, or maintain a list of variables that can improve the current $L_{maxsat}(\mathbf{x}, \lambda)$ and just flip the first variable in the list. The first alternative is fast when the search starts. By starting from a randomly generated initial assignment, it usually takes very few flips to find a variable that improves the current $L_{maxsat}(\mathbf{x}, \lambda)$. As the search progresses, there are fewer variables that can improve $L_{maxsat}(\mathbf{x}, \lambda)$. At this point, the second alternative becomes more efficient and should be applied.

(c) *Conditions for updating* λ (Line 6). The frequency in which λ is updated affects the performance of a search. The considerations here are different from those of continuous problems. In a discrete problem, descents based on discrete gradients usually make small changes in $L_{maxsat}(\mathbf{x}, \lambda)$ in each update of $\mathbf{x}$ because only one variable changes. Hence, λ should not be updated in each iteration of the search to avoid biasing the search in the Lagrange-multiplier space of λ over the original variable space of $\mathbf{x}$.

Experimental results show that a good strategy is to update λ only when $\Delta_{\mathbf{x}} L_{maxsat}(\mathbf{x}, \lambda) = 0$. At this point, a local minimum in the original variable space is reached, and the search can only escape from it by updating λ.

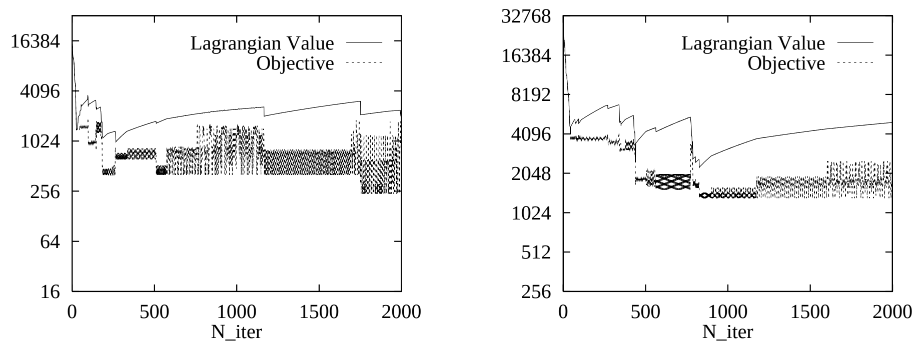
(d) *Amount of update of* λ (Line 7). A parameter c controls the magnitude of changes in λ. In general, c can be a vector of real numbers, allowing non-uniform updates of λ across different dimensions and possibly across time. In our experiments, $c = 1$ has been found to work well for most of the benchmarks tested. However, for some larger problems, a larger c results in shorter search time and better solutions.

The update rule in Line 7 results in nondecreasing λ because $U(\mathbf{x})$ is either 0 or 1. In contrast, in continuous problems λ_i of constraint $g_i(\mathbf{x}) = 0$ increases when $g_i(\mathbf{x}) > 0$ and decreases when $g_i(\mathbf{x}) < 0$.

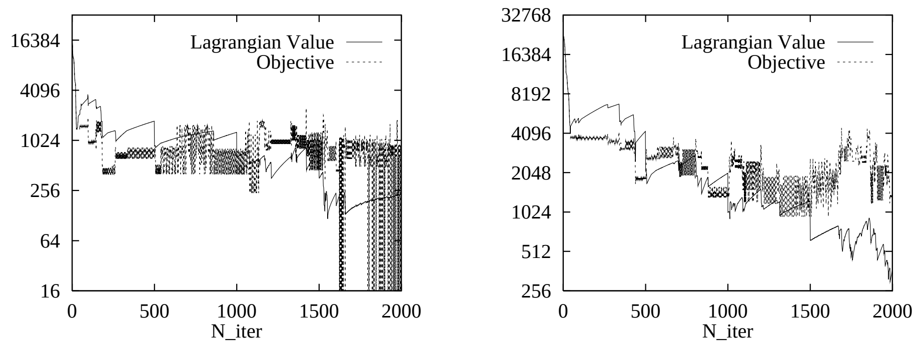
From our previous experience in solving SAT problems using DLM, we know that some λ values for some difficult problems can become very large after tens of thousands of iterations. At this point, the Lagrangian search space (6.18) becomes very rugged, and the search has difficulty in identifying an appropriate direction to move. To cope with this problem, λ should be reduced periodically to keep them small and to allow DLM to better identify good solutions.

The situation is worse in MAX-SAT because the weights of clauses, w, can be in a large range, making $L_{maxsat}(\mathbf{x}, \lambda)$ in (6.18) even more rugged and difficult to search. Here, $\lambda + w$ in MAX-SAT has a similar role as λ in SAT. Without any mechanism to reduce $\lambda + w$, $L_{maxsat}(\mathbf{x}, \lambda)$ can become very large and rugged as the search progresses.

This situation is illustrated in the upper two graphs of Figure 6.10, which shows the behavior of DLM when it is applied to solve two MAX-SAT problems. The graphs show that the Lagrangian values vary in a very large range, which makes the Lagrangian space difficult to search.



Without periodic scaling of λ and w



With periodic scaling of λ and w every 500 iterations

Figure 6.10: Execution profiles of *jnh1* (left) and *jnh10* (right). *jnh1* is satisfiable and *jnh10* is not.

One way to overcome this problem is to reduce λ and w periodically. For instance, in the lower two graphs of Figure 6.10, $\lambda + w$ was scaled down by a factor of 2 every 500 iterations. This strategy reduces $L_{maxsat}(\mathbf{x}, \lambda)$ and restricts the growth of the Lagrange multipliers, leading to faster solutions for many test problems.

Figure 6.11 shows $\mathcal{A}_{maxsat2}$, our implementation of DLM to solve MAX-SAT. $\mathcal{A}_{maxsat2}$ uses the first strategy of hill-climbing in descents in the beginning and switches to the second strategy later. A parameter, κ, is used to control the switch from the first strategy to the second. We found that $\kappa = n/3$ works well and used this value in our experiments.

$\mathcal{A}_{maxsat2}$ updates λ only when a local minimum is reached. In our experiments, we have used a simple scheme that increases λ by a constant across all the clauses.

```
Set initial $\mathbf{x}$
Set $\lambda = 0$
Set $c = 1$
Set $\kappa = n/3$, where $n$ is the number of variables
Set $\lambda$ reduction interval $I_\lambda$, e.g. 500
Set reduction ratio $r$, e.g. 2.0
Set base $\lambda$ value $\lambda_b$, e.g. 1
Set base weight value $w_b$, e.g. 1
while $\mathbf{x}$ is not a feasible solution, i.e. $N(\mathbf{x}) > 0$, and
  termination criteria are not met
  if number of iterations $\geq \kappa$ then
    Maintain a list, $l$, of variables such that
      if one of them is flipped, $L_{maxsat}(\mathbf{x}, \lambda)$ will improve.
    if $l$ is not empty then
      Update $\mathbf{x}$ by flipping the first element of $l$
    else
      Update $\lambda$: $\lambda \longleftarrow \lambda + c \cdot U(\mathbf{x})$
    end_if
  else
    if $\exists$ variable $v$ s.t. $L_{maxsat}(\mathbf{x}', \lambda) < L_{maxsat}(\mathbf{x}, \lambda)$ when flipping $v$
      in a predefined order in $\mathbf{x}$ to get $\mathbf{x}'$ then
      $\mathbf{x} \longleftarrow \mathbf{x}'$
    else
      Update $\lambda$: $\lambda \longleftarrow \lambda + c \cdot U(\mathbf{x})$
    end_if
  end_if
  if iteration index $mod\ I_\lambda = 0$ then
    Reduce $\lambda$ and weights for all clauses if possible,
    e.g. $\lambda \longleftarrow \max(\lambda_b, \lambda/r), w_i \longleftarrow \max(w_b, w_i/r)$
  end_if
  if $\mathbf{x}$ is better than incumbent, keep $\mathbf{x}$ as incumbent.
end_while
```

Figure 6.11: $\mathcal{A}_{maxsat2}$: an efficient implementation of DLM.

$\mathcal{A}_{maxsat2}$ controls the magnitude of $L_{maxsat}(\mathbf{x}, \lambda)$ by periodically reducing λ and w. More sophisticated adaptive mechanisms can be developed to lead to better performance.

6.6.3 Experimental Results

In our experiments, we evaluate DLM using some benchmarks and compare it with previous results obtained by GRASP [229]. GRASP is a greedy randomized adaptive search procedure that has been shown to quickly produce good-quality solutions for a wide variety of combinatorial optimization problems, including SAT and MAX-SAT [204,205].

The 44 MAX-SAT problems tested by GRASP were derived from DIMACS SAT benchmark *jnh* that have 100 variables and between 800 and 900 clauses. Some of them are satisfiable, while others are not. Their weights are integers generated randomly between 1 and 1000. The 44 benchmark problems can be obtained from
ftp://netlib.att.com/math/people/mgcr/data/maxsat.tar.gz.

Our DLM code was written in C. The experiments were run on a 50-MHz Sun SparcStation 10/51. The code was compiled using *gcc* with the "-O" option.

In our experiments, we first studied the effect of dynamic reduction of λ on $L_{maxsat}(\mathbf{x}, \lambda)$. We compared the results obtained by DLM with and without periodic reduction of λ. For each test problem, we ran DLM with λ scaling for 10 times from random initial points, and DLM without λ scaling between 5 and 20 times, each for 10,000 iterations. We then took the best solution of the multiple runs as our final solution.

Table 6.11 and 6.12 show the results obtained by DLM with and without periodical λ scaling. When λ was not reduced periodically, DLM found 25 optimal solutions out of the 44 problems (based on the best results of 10 runs). When λ was reduced periodically, we tried several values of I_λ, the reduction interval, and r, the reduction ratio. Using I_λ of 100, 1000, 2000, and 5000 and r of 1.5, 2, and 4, we found significant improvement in solution quality when λ was reduced periodically (except for the case when $I_\lambda = 5000$). The best combination is when $I_\lambda = 500$ and $r = 2$, in which 12 more optimal solutions were found than without λ reduction, 18 test problems were improved, and no solution is worse. Overall, we found optimal solutions in 39 out of the 44 test problems (based on best of 20 runs).

Table 6.11 and 6.12 compare the solutions obtained by DLM and by GRASP. The results of GRASP are from [205] that were run for 10,000 iterations on a 250-MHz SGI Challenge

Table 6.11: Comparison of results of DLM and those of GRASP with respect to the optimal solutions. (jnh1-19: 850 clauses, $\sum_i w_i = 420925$. GRASP was run 10,000 iterations on a 250-MHz SGI Challenge with MIPS R4400 whereas DLM with λ scaling was run 5-20 times, each for 10,000 iterations, from random starting points on a Sun SparcStation 10/51. For DLM with dynamic λ scaling, $I_\lambda = 500$ and $r = 2$.)

	Deviations from Optimal Solutions					
Problem ID	Best result of 10 runs of DLM without λ scaling	# Runs of DLM with λ scaling 5	10	20	GRASP	Optimal solution
jnh1	0	0	0	0	−188	420925
jnh4	−214	−41	−41	−41	−215	420830
jnh5	−159	−164	−131	0	−254	420742
jnh6	0	0	0	0	−11	420826
jnh7	0	0	0	0	0	420925
jnh8	0	0	0	0	−578	420463
jnh9	−198	−7	−7	−7	−514	420592
jnh10	−70	0	0	0	−275	420840
jnh11	−13	−8	0	0	−111	420753
jnh12	0	0	0	0	−188	420925
jnh13	0	0	0	0	−283	420816
jnh14	0	−77	0	0	−314	420824
jnh15	−79	−79	−79	0	−359	420719
jnh16	−30	0	0	0	−68	420919
jnh17	0	0	0	0	−118	420925
jnh18	−349	−98	0	0	−423	420795
jnh19	−334	0	0	0	−436	420759

Table 6.12: Continuation of Table 6.11. Comparison of results of DLM and those of GRASP with respect to the optimal solutions. (jnh201-220: 800 clauses, $\sum_i w_i = 394238$; jnh301-310: 900 clauses, $\sum_i w_i = 444854$.)

	Deviations from Optimal Solutions					
Problem ID	Best result of 10 runs of DLM without λ scaling	# Runs of DLM with λ scaling 5	10	20	GRASP	Optimal solution
jnh201	0	0	0	0	0	394238
jnh202	−126	0	0	0	−187	394170
jnh203	0	0	0	0	−310	394199
jnh205	0	0	0	0	−14	394238
jnh207	0	0	0	0	−137	394238
jnh208	0	0	0	0	−172	394159
jnh209	0	0	0	0	−207	394238
jnh210	0	0	0	0	0	394238
jnh211	0	0	0	0	−240	393979
jnh212	−11	0	0	0	−195	394238
jnh214	0	0	0	0	−462	394163
jnh215	0	0	0	0	−292	394150
jnh216	−115	−149	0	0	−197	394226
jnh217	0	0	0	0	−6	394238
jnh218	0	0	0	0	−139	394238
jnh219	−278	0	0	0	−436	394156
jnh220	0	0	0	0	−185	394238
jnh301	0	0	0	0	−184	444854
jnh302	−472	−510	−338	−338	−211	444459
jnh303	−258	−152	−143	−143	−259	444503
jnh304	−273	−106	0	0	−319	444533
jnh305	−290	−196	−194	−194	−609	444112
jnh306	0	0	0	0	−180	444838
jnh307	0	0	0	0	−155	444314
jnh308	−268	−103	−60	0	−502	444724
jnh309	0	0	0	0	−229	444578
jnh310	−224	0	0	0	−109	444391

Table 6.13: Comparison of average CPU time in seconds for 10,000 iterations of DLM and GRASP.

Problem	# Clauses	GRASP	DLM
jnh1-19	850	799.8	1.7
jnh201-220	800	692.9	1.7
jnh301-310	900	937.8	2.0

with MIPS R4400. In contrast, DLM with λ scaling was run between 5 and 20 times from random initial points using 10,000 iterations each. More runs are helpful in obtaining better solutions. 31, 36, and 39 optimal solutions were found when DLM with λ scaling was run 5, 10, and 20 times from random initial points, respectively.

Our results show significant improvement over GRASP in 40 of the 44 test problems. For the remaining four problems, both DLM and GRASP found optimal solutions for three of them, and GRASP found a better solution in one. It is important to note that, although the differences between GRASP's results and the optimal solutions are relatively small, improvements in this range are the most difficult for any search algorithm.

Table 6.13 shows the average time in seconds for 10,000 iterations of GRASP and DLM. Since GRASP was run on a faster machine, one iteration of DLM is around three orders of magnitude less than that of GRASP.

GRASP found better results when the number of iterations was increased. Using the ten test problems reported in [205] that were run for 100,000 iterations of GRASP, Table 6.14 compares the results of DLM and GRASP with longer runs. It shows that DLM still found better solutions in all the problems, except the cases that both found optimal solutions.

Increasing the number of iterations of DLM can also improve the best solutions found. Based on the best of 10 runs, DLM was able to find 40 out of 44 optimal solutions when the number of iterations is increased to 100,000, while it is 36 when the number of iterations is 10,000.

To summarize, we have presented DLM for solving MAX-SAT problems in this section. DLM belongs to the class of incomplete methods that attempt to find approximate solutions

Table 6.14: Comparison of solutions found by DLM and those found by GRASP with longer iterations.

Problem	GRASP		DLM	Optimal
Id	10K	100K	10K iter	Solutions
jnh1	−188	−77	0	420925
jnh10	−275	−259	0	420840
jnh11	−111	−111	0	420753
jnh12	−188	−54	0	420925
jnh201	0	0	0	394238
jnh202	−187	−141	0	394170
jnh212	−195	−50	0	394238
jnh304	−319	0	0	444533
jnh305	−609	−368	−194	444112
jnh306	−180	−63	0	444838

in a fixed amount of time. DLM improves over existing discrete local- and global-search methods in the following ways:

- DLM can escape from local minima in a continuous trajectory without restarts, hence avoiding a break in the trajectory as in methods based on restarts. This is advantageous when the trajectory is already in the vicinity of a local minimum, and a random restart may bring the search to a completely different search space.
- DLM provides a mechanism to control the growth rate and the magnitude of Lagrange multipliers, which is essential in solving difficult MAX-SAT problems.

Using a large set of test problems, we have compared the performance of DLM with that of GRASP, one of the best local-search methods for solving MAX-SAT problems. Our experimental results show that DLM solves these problems two order-of-magnitude faster than GRASP, and finds better solutions for a majority of test problems in just a few seconds.

6.7 Designing Multiplierless QMF Filter Banks

In this section, we apply *discrete Lagrangian method* (DLM) to solve an engineering application problem: the design of multiplierless quadrature-mirror-filter (QMF) filter banks. In multiplierless QMF filter banks, filter coefficients are powers-of-two (PO2) where numbers are represented as sums or differences of powers of two (also called canonical signed digit representation, or CSD). Using this representation, multiplications become additions, subtractions and shifting.

Based on a formulation similar to the real-value QMF design problem presented in Chapter 5, we formulate the design problem as a nonlinear discrete constrained optimization problem, using the reconstruction error as the objective and the other performance metrics as constraints. One of the major advantages of this formulation is that it allows us to search for designs that improve the best existing designs with respect to all performance metrics, rather than designs that trade one performance metric for another. Then, we apply DLM to solve this discrete optimization problem. In our experiments, we show that our method can find designs that improve over Johnston's benchmark designs using a maximum of three to six 1-bits in each filter coefficient. The performances of these designs are close to the best continuous solutions found by Novel in Chapter 5.

6.7.1 Introduction

Traditional FIR filters in QMF filter 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 such a 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 1-bits in the multiplier. For example, multiplying x by 0100001001 can be written as the sum of three terms, $x \times 2^8 + x \times 2^3 + x \times 2^0$, each of which can be obtained by shifting x. 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

$$H(\mathbf{x}, z) = \sum_{i=0}^{n-1} x_i z^{-i} = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{d-1} e_{i,j} 2^j \right) z^{-i} \tag{6.22}$$

where $\sum_{j=0}^{d-1} |e_{i,j}| \leq l$ for all i, $e_{i,j} = -1, 0, 1$, n is the length of the PO2 filter (the number of variables), l is the maximum number of 1-bits used in each coefficient, and d is the number of bits in each coefficient.

The design of multiplierless filters has been solved as integer programming problems that represent filter coefficients as variables with restricted values of powers-of-two. Other optimization techniques that have been applied include combinatorial search methods [211], simulated annealing [40], genetic algorithms [214], linear programming [147], and continuous Lagrange-multiplier methods in combination with a tree search [231].

The design of QMF filter banks can be formulated as a multi-objective unconstrained optimization problem or as a single-objective constrained optimization problem. In this thesis, we take the approach similar to the design of real-value QMF filter banks in Chapter 5, and formulate the design of a PO2 QMF filter bank in the most general form as a constrained nonlinear optimization problem [267]:

$$\begin{aligned} \min_{\mathbf{x} \in D} \quad & f(\mathbf{x}) = E_r(\mathbf{x})/\theta_{E_r} \\ \text{subject to} \quad & E_p(\mathbf{x})/\theta_{E_p} \leq 1 \quad E_s(\mathbf{x})/\theta_{E_s} \leq 1 \quad T_t(\mathbf{x})/\theta_{T_t} \leq 1 \\ & \delta_p(\mathbf{x})/\theta_{\delta_p} \leq 1 \quad \delta_s(\mathbf{x})/\theta_{\delta_s} \leq 1 \end{aligned} \tag{6.23}$$

where $\mathbf{x}$ is a vector of discrete coefficients in the discrete domain D, 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), and θ_{E_r} is the baseline value of reconstruction error found in the best known design. The goal here is to find designs whose performance measures are better than or equal to those of the reference design. Since the objective and constraints are nonlinear, the problem is multi-modal with many local minima.

The inequality constrained optimization problem (6.23) can be transformed into an equality constrained optimization problem as follows:

$$
\begin{aligned}
\text{Minimize} \quad & f(\mathbf{x}) = \frac{E_r(\mathbf{x})}{\theta_{E_r}} && (6.24)\\
\text{subject to} \quad & g_1(\mathbf{x}) = \max\left(\frac{E_p(\mathbf{x})}{\theta_{E_p}} - 1, 0\right) = 0 \\
& g_2(\mathbf{x}) = \max\left(\frac{E_s(\mathbf{x})}{\theta_{E_s}} - 1, 0\right) = 0 \\
& g_3(\mathbf{x}) = \max\left(\frac{\delta_p(\mathbf{x})}{\theta_{\delta_p}} - 1, 0\right) = 0 \\
& g_4(\mathbf{x}) = \max\left(\frac{\delta_s(\mathbf{x})}{\theta_{\delta_s}} - 1, 0\right) = 0 \\
& g_5(\mathbf{x}) = \max\left(\frac{T_t(\mathbf{x})}{\theta_{T_t}} - 1, 0\right) = 0
\end{aligned}
$$

where all the objective and constraints have been normalized with respect to the corresponding values of the best known design.

6.7.2 DLM Implementation Issues

DLM was mainly implemented by Mr. Zhe Wu to solve the design problem of multiplierless filter banks. The Lagrangian function of (6.24) is

$$F(\mathbf{x}, \lambda) \;=\; f(\mathbf{x}) + \sum_{i=1}^{5} \lambda_i \times g_i(\mathbf{x}) \tag{6.25}$$

where $\lambda = (\lambda_i, i = 1, \cdots, 5)$ are Lagrange multipliers. To apply DLM to solve (6.24), we first define the discrete gradient operator as follows:

$$\Delta_{\mathbf{x}} F(\mathbf{x}, \lambda) \;=\; \delta_{\mathbf{x}} \tag{6.26}$$

where $F(\mathbf{x} \ominus \delta_{\mathbf{x}}, \lambda)$ [2] has the minimum value among all $F(\mathbf{y}, \lambda)$ for $\mathbf{y}$ sufficiently close to $\mathbf{x}$.

[2] $\ominus$ is an operator for changing one point in discrete space into another with δ value. An example of $\ominus$ is the exclusive-OR operator.

```
1. Generate a starting point $\mathbf{x}$
2. Set initial value of $\lambda$
3. while $\mathbf{x}$ is not a saddle point or stopping condition has not been reached
4.     update $\mathbf{x}$ to $\mathbf{x}'$ only if this will result in $F(\mathbf{x}',\lambda) < F(\mathbf{x},\lambda)$
5.     if condition for updating $\lambda$ is satisfied then
6.         $\lambda_i := \lambda_i + c \times g_i(\mathbf{x})$, ($c > 0$ is real)
7.     end if
8. end while
```

Figure 6.12: An implementation of DLM for solving the design problem of multiplierless filter banks

Figure 6.12 shows our implementation of DLM for designing multiplierless filter banks. Mr. Zhe Wu has addressed the following issues in implementing DLM.

Generating a Starting Point

There are two alternatives to select a starting point (Line 1 in Figure 6.12): choosing an existing PO2 QMF filter bank as a starting point, or choosing a discrete approximation of an existing QMF filter bank with real coefficients. The first alterative is not possible because not many such filter banks are available in the literature. In this section, we discuss the second alternative.

Given a real coefficient and b, the maximum number of 1-bits to represent the coefficient, we first apply Booth's algorithm [33] to represent consecutive 1's using two 1-bits and then truncate the least significant bits of the coefficients. This approach generally allows more than b 1-bits to be represented in b 1-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 1-bits:

$$0.10011101100 \Longrightarrow_{\text{Booth'sAlgorithm}} 0.101000\bar{1}0\bar{1}00 \Longrightarrow_{\text{Truncation}} 2^{-1} + 2^{-3}.$$

Previous work [48, 157, 211] shows that scaling has a significant impact on the coefficient-optimization process in PO2 filters. In our case, the performance of a PO2 filter obtained

Table 6.15: Comparison of a PO2 filter bank obtained by truncating the real coefficients of Johnston's 32e QMF filter bank [134] 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

by truncating its real coefficients to a fixed maximum number of 1-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 [134] as a starting point. Table 6.15 shows the performance of two PO2 filters: Filter A was obtained by truncating each of the original coefficients to a maximum of 3 1-bits, and Filter B was obtained by multiplying each of the coefficients by 0.5565 before truncation. The example shows that Filter B performs 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 as a starting point. This example illustrates that multiplying by a scaling factor changes the bit patterns in the filter coefficients, which can improve the quality of the starting point when the coefficients are truncated.

Figure 6.13 shows a simple but effective algorithm to find the proper scaling factor. We evaluate the quality of the resulting starting point by a weighted sum of the performance metrics based on constraint violations. Since under most circumstances, the constraint on transition bandwidth is more difficult to satisfy, we assign it a weight 100 and assign 1 to the other four constraints. Our objective in finding a good scaling factor is different from that in previous work [48, 157, 211]. 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 6.13 executes fast, and the scaling factors chosen are reasonable and suitable. It is important to point out that scaling does not

```
1. LeastSum=$+\infty$
2. for scaleFactor := 0.5000 to 1.0 step 0.0001
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 := $\sum_{i=1}^{5} w_i \times g_i(\mathbf{x})$
6.     if (sum < LeastSum) then
7.         LeastSum := sum
8.         BestScale := scaleFactor
9.     endif
10. endfor
11. return BestScale
```

Figure 6.13: Algorithm for finding the best scaling factor, where w_i is the weight of constraint i.

help when the number of 1-bits allowed to represent each coefficient is large. For instance, when the maximum number of 1-bits allowed is larger than 6, the performance of all the filters is nearly the same for all scaling factors.

Updating x

The value of $\mathbf{x}$ is updated in Line 4 of DLM in Figure 6.12. There are two ways in which $\mathbf{x}$ can be updated: greedy descent and hill climbing. Hill climbing is more efficient and generally leads to good PO2 designs. Therefore, we use hill climbing as our update strategy.

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

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

For each bit $b_{i,j}$, we perturb it to be a new bit $b'_{i,j}$ that differs from $b_{i,j}$ by either the sign or the exponent or both, with the condition that $b'_{i,j}$ is not the same in sign and exponent as another bit of the ith coefficient. Using $b_{i,1}, \cdots, b_{i,j-1}, b'_{i,j}, \cdots, b_{i,n}$ while keeping other coefficients the same, we compute the new Lagrangian value $F(\mathbf{x}', \lambda)$ and accept the change if $F(\mathbf{x}', \lambda) < F(\mathbf{x}, \lambda)$.

Initializing and Updating λ

The value of λ is initialized in Line 2 of DLM in Figure 6.12. To allow our experiments to be repeated and our results be reproduced easily, we always set λ to zero as our initial point.

Line 5 of DLM in Figure 6.12 is related to the condition when λ should be updated. As we know in solving SAT and MAX-SAT problems, λ for violated constraints should be updated less frequently in DLM than in traditional Lagrangian methods for continuous problems. Thus, 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 in our implementation may not be exact.

When updating λ before the search reaches a local minimum of $F(\mathbf{x}, \lambda)$, we set c in Line 6 of DLM to be a normalized value as follows:

$$c = \frac{\theta_{speed}}{\max_{i=1}^{5} g_i(\mathbf{x})} \tag{6.27}$$

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

When the search reaches a local minimum, a more efficient rule for updating λ has been developed by Mr. Zhe Wu to bring the search out of the local minimum. A proper value of c different from that in (6.27) is calculated. If λ is increased too fast, then the search will restart from a random starting point. On the other hand, if the increase of λ is too small, then the local minimum is not changed, and all surrounding points are still worse. Hence, we would like to set c large enough so that some of its neighbor points will be better. This means that, after λ has been changed to λ', there exists $\mathbf{x}'$ in the neighborhood of x such that

$$F(\mathbf{x}, \lambda) \leq F(\mathbf{x}', \lambda) \quad \text{and} \quad F(\mathbf{x}', \lambda') < F(\mathbf{x}, \lambda') \tag{6.28}$$

Replacing $F(\mathbf{x}, \lambda)$ by $f(\mathbf{x}) + \sum_{i=0}^{5} \lambda_i \times g_i(\mathbf{x})$ in (6.28), we get the following condition before λ changes:

$$F(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^{5} \lambda_i \times g_i(\mathbf{x}) \leq F(\mathbf{x}', \lambda) = f(\mathbf{x}') + \sum_{i=1}^{5} \lambda_i \times g_i(\mathbf{x}') \tag{6.29}$$

and that after λ_i is updated to $\lambda_i' = \lambda_i + c \times g_i(\mathbf{x})$:

$$\begin{aligned} F(\mathbf{x}, \lambda') &= f(\mathbf{x}) + \sum_{i=1}^{5} (\lambda_i + c \times g_i(\mathbf{x})) \times g_i(\mathbf{x}) \qquad (6.30) \\ > \quad F(\mathbf{x}', \lambda') &= f(\mathbf{x}') + \sum_{i=1}^{5} (\lambda_i + c \times g_i(\mathbf{x})) \times g_i(\mathbf{x}') \end{aligned}$$

where $g_i(\mathbf{x}')$ is the new violation value of the ith constraint at $\mathbf{x}'$. From (6.29) and (6.30), we get

$$\begin{aligned} & 0 \leq F(\mathbf{x}', \lambda) - F(\mathbf{x}, \lambda) < c \times \sum_{i=0}^{5} \lambda_i \times g_i(\mathbf{x}) \times (g_i(\mathbf{x}) - g_i(\mathbf{x}')) \\ \Longrightarrow \quad & c > \frac{F(\mathbf{x}', \lambda) - F(\mathbf{x}, \lambda)}{\sum_{i=0}^{5} \lambda_i \times g_i(\mathbf{x}) \times (g_i(\mathbf{x}) - g_i(\mathbf{x}'))} \qquad (6.31) \end{aligned}$$

When c is large enough to satisfy (6.31) for all $\mathbf{x}'$, after updating λ to λ', there exists some point the neighborhood of $\mathbf{x}$ that has smaller Lagrangian value.

As an example, consider in Figure 6.14 the constraint violation of transition bandwidth T_t in a typical search based on constraints derived from Johnston's 32e filter bank [134]. Figure 6.14 shows that the value of violation on T_t can be extremely small, in the order of 10^{-5} in the later part of the search. For such small violation values, the corresponding Lagrange multiplier λ_{T_t} increases very slow without a large c. Using c defined in (6.31) to increase λ_{T_t}, we see in Figure 6.14 that λ_{T_t} was increased significantly three times when the

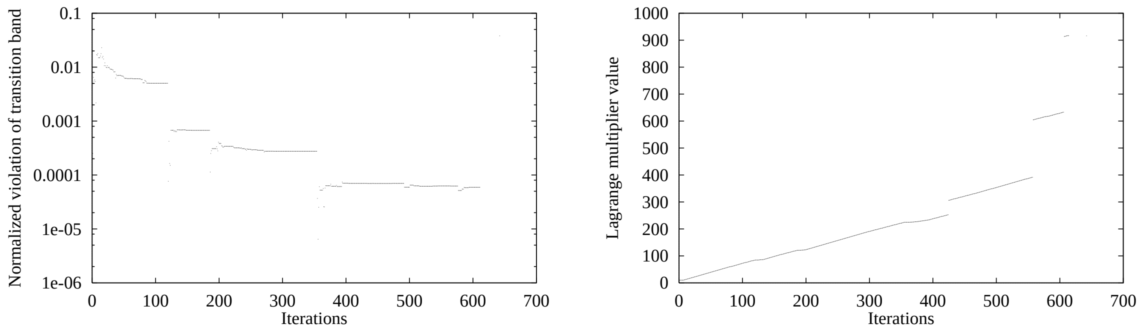


Figure 6.14: The violation values of T_t during a search (left) and the corresponding value of λ_{T_t} (right).

condition for updating λ was satisfied. These saved at least half of the total search time for the solution.

Finally, we notice in Line 6 of DLM that λ is nondecreasing. This means that as the search progresses, the λs grow, and more emphasis is placed on getting the search out of local minima. This approach fails when the λs are so large that the search is brought to a totally new terrain when Line 6 of DLM is applied. To cope with this problem, we measure the relative values between $f(\mathbf{x})$ and $\sum_{i=1}^{5} \lambda_i \times g_i(\mathbf{x})$ and keep them within a reasonable range:

$$0 \leq \frac{\sum_{i=1}^{5} \lambda_i \times g_i(\mathbf{x})}{f(\mathbf{x})} \leq \theta_{threshold} \tag{6.32}$$

If the ratio exceeds $\theta_{threshold}$, then we divide all the λs by a constant r to regain the balance. In our experiments, we set $\theta_{threshold}$ to be 250 and r be 1.3 and check $\theta_{threshold}$ every time λ is updated.

6.7.3 Experimental Results

In this section, we compare the performance of PO2 QMF filter banks designed by DLM and those by Johnston [134], Chen *et al.* [48], Novel, simulated annealing, and genetic algorithms [267].

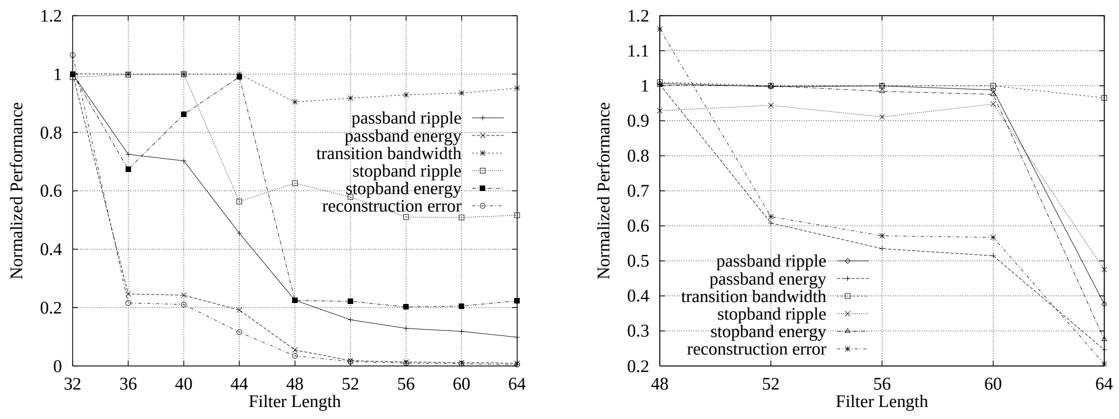


Figure 6.15: Normalized performance with respect to Johnston's 32e (left) and 48e (right) QMF filter banks for PO2 filters with a maximum of 3 1-bits per coefficient and different number of filter taps.

There are two parameters in a PO2 filter bank design: the maximum number of 1-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 values of performance metrics. In contrast, Johnston [134] used sampling in computing energy values. Hence, designs found by Johnston are not necessarily at the local minima in a continuous sense.

We have evaluated PO2 designs obtained by DLM with respect to Johnston's designs whose coefficients are 32-bit real numbers. Using the performance of Johnston's 32e design as constraints [134], we ran DLM from 10 different starting points obtained by randomly perturbing 1% of all the coefficients of Johnston's design. Each run was limited so that each 1-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 6.15, which shows the normalized performance of PO2 designs with increasing number of filter taps, while each filter coefficient has a maximum of 3 1-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's 32e 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 1-bits,

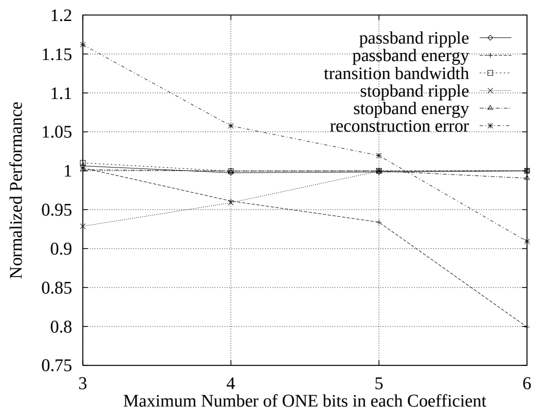


Figure 6.16: Normalized performance with respect to Johnston's 48e QMF filter bank for PO2 filters with 48 taps and different maximum number of 1-bits per coefficient.

and set the filter coefficients of the remaining taps to zeroes initially. The starting points of 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 [134], we set a limit so that each 1-bit of the coefficient was processed in a round-robin fashion 800 times, and ran DLM once from the truncated Johnston's 48e design. (The scaling factor was 0.5584 for filters with 48, 52, 56, and 60 taps. The scaling factor was 0.6486 for the filter with 64 taps.) Our results show that our 48-tap PO2 design is slightly worse than that of Johnston's, while our PO2 designs with 52 taps or larger 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 1-bits in each coefficient from 3 to 6. We set a limit so that each 1-bit of the coefficient was processed in a round-robin fashion 800 times, and ran DLM once from the truncated Johnston's 48e design. Figure 6.16 shows a design that is

Table 6.16: Comparison of normalized performance of PO2 filter banks designed by DLM with respect to those designed by Johnston and Chen. Columns 2-4 show the performance of DLM using 3 1-bits for 32-tap filters and 6 1-bits for 64-tap filters normalized with respect to that of Johnston's 32e, 64d, and 64e filter banks [134]. Columns 5-6 show the performance of DLM using 3 1-bits normalized with respect to that of Chen *et al.*'s 64-tap and 80-tap filter banks [48].

	Comparing with Johnston's			Comparing with Chen *et al.*'s	
Metrics	DLM–32e	DLM–64d	DLM–64e	DLM–64	DLM–80
E_r	0.83	0.90	0.89	0.91	0.95
E_p	1.00	0.82	0.83	0.80	0.96
E_s	1.00	1.00	1.00	1.00	0.86
δ_p	1.00	0.97	1.00	1.00	1.00
δ_s	0.99	0.75	1.00	1.00	1.00
T_t	1.00	1.00	1.00	1.00	1.00

better than Johnston's 48e design when the maximum number of 1-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.)

With respect to Johnston's 64d and 64e designs, Table 6.16 shows improved PO2 designs obtained by DLM using a maximum of 6 1-bits per coefficient and 64 taps. No improvements were found when the maximum number of 1-bits is less than 6.

Table 6.16 further shows improved PO2 designs obtained by DLM with respect to Chen *et al.*'s PO2 designs (with 3 1-bit coefficients) with 64 and 80 taps, respectively, and a maximum of 3 1-bits per coefficient. In these designs, we used Chen *et al.*'s designs as starting points and ran DLM once with a limit so that each 1-bit was processed in a round-robin fashion 1,000 times.

Finally, we compare in Table 6.17 the performance of 32e PO2 filter banks obtained by DLM with a maximum of 3 1-bits per coefficient, and the performance of 32e real-value filter banks obtained by Novel, simulated annealing (SA), and evolutionary algorithms (EAs) in Chapter 5. Novel and DLM find solutions that is better than or equal to Johnston's solution in all six performance metrics, although DLM's solution has larger reconstruction

Table 6.17: Comparison of normalized performance of 32e PO2 QMF filter banks designed by DLM with respect to those designed by Novel, simulated annealing (SA), and genetic algorithms (EA-Ct and EA-Wt).

Metrics	DLM	Novel	SA	EA-Ct	EA-Wt
E_r	0.83	0.71	0.50	0.094	0.51
E_p	1.00	0.90	0.58	1.54e5	0.59
E_s	1.00	1.00	1.00	1.26e3	1.00
δ_p	1.00	1.00	1.00	1.69e3	1.00
δ_s	0.99	1.00	1.00	17.3	1.00
T_t	1.00	1.00	1.01	0.00	1.01

error than Novel's. SA and EA-Wt result in larger transition bandwidths than Johnston's. EA-Ct converges to designs with very small reconstruction errors, while other constraints are violated significantly. Considering the fact that the DLM design uses a maximum of two additions in each tap rather than a complex carry save adder or a 32-bit multiplier, the design obtained by DLM has much lower implementation cost and can achieve faster speed.

To summarize, we have applied DLM to design multiplierless powers-of-two (PO2) QMF filter banks. Our design method is unique because it starts from a constrained formulation with the objective of finding a design that improves over the 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.

DLM effectively finds high quality filter-bank designs with very few 1-bits for each filter coefficient, allowing the design to be implemented in a fraction of the cost. For example a design with a maximum of 3 1-bits per coefficient can be implemented in two additions in each tap. In contrast, a full 32-bit multiplier is close to one order of magnitude more expensive in terms of hardware cost and computational delay.

6.8 Summary

In this chapter, we have applied our new discrete Lagrangian method (DLM) to solve discrete optimization problems that include the satisfiability (SAT), the maximum satisfiability (MAX-SAT) and the multiplierless QMF filter-bank design problems. These problems have all been formulated as discrete constrained optimization problems.

We extend the theory of Lagrange multipliers for continuous problems to discrete problems. With respect to problems in discrete space, we define the concept of saddle points, derive the Saddle Point Theorem, propose methods to compute discrete gradients, and investigate various heuristics in implementing DLM. We show the Fixed Point theorem which guarantees that the algorithm will continue to search until a saddle point is found. DLM belongs to the class of incomplete methods that attempts to find a saddle point if one exists, but will not terminate if the problem is infeasible.

In our experiments, we have compared the performance of DLM with respect to some of the best existing results in all three applications. Using a large set of SAT benchmark problems archived in DIMACS, we have shown that DLM usually solves these problems substantially faster than other competing local-search and global-search methods, and can solve some difficult problems that other methods cannot. In solving MAX-SAT problems, we have compared the performance of DLM with that of GRASP, one of the best methods for solving MAX-SAT. Our experimental results show that DLM solves these problems two orders-of-magnitude faster and found better solutions for a majority of test problems in a few seconds. Finally, in designing multiplierless QMF filter banks, our experimental results show that DLM finds better designs than existing methods. Comparing with real-value filter banks, the multiplierless filter banks designed by DLM have little performance degradation and with very few 1-bits for each filter coefficient, allowing the design to be implemented in a fraction of the cost and execute much faster.

To summarize, DLM is a generalization of local search schemes that optimize the objective alone or optimize the constraints alone. When the search reaches a local minimum, the Lagrange multipliers in DLM lead the search out of the local minimum and move it in the direction provided by the multipliers. DLM also uses the value of an objective function (the number of violated constraints in the case of SAT problems) to provide further guidance in addition to the constraint violations. The dynamic shift in emphasis between the objective and the constraints, depending on their relative values, is the key of Lagrangian methods. DLM improves over existing discrete local- and global-search methods in the following aspects.

- DLM can escape from local minima without random restarts. When a constraint is violated but the search is in a local minimum, the corresponding Lagrange multipliers in DLM provide a force that grows with the amount of time that the constraint is violated, eventually bringing the search out of the local minimum. DLM may rely on systematic restarts (that are not based on random numbers) in the Lagrangian space when the Lagrange multipliers grow large and the search space becomes rugged.
- DLM escapes from local minima in a continuous trajectory, hence avoiding a break in the trajectory as in methods based on restarts. This is advantageous when the trajectory is already in the vicinity of a local minimum, and a random restart may bring the search to a completely different search space.
- DLM is more robust than other local-search methods and is able to find feasible solutions irrespective of its initial starting points. In contrast, descent methods have to rely on properly chosen initial points and on a good sampling procedure to find new starting points or by adding noise in order to bring the search out of local minima.

In short, the Lagrangian formulation and our discrete Lagrangian methods are based on a solid theoretical foundation, and can be used to develop better heuristic algorithms for solving discrete constrained optimization problems.

7. CONCLUSIONS AND FUTURE WORK

7.1 Summary of Work Accomplished

Many important engineering and industrial applications can be formulated as nonlinear, constrained or unconstrained, optimization problems. Improved solutions to these problems can lead to significant savings. In this thesis, we have designed efficient methods to handle nonlinear constraints and to overcome local minima. Using these methods, we have improved existing nonlinear optimization methods in two forms: finding better solutions at the same cost, or finding solutions of similar quality at less cost.

To summarize, we have made the following contributions in this thesis:

- We have developed a new Lagrangian method with dynamic weight adaptation to handle equality and inequality constraints in nonlinear optimization problems, and to improve the convergence speed and solution quality of traditional Lagrangian methods (Chapter 2).

- We have extended the traditional Lagrangian theory for the continuous space to the discrete space, and have developed efficient discrete Lagrangian methods. Our discrete Lagrangian theory provides the mathematical foundation of discrete Lagrangian methods that does not exist in existing methods for handling nonlinear discrete constraints (Chapter 2).

- We have developed an innovative trace-based global-search method to overcome local minima in a nonlinear function space. The trace-based global search relies on an external traveling trace to pull a search trajectory out of a local optimum in a continuous fashion without having to restart it from a new starting point. Good starting points identified in the global search are used in the local search to identify true local optima (Chapter 3).

- We have developed a prototype, called Novel (Nonlinear Optimization Via External Lead), that incorporates our new methods and solves nonlinear constrained and unconstrained problems in a unified framework. In Novel, constrained problems are first transformed into Lagrangian functions. Then, the nonlinear functions are searched for optimal solutions by a combination of global- and local-search methods (Chapter 3).

- We have applied Novel to solve neural-network design problems and have obtained significant improvements in learning feedforward neural networks. Artificial neural networks have many applications, but existing design methods are far from optimal. We formulate the neural-network learning problem as an unconstrained nonlinear optimization problem, and solve it using Novel. Novel has designed neural networks with higher quality than existing methods, or have found much smaller neural networks with similar quality (Chapter 4).

- We have applied Novel to design digital filter banks and have obtained improved solutions. Digital filter banks have been used in many digital signal processing and communication applications. We formulate the design of filter banks as a constrained nonlinear optimization problem and solve it using Novel. Novel has found better designs than the best existing solutions across all the performance metrics (Chapter 5).

- We have applied Novel to solve discrete problems, including the satisfiability problem (SAT), the maximum satisfiability problem (MAX-SAT), and the design of multiplierless filter banks. Many problems in computer science, management, and decision science can be formulated as discrete optimization problems. By formulating these problem (SAT, MAX-SAT, and the design of multiplierless filter banks) into discrete constrained optimization problems, Novel, particularly discrete Lagrangian method

(DLM), has obtained significant improvements over the best existing methods (Chapter 6).

7.2 Future Work

In this section, we present some possible avenues for research in the future.

- *Development of Trace-based Global Search for Solving Discrete Optimization problems.* We plan to extend the idea of our trace-based global search to solve discrete problems. Nonlinear discrete optimization problems have many local minima that trap local search methods. A trace-based global search can help the search escape from local minima, and avoid expensive local search in unpromising regions. We plan to study different forms of trace functions, identify their effectiveness, and investigate various implementation issues.

- *Extension to Mixed-integer Programming Problems.* We have developed methods that solve continuous and discrete problems, respectively. Many application problems are mixed-integer programming problems, which have mixed variable domains – some continuous and some discrete. We plan to extend our methods to solve these problems.

- *Applications of Novel.* There are many applications that can be formulated as nonlinear optimization problems. The merits of our method lie in its ability to find better solutions for real-world applications. Different applications have different characteristics, requirements, and implementation considerations. The theoretical work and experimental results presented in this thesis provide a solid foundation for applying Novel to new applications.

APPENDIX A. EXPERIMENTAL RESULTS OF NOVEL ON SOME TEST FUNCTIONS

In this appendix, we demonstrate Novel's capability of finding good solutions by solving some multidimensional nonlinear optimization problems. We first describe these problems, which include 11 widely used multi-modal simple-bounded test functions with known optimal solutions [253]. Then, we report the experimental results of Novel, and compare the results with those obtained by multi-starts of gradient descent.

(1) **Six Hump Camel Back (C)**

$$f_C(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

Domain:

$$-5 \le x_i \le 5, \quad i = 1, 2$$

There are 2 global minima at

$$\begin{aligned} f^* &= -1.0316285 \\ x^* &= ((0.0898400, -0.712656), (-0.0898400, 0.712656)). \end{aligned}$$

There are 4 local minima in the minimization region.

(2) **Branin (BR)**

$$f_{BR}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$

Domain:

$$-5 \leq x_1 \leq 10, \quad 0 \leq x_2 \leq 15$$

There are 3 global minima at

$$\begin{aligned} f^* &= 0.397887 \\ x^* &= ((-3.14159, 12.27500), (3.14159, 2.27500), (9.42478, 2.47500)) \end{aligned}$$

(3) **Goldstein-Price (GP)**

$$\begin{aligned} f_{GP} &= [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times \\ &\quad [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \end{aligned}$$

Domain:

$$-2 \leq x_i \leq 2, \quad i = 1, 2$$

The global minimum is

$$\begin{aligned} f^* &= 3 \\ x^* &= (0, -1) \end{aligned}$$

There are 4 local minima in the minimization region.

(4) **Shekel 5 (S5)**

$$f_{S5} = -\sum_{i=1}^{5} \frac{1}{(x - a_i)(x - a_i)^T + c_i}$$

Domain:

$$0 \leq x_i \leq 10, \quad i = 1, \cdots, 4$$

Coefficients:

i	a_i	c_i	i	a_i	c_i
1	(4.0, 4.0, 4.0, 4.0)	0.1	4	(6.0, 6.0, 6.0, 6.0)	0.4
2	(1.0, 1.0, 1.0, 1.0)	0.2	5	(3.0, 7.0, 3.0, 7.0)	0.4
3	(8.0, 8.0, 8.0, 8.0)	0.2			

The global minimum is

$$\begin{aligned} f^* &= -10.1532 \\ x^* &= (4.00004, 4.00013, 4.00004, 4.00013) \end{aligned}$$

There are 5 local minima with $f \approx -1/c_i$ approximately at a_i.

(5) **Shekel 7 (S7)**

$$f_{S7}(x) = -\sum_{i=1}^{7} \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

Domain:

$$0 \leq x_i \leq 10, \quad i = 1, \cdots, 4$$

Coefficients

i	a_i	c_i	i	a_i	c_i
1	(4.0, 4.0, 4.0, 4.0)	0.1	5	(3.0, 7.0, 3.0, 7.0)	0.4
2	(1.0, 1.0, 1.0, 1.0)	0.2	6	(2.0, 9.0, 2.0, 9.0)	0.6
3	(8.0, 8.0, 8.0, 8.0)	0.2	7	(5.0, 5.0, 3.0, 3.0)	0.3
4	(6.0, 6.0, 6.0, 6.0)	0.4			

The global minimum is

$$\begin{aligned} f^* &= -10.4029 \\ x^* &= (4.00057, 4.00069, 3.99949, 3.99961) \end{aligned}$$

There are 7 local minima with $f \approx -1/c_i$ approximately at a_i.

(6) **Shekel 10 (S10)**

$$f_{S10}(x) = -\sum_{i=1}^{1} 0 \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

Domain:

$$0 \leq x_i \leq 10, \quad i = 1, \cdots, 4$$

Coefficients:

i	a_i	c_i	i	a_i	c_i
1	(4.0, 4.0, 4.0, 4.0)	0.1	6	(2.0, 9.0, 2.0, 9.0)	0.6
2	(1.0, 1.0, 1.0, 1.0)	0.2	7	(5.0, 5.0, 3.0, 3.0)	0.3
3	(8.0, 8.0, 8.0, 8.0)	0.2	8	(8.0, 1.0, 8.0, 1.0)	0.7
4	(6.0, 6.0, 6.0, 6.0)	0.4	9	(6.0, 2.0, 6.0, 2.0)	0.5
5	(3.0, 7.0, 3.0, 7.0)	0.4	10	(7.0, 3.6, 7.0, 3.6)	0.5

The global minimum is

$$\begin{aligned} f^* &= -10.5364 \\ x^* &= (4.00075, 4.00059, 3.99966, 3.99951) \end{aligned}$$

There are 10 local minima with $f \approx -1/c_i$ approximately at a_i.

(7) **Rastrigin (R)**

$$f_R(x) = x_1^2 + x_2^2 - \cos 18x_1 - \cos 18x_2$$

Domain:

$$-1 \leq x_i \leq 1, \quad i = 1, 2$$

The global minimum is

$$\begin{aligned} f^* &= -2 \\ x^* &= (0, 0) \end{aligned}$$

There are approximately 50 local minima in the minimization region.

(8) **Griewank 2 (G2)**

$$f_{G2}(x) = \frac{x_1^2 + x_2^2}{200} - \cos x_1 \cdot \cos \frac{x_2}{\sqrt{2}} + 1$$

Domain:

$$-100 \leq x_i \leq 100, \quad i = 1, 2$$

The global minimum is

$$\begin{aligned} f^* &= 0 \\ x^* &= (0, 0) \end{aligned}$$

There are approximately 500 local minima in the minimization region.

(9) **Griewank 10 (G10)**

$$f_{G10}(x) = \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos \frac{x_i}{\sqrt{i}} + 1$$

Domain:

$$-600 \leq x_i \leq 600, \quad i = 1, \cdots, 10$$

The global minimum is

$$\begin{aligned} f^* &= 0 \\ x^* &= (0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \end{aligned}$$

There are several thousand local minima in the minimization region.

(10) **Hartman 3 (H3)**

$$f_{H3}(x) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{3} \alpha_{ij}(x_j - p_{ij})^2}$$

Domain:

$$0 \leq x_i \leq 1, \quad i = 1, \cdots, 3$$

Coefficients:

i	α_i	c_i	p_i
1	(3.0, 10.0, 30.0)	1.0	(0.36890, 0.11700, 0.26730)
2	(0.1, 10.0, 35.0)	1.2	(0.46990, 0.43870, 0.74700)
3	(3.0, 10.0, 30.0)	3.0	(0.10910, 0.87320, 0.55470)
4	(0.1, 10.0, 35.0)	3.2	(0.03815, 0.57430, 0.88280)

The global minimum is

$$\begin{aligned} f^* &= -3.86278 \\ x^* &= (0.114614, 0.555649, 0.852547) \end{aligned}$$

(11) **Hartman 6 (H6)**

$$f_{H6}(x) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{6} \alpha_{ij}(x_j - p_{ij})^2}$$

Domain:

$$0 \leq x_i \leq 1, \quad i = 1, \cdots, 6$$

Coefficients:

i	α_i	c_i	p_i
1	(10, 3, 17, 3.5, 1.7, 8)	1	(0.1312, 0.1696, 0.5569, 0.0124, 0.8283, 0.5886)
2	(0.05, 10, 17, 0.1, 8, 14)	1.2	(0.2329, 0.4135, 0.8307, 0.3736, 0.1004, 0.9991)
3	(3, 3.5, 1.7, 10, 17, 8)	3	(0.2348, 0.1451, 0.3522, 0.2883, 0.3047, 0.6650)
4	(17, 8, 0.05, 10, 0.1, 14)	3.2	(0.4047, 0.8828, 0.8732,0.5743, 0.1091, 0.0381)

The global minimum is

$$\begin{aligned} f^* &= -3.32237 \\ x^* &= (0.201690, 0.150011, 0.476874, 0.275332, 0.311652, 0.657300) \end{aligned}$$

Table A.1: A set of multi-modal test functions.

Id.	Name	Dimension	# Global minima	# Local minima
1	Six Hump Camel Back	2	2	4
2	Branin	2	3	0
3	Goldstein-Price	2	1	4
4	Shekel 5	4	1	5
5	Shekel 7	4	1	7
6	Shekel 10	4	1	10
7	Rastrigin	2	1	≈ 50
8	Griewank 2	2	1	≈ 500
9	Griewank 10	10	1	> 1000
10	Hartman 3	3	1	N/A
11	Hartman 6	6	1	N/A

Table A.1 lists the summary of these test functions. Most of them are low-dimensional problems, with dimensions ranging from 2 to 10. Problems 1 to 6 only have a few local minima, while Problems 7, 8, and 9 have many. The number of local minima of problems 10 and 11 are unknown.

First, we use multi-starts of gradient descent to solve these problems. The results of multi-starts reveal the degree of difficulty in finding global optima. For each problem, we run gradient descent 10 times from randomly generated initial points. Table A.2 shows the number of runs that find the optimal solutions. If the optimal solution is not found in 10 runs, the best solution obtained is shown in the parenthesis. A large number of success implies that the global optimum is easy to find by gradient descents from random initial points. Table A.2 shows that multi-starts find the optimal solutions of Problems 1 through 6, 10, and 11. The optimal solutions of Problems 7, 8, and 9 are not found.

A simple one-global-stage Novel is applied to solve these test functions. The trace-based global-search phase of Novel has one stage, which is run for 10 time units. Initial points for the local-search phase, which uses gradient descent, are selected based on the trajectory generated by the global-search stage. One initial point is selected in every one time unit of global search. From these initial points, a total of 10 gradient descents are performed. For each problem, the range of each dimension is mapped to $[-1, 1]$, in which Novel performs the

Table A.2: Experimental results of multi-starts of gradient descent on the test functions. The number of runs that find the optimal solutions are shown in the table (out of 10 runs). If the optimal solution is not found, the best solution obtained is shown in the parenthesis.

Problem Id.	1	2	3	4	5	6	7	8	9	10	11
No. of Success	4	10	4	3	2	1	(-1.87)	(7.13)	(21.0)	5	5

Table A.3: Experimental results of Novel on the test functions. The index of the first local descent of Novel that finds the global minimum is shown in the table (out of 10 descents).

Problem Id.	1	2	3	4	5	6	7	8	9	10	11
Index of First Success	1	1	5	1	6	2	2	(0.54)	(1.70)	1	1

global search. Novel always starts from the origin if it is not a global optimum. Otherwise, Novel starts from 1/2 in each dimension.

The execution times of Novel and multi-starts are comparable. In Novel, the execution time of global-search phase is only a small fraction of the time spent in gradient descents. One gradient descent in Novel takes similar amount of time to one descent in multi-starts. Hence, since both Novel and multi-starts perform the same number of descents, the execution time of Novel is slightly longer.

Table A.3 shows the index of the first local descent of Novel that finds the global optimal solution. For example, it takes one descent for Novel to find the optimal solution of Problem 1, and it takes 5 descents to find the optimal solution of Problem 3. For Problems 8 and 9, the optimal solutions are not found by Novel. The best solutions obtained are shown in the parenthesis.

To summarize, Novel finds better solutions for the difficult problems than multi-starts in comparable amount of time. Novel finds the optimal solutions of Problem 7, while multi-starts do not. For Problems 8 and 9, Novel finds better solutions than multi-starts, although both do not find the optimal solutions. Both Novel and multi-starts of gradient descent find optimal solutions for the easy problems: Problems 1 to 6, and Problems 10 and 11.

REFERENCES

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

[2] E. H. L. Aarts and P. J. van Laarhoven. *Simulated Annealing: Theory and Practice.* Wiley, New York, 1987.

[3] J. Abadie and J. Carpentier. Generalization of the wolfe reduced gradient method to the case of nonlinear constraints. In R. Fletcher, editor, *Optimization*, pages 37–47. Academic Press, London, 1969.

[4] A. N. Akansu and R. A. Haddad. *Multiresolution Signal Decomposition.* Academic Press, Inc., San Diego, CA, 1992.

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

[6] L. Armijo. Minimization of functions having continuous partial derivatives. *Pacific J. Math.*, 16:1–3, 1966.

[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] M. Avriel. *Nonlinear Programming: Analysis and Methods.* Prentice-Hall, 1976.

[9] M. R. Azimi-Sadjadi and R.-J. Liou. Fast learning process of multilayer neural networks using recursive least squares. *IEEE Transactions on Signal Processing*, 40(2):446–450, February 1992.

[10] N. Baba, Y. Mogami, M. Kohzaki, Y. Shiraishi, and Y. Yoshida. A hybrid algorithm for finding the global minimum of error function of neural networks and its applications. *Neural Networks*, 7(8):1253–1265, 1994.

[11] E. Balas and A. Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. *Mathematical Programming Study*, 12:37–60, 1980.

[12] E. Balas and P. Toth. Branch and bound methods. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem*, pages 361–402. Wiley, New York, NY, 1985.

[13] P. Baldi. Gradient descent learning algorithm overview: a general dynamical systems perspective. *IEEE Transactions on Neural Networks*, 6(1):182–195, January 1995.

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

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

[16] E. Barnard. Optimization for training neural nets. *IEEE Transactions on Neural Networks*, 3(2):232–240, 1992.

[17] R. Battiti. First- and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, 4(2):141–166, 1992.

[18] R. Battiti. The reactive tabu search. *ORSA J. Computing*, 6(2):126–140, 1994.

[19] R. Battiti and G. Tecchiolli. Parallel biased search for combinatorial optimization: Genetic algorithms and TABU. *Microprocessors and Microsystems*, 16:351–367, 1992.

[20] R. Battiti and G. Tecchiolli. Training neural nets with the reactive tabu search. *IEEE Transactions on Neural Networks*, 6(5):1185–1200, September 1995.

[21] Benchmarks on nonlinear mixed optimization problems. available through ftp (ftp://elib.zib-berlin.de/pub/mp-testdata/ip/index.html), 1992.

[22] H. S. M. Beigi and C. J. Li. Learning algorithms for neural networks based on quasi-Newton with self-scaling. *Intelligent Control Systems*, 23:23–28, 1990.

[23] A. Bellen. Parallelism across the steps for difference and differential equations. In C. W. Gear A. Bellen and E. Russo, editors, *Numerical Methods for Ordinary Differential Equations. Lecture Notes in Mathematics #1386*, pages 22–35. Springer, Berlin, 1989.

[24] B. Betrò and F. Schoen. Sequential stopping rules for the multistart algorithm in global optimization. *Mathematical Programming*, 38, 1987.

[25] B. Betrò and F. Schoen. Optimal and sub-optimal stopping rules for the multistart algorithm in global optimization. *Mathematical Programming*, 57:445–458, 1992.

[26] M. Bianchini, M. Gori, and M. Maggini. On the problem of local minima in recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(2):167–177, March 1994.

[27] L. G. Birta and O. Abou-Rabia. Parallel block predictor-corrector methods for ODEs. *IEEE Trans. on Computers*, C-36:299–311, 1987.

[28] C. G. E. Boender and A. H. G. Rinnooy Kan. Bayesian stopping rules for multi-start global optimization methods. *Mathematical Programming*, 36, 1987.

[29] C. G. E. Boender and A. H. G. Rinnooy Kan. On when to stop sampling for the maximum. *Journal of Global Optimization*, 1(4):331–340, 1991.

[30] C. G. E. Boender, A. H. G. Rinnooy Kan, L. Stougie, and G. T. Timmer. A stochastic method for global optimization. *Mathematical Programming*, 22:125–140, 1982.

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

[32] I. O. Bohachevsky, M. E. Johnson, and M. L. Stein. Generalized simulated annealing for function optimization. *Technometrics*, 28:209–217, 1986.

[33] A. D. Booth. A signed binary multiplication technique. *Quart. J. Mech. Appl. Math.*, 4:236–240, 1951.

[34] C. G. Broyden. Quasi-newton methods. In W. Murray, editor, *Numerical Methods for Unconstrained Optimization*, pages 87–106. Academic Press, New York, 1972.

[35] R. Brunelli. Training neural nets through stochastic minimization. *Neural Networks*, 7(9):1405–1412, 1994.

[36] K. Burrage. The search for the holy grail, or: Predictor-corrector methods for solving ODEIVPs. *Appl. Numer. Math.*, 11:125–141, 1993.

[37] R. H. Byrd, T. Derby, E. Eskow, K. P. B. Oldenkamp, R. B. Schnabel, and C. Triantafillou. Parallel global optimization methods for molecular configuration problems. In *Proc. Sixth SIAM Conf. of Parallel Processing for Scientific Computation*, pages 165–169, Philadelphia, 1993.

[38] R. H. Byrd, C. L. Dert, A. H. G. Rinnooy Kan, and R. B. Schnabel. Concurrent stochastic methods for global optimization. *Mathematical Programming*, 46:1–30, 1990.

[39] R. H. Byrd, E. Eskow, R. B. Schnabel, and S. L. Smith. Parallel global optimization: numerical methods, dynamic scheduling methods, and applications to molecular configuration. In B. Ford and A. Fincham, editors, *Parallel Computation*, pages 187–207. Oxford University Press, 1993.

[40] R. A. Caruana and B. J. Coffey. Searching for optimal FIR multiplierless digital filters with simulated annealing. *Technical report, Philips Laboratories*, 1988.

[41] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[42] 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.

[43] B. C. Cetin, J. W. Burdick, and J. Barhen. Global descent replaces gradient descent to avoid local minima problem in learning with artificial neural networks. In *IEEE Int'l Conf. on Neural Networks*, 1993.

[44] J. Chakrapani and J. Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. *Annals of Operation Research*, 41:327–341, 1993.

[45] Y. Chang and B. W. Wah. Lagrangian techniques for solving a class of zero-one integer linear programs. In *Proc. Computer Software and Applications Conf.*, 1995.

[46] D. Chazan and W. L. Miranker. A nongradient and parallel algorithm for unconstrained minimization. *SIAM Journal on control*, 8:207–217, 1970.

[47] 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.

[48] 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.

[49] O. T. Chen and B. J. Sheu. Optimization schemes for neural network training. In *Proc. IEEE Int'l Conf. on Neural Networks*, pages 817–822, Orlando, 1994.

[50] A. Cichocki and R. Unbehauen. Switched-capacitor artificial neural networks for nonlinear optimization with constraints. In *Proc. of 1990 IEEE Int'l Symposium on Circuits and Systems*, pages 2809–2812, 1990.

[51] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. of 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[52] S. A. Cook. An overview of computational complexity. *Comm. of the ACM*, 26(6):401–408, June 1983.

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

[54] C. D. Creusere and S. K. Mitra. A simple methor for designing high-quality prototype filters for m-band pseudo QMF banks. *IEEE Trans. on Signal Processing*, 43(4):1005–1007, April 1995.

[55] M. R. Crisci, B. Paternoster, and E. Russo. Fully parallel Runge-Kutta-Nyström methods for ODEs with oscillating solutions. *Appl. Numer. Math.*, 11:143–158, 1993.

[56] A. Croisier, D. Esteban, and C. Galand. Perfect channel splitting by use of interpolation/decimation/tree decomposition techniques. In *Proc. of Int'l Conf. on Information Science and Systems*, pages 443–446, 1976.

[57] H. Crowder, E. L. Johnson, and M. W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

[58] E. D. Dahl. Accelerated learning using the generalized delta rule. In *Proc. Int'l Conf. on Neural Networks*, volume II, pages 523–530. IEEE, 1987.

[59] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[60] A. Davenport, E. Tsang, C. Wang, and K. Zhu. Genet: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proc. of the 12th National Conf. on Artificial Intelligence*, pages 325–330, Seattle, WA, 1994.

[61] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.

[62] J. E. Dennis and J. J. More. Quasi-newton methods: motivation and theory. *SIAM Rev.*, 19:46–89, 1977.

[63] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice-Hall, Englewood Cliffs, NJ, 1983.

[64] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice Hall, 1983.

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

[66] Dimacs sat benchmarks. available through ftp (ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/cnf), 1994.

[67] L. C. W. Dixon. Neural networks and unconstrained optimization. In E. Spedicato, editor, *Algorithms for Continuous Optimization: The State of the Art*, pages 513–530. Kluwer Academic Publishers, Netherlands, 1994.

[68] J. Engel. Teaching feedforward neural networks by simulating annealing. *Complex System*, 2:641–648, 1988.

[69] 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.

[70] S. Ergezinger and E. Thomsen. An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer. *IEEE Transactions on Neural Networks*, 6(1):31–42, January 1995.

[71] D. Esteban and C. Galand. Application of quadrature mirror filters to split band voice coding schemes. In *Proc. of IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, pages 191–195, 1977.

[72] 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.

[73] D. K. Fadeev and V. N. Fadeev. *Computational Methods of Linear Algebra.* Freeman, 1963.

[74] S. E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In D. Touretzky, G. E. Hinton, and T. J. Sejnowski, editors, *Proc. Connectionist Models Summer School*, pages 38–51, Palo Alto, CA, 1988. Morgan Kaufmann.

[75] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, San Mateo, CA, 1990.

[76] T. Feo and M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, (8):67–71, 1989.

[77] T. Feo and M. Resende. Greedy randomized adaptive search procedures. Technical report, AT&T Bell Laboratories, Murray Hill, NJ 07974, February 1994.

[78] H. Fischer and K. Ritter. An asynchronous parallel Newton method. *Mathematical Programming*, 42:363–374, 1988.

[79] R. Fletcher. *Pratical Methods of Optimization.* John Wiley and Sons, New York, 1987.

[80] R. Fletcher and M.J.D. Powell. A rapidly convergent descent method for minimization. *Computer Journal*, 6:163–168, 1963.

[81] N. J. Fliege. *Multirate Digital Signal Processing.* John Wiley and Sons, 1994.

[82] 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.

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

[84] C. A. Floudas and V. Visweswaran. Quadratic optimization. In Reiner Horst and P. M. Pardalos, editors, *Handbook of Global Optimization*, pages 217–269. Kluwer Academic Publishers, 1995.

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

[86] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, San Francisco, CA, 1979.

[87] B. Gavish and S. L. Hantler. An algorithm for optimal route selection in SNA networks. *IEEE Transactions on Communications*, pages 1154–1161, 1983.

[88] 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.

[89] M. R. Genesereth and N. J. Nilsson. *Logical Foundation of Artificial Intelligence.* Morgan Kaufmann, 1987.

[90] I. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proc. of the 11th National Conf. on Artificial Intelligence*, pages 28–33, Washington, DC, 1993.

[91] A. M. Geoffrion. Lagrangian relaxation and its uses in integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

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

[93] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.

[94] W. L. Goffe, G. D. Ferrier, and J. Rogers. Global optimization of statistical functions with simulated annealing. *Journal of Econometrics*, 60:65–99, 1994.

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

[96] R. P. Gorman and T. J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[97] D. Gorse and A. Shepherd. Adding stochastic search to conjugate gradient algorithms. In *Proc. of the 3rd Intl Conf on Parallel Applications in Statistics and Economics*, Prague, Tifkarenfke Zacody, 1992.

[98] S. M. Le Grand and Jr. K. M. Merz. The application of the genetic algorithm to the minimization of potential energy functions. *Journal of Global Optimization*, 3:49–66, 1993.

[99] A. O. Griewank. Generalized descent for global optimization. *Journal of Optimization Theory and Applications*, 34:11–39, 1981.

[100] J. Gu. *Parallel Algorithms and Architectures for Very Fast AI Search.* PhD thesis, Dept. of Computer Science, University of Utah, August 1989.

[101] J. Gu. How to solve very large-scale satisfiability (VLSS) problems. Technical Report UCECE-TR-90-002, Univ. of Calgary, Canada, October 1990.

[102] J. Gu. Efficient local search for very large-scale satisfiability problems. *SIGART Bulletin*, 3(1):8–12, January 1992.

[103] J. Gu. On optimizing a search problem. In N. G. Bourbakis, editor, *Artificial Intelligence Methods and Applications.* World Scientific Publishers, 1992.

[104] J. Gu. The UniSAT problem models (appendix). *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(8):865, Aug 1992.

[105] J. Gu. Local search for satisfiability (SAT) problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(4):1108–1129, 1993.

[106] J. Gu. Global optimization for satisfiability (SAT) problems. *IEEE Trans. on Knowledge and Data Engineering*, 6(3):361–381, Jun 1994.

[107] J. Gu. *Constraint-Based Search.* Cambridge University Press, New York, to appear.

[108] J. Gu and Q.-P. Gu. Average time complexities of several local search algorithms for the satisfiability problem (sat). Technical Report UCECE-TR-91-004, Univ. of Calgary, Canada, 1991.

[109] J. Gu and Q.-P. Gu. Average time complexity of the sat1.3 algorithm. Technical report, Tech. Rep., Univ. of Calgary, Canada, 1992.

[110] J. Gu and W. Wang. A novel discrete relaxation architecture. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(8):857–865, August 1992.

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

[112] P. Hansen and R. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[113] S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In D. Z. Anderson, editor, *Proc. Neural Information Processing Systems*, pages 177–185, New York, 1988. American Inst. of Physics.

[114] S. A. Harp, T. Samad, and A. Guha. Designing application-specific neural network using the genetic algorithm. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 447–454. Morgan Kaufmann, San Mateo, CA, 1990.

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

[116] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 6:62–88, 1971.

[117] M. R. Hestenes. *Conjugate Direction Methods in Optimization.* Springer-Verlag, 1980.

[118] D. Hilbert. Ueber stetige abildung einer linie auf ein flachenstuck. *Mathematische Annalen*, 38:459–460, 1891.

[119] A. C. Hindmarsh. ODEPACK, a systematized collection of ode solvers. In R. S. Stepleman et al., editor, *scientific computing*, pages 55–64. north-holland, amsterdam, 1983.

[120] A. C. Hindmarsh. ODEPACK, a systematized collection of ODE solvers. In R. S. Stepleman, editor, *Scientific Computing*, pages 55–64. North Holland, Amsterdam, 1983.

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

[122] J. N. Hooker. Resolution vs. cutting plane solution of inference problems: some computational results. *Operations Research Letters*, 7:1–7, 1988.

[123] B. R. Horng and A. N. Willon, 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.

[124] R. Horst. A general class of branch-and-bound methods in global optimization with some new approaches for concave minimization. *Journal of Optimization Theory and Applications*, 51:271–291, 1986.

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

[126] Reiner Horst and P. M. Pardalos, editors. *Handbook of Global Optimization.* Kluwer Academic Publishers, 1995.

[127] E. C. Housos and O. Wing. Parallel nonlinear minimization by conjugate gradients. In *Proc. Intl. Conf. on Parallel Processing*, pages 157–158, Los Alamitos, CA, 1980.

[128] J. Hwang, S. Lay, M. Maechler, R. D. Martin, and J. Schimert. Regression modeling in back-propagation and projection pursuit learning. *IEEE Transactions on Neural Networks*, 5(3):1–24, May 1994.

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

[130] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4):195–308, 1988.

[131] A. K. Jain and J. Mao. Artificial neural networks: A tutorial. *IEEE Computer*, 29:31–44, March 1996.

[132] Y. Jiang, H. Kautz, and B. Selman. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In *Proc. of 1st Int'l Joint Workshop on Artificial Intelligence and Operations Research*, Timberline, Oregon, 1995.

[133] J. D. Johnston. A filter family designed for use in quadrature mirror filter banks. In *Proc. of Int'l Conf. on ASSP*, pages 291–294, 1980.

[134] J. D. Johnston. A filter family designed for use in quadrature mirror filter banks. *Proc. of Int'l Conf. on ASSP*, pages 291–294, 1980.

[135] 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.

[136] S. Joy, J. Mitchell, and B. Borchers. A branch and cut algorithm for MAX-SAT and weighted MAX-SAT. In *Satisfiability Problem: Theory and Applications. DIMACS Series on Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society (to appear), 1997.

[137] J. E. Dennis Jr. and V. Torczon. Direct search methods on parallel computers. *SIAM Journal on Optimization*, 1:448–474, 1991.

[138] 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.

[139] A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.

[140] A. H. G. Rinnooy Kan and G. T. Timmer. Stochastic methods for global optimization. *American Journal of Mathematical and Management Sciences*, 4:7–40, 1984.

[141] R. M. Karp. The probabilistic analysis of some combinatorial search algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Direction and Recent Result*, pages 1–19. Academic Press, New York, NY, 1976.

[142] R. M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, NY, 1992.

[143] J. A. Kinsella. Comparison and evaluation of variants of the conjugate gradient methods for efficient learning in feed-forward neural networks with backward error propagation. *Neural Networks*, 3:27–35, 1992.

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

[145] J. Klincewicz. Avoiding local optima in the p-hub location problem using tabu search and GRASP. *Annals of Operations Research*, (40):283–302, 1992.

[146] Donald E. Knuth. *The Art of Computer Programming, Volume II.* Addison-Wesley, Reading, MA, 1981.

[147] 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.

[148] 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.

[149] R. E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[150] V. Kumar and L. Kanal. The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. *Search in Artificial Intelligence (eds L. Kanal, V. Kumar)*, pages 1–27, 1988.

[151] P. J. van Laarhoven. Parallel variable metric methods for unconstrained optimization. *Mathematical Programming*, 33:68–81, 1985.

[152] R. W. Beckerand G. W. Lago. A global optimization algorithm. In *Proc. of the 8th Allerton Conf. on Circuits and Systems Theory*, pages 3–12, Monticello, Illinois, 1970.

[153] L. S. Lasdon, A. D. Warren, A. Jain, and M. Ratner. Design and testing a generalized reduced gradient code for nonlinear programming. *ACM Trans. Math. Software*, 4:34–50, 1978.

[154] E. L. Lawler and D. W. Wood. Branch and bound methods: A survey. *Operations Research*, 14:699–719, 1966.

[155] Y. Le Cun. A theoretical framework for back-propagation. In D. Touretzky, G. E. Hinton, and T. J. Sejnowski, editors, *Proc. Connectionist Models Summer School*, pages 21–28, Palo Alto, CA, 1988. Morgan Kaufmann.

[156] A. V. Levy, A. Montalvo, S. Gomez, and A. Calderon. *Topics in Global Optimization, Lecture Notes in Mathematics No. 909.* Springer-Verlag, New York, 1981.

[157] 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.

[158] F. A. Lootsma. State-of-the-art in parallel unconstrained optimization. In M. Feilmeir, E. Joubert, and V. Schendel, editors, *Parallel Computing 85*, pages 157–163. North-Holland, Amsterdam, Holland, 1986.

[159] S. Lucidi and M. Piccioni. Random tunneling by means of acceptance-rejection sampling for global optimization. *Journal of Optimization Theory and Applications*, 62:255–277, 1989.

[160] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Wiley, 1973.

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

[162] F. Maffioli. The complexity of combinatorial optimization algorithms and the challenge of heuristics. *Combinatorial Optimization*, pages 107–128, 1979.

[163] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

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

[165] Z. Michalewicz. Genetic algorithsm, numerical optimization and constraints. In L. J. Eshelman, editor, *Proc. Int'l Conf. on Genetic Algorithms*, pages 98–108. Morgan Kaufmann, 1995.

[166] S. Minton, M. D. Johnson, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, Dec 1992.

[167] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. of the 10th National Conf. on Artificial Intelligence*, pages 459–465, 1992.

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

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

[170] M. S. Moller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–534, 1993.

[171] D. J. Montana and L. Davis. Training feedforward networks using genetic algorithms. In *Proc. Int. Joint Conf. Artificial Intelligence*, pages 762–767, San Mateo, CA, 1989.

[172] 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.

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

[174] S. G. Nash and A. Sofer. Block truncated-Newton methods for parallel optimization. *Mathematical Programming*, 45:529–546, 1989.

[175] S. G. Nash and A. Sofer. A general-purpose parallel algorithm for unconstrained optimization. *SIAM Journal on Optimization*, 1:530–547, 1991.

[176] D. S. Nau, V. Kumar, and L. Kanal. General branch and bound, and its relation to A* and AO*. *Artificial Intelligence*, 23:29–58, 1984.

[177] 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.

[178] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[179] O. Nerrand, P. Roussel-Ragot, D. Urbani, L. Personnaz, and G. Dreyfus. Training recurrent neural networks: why and how? An illustration in dynamical process modeling. *IEEE Transactions on Neural Networks*, 5(2):178–184, March 1994.

[180] T. Q. Nguyen. Digital filter bank design quadratic-constrained formulation. *IEEE Trans. on Signal Processing*, 43(9):2103–2108, September 1995.

[181] T. Q. Nguyen and P. P. Vaidyanathan. Two-channel perfect-reconstruction FIR QMF structures which yield linear-phase analysis and synthesis filters. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 37(5):676–690, May 1989.

[182] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, 1970.

[183] E. R. Panier and A. L. Tits. A superlinearly convergence feasible method for the solution of inequality constrained optimization problems. *SIAM Journal on control and Optimization*, 25(4):934–950, 1987.

[184] E. R. Panier and A. L. Tits. On combining feasibility, descent and superlinear convergence in inequlaity constrained optimization. *Mathematical Programming*, 59(2):261–276, 1993.

[185] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization.* Prentice-Hall, 1982.

[186] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*, pages 55–58. Prentice-Hall, 1982.

[187] P. M. Pardalos. *Complexity in numerical optimization.* World Scientific, Singapore and River Edge, N.J., 1993.

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

[189] D. B. Parker. Optimal algorithms for adaptive networks: Second order back propagation, second order direct propagation, and second order Hebbian learning. In *Proc. Int'l Conf. on Neural Networks*, volume II, pages 593–600. IEEE, 1987.

[190] R. G. Parker and R. L. Rardin. *Discrete Optimization.* Academic Press, Inc., San Diego, CA, 1988.

[191] A. G. Parlos, B. Fernandez, A. F. Atiya, J. Muthusami, and W. Tsai. An accelerated learning algorithm for multilayer perceptron networks. *IEEE Transactions on Neural Networks*, 5(3):493–497, May 1994.

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

[193] G. Peano. Sur une courbe, qui remplit toute une aire plaine. *Mathematische Annalen*, 36:157–160, 1890.

[194] B. A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: a survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, September 1995.

[195] M. Piccioni. A combined multistart-annealing algorithm for continuous global optimization. Technical Report 87-45, Systems and Research Center, The University of Maryland, College Park MD, 1987.

[196] F. J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, 1987.

[197] F. J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, 1987.

[198] V. W. Porto, D. B. Fogel, and L. J. Fogel. Alternative neural network training methods. *IEEE Expert*, pages 16–22, June 1995.

[199] D. Powell and M. M. Skolnick. Using genetic algorithms in engineering design optimization with nonlinear constraints. In S. Forrest, editor, *Proc. of the Fifth Int'l Conf. on Genetic Algorithms*, pages 424–431. Morgan Kaufmann, 1993.

[200] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 7:155–162, 1964.

[201] B. N. Pschenichny. Algorithms for general problems of mathematical programming. *Kibernetica*, 6:120–125, 1970.

[202] B.N. Pshenichnyj. *The Linearization Method for Constrained Optimization.* Springer-Verlag, 1994.

[203] P. W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.

[204] M.G.C. Resende and T. Feo. A GRASP for satisfiability. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26, pages 499–520. DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996.

[205] M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In *Satisfiability Problem: Theory and Applications. DIMACS Series on Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, 1997.

[206] A. J. Robinson. *Dynamic Error Propagation Networks.* PhD thesis, Cambridge University, 1989.

[207] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.*, pages 23–41, 1965.

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

[209] D. E. Rumelhart, G.E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition. In *Parallel distributed processing, Vol. 1*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[210] D. E. Rumelhart and J. L. McClelland, editors. *Parallel distributed processing, vol. 1.* MIT Press, Cambridge, MA, 1986.

[211] 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.

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

[213] R. S. Scalero and N. Tepedelenlioglu. A fast new algorithm for training feedforward neural networks. *IEEE Transactions on Signal Processing*, 40(1):202–210, January 1992.

[214] 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.

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

[216] I. P. Schagen. Internal modeling of objective functions for global optimization. *Journal of Optimization Theory and Applications*, 51(2), 1986.

[217] R. B. Schnabel. Parallel computing in optimization. In K. Schittkowsky, editor, *Computational Mathematical Programming*, pages 357–381. Springer-Verlag, Berlin, Germany, 1985.

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

[219] R. Sebastiani. Applying GSAT to non-clausal formulas. *Journal of Artificial Intelligence Research*, 1:309–314, 1994.

[220] T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1(1):145–168, February 1987.

[221] B. Selman, 1995. private communication.

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

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

[224] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. of the 12th National Conf. on Artificial Intelligence*, pages 337–343, Seattle, WA, 1994.

[225] B. Selman and H. A. Kautz. An empirical study of greedy local search for satisfiability testing. In *Proc. of the 11th National Conf. on Artificial Intelligence*, pages 46–51, Washington, DC, 1993.

[226] 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.

[227] Y. Shang and B. W. Wah. Global optimization for neural network training. *IEEE Computer*, 29:45–54, March 1996.

[228] Y. Shang and B. W. Wah. A discrete Lagrangian-based global-search method for solving satisfiability problems. *J. of Global Optimization*, (accepted to appear) 1997.

[229] Y. Shang and B. W. Wah. Discrete Lagrangian-based search for solving MAX-SAT problems. In *Proc. Int'l Joint Conf. on Artificial Intelligence.* IJCAI, (accepted to appear) Aug. 1997.

[230] S. Shekhar and M. B. Amin. Generalization by neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):177–185, April 1992.

[231] J.-J. Shyu and Y.-C. Lin. A new approach to the design of discrete coefficient FIR digital filters. *IEEE Transactions on Signal Processing*, 41(1), 1 1995.

[232] D. M. Simmons. *Nonlinear Programming for Operations Research.* Prentice-Hall, Englewood Cliffs, NJ, 1975.

[233] S. Singhal and L. Wu. Training multilayer perceptrons with the extended Kalman algorithm. In D. Z. Anderson, editor, *Proc. Neural Information Processing Systems*, pages 133–140, New York, 1988. American Inst. of Physics.

[234] 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.

[235] S. Smith, E. Eskow, and R. B. Schnabel. Adaptive, asynchronous stochastic global optimization for sequential and parallel computation. In T. Coleman and Y. Li, editors, *Large Scale Numerical Optimization*, pages 207–227. SIAM, Philadelphia, 1990.

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

[237] 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.

[238] 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.

[239] 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.

[240] Y. Song. Convergence and comparisons of waveform relaxation methods for initial value problems of linear ODE systems. *Computing*, 50:337–352, 1993.

[241] R. Sosič and J. Gu. A polynomial time algorithm for the n-queens problem. *SIGART Bulletin*, 1(3):7–11, October 1990.

[242] R. Sosič and J. Gu. 3,000,000 queens in less than one minute. *SIGART Bulletin*, 2(2):22–24, April 1991.

[243] 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.

[244] 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.

[245] J. Sprave. Linear neighborhood evolution stategies. In *Proc. of the third Annual Conf. on Evolutionary Programming*, San Diego, CA, 1994. World Scientific.

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

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

[248] C. Sutti. Nongradient minimization methods for parallel processing computers, Part 1. *Journal of Optimization Theory and Applications*, 39:465–474, 1983.

[249] Z. Tang and G. Koehler. Deterministic global optimal FNN training algorithms. *Neural Networks*, 7(2):301–311, 1994.

[250] A. Törn. Global optimization as a combination of global and local search. *Gothenburg Business Adm. Studies*, 17:191–206, 1973.

[251] A. Törn. Cluster analysis using seed points and density determined hyperspheres as an aid to global optimization. *IEEE Trans. Syst. Men and Cybernetics*, 7:610–616, 1977.

[252] A. Törn and S. Viitanen. Topographical global optimization. In C. A. Floudas and P. M. Pardalos, editors, *Recent Advances in Global Optimization*, pages 385–398. Princeton University Press, 1992.

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

[254] A. Torn and A. Zilinskas. *Global Optimization.* Springer-Verlag, 1987.

[255] E. Tsang. *Foundations of Constraint satisfaction.* Academic Press, San Diego, CA, 1993.

[256] 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.

[257] P. P. Vaidyanathan. Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial. *Proc. of the IEEE*, 78(1):56–93, January 1990.

[258] P. P. Vaidyanathan. *Multirate Systems and Filter Banks.* Printice-Hall Inc., 1993.

[259] D. Vanderbilt and S. G. Louie. A Monte Carlo simulated annealing approach to optimization over continuous variables. *Journal of Computational Physics*, 56:259–271, 1984.

[260] 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.

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

[262] 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.

[263] 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, *Proc. DIMACS Workshop on Satisfiability Problem: Theory and Applications.* American Mathematical Society, March 1996.

[264] B. W. Wah and Y. Shang. A new global optimization method for neural network training. In *Proc. Int'l Conf. on Neural Networks*, volume Plenary, Panel and Special Sessions, pages 7–11, Washington, DC, June 1996. (Plenary Speech) IEEE.

[265] B. W. Wah, Y. Shang, T. Wang, and T. Yu. Global optimization design of QMF filter banks. In *Proc. IEEE Midwest Symposium on Circuits and Systems*, Iowa City, Iowa, (accepted to appear) Aug. 1996. IEEE.

[266] B. W. Wah, Y. Shang, T. Wang, and T. Yu. Global optimization of QMF filter-bank design using Novel. In *Proc. Int'l Conf. on Acoustics, Speech and Signal Processing*, volume 3, pages 2081–2084. IEEE, April 1997.

[267] 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*. IEEE, (accepted to appear) July 1997.

[268] B. W. Wah, T. Wang, Y. Shang, and Z. Wu. Improving the performance of weighted Lagrange-multiple methods for constrained nonlinear optimization. In *Proc. 9th Int'l Conf. on Tools for Artificial Intelligence*. IEEE, (invited) Nov. 1997.

[269] G. R. Walsh. *Methods of Optimization*. John Wiley and Sons, 1975.

[270] F. H. Walters, L. R. Parker, S. L. Morgan, and S. N. Deming. *Sequential simplex optimization*. CRC Press, 1991.

[271] T. Wang and B. W. Wah. Handling inequality constraints in continuous nonlinear global optimization. *Journal of Integrated Design and Process Science*, 1, (accepted to appear) 1997.

[272] Tao Wang and B. W. Wah. Handling inequality constraints in continuous nonlinear global optimization. In *Proc. 2nd World Conference on Integrated Design and Process Technology*, volume 1, pages 267–274, Austin, TX, December 1996.

[273] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computation*, 14:347–361, 1990.

[274] R. J. Williams and J. Peng. An efficient gradient-based algorithm for on-line training of recurrent networks trajectories. *Neural Computation*, 2(4):490–501, 1990.

[275] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[276] P. Wolfe. Methods of nonlinear programming. In J. Abadie, editor, *Nonlinear Programming*, pages 97–131. John Wiley, New York, 1967.

[277] P. Wolfe. Convergence conditions for ascent methods. *SIAM Rev.*, 11:226–235, 1969.

[278] J. W. Woods, editor. *Subband Image Coding*. Kluwer Academic Publishers, 1991.

[279] Ting Yu. *QMF Filter Bank Design Using Nonlinear Optimization*. M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 1997.

[280] X. Yu, G. Chen, and S. Cheng. Dynamic learning rate optimization of the backpropagation algorithm. *IEEE Transactions on Neural Networks*, 6(3):669–677, May 1995.

[281] Z. B. Zabinsky and R. L. Smith. Pure adaptive search in global optimization. *Mathematical Programming*, 53:323–338, 1992.

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

[283] S. Zhang and A. G. Constantinides. Lagrange programming neural networks. *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 39(7):441–452, 1992.

VITA

Yi Shang received his B.S. degree in computer science from the University of Science and Technology of China, Hefei, China, in 1988, and his M.S. degree in computer science from the Institute of Computing Technology, Academia Sinica, Beijing, China in 1991. He is currently a candidate for Doctor of Philosophy in Computer Science at The University of Illinois at Urbana-Champaign. After completing his doctoral degree, he will become an assistant professor in the Department of Computer Engineering and Computer Science, University of Missouri, Columbia, Missouri.

His research interests include optimization, machine learning, artificial intelligence, and distributed and parallel systems.