THE DISCRETE LAGRANGIAN THEORY AND ITS APPLICATION TO SOLVE NONLINEAR DISCRETE CONSTRAINED OPTIMIZATION PROBLEMS

BY

ZHE WU

B.E., University of Science & Technology of China, 1996

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1998

Urbana, Illinois

Abstract

In this research we present new results on discrete Lagrangian methods (DLM) and extend our previous (incomplete and highly simplified) theory on the method. Our proposed method forms a strong mathematical foundation for solving general nonlinear discrete optimization problems. Specifically, we show for continuous Lagrangian methods the relationship among local minimal solutions satisfying constraints, solutions found by the first-order necessary and second-order sufficient conditions, and saddle points. Since there is no corresponding definition of gradients in discrete space, we propose a new vector-based definition of gradient, develop first-order conditions similar to those in continuous space, propose a heuristic method to find saddle points, and show the relationship between saddle points and local minimal solutions satisfying constraints. We then show, when all the constraint functions are non-negative, that the set of saddle points is the same as the set of local minimal points satisfying constraints. Our formal results for solving discrete problems are stronger than the continuous counterparts because saddle points in discrete space are equivalent to local minima satisfying constraints, whereas local minimal solutions satisfying constraints in continuous space do not necessarily satisfy the first- and second-order conditions. To verify our method, we apply it to solve discrete benchmark problems, design multiplierless QMF filter banks, and solve some mixed nonlinear integer optimization benchmark problems. We also study methods to speed up convergence while maintaining solution quality. Our experimental results demonstrate that DLM is both effective and efficient.

To my wife, my parents, my brother and my friends

Acknowledgments

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

I would like to thank all the members in our research group for providing crucial comments on the work and for providing a congenial environment for me to work in.

Special thanks go to Dr. Yi Shang who first tried applying DLM to solve SAT and MAX-SAT problems. I would like to thank Mr. Tao Wang for his research in dynamic weight adaptation, benchmark problems, and the design of Novel.

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

Table of Contents

Chapter

List of Tables

List of Figures

Chapter 1

Introduction

Many applications in engineering, decision science and operations research can be formulated as nonlinear discrete optimization problems, whose variables are restricted to discrete values and whose objective and constraint functions are nonlinear.

1.1 Problem Definition

The general formulation of a nonlinear discrete constrained optimization problem is as follows:

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

where $f(x)$ is the objective function, $g(x) = [g_1(x), \ldots, g_k(x)]^T$ is a k-component vector of inequality constraints, $h(x) = [h_1(x), \ldots, h_m(x)]^T$ is an m-component vector of equality constraints, and x is a set of discrete variables. The objective and constraint functions are nonlinear and can be either discrete or continuous. Without loss of generality, we consider only minimization problems here, knowing that maximization problems can be converted to minimization problems by negating their objectives.

1.2 Basic Definitions

The possible solutions for constrained minimization problems are local minima that satisfy all the constraints defined in the variable space. To formally specify the solutions we are interested to find in (1.1), we need to define neighborhoods in discrete space.

Definition 1. $\mathcal{N}(x)$, the *neighborhood* of point x in discrete space, is the set of all points y that satisfy a user-defined property $P(x, y)$; that is, given point x, its neighborhood is the set of all points y that make $P(x, y)$ $TRUE$.

For example, in $\{0, 1\}^n$ space in which variables are represented as a vector of binary elements, each in the set $\{0,\ 1\}$, the neighborhood of x can be defined as points whose Hamming distance between x and y is less than 2. In modulo-integer space in which variables are vectors of integer elements, each in the set $\{0, 1, \ldots, k-1\}$, $P(x, y)$ can be defined to be $TRUE$ when $\mod(y_1 - x_1, k) + \ldots + \mod(y_n - x_n, k)$ is less than 2. In general, the choice of property function P is application-dependent and may affect the efficiency of the search. Its effect is similar to the choice of the state-space operator in combinatorial searches.

A local minimum solution to (1.1) can be defined with respect to its neighborhood.

Definition 2. A point x is a *local minimum subject to constraints* if and only it satisfies the following two properties:

- x is a feasible point, implying that x satisfies all the constraints;
- For all $x' \in \mathcal{N}(x)$ and x' is also feasible, $f(x') \geq f(x)$ holds true.

A special case in the second condition of the definition needs to be emphasized. When all neighboring points of x are infeasible, x is still a local minimum subject to constraints, since it is the only feasible point surrounded by infeasible points.

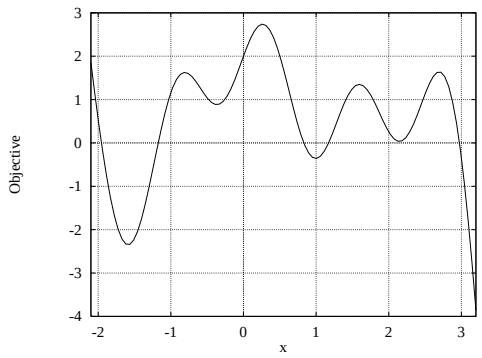

Figure 1.1: The objective function defined in (1.2)

.

Example 1. Consider the following one-dimensional nonlinear discrete constrained optimization problem whose constraints are satisfied at integer points between -2 and 3.

$$\begin{aligned} \text{minimize} \quad & f(x) = 2 - 0.4x - 2.0x^2 + 0.75x^3 + 0.4x^4 - 0.15x^5 + \sin(5x) \\ \text{subject to} \quad & h(x) = 0 \\ \text{where} \quad & h(x) = \begin{cases} \sin(\pi x) & \text{if } -2 \le x \le 3 \\ 1 & \text{otherwise} \end{cases} \end{aligned} \tag{1.2}$$

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

Applying the definitions on the example, the neighborhood of x can be defined as $\{x-1, x+1\}$, and $x = -2, 1, 3$ are all local minima subject to constraint in (1.2), since $x = -3$ is infeasible, $f(-1) > f(-2)$, $f(0) > f(1)$, $f(2) > f(1)$, $f(2) > f(3)$, and $x = 4$ is infeasible. Out of the six points in the feasible region, the global minimum is at $x = 1$ where $f(1) = -0.359$. ■

In short, our goal in solving (1.1) is to find the best local minima that satisfy all the constraints. It is usually very difficult to find the global minima for many of these problems because they are NP hard [15] and have extremely large search spaces bounded by highly nonlinear functions.

1.3 Previous Work

There exist a large number of methods for solving the nonlinear discrete constrained optimization problem defined in (1.1), such as simulated annealing (SA) [28], genetic algorithm (GA) [19], tabu

search [18], gradient descent [1], penalty methods, Hopfield networks [22], and branch-and-bound search [16]. Unfortunately, they all have some shortcomings. SA and GA are not effective for finding feasible solutions because random perturbations may not be able to locate feasible regions that satisfy all the defined constraints. Tabu search tries to avoid perturbing the same set of variables in a search but does not address the problem of constraint satisfaction. Likewise, gradient descent, such as Newton's method, is good for solving unconstrained problems but can get stuck easily in local minima in the objective space. On the other hand, penalty methods attempt to penalize constraints that are hard to satisfy using penalties. The concept they implement is similar to that in Hopfield networks that model their energy functions as weighted sums of objective and constraints. The difficulty, however, is in setting the relative magnitudes of the penalties (or weights) that are critical in convergence. In a general penalty-based method, penalties should be allowed to change and adapt to the degree of violation of each individual constraint. Finally, branch-and-bound search has been adapted to work in nonlinear constrained problems by finding lower bounds using linearized constraints. However, the method does not work for general nonlinear constraints.

Previous research indicates that any heuristic method for solving nonlinear discrete constrained optimization problems must have schemes that examine violated constraints and force them into satisfaction. Although penalty-based methods seem to be the most viable, the current literature does not have any dynamic method with a formal mathematical foundation that can adapt penalties to degrees of violation.

1.4 Outline of this Thesis

In this thesis, we present an extension of continuous Lagrangian methods and show how it can be extended to work in discrete space. The nontrivial extension requires the development of a new definition of gradient in discrete space and the proof of convergence of the search process. In Chapter 2, we summarize the theory of continuous Lagrangian methods and describe their limitations. Chapter 3 presents the mathematical theory behind our proposed discrete Lagrangian method (DLM) and presents an enhanced DLM by including an adaptive weight that balances the relative importance between the objective and the constraints. Finally, Chapter 4-6 show

experimental results on applying DLM to solve discrete optimization problems, multiplierless filter bank design problems, and mixed integer programming problems, respectively.

Chapter 2

Lagrangian Methods for Continuous Problems

Traditionally, Lagrangian methods were developed for solving continuous optimization problems. In this section, we examine the relationship among the solution spaces of these methods.

Our results here extend our previous work [43] that did not study these relationships. The relationships are important because they show that some local minima subject to constraints cannot be found by existing continuous Lagrangian methods. We show in Chapter 3 that it is possible to overcome this problem in Lagrangian methods that work in discrete solution space.

2.1 Basic Definitions

A general *continuous equality-constrained optimization problem* is formulated as follows:

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad h(x) = 0 \qquad x = (x_1, x_2, \ldots, x_n) \end{aligned} \tag{2.1}$$

where $h(x) = [h_1(x), \ldots, h_m(x)]^T$ is an m-component vector, $f(x)$ is the objective function, and x is a vector of continuous variables. Both $h(x)$ and $f(x)$ are nonlinear continuous functions.

In continuous space, the solutions to (2.1) are also called local minima subject to constraints. However, their definition is slightly different from that in discrete space defined in Definition 2.

Definition 3. A point x in continuous space is a *local minimum subject to constraints* [33] if and only if there exists a small $\varepsilon > 0$ such that for all x' that satisfy $|x' - x| < \varepsilon$ and that x' is also a feasible point, $f(x') \geq f(x)$ holds true.

The following two definitions define the Lagrangian function as a weighted sum of the objective and the constraints, and the saddle point as a point in the Lagrangian space where the Lagrange multipliers are at their local maxima and the objective function is at a local minimum.

Definition 4. The *Lagrangian function* of the general continuous constrained optimization problem defined in (2.1) is $L(x, \lambda) = f(x) + \lambda^T h(x)$, where λ is a vector of Lagrange multipliers.

Definition 5. A *saddle-point* (x^*, λ^*) of $L(x, \lambda)$ is defined as one that satisfies the following:

$$L(x^*, \lambda) \leq L(x^*, \lambda^*) \leq L(x, \lambda^*) \tag{2.2}$$

for all (x^*, λ) and all (x, λ^*) sufficiently close to (x^*, λ^*).

In general, there is no efficient method to find saddle points in continuous space.

2.2 First-Order Necessary Conditions and Methods for Continuous Problems

There are various continuous Lagrangian methods that can be used to locate local minima subject to constraints. They are all based on solving two differential equations based on the first-order necessary conditions. To state these conditions, we first define the concept of regular points.

Definition 6. A point x satisfying constraints $h(x) = 0$ is said to be a *regular point* [33] of the constraints if gradient vectors $\nabla h_1(x), \cdots, \nabla h_m(x)$ are linearly independent.

Theorem 1. (*First-Order Necessary Conditions for Continuous Problems*) Let x be a local extremum point of $f(x)$ subject to constraints $h(x) = 0$. Further, assume that $x = (x_1, \ldots, x_n)$ is a regular point of these constraints. Then there exists $\lambda \in E^m$ such that

$$\nabla_x f(x) + \lambda^T \nabla_x h(x) = 0 \tag{2.3}$$

Based on the definition of Lagrangian function, the necessary conditions can be expressed as follows:

$$\begin{aligned} \nabla_x L(x,\lambda) &= 0 \\ \nabla_\lambda L(x,\lambda) &= 0 \end{aligned} \tag{2.4}$$

Proof. Since the proof is well known in the literature [33], we only outline its three major parts here. (We show later that there is no similar proof for problems in discrete space.) First, assuming that $h(x) = (h_1(x), \ldots, h_m(x))$ and $m \leq n$, we use the definition of regular point in Definition 6 and the implicit-function theorem to eliminate the constraints by solving them for the first m variables in terms of the remaining $n - m$ variables. Second, we substitute the first m variables in terms of the other $n - m$ ones into the objective function. Using the local extremum's property ($\frac{\partial f}{\partial x} = 0$) on the new function, we prove $\nabla_x L(x,\lambda) = 0$ for the first m variables. Finally, we apply the chain rule to prove that $\nabla_x L(x,\lambda) = 0$ is true for the remaining $n - m$ variables. ■

Based on the first-order necessary conditions in continuous space, there are a number of methods for solving constrained optimization problems. These include the first-order method, Newton's method, modified Newton's methods, quasi-Newton methods [33], and sequential quadratic programming [24]. Among these methods, the most widely used is the first-order method represented as an iterative process:

$$x^{k+1} = x^k - \alpha_k \nabla L_x(x^k, \lambda^k)^T \tag{2.5}$$

$$\lambda^{k+1} = \lambda^k + \alpha_k h(x^k) \tag{2.6}$$

where α_k is a step-size parameter to be determined.

Intuitively, these two equations represent two counter-acting forces to resolve constraints and find high-quality solutions. When any of the constraints is violated, its degree of violation is used in (2.6) to increase the corresponding Lagrange multiplier in order to increase the penalty on the unsatisfied constraint and to force it into satisfaction. When the constraint is satisfied, its Lagrange multiplier stops to grow. Hence, (2.6) performs ascents in the Lagrange-multiplier space until each Lagrange multiplier is at its maximum. In contrast, (2.5) performs descents in the objective space when all the constraints are satisfied and stops at an extremum in that space. By using a combination of simultaneous ascents and descents, an equilibrium is eventually reached in which all the constraints are satisfied and the objective is at a local extremum.

To guarantee that the equilibrium point is a local minimum, there are second-order sufficient conditions to make the solution a strict relative minimum subject to constraints. Since it is irrelevant to the Lagrangian method for solving discrete problems, we do not discuss them here.

It may seem from (2.2) that saddle points are the same as points satisfying the first-order conditions. However, we show in the next section that they are actually not equivalent.

2.3 Relationships among Solution Spaces

In this section, we prove six lemmas to show the relationships among the three solution spaces: the set of local minima subject to constraints (**A**), the set of solutions satisfying the first-order necessary and second-order sufficient conditions (**B**), and the set of saddle points (**C**).

The following two lemmas show that **C** is a proper subset of **A**. Hence, any method that aims at locating saddle points are theoretically deficient because they may miss some local minima subject to constraints.

Lemma 1. A saddle point is a local minimum subject to constraints, *i.e.*, $x \in \mathbf{C} \Rightarrow x \in \mathbf{A}$.

Proof. Let (x^*, λ^*) be a saddle point. From the definition of saddle points, we know that $L(x^*, \lambda) \leq L(x^*, \lambda^*)$ is true for any λ sufficiently close to λ^*, and that $L(x^*, \lambda^*) \leq L(x, \lambda^*)$ is true for any x sufficiently close to x^*. Hence, all the constraints at the saddle point must be satisfied. For any x sufficiently close to x^* such that x is also feasible, $f(x^*) = L(x^*, \lambda^*) \leq L(x, \lambda^*) = f(x)$. (Infeasible points do not have to be considered according to the definition of local minima subject to constraints.) Hence, x^* is a local minimum subject to constraints. ■

Lemma 2. A local minimum x^* subject to constraints is not necessarily a saddle point, *i.e.*, $x \in \mathbf{A} \not\Rightarrow x \in \mathbf{C}$.

Proof. The proof is done by showing the following counter example that it is not possible to find a corresponding λ^* to make (x^*, λ^*) a saddle point:

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

Obviously, $x^* = 0$ is the only feasible point to this problem and also a local minimum subject to constraints from the definition. Assuming λ^* such that (x^*, λ^*) is a saddle point, it follows that for all (x, λ) sufficiently close to (x^*, λ^*), $L(x^*, \lambda^*) \leq L(x, \lambda^*)$. Then there must exist a small $\alpha > 0$ such that

$$\begin{aligned} 0 &= L(x^*, \lambda^*) \leq L(\alpha, \lambda^*) = -\alpha^2 + \lambda^* \alpha^5 \\ 0 &= L(x^*, \lambda^*) \leq L(-\alpha, \lambda^*) = -\alpha^2 - \lambda^* \alpha^5 \end{aligned}$$

These lead to a contradiction that $-2\alpha^2 \geq 0$ when the above two equations are added together. ■

The following two lemmas show that **B** is a proper subset of **A**. Hence, any search method based on the first-order necessary conditions may not find all the local minima subject to constraints.

Lemma 3. A solution satisfying the first-order necessary and second-order sufficient conditions must also be a local minimum subject to constraints, *i.e.*, $x \in \mathbf{B} \Rightarrow x \in \mathbf{A}$.

Proof. Please refer to the proof in the reference [33]. ■

Lemma 4. A local minimum x^* subject to constraints may not satisfy the first-order necessary conditions, *i.e.*, $x \in \mathbf{A} \not\Rightarrow x \in \mathbf{B}$.

Proof. The proof is done by showing a counter example that it is impossible to find a corresponding λ to make (x^*, λ^*) a solution to the first-order necessary conditions.

$$\begin{aligned} \text{minimize} \quad & f(x) = (x+1)^4 \\ \text{subject to} \quad & h(x) = 0 \\ \text{where} \quad & h(x) = \begin{cases} x^4 & x < 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2.7}$$

Both $f(x)$ and $h(x)$ have continuous second-order derivatives, and $x^* = 0$ is a local minimum subject to the equality constraint. However, since $\nabla_x f(x^*) \neq 0$ and $\nabla_x h(x^*) = 0$, there cannot exist any λ^* that satisfies the first-order necessary conditions (2.3). ■

The next two lemmas show that sets **B** and **C** are not equal. Hence, methods based on the first-order necessary conditions cannot be used to find all the saddle points of a problem.

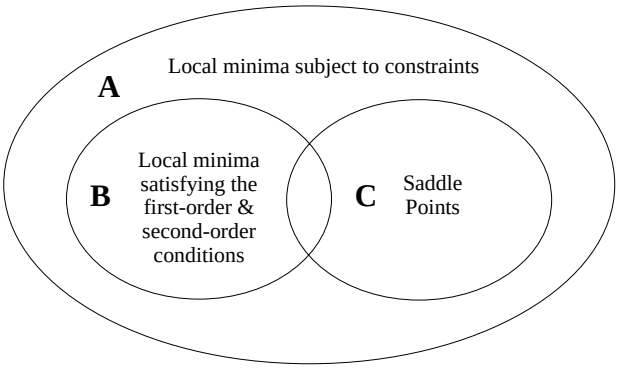

Figure 2.1: Relationship among solution sets in continuous problems.

Lemma 5. A solution (x^*, λ^*) to the first-order necessary conditions may not be a saddle point, *i.e.*, $x \in \mathbf{B} \not\Rightarrow x \in \mathbf{C}$.

Proof. Using the same example defined in the proof of Lemma 2 as a counter example, it is obvious that $x^* = 0$ together with any λ will satisfy the first-order necessary conditions because $\nabla_x f(x) = \nabla_x h(x) = 0$ at $x^* = 0$. However, as proved in Lemma 2, there cannot exist λ^* to make (x^*, λ^*) a saddle point. ■

Lemma 6. A saddle point may not satisfy the first-order necessary conditions, *i.e.*, $x \in \mathbf{C} \not\Rightarrow x \in \mathbf{B}$.

Proof. The proof is done by using the following example.

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

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

Obviously, $(0, 0)$ is a saddle point. However there is no continuous gradient at point $x^* = 0$. Hence, the first-order necessary conditions are not satisfied. ■

The following theorem summarizes the results proved in the lemmas.

Theorem 2. The relationships among the solutions set $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ as as follows: (a) $\mathbf{B} \subset \mathbf{A}$, (b) $\mathbf{C} \subset \mathbf{A}$, and $\mathbf{B} \neq \mathbf{C}$. Figure 2.1 depicts the relationships.

2.4 Summary

In this chapter, we first introduce the general Lagrangian methods for solving nonlinear continuous constrained optimization problems. We then discover the relationships among three solution spaces: the set of local minima subject to constraints, the set of solutions satisfying the first-order necessary and second-order sufficient conditions, and the set of saddle points. From the relationships discovered, we claim that continuous Lagrangian methods are defficient in the sense that the solutions they can find are just a proper subset of all the solutions in the search space. In the next chapter, we extend continuous Lagrangian methods to discrete space and reconsider the relationships among the three solution spaces metioned above. We prove that, by adding some constraints, we can overcome the deficiency of continuous Lagrangian methods. That is, the set of all solutions found by our proposed Discrete Lagrangian methods is the same as the set of all possible solutions to the optimization problem.

Chapter 3

Lagrangian Methods for Discrete Problems

In discrete space, we like to find first-order conditions similar to those derived in continuous space and establish the relationships among different solution sets. In this section, we prove the corresponding first-order conditions and propose the corresponding first-order procedure. We also show that, when all constraint functions are non-negative, solutions satisfying the first-order conditions are equivalent to all local minima subject to constraints.

Our work here extends substantially our previous results in this area [43]. In our previous work, we showed that if a point was a discrete saddle point, then it was a local minimum subject to constraints. We did not show the relationship between the two solution spaces and the condition under which they can be equivalent. Our previous proof was based on a discrete gradient operator using a special neighborhood function of Hamming distance 1, without considering more general neighborhood functions. Finally, the discrete search procedure proposed earlier was based on finding discrete saddle points. Here, we develop a similar procedure based on the (new) discrete-space first-order necessary conditions derived from the concept of discrete saddle points. We further show that the heuristic procedure proposed before is one way of finding solutions that satisfy the discrete-space first-order conditions. Our current approach, therefore, provides a complete formal mathematical foundation for solving nonlinear optimization problems in discrete space.

3.1 Basic Definitions

In a way similar to that in continuous problems, the *discrete Lagrangian function* [43] of a nonlinear discrete optimization problem with equality constraints is defined as follows:

$$L_d(x, \lambda) = f(x) + \lambda^T h(x). \tag{3.1}$$

In discrete space, there is currently no accepted definition of discrete gradients. The following definition is one that we use in discrete Lagrangian space.

Definition 7. Given the discrete Lagrangian function $L_d(x, \lambda)$, we define $\Delta_x L_d(x, \lambda)$, the *discrete gradient* of $L_d(x, \lambda)$ at point x for fixed λ, as a vector[1] that points from x to a neighborhood point of $x \in \mathcal{N}(x) \cup \{x\}$ with the minimum L_d value. That is,

$$\Delta_x L_d(x, \lambda) = \vec{\nu}_x = y \ominus x = (y_1 - x_1, y_2 - x_2, \ldots, y_n - x_n) \tag{3.2}$$
$$\text{where } y \in \mathcal{N}(x) \cup \{x\} \text{ and } L_d(y, \lambda) = \min_{\substack{x' \in \mathcal{N}(x) \\ \cup \{x\}}} L_d(x', \lambda).$$

Here, $\ominus$ is the vector-subtract operator for changing x in discrete space to one of its "user-defined" neighborhood points $\mathcal{N}(x)$. Intuitively, $\vec{\nu}_x$ is a vector pointing from x to y, the point with the minimum L_d among all neighboring points of x, including x itself. That is, when x itself has the minimum L_d, then $\vec{\nu}_x = \vec{0}$. Note that this definition is similar to the concept of gradients in continuous space.

It is important to emphasize that, with this definition of discrete gradients, gradients cannot be added/subtracted in discrete space. This is shown in the following lemma.

Lemma 7. There is no addition operation for the discrete gradient defined in Definition 7.

Proof. In general, the gradient of $[f_1(x) + f_2(x)]$ is not the same as the summation of the gradient of $f_1(x)$ and the gradient of $f_2(x)$. The following example illustrates the point.

$$\begin{array}{lllll} f_1(0,0) = 0 & f_1(0,1) = -3 & f_1(1,0) = 0 & f_1(-1,0) = -2 & f_1(0,-1) = 0 \\ f_2(0,0) = 0 & f_2(0,1) = 0 & f_2(1,0) = -3 & f_2(-1,0) = -2 & f_2(0,-1) = 0 \end{array}$$

[1] We assume that points in the x space are represented as vectors without explicitly denoting them using the vector notation, and that λ is scalar.

From the definition of discrete gradients, we know that $\Delta_x f_1(0,0) = (0,1)$, $\Delta_x f_2(0,0) = (1,0)$ and $\Delta_x(f_1 + f_2)(0,0) = (-1,0)$. Hence, $\Delta_x(f_1 + f_2)(0,0) \neq \Delta_x f_1(0,0) \oplus \Delta_x f_2(0,0)$. ■

The result of this lemma implies that the addition of two gradients in (2.3) cannot be carried out, rendering it impossible to prove the first-order conditions similar to those proved in Theorem 1 for continuous problems.

The concept of saddle points can also be extended to discrete problems [43].

Definition 8. A *saddle point* (x^*, λ^*) of $L(x, \lambda)$ is defined as one that satisfies the following:

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*) \tag{3.3}$$

for all $x \in \mathcal{N}(x)$ and all λ sufficiently close to λ^*.

The concept of saddle points is of great importance to discrete problems because starting from saddle points, we can derive similar first-order necessary conditions for discrete problems and develop efficient procedures for finding local minima subject to constraints.

3.2 First-Order Necessary Conditions and Methods for Discrete Problems

In this section, we prove the first-order necessary conditions for solving equality-constrained discrete problems and present the first-order search procedure. Although the conditions are similar to those for continuous problems, they are derived from the concept of saddle points rather than regular points.

Theorem 3. (*First-Order Necessary Conditions for Discrete Problems*) In discrete space, the set of all saddle points is equal to the set of all solutions that satisfy the following two equations:

$$\Delta_x L_d(x, \lambda) = \Delta_x[f(x) + \lambda^T h(x)] = 0 \tag{3.4}$$

$$\nabla_\lambda L_d(x, \lambda) = 0 \tag{3.5}$$

Note that the gradient in (3.4) is discrete, that the discrete gradient operator *cannot* be distributed into the two terms of the addition according to Lemma 7, and that the gradient in (3.5) is continuous.

Proof. The proof is done in two parts:

"$\Rightarrow$" part: Given a saddle point (x^*, λ^*), we like to prove it to be a solution to (3.4) and (3.5). Eq. (3.4) is true because L_d cannot be improved among $\mathcal{N}(x^*)$ from the definition of saddle points. Hence, $\Delta_x L_d(x^*, \lambda^*) = 0$ must be true from the definition of discrete gradient. Eq. (3.5) is true because L_d is a linear function of λ, and $h(x^*) = 0$ at point (x^*, λ^*). Hence, $\nabla_\lambda L_d(x^*, \lambda^*) = h(x^*) = 0$.

"$\Leftarrow$" part: Given a solution (x^*, λ^*) to (3.4) and (3.5), we like to prove it to be a discrete saddle point. The first condition $L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*)$ holds for all $x \in \mathcal{N}(x^*)$ because $L_d(x, \lambda^*) - L_d(x^*, \lambda^*) \geq |\Delta_x L_d(x^*, \lambda^*)| = 0$. (The last equality is true according to the definition of discrete gradient.) Hence, no improvement of L_d can be found in the neighborhood of x^*. The second condition $L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*)$ is true for all λ because $h(x^*) = \nabla_\lambda L_d(x^*, \lambda^*) = 0$ according to (3.5). Thus, (x^*, λ^*) is a saddle point in discrete space. ■

Note that there is no need to develop second-order sufficient conditions (as in the continuous case) for solving discrete problems because discrete gradients always point in descending directions, and local extremum found are always local minima.

The discrete-space first-order conditions provide a basis for an efficient search procedure. In a way similar to that in continuous space, we have an iterative first-order procedure that looks for saddle points in discrete space.

Procedure 1. Discrete Lagrangian Method (DLM)

$$x^{k+1} = x^k \oplus \Delta_x L_d(x^k, \lambda^k) \tag{3.6}$$

$$\lambda^{k+1} = \lambda^k + c_1 h(x^k) \tag{3.7}$$

where $\oplus$ is the vector-add operator ($x \oplus y = (x_1 + y_1, \ldots x_n + y_n)$), and c_1 is a positive real number controlling how fast the Lagrange multipliers change.

Note that (3.6) uses vector addition instead of arithmetic addition as in (2.5) in the continuous case, and that (3.7) is a discrete approximation to implement (3.5). The method presented here is more general than the one presented earlier [43]. In our earlier method, we considered a special neighborhood function of Hamming distance one and arithmetic addition instead of vector addition.

It is easy to see that the necessary condition for DLM to converge is when $h(x) = 0$, implying that x is a feasible solution to the original problem. If any of the constraints in $h(x)$ is not satisfied, then λ will continue to evolve to suppress the unsatisfied constraints. Further, as in continuous Lagrangian methods, the time for DLM to find a saddle point can only be determined empirically, even when there exist feasible solutions.

Finally, we show that DLM stated in Procedure 1 can be used to locate saddle points. The result has been proved before [43] and is stated here for completeness.

Theorem 4. (*Fixed-Point Theorem for Discrete Problems*) A saddle point (x^*, λ^*) of (3.1) is reached if and only if Procedure 1 terminates [43].

Proof. The proof consists of two parts.

"$\Rightarrow$" part: We prove that at a saddle point DLM will stop. This means that if (x^*, λ^*) is a saddle point, then it is also a fixed point of the iterative process. Given a saddle point (x^*, λ^*), we know from its definition that $L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*)$ holds true for all $x \in \mathcal{N}(x^*)$. Thus, from the definition of discrete gradient, we conclude that $x_{k+1} = x_k$. In addition, since x^* is feasible, we conclude that $h(x_k) = 0$. Since $x_{k+1} = x_k$ and $\lambda_{k+1} = \lambda_k$, Procedure 1 will stop at (x^*, λ^*).

"$\Leftarrow$" part: We prove the point that Procedure 1 stops at must be a saddle point of $L_d(x, \lambda) = f(x) + \lambda^T h(x)$. This also means that the procedure will not stop at points other than saddle points. Since the procedure stops at (x_k, λ_k), it implies that $x_{k+1} = x_k$ and $\lambda_{k+1} = \lambda_k$. These conditions imply that $\Delta_x L_d(x_k, \lambda_k) = \vec{0}$, meaning that $L_d(x_k, \lambda_k) \leq L_d(x'_k, \lambda_k)$ for any $x'_k \in \mathcal{N}(x_k)$. Moreover, $h(x_k) = 0$ is true because $\lambda_{k+1} = \lambda_k$. Hence, x_k is a feasible point, and we conclude that (x_k, λ_k) is a saddle point. ■

DLM can help locate all saddle points for a discrete constrained optimization problem. However, it cannot locate all local minima subject to constraints. This is proved in the following lemma.

Lemma 8. Given the definition of discrete Lagrangian function (3.1), a local minimum x^* subject to constraints may not be a saddle point.

Proof. The proof is done by showing a counter example that it is not always possible to find λ^* to make (x^*, λ^*) a saddle point even when x^* is a feasible local minimum in discrete space.

Consider a two-dimensional discrete equality-constrained problem with objective function $f(x)$ and constraint $h(x) = 0$.

$$f(0,0) = 0 \quad h(0,0) = 0 \quad f(0,1) = 1 \quad h(0,1) = 0 \quad f(1,0) = 0 \quad h(1,0) = 0$$
$$f(-1,0) = -1 \quad h(-1,0) = -1 \quad f(0,-1) = 0 \quad h(0,-1) = 1$$

Obviously, $(x^*, y^*) = (0,0)$ is a local minimum subject to constraints. Furthermore, from the definition of Lagrangian function, we know that $L_d((0,0), \lambda) = 0$ holds true for any λ because $h(0,0) = f(0,0) = 0$.

To draw a contradiction, assume that $(0,0)$ is a saddle point. Hence, there is λ^* such that $L_d((0,0), \lambda^*) \leq L_d((-1,0), \lambda^*)$, and $L_d((0,0), \lambda^*) \leq L_d((0,-1), \lambda^*)$. After substitution, we get the following equations:

$$\begin{aligned} 0 \leq f(-1,0) + \lambda^* \times h(-1,0) &= -1 + \lambda^* \times (-1) \\ 0 \leq f(0,-1) + \lambda^* \times h(0,-1) &= 0 + \lambda^* \times 1 \end{aligned} \tag{3.8}$$

Since there is no solution for λ^* in (3.8), the example shows that $(0,0)$ is a local minimum subject to constraints but not a saddle point. ■

3.3 Special Case of Non-negative Equality Constraint Functions

In this section, we consider a special case in which all the constraint functions are non-negative; that is, $h(x) \geq 0$ for all x, and the constraints are satisfied when $h(x) = 0$. In this case, we show that the set of saddle points is the same as the set of local minima subject to constraints (thereby overcoming the limitation shown in Lemma 8).

The condition of non-negative constraint functions is not difficult to achieve in practice. The next section shows a transformation for inequality constraints. For general equality constraint functions, we can design a transformation T to convert $h(x) = 0$ into a non-negative equality

constraint as follows:

$$h(x) = 0 \quad \Rightarrow \quad T(h(x)) = 0 \tag{3.9}$$
$$\text{where} \begin{cases} h(x) = 0 & \Rightarrow \quad T(h(x)) = 0 \\ T(h(x)) = 0 & \Rightarrow \quad h(x) = 0 \\ 0 \leq T(h(x)) \end{cases}$$

Two examples of T are the absolute function $T(h(x)) = |h(x)|$ and the square function $T(h(x)) = h^2(x)$. The definitions of Lagrangian function and saddle point remain unchanged.

The following lemma shows that, when all the constraint functions are non-negative, all local minima subject to constraints are also saddle points. Hence, methods for locating saddle points can be used to solve the original discrete optimization problem.

Lemma 9. When all the constraint functions are non-negative, *i.e.*, $h(x) \geq 0$, and the constraints are satisfied when $h(x) = 0$, there exists a finite λ^* such that (x^*, λ^*) is a saddle point for any local minimum x^* subject to constraints in discrete space.

Proof. To prove the lemma, we need to construct λ^* for every local minimum x^* subject to constraints in order to make (x^*, λ^*) a saddle point. This λ^* must be bounded and be found in finite time in order for the procedure to be useful.

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

$$\lambda_i^* \rightarrow \max\left(\lambda_i^*, \frac{f(x^*) - f(x)}{h_i(x)}\right) \tag{3.10}$$

The update defined in (3.10) is repeated for every unsatisfied constraint of x and every $x \in \mathcal{N}(x^*)$ until no further update is possible. Since $\mathcal{N}(x^*)$ has finite number of elements in discrete space, (3.10) will terminate in finite time and result in finite λ^* values.

(b) Proving that (x^*, λ^*) *is a saddle point.* To prove that (x^*, λ^*) is a saddle point, we need to prove that, for all $x \in \mathcal{N}(x^*)$, $L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*)$, where $L_d(x, \lambda) = f(x) + \lambda^T h(x)$. The first inequality is trivial because λ continues to increase in (3.10) until it stops. In the second inequality, for all $x \in \mathcal{N}(x^*)$ such that $h(x) = 0$, it is obvious that $L_d(x^*, \lambda^*) = f(x^*) \leq f(x) =$

$L_d(x, \lambda^*)$ since x^* is a local minimum subject to constraints. For all $x \in \mathcal{N}(x^*)$ such that $h(x) \neq 0$, there must be at least one constraint that is not satisfied, say $h_i(x) > 0$. Moreover, from the construction method, we know that $\lambda_i^* \geq \frac{f(x^*)-f(x)}{h_i(x)}$. Consequently,

$$L_d(x^*, \lambda^*) = f(x^*) \leq f(x) + \lambda_i^* h_i(x).$$

Further, since $\sum_{j=1, j\neq i}^{m} \lambda_j^* h_j(x)$ is non-negative, it is obvious that

$$L_d(x^*, \lambda^*) = f(x^*) \leq f(x) + \sum_{j=1}^{m} \lambda_j^* h_j(x) = L_d(x, \lambda^*).$$

Hence, (x^*, λ^*) is a saddle point. ■

Corollary 1. Given λ^* defined in Lemma 9, (x^*, λ') is a saddle point for any $\lambda' \geq \lambda^*$, where $\lambda' \geq \lambda^*$ means that every element of λ' is not less than the corresponding element of λ^*.

The proof of the corollary is similar to that of Lemma 9 and will not be shown here. The corollary is important because in practice a search algorithm may only be able to find $\lambda \geq \lambda^*$ but not the exact λ^*.

It is easy to see that the first-order necessary conditions defined in Theorem 3 are satisfied in the special case when all the constraint functions are non-negative. The theorem is true because the proof does not depend on whether the constraints are positive or not.

Similar to the continuous case, we denote **A** to be the set of all local minima subject to constraints, **B** to be the set of all solutions that satisfy the discrete-space first-order necessary conditions (3.4) and (3.5), and **C** to be the set of all discrete saddle points satisfying (3.3). The following theorem shows the relationships among the solution spaces for the special case of non-negative equality constraint functions. Its proof follows from Theorem 3 and Lemma 9.

Theorem 5. In discrete space, when all the constraint functions are non-negative, $\mathbf{A} = \mathbf{B} = \mathbf{C}$.

Figure 3.1 illustrates the relationships in Theorem 5. Comparing with Figure 2.1, the relationships provide a formal mathematical foundation for solving nonlinear discrete constrained optimization problems. Moreover, from Theorems 4 and 5, we conclude that when all constraint functions

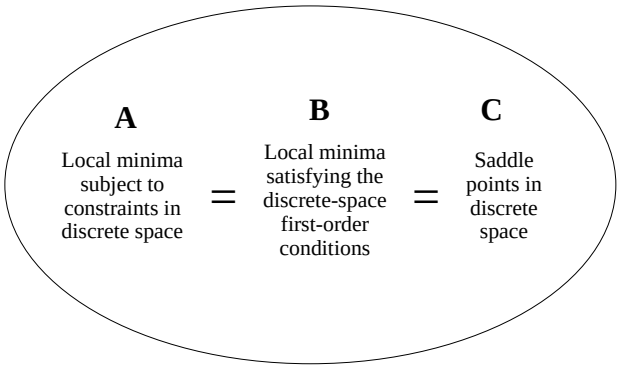

Figure 3.1: Relationship among solution sets in discrete space when all constraint functions are non-negative.

are non-negative, finding all saddle points is equivalent to solving the constrained optimization problem.

3.4 Transforming Inequality Constraints to Equality Constraints

The results and procedures we have discussed so far apply only to discrete constrained optimization problems with equality constraints. To handle problems (1.1) with inequality constraints, we need to transform inequality constraints $g(x)$ into equality constraints.

One general method for handling inequality constraint $g_j(x) \leq 0$ in continuous problems is to add a slack variable to the constraint to make it an equality constraint $g_j(x) + y_j^2 = 0$. Although the transformation works in discrete problems, the resulting function $g_j(x) + y_j^2$ may be positive or negative. Hence, as proved in Lemma 8, there are local minima subject to constraints that cannot be found by Procedure 1.

Another method of transforming inequality constraint $g_j(x) \leq 0$ is to apply a maximum function to convert the constraint into $\max(g_j(x), 0) = 0$. Obviously, the new constraint is satisfied if and only if $g_j(x) = 0$. Moreover, the new constraint function is non-negative, implying that saddle points are equivalent to local minima subject to constraints (see Theorem 5).

This transformation does not work for continuous problems because the transformed function may not be continuous at $g_j(x) = 0$, rendering non-existent derivatives at that point. The problem can be overcome in continuous space by using a transformation $(\max(g_j(x), 0))^q = 0$, where $q >$

1 [54]. It is not an issue in discrete problems because gradients are not required to be continuous functions in discrete space.

The discrete Lagrangian function using the maximum transformation for inequality constraints is as follows:

$$L_d(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \sum_{i=1}^{k} \mu_i \max(0, g_i(x)) \tag{3.11}$$

Using the maximum transformation, the general discrete Lagrangian method is given below.

Procedure 2. General Discrete Lagrangian Method

$$x^{k+1} = x^k \oplus \Delta_x[f(x^k) + \lambda^k h(x^k) + \mu^k \max(0, g(x^k))] \tag{3.12}$$

$$\lambda^{k+1} = \lambda^k + c_1 h(x^k) \tag{3.13}$$

$$\mu^{k+1} = \mu^k + c_2 \max(0, g(x^k)) \tag{3.14}$$

where $\oplus$ is the vector-add operator, and c_1 and c_2 are positive real numbers controlling how fast the Lagrange multipliers change.

3.5 Weighted Discrete Lagrangian Methods

As discussed in Section 2.2, Lagrangian methods rely on ascents in the Lagrange-multiplier space and descents in the objective space in order to reach equilibrium. The convergence speed and solution quality, however, depends on the balance between objective $f(x)$ and constraints $h(x)$ and $g(x)$. Although changes in λ and μ lead to different balance between ascents and descents, convergence can be improved by introducing a weight on the objective function. These considerations lead to a new Lagrangian function as follows.

$$L_d(x, \lambda, \mu) = w\ f(x) + \lambda^T h(x) + \sum_{i=1}^{k} [\mu_i\ \max(0, g_i(x))] \tag{3.15}$$

where $w > 0$ is a user-controlled weight on the objective. In the following subsection, we illustrate the effect of w on convergence speed and solution quality.

By applying DLM on (3.15), we observe four possible behaviors of the search trajectory:

- The trajectory converges without oscillations.

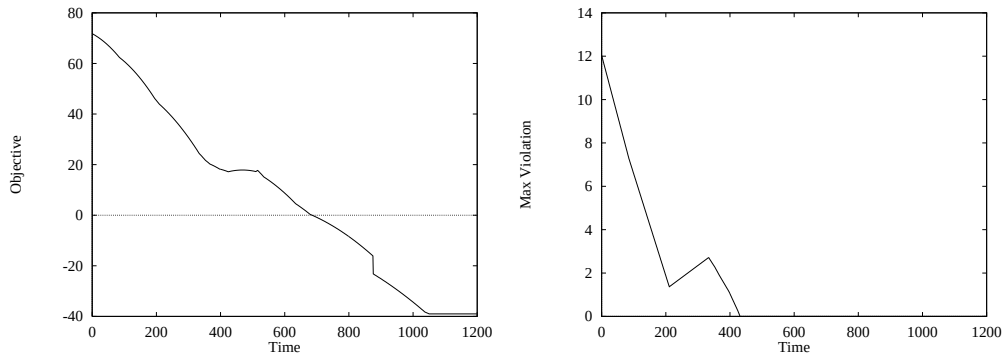

Figure 3.2: Objective and maximal violation converge without oscillations when w is fixed at 0.00001

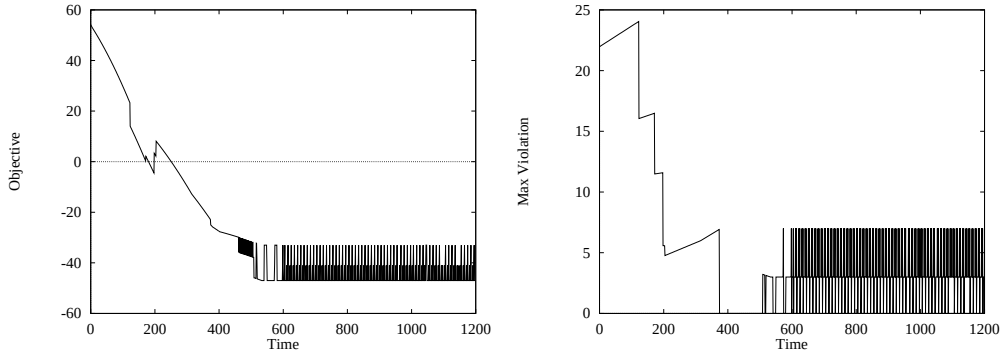

Figure 3.3: Objective and maximal violation oscillate when w is fixed at 1000

- The trajectory gradually reduces in oscillations and eventually converges.
- The trajectory oscillates within some range but never converges.
- The magnitude of oscillations increases, and the trajectory eventually diverges.

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

To demonstrate the four behaviors, we evaluate Problem 2.6 in [14] under different weights. The original problem is a nonlinear, continuous optimization problem with 10 variables and 25 inequality constraints. Since it uses continuous variables in the range $[0, 1]$, we convert it into a discrete optimization problem by dividing each variable x_i by 1000 and restricting the new variable $x_i/1000$ to integer values. (The details of the problem are not shown here due to space limitation.)

Table 3.1: Effects of w on convergence time and solution quality from 20 randomly generated starting points for the discretized version of Problem 2.6 in [14].

Fixed w	Avg. Conv. Time	Fraction Converged	Avg Sol.	Best Sol.
0.00001	1099	100%	-24.7	-39.0
0.0001	1247	100%	-29.4	-39.0
0.001	1249	100%	-29.3	-39.0
0.01	1254	100%	-29.4	-39.0
0.1	1267	100%	-28.3	-39.0
1	1230	90%	-27.4	-39.0
10	1203	75%	-27.8	-39.0
100	972	40%	-34.8	-39.0
1000	679	15%	-39.0	-39.0
10000	–	0%	–	-
100000	–	0%	–	–

We apply DLM based on (3.15) to solve the discretized problem. The first convergence behavior is demonstrated by setting w to 10^{-5}. Figure 3.2 shows the maximum violation and the objective of the trajectory. The search first starts from an infeasible region and gets into a feasible region eventually, where the objective function stabilizes to a constant. During this time, the maximum violation keeps decreasing until it reaches zero. The third convergence behavior is illustrated by setting w to 10^3. In this case, the behavior shown in Figure 3.3 is quite different. After getting into a feasible region, the search continues to oscillate around the boundary of the region without subsiding. The last convergence behavior can be shown by setting w to larger than 10^5. In this case, the search diverges without moving into a feasible region. The second convergence behavior cannot be illustrated by this benchmark problem.

Table 3.1 shows the complete results on evaluating the discretized Problem 2.6 using different w from 20 randomly generated starting points. When w is small, the trajectory converges for all the starting points but the average solution quality is not good. When w is larger, the fraction of converged searches drops gradually, while the average solution improves until it is the same as the best solution when $w = 1000$. Further increases in w do not result in any converged searches.

1. Set control parameters:
   time interval δ_t
   initial weight $w(t = 0)$
   maximum number of iterations i_{max}
2. Set window size $N_u = 100$ or 10
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 δ_t to get to (x_j, λ_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** $((j \mod N_u) == 0)$ **then**
   /* Test whether w should be modified at the end of a window */
9. compute performance metrics based on data collected
10. change w when the conditions defined in Table 3.3 are satisfied
    end if
11. **end while**

Figure 3.4: Framework of a dynamic weight-adaptation algorithm

These results demonstrate that the choice of w is critical in controlling both the convergence time and the solution quality. There is, however, no effective method to choosing a fixed w except by trial and error.

3.6 Dynamic Weight Adaptation

In this section, we propose a strategy to adapt w based on the behavior of the dynamic system defined in (3.12)-(3.14) in order to obtain high-quality solutions with short convergence time.

In general, changing w may speed up or delay convergence before a trajectory reaches an equilibrium point, and may bring the trajectory out of equilibrium after it reaches there. In this section, we design weight-adaptation algorithms to speed up convergence. Studies to bring a trajectory out of equilibrium by modifying w will be studied in the future.

Figure 3.4 outlines the dynamic weight-adaptation algorithm. Its basic idea is to first estimate the initial weight $w(t = 0)$ (Step 1), measure the performance metrics of the search trajectory $(x(t), \lambda(t))$ 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 δ_t so that the maximum number of iterations is t_{max}/δ_t. Further, assume a stopping condition if the search were stopped before t_{max} (Step 4). Given a starting point $x(0)$, we set the initial Lagrange multipliers to be zero; *i.e.*, $\lambda(t = 0) = 0$. Let (x_i, λ_i) be the point of the i^{th} iteration, and $v_{max}(i)$ be its maximum violation over all the m constraints:

$$v_{max}(i) = \max\left[\max_{1\le j\le m}\{|h_j(x(t))|\}, \max_{1\le j\le k}\{g_j(x(t), 0\}\right] \tag{3.16}$$

To monitor the progress of the search, we divide time into non-overlapping windows of size N_u iterations each (Step 2). In each window, we compute some metrics to measure the progress of the search relative to that of previous windows. In the u^{th} window $(u = 1, 2, \cdots)$, we calculate the average (or median) of $v_{max}(i)$ over all the iterations in the window,

$$\bar{v}_u = \frac{1}{N_u}\sum_{i=(u-1)N_u+1}^{uN_u} v_{max}(i) \quad \text{or} \quad \bar{v}_u = \underset{\substack{(u-1)N_u+1\\ \le i \le uN_u}}{\text{median}}\{v_{max}(i)\} \tag{3.17}$$

and the average (or median) of the objective $f_i(x)$.

$$\bar{f}_u = \frac{1}{N_u}\sum_{i=(u-1)N_u+1}^{uN_u} f_i(x) \quad \text{or} \quad \bar{f}_u = \underset{\substack{(u-1)N_u+1\\ \le i \le uN_u}}{\text{median}}\{f_i(x)\} \tag{3.18}$$

During the search, we apply an algorithm to solve the dynamic system (3.12)-(3.14), and advance the trajectory by time interval δ_t in each iteration in order to arrive at point (x_j, λ_j) (Step 5).

At this point, we test whether the trajectory diverges or not (Step 6). Divergence happens when the maximum violation $v_{max}(i)$ 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 $v_{max}(i)$ and $f_i(x)$, that will be used to calculate the performance metrics for each window (Step 7).

At the end of each window or every N_u iterations (Step 8), we decide whether to update w based on the performance metrics (3.17) and (3.18) (Step 9). In our current implementation, we use the averages (or medians) of maximum violation $v_{max}(i)$ and objective $f_i(x)$. In general, other

application-specific metrics can be used, such as the number of oscillations of the trajectory. Based on these measurements, we adjust w accordingly (Step 10).

To understand how weights should be updated in Step 10, we examine all the possible behaviors of the resulting search trajectory in successive windows. We have identified four possible cases. First, the trajectory does not stay within a feasible region, but goes from one feasible region to another through an infeasible region. During this time, the maximum violation $v_{max}(i)$ is zero when the trajectory is in the first feasible region, increased when it travels from the first feasible region to an infeasible region, and decreased when going from the infeasible region to the second feasible region. No oscillations will be observed because oscillations normally occur around an equilibrium point in one feasible region. In this case, no change of w is required.

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

Third, the search trajectory moves very slowly within a feasible region. This happens when w is very small, and the constraints dominate the search process. As a result, the objective value is improved very slowly and may eventually converge to a poor value. This situation can be identified when the trajectory remains within a feasible region in two successive windows, and there is little improvement in the objective. Obviously, we need to increase w in order to speed up the improvement of the objective. If the objective remains unchanged, then the trajectory has converged, and no further modification of w is necessary.

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

Given the four convergence behaviors, Table 3.3 shows a comprehensive list of conditions to change w for discrete problems. Scaling factors $0 < \alpha_0, \alpha_1 < 1$ represent how fast w is updated.

Table 3.2: Effects of dynamic weighting on convergence time and solution quality from 20 randomly generated starting points for the discretized version of Problem 2.6 in [14].

Initial w	Avg. Conv. Time	Fraction Converged	Avg. Sol.	Best Sol.
0.00001	1249	100%	-29.0	-39.0
0.0001	1249	100%	-29.0	-39.0
0.001	1248	100%	-29.3	-39.0
0.01	1249	100%	-29.3	-39.0
0.1	1265	100%	-28.8	-39.0
1	1289	100%	-27.4	-39.0
10	1377	100%	-28.1	-39.0
100	1902	100%	-29.3	-39.0
1000	1409	100%	-38.5	-39.0
10000	2182	100%	-38.4	-39.0
100000	2289	100%	-33.3	-39.0

Because we use difference equations to solve the dynamic system defined in (3.12)-(3.14), a trajectory in window u is said to satisfy all the constraints when $\bar{v}_u < \delta$, 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 $u-1$ and u, 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 when we observe the third convergence behavior. At this time, the trajectory is within a feasible region, and the objective is improved in successive windows (Condition a_1 in Table 3.3). Further, the improvement of the objective in the feasible region is not fast enough and is below an upper bound (Condition a_2).

Weight w should be decreased when we observe the second convergence behavior (the trajectory oscillating around an equilibrium point) or the fourth convergence behavior (the trajectory moving slowly back to the feasible region). In this case, the trajectory is either oscillating or not oscillating (Condition b_3), is not in a feasible region (Condition b_1), and the trend of the maximum violation does not decrease (Condition b_2).

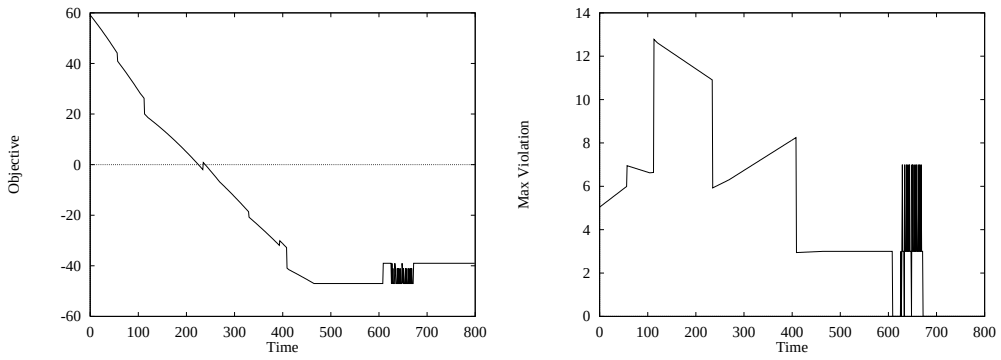

Figure 3.5: Objective and maximal violation eventually converge after some oscillations when w is dynamically adapted with initial value of 10000.

Table 3.3: Conjunctive conditions under which weights will be changed in Step 10 of Figure 3.4, given performance measurements in the u^{th} window: $\bar{v}_u$, $\bar{f}_u$, and NO_u (number of oscillations) and application-specific constants α_0, α_1, β_0, β_1, δ, and ϵ.

ID	Condition to increase w to $\frac{w}{\alpha_0}$	ID	Condition to decrease w to $w \times \alpha_1$
a_1	$\bar{v}_{u-1}, \bar{v}_u < \delta$	b_1	$\bar{v}_u \geq \delta$
a_2	$\beta_0 \lvert\overline{f}_{u-1}\rvert > \overline{f}_{u-1} - \overline{f}_u > \beta_1 \lvert\overline{f}_{u-1}\rvert$	b_2	$\bar{v}_{u-1} - \bar{v}_u \leq \beta_0 \bar{v}_{u-1}$
		b_3	$NO_u \geq \epsilon$

Table 3.2 shows the results of applying dynamic weight adaptation in DLM to solve the same benchmark problem presented in Section 3.5. Comparing with Table 3.1, we see that all the searches can now converge with better average solutions.

Figure 3.5 plots the trajectory when w was set to 10^{-4} initially and dynamically adjusted during the search. There were some oscillations during the search before they eventually subsided and the search converged to the optimal solution. These behaviors are very much improved over those in Figure 3.3 and the case in Table 3.1 when the initial weight was 10^4.

3.7 Summary

In this chapter, we propose the discrete Lagrangian methods (DLM) and extend our previous (incomplete and highly simplified) theory on the method. Our proposed discrete Lagrangian method

forms a strong mathematical foundation for solving general nonlinear discrete optimization problems. We invent a new vector-based definition of gradient, develop in discrete space first-order conditions similar to those in continuous space, propose a heuristic method to find saddle points, and show the relationship between saddle points and local minimal solutions satisfying constraints. We then show, when all the constraint functions are non-negative, that the set of saddle points is the same as the set of local minimal points satisfying constraints.

In the following three chapters, we apply DLM to three applications: discrete benchmark problems, multiplierless filter bank designs, and mixed integer optimizaiton problems. We find promising results for all these three applications after DLM is being applied.

Chapter 4

Application 1-Discrete Benchmark Problems

In this chapter, we present experimental results on evaluating DLM using some discrete nonlinear benchmark problems. These benchmarks were obtained by converting a set of continuous benchmarks problems in [14].

4.1 Constructing Discrete Optimization Problems from Continuous Benchmark Problems

Starting from a continuous benchmark problem, we first normalize all variable ranges to $[0, 1]$, multiply each variable by a predefined range value (e.g. 1000, or 10000) and restrict all the variables to integers during the search. In this way, the continuous problems are transformed to discrete problems.

For example, Problem 3.2.1 has the following form.

$$\begin{aligned} &\textit{Minimize} && f(X) = 37.293239x_1 + 0.8356891x_1x_5 + 5.3578547x_3^2 - 40792.141 \\ &\textit{subject to} && max(0, g(X)) = 0 \end{aligned}$$

where $g(x)$ is defined as

$$g(x) = \begin{cases} -0.0022053x_3x_5 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 6.665593 & \leq 0.0, \\ 0.0022053x_3x_5 - 0.0056858x_2x_5 - 0.0006262x_1x_4 - 85.334407 & \leq 0.0, \\ 0.0071317x_2x_5 + 0.0021813x_3x_3 + 0.0029955x_1x_2 - 29.48751 & \leq 0.0, \\ -0.0071317x_2x_5 - 0.0021813x_3x_3 - 0.0029955x_1x_2 + 9.48751 & \leq 0.0, \\ 0.0047026x_3x_5 + 0.0019085x_3x_4 + 0.0012547x_1x_3 - 15.699039 & \leq 0.0, \\ -0.0047026x_3x_5 - 0.0019085x_3x_4 - 0.0012547x_1x_3 + 10.699039 & \leq 0.0 \end{cases}$$

Also, the variables are all inside predefined ranges such as $x_1 \in [78.0, 102.0], x_2 \in [33.0, 45.0], x_3 \in [27.0, 45.0], x_4 \in [27.0, 45.0], x_5 \in [27.0, 45.0]$. After transformation, the objective function is changed to

$$\begin{aligned} f(x) \;=\; & 1.43656\; x_1 \;+\; 5.20783\; x_3 \;+\; 1.17331\; x_5 \;+\; 0.000361018\; x_1x_5 \;+ \\ & 0.00173594\; x_3^2 \;- 32217.4 \end{aligned}$$

The first inequality constraint is changed to

$$\begin{aligned} & 0.000405778\; x_1 \;+ 0.0018422\; x_2 \;-\; 0.00107178\; x_3 \;+\; 0.000879185\; x_4 \;+\; 0.00230559\; x_5 \;+ \\ & 2.70518 \times 10^{-7}\; x_1x_4 \;+ 1.22813 \times 10^{-6}\; x_2x_5 \;-\; 7.14517 \times 10^{-7}\; x_3x_5 \;- 1.88843 \;\leq\; 0 \end{aligned}$$

where all $x_1, x_2, \ldots, x_5$ are now integers in the range $[0, 1000]$. (Note that other inequality constraint functions are omitted due to space considerations.)

It is obvious that the transformed problems are have smaller search space. However the search space is still extremely large. For example, assuming a problem with 20 variables, each has an integer range of $[0, 10000]$, then there are 10^{80} points in the search space. After transformation, it is possible that the global optimal is excluded or there may be no feasible points at all.

4.2 Implementation Issues

During the search, all the variables are processed in a round-robin manner. For each variable, new values are tried to see if the Lagrangian value can be reduced. If so, the new value will be accepted. At the end of processing each variable, we update the Lagrangian multipliers according to the dynamic system specified in DLM and invoke the dynamic weight-adaptation algorithm to modify the weight w in order to balance between the objective and the constraints.

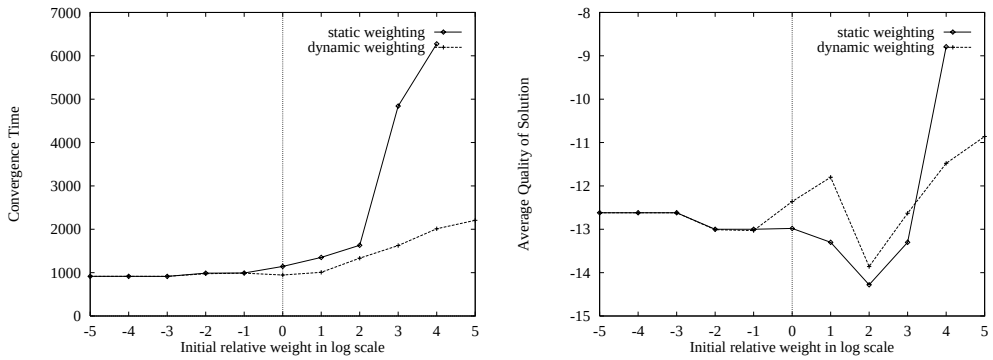

Figure 4.1: Comparison of average convergence time and average solution quality between static weighting and dynamic weighting for discretized benchmark problem 2.3. Note that the base of the log scale is 10.

We set a time limit on the search process because the search may not converge to a feasible point from a given starting point. The search will be interrupted automatically after the time limit is exceeded. For the benchmark problems in [14], we ran the DLM from several different randomly generated starting points in order to get rid of the bias of a given starting point.

4.3 Experimental Results

To test the dynamic weight adaptation strategy, we selected three problems 2.3, 2.6, and 3.4 from the benchmarks collection [14]. First, we discretized these continuous problems and generated randomly 20 discrete starting points with initial values of Lagrange multipliers set to zero. For each starting point, we apply DLM using both static weights and dynamic weight adaptation. We stop the search when either the convergence condition is satisfied or time is used up.

We set the initial weight w in the range $[10^{-5}, 10^{5}]$, which is large enough to test the robustness of our dynamic weight adaptation strategy. The parameters for our algorithm are: $N_w = 320, \alpha_0 = 0.8, \alpha_1 = 0.5, \beta_0 = 10^{-8}$, and $\beta_1 = 10^{-4}$.

We use both the average convergence time from the 20 random starting points and the average solution quality when the search stops as performance metrics.

Figures 4.1-4.3 show the results. Obviously, the dynamic weight adaptation strategy improves a lot in convergence time over that with static weight. When w is larger than 100.0 for Problem

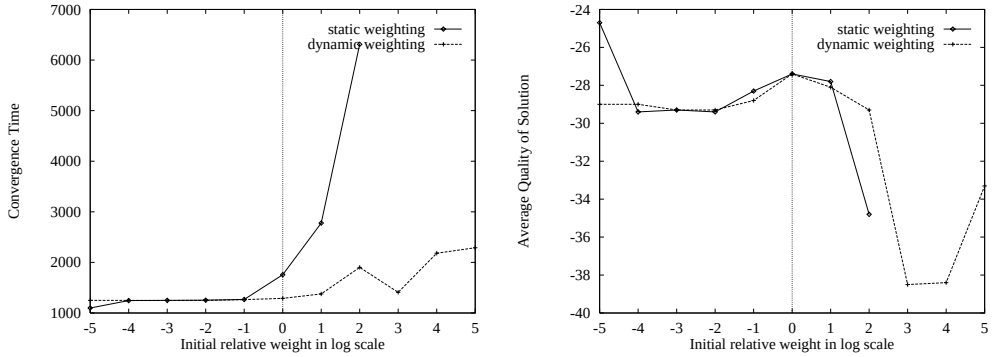

Figure 4.2: Comparison of average convergence time and average solution quality between static weighting and dynamic weighting for discretized benchmark problem 2.6. Note that the base of the log scale is 10.

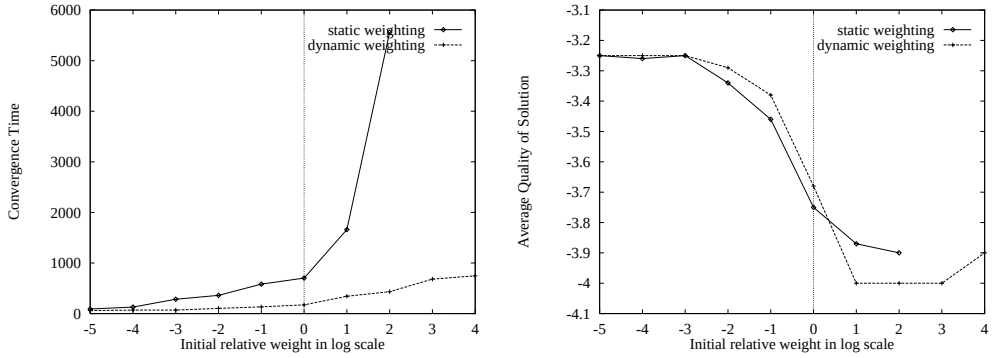

Figure 4.3: Comparison of average convergence time and average solution quality between static weighting and dynamic weighting for discretized benchmark problem 3.4. Note that the base of the log scale is 10.

2.6, DLM with static weights is unable to converge even after 10 hours. In contrast, DLM with dynamic weight adaptation converges in less than 1 hour for all different initial weights.

The complete experimental results for all the discrete benchmark problems are listed in Table 4.1. The first column is the problem identifier. The second and third columns are the number of all the constraints and equality constraints, respectively. The number of variables is shown in the fourth column. In the fifth column, the best origial solutions for the continuous problems are listed. However, it is possible that these solutions may not be achievable after the discretization. The sixth column shows the best DLM solution which is the best quality solution among all the solutions found from 100 randomly generated starting points. The feasibility ratio, shown in the

seventh column, is the ratio of the number of feasible solutions found to 100. The last column shows the average running time in minutes. The computers we used are Pentinum 200MHz machines with Linux as the operating system. As shown in Table 4.1, for most of the discretized problems, feasible solutions have been found. Due to the discretization, some solutions do not have the same good quality as their continuous counterparts. However, they are very close, such as the solutions to problems 2.5, 3.1, 4.6 and so on. For problems with lots of equality constraints such as problems 5.2, 5.4, 7.2, 7.3 and 7.4, no feasible solutions are found. However, the minimal maximum violations are very small, as shown by values in the seventh column with a '*'. By checking the solutions found by DLM, we observe that the solutions are very close to the continuous solutions. But due to precision limitation of discrete variable space, the same solutions can not be found.

4.4 Summary

In this chapter, we apply DLM and the dynamic weight adaptation to discrete nonlinear constrained benchmark problems. We show by examples that DLM with dynamic weight adaptation is quite efficient and generally can find high quality solutions. Morever, the convergence is fast due to the dynamic weight adaptation. For many of the discretized benchmark problems, we can find solutions having the same quality as the best-known ones to the original continuous problems in a short time (less than a few minutes). Also, from most randomly generated starting points, feasible solutions can be found, which proves the robustness of DLM. In the following chapter, we apply DLM to filter bank designs where the search space is also discrete, but in the forms of powers-of-two.

Table 4.1: Experimental results of applying DLM to solve benchmark nonlinear discrete optimization problems. For the problem whose feasibility value has a '*', no feasible solutions have been found and the value is actually the minimal maximum violation.

Problem	Constraints		Variables	Best Orig.	Best DLM	Feasibility	Avg.
ID	Total	Equality	Total	Solution	Solution	Ratio	Time
2.1	11	0	5	-17.0	-17.0	100	0.01
2.2	2	0	6	-213.0	-213.0	100	0.01
2.3	35	0	13	-15.0	-15.0	100	0.02
2.4	17	0	6	-11.0	-11.0	100	0.02
2.5	31	0	10	-268.0	-267.9	100	0.1
2.7.1	50	0	20	-394.75	-394.67	100	1.2
2.7.2	50	0	20	-884.75	-883.91	100	2.4
2.7.3	50	0	20	-8695.0	-8615.8	100	2.4
2.7.4	50	0	20	-754.75	-754.67	100	4.8
2.7.5	50	0	20	-4150.4	-4150.0	100	4.2
2.8	10	10	24	15990	20762	50	4.2
3.1	22	0	8	7049.25	7053.72	99	0.15
3.2	16	0	4	-30665.5	-30665.5	100	0.03
3.3	18	0	6	-310	-310	100	0.03
3.4	9	0	3	-4.0	-4.0	100	0.01
4.3	9	1	4	-4.51	-0.47	55	0.06
4.4	9	1	4	-2.217	-2.216	60	0.06
4.5	15	3	6	-11.96	-12.80	40	0.4
4.6	6	0	2	-5.51	-5.506	100	0.01
4.7	5	1	2	-16.74	-16.74	10	0.04
5.2	86	36	46	1.567	3.08	0.14*	18.0
5.4	58	26	32	1.86	1.863	0.003*	4.8
6.2	17	4	5	-400.0	-319.8	8	1.2
6.3	17	4	5	-600.0	-0.20	2	3.8
6.4	17	4	5	-750.0	-373.2	5	1.2
7.2	41	13	16	56285.0	81317.0	0.164*	7.2
7.3	64	19	27	46266.0	51438.0	1.94*	7.0
7.4	90	23	38	35920.0	35925.1	0.276*	24.0

Chapter 5

Application 2-Multiplierless Filter Bank Design

In this chapter, we apply DLM to design multiplierless QMF (quadrature mirror filtering) filter banks [52, 53]. The filter coefficients in these filter banks are in powers-of-two (PO2), where numbers are represented as sums or differences of powers of two (also called Canonical Signed Digit–CSD–representation), and multiplications are carried out as additions, subtractions and shifting. We formulate the design problem as a nonlinear discrete constrained optimization problem, using reconstruction error as the objective, and stopband and passband energies, stopband and passband ripples and transition bandwidth as constraints. Using the performance of the best existing designs as constraints, we search for designs that improve over the best existing designs with respect to all the performance metrics. Experimental results show that DLM can find designs that improve over Johnston's benchmark designs using a maximum of three to six ONE bits in each filter coefficient instead of using floating-point representations. Further, our approach is general and is applicable to the design of other types of multiplierless filter banks.

5.1 Introduction

Digital filter banks have been applied in many engineering fields. Their design objectives consist of their overall metrics and the metrics of each individual filter. Figure 5.1 summarizes the various

Filter	Design Objectives
Overall Filter Bank	Minimize amplitude distortion
	Minimize aliasing distortion
	Minimize phase distortion
Single Filter	Minimize stopband ripple (δ_s)
	Minimize passpand ripple (δ_p)
	Minimize transition bandwidth (T_t)
	Minimize stopband energy (E_s)
	Maximize passband flatness (E_p)

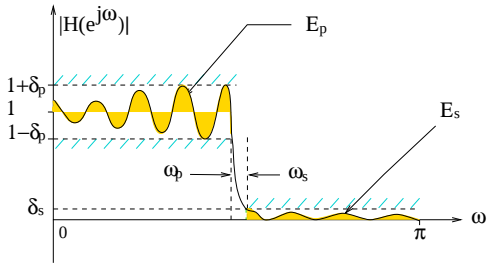

Figure 5.1: Possible design objectives of filter banks and an illustration of the design objectives of a single low-pass filter. ($[0, \omega_p]$ is the pass band, $[\omega_s, \pi]$, the stop band, $[\omega_p, \omega_s]$, the transition band.)

design objectives for measuring design quality. In general, filter bank-design problems are multi-objective, continuous, nonlinear optimization problems.

Algorithms for designing filter banks are either optimization-based or non-optimization based. In optimization-based methods, a design problem is formulated as a multi-objective nonlinear optimization problem [48] whose form may be application- and filter-dependent. The problem is then converted into a single-objective optimization problem and solved by existing optimization methods, such as gradient-descent, Lagrange-multiplier, quasi-Newton, simulated-annealing, and genetics-based methods [26, 23]. On the other hand, filter bank-design problems have been solved by non-optimization-based algorithms, which include spectral factorization [30, 49] and heuristic methods (as in IIR-filter design [37, 38]). These methods generally do not continue to find better designs once a suboptimal design has been found [49].

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

$$\hat{X}(z) = \frac{1}{2}\left[H_0(z)F_0(z) + H_1(z)F_1(z)\right] X(z) + \frac{1}{2}\left[H_0(-z)F_0(z) + H_1(-z)F_1(z)\right] X(-z) \quad (5.1)$$

where $X(z)$ is the original signal, and $H_i(z)$ and $F_i(z)$ are, respectively, the response of the analysis and synthesis filters. To perfectly reconstruct the original signal based on $\hat{X}$, we have to eliminate aliasing, amplitude, and phase distortions. QMF FIR filter banks implement perfect reconstruction by setting $F_0(z) = H_1(-z)$, $F_1(z) = -H_0(-z)$ and $H_1(z) = H_0(-z)$, leading to a filter bank with one prototype filter $H_0(z)$, linear phase, and no aliasing distortions.

Let $\mathsf{h}(n)$ be the filter parameters. If $\mathsf{h}_0(n)$ is symmetric, then $\mathsf{h}_1(n) = (-1)^n \mathsf{h}_0(n)$ is antisymmetric, and the system has linear phase. This leads to zero aliasing and phase distortions in QMF filter banks.

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 this limitation, *multiplierless* (*powers-of-two* or *PO2*) filters have been proposed. These filters use filter coefficients which have only a few bits that are ones. When multiplying a filter input (multiplicand) with one such coefficient (multiplier), the product can be performed by adding and shifting the multiplicand a number of times corresponding to the number of ONE bits in the multiplier. For example, the multiplication of y by 0100001001 can be written as the sum of three terms, $y \times 2^8 + y \times 2^3 + y \times 2^0$, each of which can be obtained by shifting y. A limited sequence of shifts and adds are usually much faster than full multiplications. Without using full multiplications, each filter tap takes less area to implement in VLSI, and more filter taps can be accommodated in a given area to implement filter banks of higher performance.

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

$$H(z) = \sum_{i=0}^{\gamma-1} x_i z^{-i} = \sum_{i=0}^{\gamma-1} \left(\sum_{j=0}^{d-1} e_{i,j} 2^j \right) z^{-i} \quad \text{where} \quad \sum_{j=0}^{d-1} |e_{i,j}| \leq l \text{ for all } i, \;\; e_{i,j} = -1, 0, 1. \tag{5.2}$$

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

The design of multiplierless filters has been solved 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 [40], simulated annealing [5], genetic algorithms [42], linear programming [29], and continuous Lagrange-multiplier methods in combination with a tree search [44].

5.2 Problem Formulation

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

5.2.1 Multi-Objective Unconstrained Formulation

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

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

One possible formulation using a subset of the measures shown in Figure 5.1 is as follows:[1]

$$\begin{aligned} &\text{Minimize} \quad E_r \quad \text{and} \quad E_s \\ &\text{where} \quad E_r = \int_{\omega=0}^{\frac{\pi}{2}} (|H_0(e^{j\omega})|^2 + |H_0(e^{j(\omega-\pi)})|^2 - 1)^2 \, d\omega \quad \text{and} \quad E_s = \int_{\omega=\omega_s}^{\pi} |H_0(e^{j\omega})|^2 d\omega \end{aligned} \tag{5.3}$$

Unfortunately, optimal solutions to (5.3) are not necessarily optimal solutions to the original problem that considers all the performance measures. Oftentimes, performance measures not included in the formulation are compromised.

In general, optimal solutions of a multi-objective problem form a *Pareto optimal frontier* such that one solution on this frontier is not dominated by another. One approach to find a point on the frontier is to optimize a weighted sum of all the objectives [26, 12, 48, 6, 36]. This approach has difficulty when frontier points of certain characteristics are desired, such as those with certain transition bandwidth. Different combinations of weights must be tested by trial and error until a desired solution is found. When the desired characteristics are difficult to satisfy, trial and error is not effective in finding feasible designs. Instead, constrained formulations should be used.

5.2.2 Single-Objective Constrained Formulation

Another approach to solve a multi-objective problem is to turn all but one objectives into constraints, and define the constraints with respect to a reference design. The specific measures constrained may be application- and filter-dependent [48].

[1] Note that in QMF filter banks, E_r is non-zero. A multi-rate filter bank that enforces perfect reconstruction ($E_r = 0$) can be formulated as a constrained optimization problem with a goal of minimizing E_s [25, 23].

Constraint-based methods have been applied to design QMF filter banks in both the frequency [26, 6, 9, 30, 45, 47] and the time domains [35, 46]. In the frequency domain, the most often considered objectives are E_r, the reconstruction error, and δ_s, the stopband ripple. As stopband ripples cannot be formulated in closed form, stopband attenuation is used instead (represented as E_s in Figure 5.1). In the time domain, Nayebi [35] gave a time-domain formulation with constraints in the frequency domain and designed filter banks using an iterative time-domain design algorithm.

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

$$\begin{aligned} &\text{Minimize} \quad E_r \\ &\text{subject to} \quad E_p \le \theta_{E_p} \qquad E_s \le \theta_{E_s} \qquad T_t \le \theta_{T_t} \\ &\qquad\qquad \delta_p \le \theta_{\delta_p} \qquad \delta_s \le \theta_{\delta_s} \end{aligned} \tag{5.4}$$

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

The original optimization problem with inequality constraints (5.4) can be transformed into an optimization problem with equality constraints as follows:

$$\text{Minimize} \quad f(X) = V_{E_r} = \frac{E_r - \theta_{E_r}}{\theta_{E_r}} \quad ; \tag{5.5}$$

$$\begin{aligned} \text{subject to} \quad V_{E_p} &= \max\left(\frac{E_p - \theta_{E_p}}{\theta_{E_p}}, 0\right) = 0 \qquad V_{E_s} = \max\left(\frac{E_s - \theta_{E_s}}{\theta_{E_s}}, 0\right) = 0 \\ V_{\delta_p} &= \max\left(\frac{\delta_p - \theta_{\delta_p}}{\theta_{\delta_p}}, 0\right) = 0 \qquad V_{\delta_s} = \max\left(\frac{\delta_s - \theta_{\delta_s}}{\theta_{\delta_s}}, 0\right) = 0 \\ V_{T_t} &= \max\left(\frac{T_t - \theta_{T_t}}{\theta_{T_t}}, 0\right) = 0 \end{aligned} \tag{5.6}$$

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

The problem here is to find whether there exists a filter of a finite word length that satisfies the constraints defined in (5.6).

1. Generate a starting point X
2. Set initial value of λ
3. **while** X is not a saddle point or stopping condition has not been reached
4. update X to X' only if this will result in $L_o(X', \lambda) < L_o(X, \lambda)$
5. **if** condition for updating λ is satisfied **then**
6. $\lambda_i := \lambda_i + c \times g_i$, ($c > 0$ is real)
7. **end if**
8. **end while**

Figure 5.2: $\mathcal{A}$: An implementation of DLM

5.3 Discrete Lagrangian Method for Designing QMF Filter Banks

The discrete Lagrangian function for optimizing PO2 filter banks is:

$$L_d(X, \lambda) = f(X) + \sum_{i \in \{E_p, E_s, \delta_p, \delta_s, T_t\}} \lambda_i \times max(0, V_i) \tag{5.7}$$

where X is a vector of integers, each of which is restricted to the form of the sum of several signed binary bits, such as $2^{-1} + 2^{-3} - 2^{-6}$. V_i here stands for the violation of each inequality constraints. In QMF Filter Bank Design, we use MaxQ's method (3.11) to transform the inequality constraints to equality constraints. The dynamic system (iterative process) to find solutions for this discrete constrained optimization problem is the same as the one specified in DLM.

Figure 5.2 shows our implementation of DLM for designing multiplierless filter banks. We explain each step in detail in this section.

5.3.1 Generating a Starting Point

There are two alternatives in selecting a starting point (Line 1 in Figure 5.2): using the parameters of an existing PO2 QMF filter bank, or using a discrete approximation of an existing QMF filter bank with real coefficients. The first alterative is not always possible because not many such filter banks are available in the literature. So we focus on the second alternative.

In the second approach, we first transform the real coefficients of the best-known design to PO2 forms using CSD representation. Given a real coefficient and b, the maximum number of ONE bits to represent the coefficient, we first apply Booth's algorithm [3] to represent consecutive 1's using

Table 5.1: Comparison of a PO2 filter bank obtained by truncating the real coefficients of Johnston's 32e QMF filter bank [26] 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

two ONE bits and then truncate the least significant bits of the coefficients. This approach generally allows a number to be represented in a few ONE bits. As an example, consider a binary fixed-point number 0.10011101100. After applying Booth's algorithm and truncation, we can represent the number in 2 ONE bits :

$$0.10011101100 \quad \underset{\text{Booth's Algorithm}}{\Longrightarrow} \quad 0.101000\bar{1}0\bar{1}00 \quad \underset{\text{Truncation}}{\Longrightarrow} \quad 2^{-1} + 2^{-3}.$$

Previous work [32, 40, 7] shows that scaling has a significant impact on the optimization of coefficients in PO2 filters. That is, if each coefficient is scaled properly before the search starts (based on a heuristic objective), the quality of the final design can be improved significantly. In our case, the performance of a PO2 filter obtained by truncating its real coefficients to a fixed maximum number of ONE bits is not as good as one whose real coefficients were first multiplied by a scaling factor. We illustrate this observation in the following example.

Consider Johnston's 32e filter bank [26] as a starting point. Table 5.1 shows the metrics of two PO2 filters: Filter Bank A was obtained by truncating each of the original coefficients to a maximum of 3 ONE bits, whereas Filter Bank B was obtained by multiplying each of the coefficients by 0.5565 before truncation. Filter Bank B performs better and is almost as good as the original design with real coefficients. In fact, a design that is better than Johnston's 32e design can be obtained by using Filter B as a starting point, but no better designs were found using Filter A. This example illustrates that multiplying the filter coefficients by a scaling factor changes the bit patterns of the coefficients, which can improve the quality of the starting point when the coefficients are truncated.

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: $sum := \sum_{i=1}^{5} w_i \times g_i$
6. **if** $(sum < LeastSum)$ **then**
7. $LeastSum := sum$
8. $BestScale := ScaleFactor$
9. **endif**
10. **endfor**
11. **return** *BestScale*

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

To find the best scaling factor, we enumerate over different scaling constants and scale all the coefficients by a common constant before the search begins. Experiments also show that it is possible to find good designs without requiring the PO2 coefficients to have the same degree of precision as that of continuous coefficients. For instance, in our experiments, we restrict the minimum exponent of the ONE bits in each coefficient (in the range $[-1, 1]$) to be -22, even though the real coefficients have a minimum exponent of -31.

Figure 5.3 shows a simple but effective algorithm to find the proper scaling factor to be multiplied before the coefficients are truncated. We evaluate the quality of the resulting starting point by a weighted sum of its performance metrics. Since under most circumstances, the constraint on transition bandwidth is more difficult to satisfy, we give it a weight 100 and use 1 for the other four metrics. Note that our objective in finding a good scaling factor is different from that in previous work [32, 40, 7]. Further, note that the filter output in the final design will need to be divided by the same scaling factor.

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

coefficient is large. For instance, when the maximum number of ONE bits allowed is larger than 6, the performance of all the filters is nearly the same for all scaling factors.

As an illustration, consider the design of a PO2 QMF filter bank [52] based on Johnston's 32d design [26] as our constraints. Assuming a minimum exponent of -22 in each ONE bit, we enumerate and find the best scaling factor for all the coefficients to be 0.9474.

5.3.2 Updating variable X

The value of X is updated in Line 4 of DLM in Figure 5.2. There are two ways in which X can be updated: greedy update and hill climbing. In greedy updates, the update of X leading to the maximum improvement of $L_o(X, \lambda)$ in (3.15) is found before an update is made. This approach is very time consuming and may not lead to the best filter bank when DLM stops. On the other hand, in hill climbing, X is updated as soon as an improvement in $L_o(X, \lambda)$ is found. This approach is efficient and generally leads to good designs. For this reason, we use hill climbing as our update strategy.

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

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

For each elements $b_{i,j}$, we perturb it to be a new elements $b'_{i,j}$ that differs from $b_{i,j}$ by either the sign or the exponent or both, while maintaining $b'_{i,j}$ to be not the same in sign and exponent as another elements of the i^{th} coefficient. Using $b_{i,1}, \cdots, b_{i,j-1}, b'_{i,j}, \cdots, b_{i,\gamma}$ while keeping other coefficients the same, we compute the new value $L_o(X', \lambda)$ using (3.15) and accept the change if $L_o(X', \lambda) < L_o(X, \lambda)$.

5.3.3 Initializing and Updating λ

The value of λ is initialized in Line 2 of DLM in Figure 5.2. 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 5.2 is related to the condition when λ should be updated. In traditional Lagrangian methods on continuous variables, λ is updated in every iteration. This approach does

not work in DLM because if λ were updated after each update of X, then the search behaves like random probing and restarts from a new starting point even before a local minimum is reached. For this reason, λ for violated constraints should be updated less frequently, only when no further improvement in $L_o(X, \lambda)$ can be made in Line 4 of DLM for all the bits in all the coefficients. This is the approach we have taken in solving satisfiability problems [43, 50]. However, we have found that more frequent updates of λ may lead to better PO2 filters. In our implementation, we update λ every time three coefficients have been processed. Since λ is updated before all the filter coefficients have been perturbed, the guidance provided by λ may not be exact.

When updating λ before the search reaches a local minimum of $L_o(X, \lambda)$, we set c in Line 6 of Figure 5.2 to be a normalized value as follows:

$$c = \frac{\theta_{speed}}{\max_{i=1}^{n} g_i} \tag{5.8}$$

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 of $L_o(X, \lambda)$, perturbing all the bits in all the coefficients will result in no improvement of $L_o(X, \lambda)$. At this point, we need to update λ differently in order to bring the search out of the local minimum. This is done by choosing a proper value of c in Line 6 of DLM. If λ is increased too fast, then the search will restart from a random starting point. On the other hand, if λ is increased too slowly, then the trajectory remains in the current local minimum, and updates of X in the next iteration of DLM will bring the search to the same local minimum! Hence, we would like to set c so that it will bring the search out of the current local minimum in one step, and local descents in the next iteration will start in an adjacent local minimum. This means that, after λ has been changed to λ', there exists X' in the neighborhood of X such that

$$L_o(X, \lambda) \leq L_o(X', \lambda) \quad \text{and} \quad L_o(X', \lambda') < L_o(X, \lambda') \tag{5.9}$$

Replacing $L_o(X, \lambda)$ by $f(X) + \sum_{i=1}^{n} \lambda_i \times h_i(X)$ in (5.9), we get the condition before λ changes:

$$f(X) + \sum_{i=1}^{n} \lambda_i \times h_i(X) \leq f(X') + \sum_{i=1}^{n} \lambda_i \times h_i(X') \tag{5.10}$$

and that after λ_i is updated to $\lambda_i' = \lambda_i + c \times h_i(X)$:

$$f(X) + \sum_{i=1}^{n} (\lambda_i + c \times h_i(X)) \times h_i(X) > f(X') + \sum_{i=1}^{n} (\lambda_i + c \times h_i(X)) \times h_i(X') \tag{5.11}$$

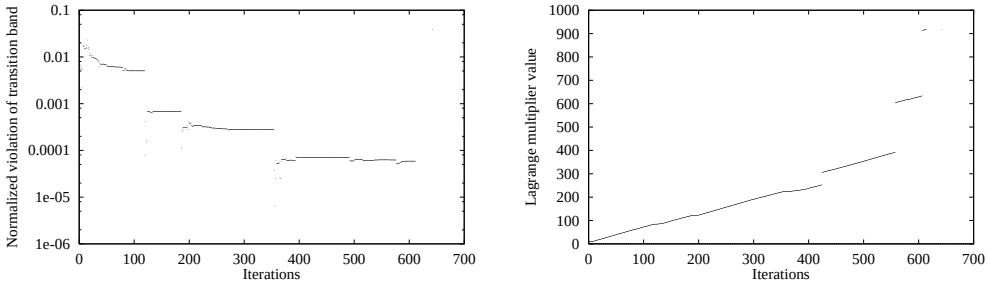

Figure 5.4: The violation values of T_t during a search (left) and the corresponding λ_{T_t} (right).

where $h_i(X')$ is the new violation value of the i^{th} constraint at X'. After transformations, we get

$$0 \leq L_o(X', \lambda) - L_o(X, \lambda) < c \times \sum_{i=1}^{n} h_i(X) \times (h_i(X) - h_i(X'))$$

$$\text{or} \quad c > \frac{L_o(X', \lambda) - L_o(X, \lambda)}{\sum_{i=1}^{n} h_i(X) \times (h_i(X) - h_i(X'))} \tag{5.12}$$

When c is large enough to satisfy (5.12) for all X', and the λ values are increased according to Line 6 of DLM in Figure 5.2, we are assured that in the next iteration, there exists a new X' that will cause the Lagrangian value to decrease.

As an example, consider in Figure 5.4a the violation of transition bandwidth T_t in a typical search based on the constraints derived from Johnston's 32e filter bank [26]. Figure 5.4a shows that the value of the 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 update of λ_{T_t} using c defined in (5.8) will result in a large number of iterations before the violation can be overcome. Using c defined in (5.12) to increase λ_{T_t}, we see in Figure 5.4b that λ_{T_t} jumps three times when the condition for updating λ was satisfied. These saved at least half of the total search time in order to find the solution.

Finally, we notice in Line 6 of DLM that λ is nondecreasing. This means that as the search progresses, the λ's grow, and more emphasis is placed on getting the search out of local minima. This approach fails when the λ's 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 and $\sum_{i=1}^{n} \lambda_i \times h_i$ and scale the λ's in order to keep the ratio within the following range.

$$0 \leq \frac{\sum_{i=1}^{n} \lambda_i \times h_i(X)}{f(X)} \leq \theta_{threshold} \tag{5.13}$$

If the ratio exceeds $\theta_{threshold}$, then we divide all the λ's by a constant r. In our experiments, we set $\theta_{threshold}$ to be 250 and r to be 1.3, and check $\theta_{threshold}$ every time when λ is updated.

Table 5.2: Multiplierless 32d QMF filter banks found by DLM with static and adaptive weights. (The objective is the reconstruction error E_r.)

Weight	Static Weight		Adaptive Weight	
w	Objective	Time (minutes)	Objective	Time (minutes)
10000.0	-	-	0.998	195.9
1000.0	-	-	0.998	168.3
100.0	-	-	0.885	277.2
10.00	-	-	0.883	119.3
1.00	-	-	0.836	190.8
0.1	0.87	197.1	0.837	161.4
0.01	0.87	115.2	0.87	124.3
0.001	0.87	4.8	0.87	34.5
0.0001	0.88	12.0	0.878	10.8
0.00001	0.9	23.7	0.924	12.0

5.4 Experimental Results

We have applied DLM to solve the QMF filter-bank design problems formulated by Johnston [26]. In this section, we compare the performance of PO2 QMF filter banks designed by DLM and those by Johnston [26], Chen *et al.* [7] , *Novel* [51], simulated annealing, and genetic algorithms.

Our goal is to find designs that are better than the baseline results across all six performance measures. Hence, we use (5.5) with the constraint bounds defined by those of the baseline designs.

5.4.1 Improvements Using Dynamic Weight Adaptation

In this section, we use a QMF benchmark design problem 32d as an example to demonstrate the functionality of our dynamic adaptation strategy (section 3.5).

The second and third columns of Table 5.2 show the objective-function values of the designs found and the corresponding convergence times of DLM with static weights. Two things can be observed from Table 5.2. One is that when initial weight for objective is too larger, it is very hard for search to converge or find a feasible solution. The other is that when initial weight for objective is too small, the search trajectory converges very fast however the quality of solution may not be

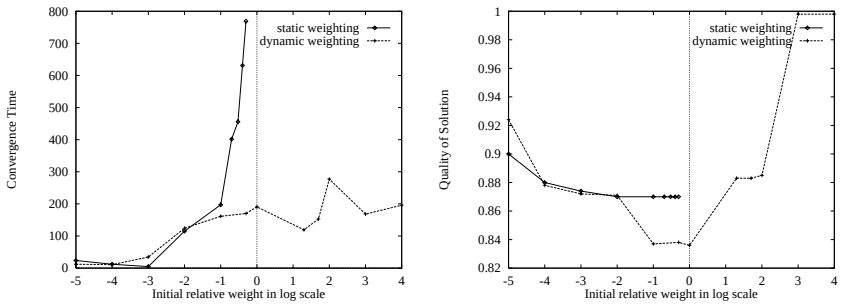

Figure 5.5: Comparison of convergence time and quality of solution between static weighting and dynamic weighting for multiplierless QMF design problem 32d

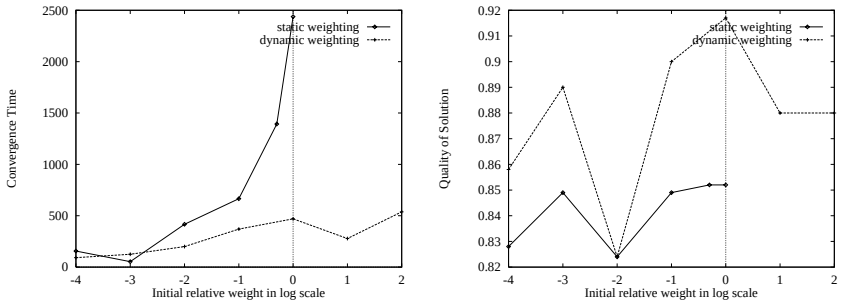

Figure 5.6: Comparison of convergence time and quality of solution between static weighting and dynamic weighting for multiplierless QMF design problem 48e

satisfying. Therefore, the relative weight between the objective function and constraints is also quite critical for Multiplierless QMF Filter Bank Designs.

After we have applied the dynamic weight adaptation strategy defined in Table 3.3, we have found significant improvement in convergence time. We allow the maximum number of ONE bits to be 6 and the minimum exponent to be -22 for each filter coefficient. The Lagrangian method uses both static weights and dynamic weights to solve 32d and 48e problems. We compare both the convergence time and the quality of solution in terms of reconstruction error. The starting points were obtained from Johnston's design, and the control parameters were the same as those used in the previous subsection except that the window size N_u is 10.

A comparison between DLM with static weights and DLM with dynamic weights is shown in Figures 5.5 and 5.6, respectively. Even though the initial weights have very large ranges, $[10^{-5}, 10^{4}]$ for 32d and $[10^{-4}, 10^{2}]$ for 48e, the dynamic weight-adaptation algorithm converges in less than 300 minutes for the 32d problem and 510 minutes for 48e. However, using DLM with static weights,

Table 5.3: Experimental results of DLM in solving QMF filter bank design problems, starting from six ONE-BIT expressions of scaled Johnston's solutions.

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

when the initial w is larger than 1.0, the search cannot converge within 15 hours for 32d and 32 hours for 48e.

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

We run DLM over all of the benchmark problems for significant long time (> 10 hours). Table 5.3 shows the results of solving all the Johnston's benchmarks using filter coefficients with a maximum of six ONE bit. Our results show that we were able to find designs that have better reconstruction errors, while the other performance metrics are either the same or better.

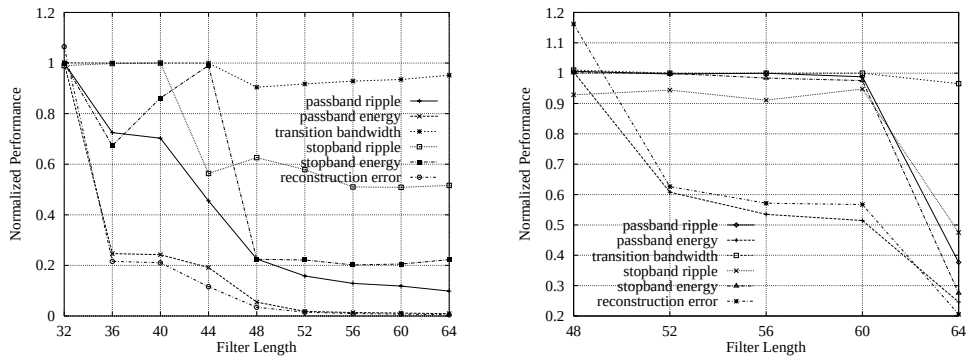

Figure 5.7: Normalized performance with respect to Johnston's 32e (left) and 48e (right) QMF filter banks [26] for PO2 filters with a maximum of 3 ONE bits per coefficient and different number of filter taps.

5.4.2 Comparison of DLM with other optimization methods in QMF Design

In this section, we compare the performance of PO2 QMF filter banks designed by DLM and those by Johnston [26], Chen *et al.* [7], *Novel* [51], simulated annealing, and genetic algorithms.

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

We have used closed-form integration to compute the performance values. In contrast, Johnston [26] used sampling to compute energies. Hence, designs found by Johnston are not necessarily at the local minima in a continuous sense. To demonstrate this, we applied local search in a continuous formulation of the 24D design, starting from Johnston's design. We found a design with a reconstruction error of 3.83E-05, which is better than Johnston's result of 4.86E-05. By applying global search, we can further improve the design to have a reconstruction error of 3.66E-05.

We have evaluated PO2 designs obtained by DLM with respect to Johnston's designs whose coefficients are 32-bit real numbers. Using the performance of Johnston's 32e design as constraints [26], we ran DLM from 10 different starting points obtained by randomly perturbing 1% of all the coefficients of Johnston's design [26]. Each run was limited so that each ONE bit of the coefficient was processed in a round robin fashion 400 times. We then picked the best solution of the 10 runs and plotted the result in Figure 5.7, which shows the normalized performance of PO2 designs with increasing number of filter taps, while each filter coefficient has a maximum of 3 ONE bits.

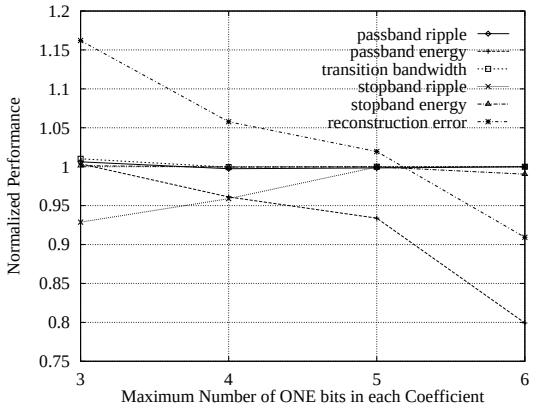

Figure 5.8: Normalized performance with respect to Johnston's 48e QMF filter bank [26] for PO2 filters with 48 taps and different maximum number of ONE bits per coefficient.

(The best design is one with the minimum reconstruction error if all the constraints are satisfied; otherwise, the one with the minimum violation is picked.) Our results show a design with 32 taps that is nearly as good as Johnston 32e's design. For filters with 32, 36, 40 and 44 taps, we used a starting point derived from Johnston's 32e design with filter coefficients first scaled by 0.5565 and truncated to a maximum of 3 ONE bits, and the filter coefficients of the remaining taps set to zeroes initially. Starting points for filters with longer than 44 taps were generated similarly, except that a scaling factor of 0.5584 was used instead. Our results show that, as the filter length is increased, all the performance metrics improve, except the transition bandwidth, which remains close to that of the benchmark design.

With respect to Johnston's 48e design [26], we set a limit so that each ONE 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 filters with 64 taps.) Our results show that our 48-tap PO2 design is slightly worse than that of Johnston's, while PO2 designs with 52 taps or longer have performance metrics 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.

Table 5.4: Comparison of normalized performance of PO2 filter banks designed by DLM with respect to those designed by Johnston, Chen, *Novel*, simulated annealing (SA), and genetic algorithms (EA-Ct and EA-Wt). Columns 2-4 show the performance of DLM using 3 ONE bits for 32-tap filters and 6 ONE bits for 64-tap filters normalized with respect to that of Johnston's 32e, 64d, and 64e filter banks [26]. Columns 5-6 show the performance of DLM using 3 ONE bits normalized with respect to that of Chen *et al.*'s 64-tap and 80-tap filter banks [7]. Columns 7-10 show the performance of 32-tap filter banks designed using *Novel* [51], SA, and EA, normalized with respect to that of Johnston's 32e filter bank and using Johnston's design as constraints.

	Johnston's			Chen *et al.*		Other Search Methods wrt 32e			
Metrics	DLM–32e	DLM–64d	DLM–64e	DLM–64	DLM–80	*Novel*	SA	EA-Ct	EA-Wt
E_r	0.83	0.90	0.89	0.91	0.95	0.712	0.500	0.724	0.507
E_p	1.00	0.82	0.83	0.80	0.96	0.896	0.582	0.905	0.590
E_s	1.00	1.00	1.00	1.00	0.86	1.000	1.000	1.000	0.999
δ_p	1.00	0.97	1.00	1.00	1.00	1.000	1.000	1.000	0.997
δ_s	0.99	0.75	1.00	1.00	1.00	1.000	1.000	1.000	0.999
T_t	1.00	1.00	1.00	1.00	1.00	1.000	1.013	1.000	1.013

In the next set of experiments, we kept the same number of taps as Johnston's 48e design and increased the maximum number of ONE bits in each coefficient from 3 to 6. We set a limit so that each ONE bit of the coefficient was processed in a round-robin fashion 800 times, and ran DLM once from the truncated Johnston's 48e design. Figure 5.8 shows a design that is better than Johnston's 48e design when the maximum number of ONE bits per coefficient is 6. In this case, the reconstruction error is 91% of Johnston's 48e design. (The scaling factors used are 0.5584 for 3 bits, 0.8092 for 4 bits, 0.7409 for 5 bits, and 1.0 for 6 bits.)

With respect to Johnston's 64d and 64e designs, Table 5.4 shows improved PO2 designs obtained by DLM using a maximum of 6 ONE bits per coefficient and 64 taps. No improvements were found when the maximum number of ONE bits is less than 6.

Table 5.4 further shows improved PO2 designs obtained by DLM with respect to Chen *et al.*'s PO2 designs (with 3 ONE-bit coefficients) with 64 and 80 taps, respectively, and a maximum of 3

ONE 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 ONE bit was processed in a round-robin fashion 1,000 times.

Finally, we compare in Table 5.4 the performance of 32e PO2 filter banks obtained by DLM with a maximum of 3 ONE bits per coefficient, and those obtained by *Novel*, simulated annealing (SA), and evolutionary algorithms (EAs). *Novel* uses a continuous trace function to bring a search out of local minima rather than restarting the search from a new starting point when the search finds a feasible design. The SA we have used is SIMANN from netlib that works on a weighted-sum formulation. The EA is Sprave's Lice (Linear Cellular Evolution) that can be applied to both constrained and weighted-sum formulations. SA and EA-Wt use weighted-sum formulation with weight 1 for reconstruction error and weight 10 for the remaining metrics. EA-Ct works on the same constrained formulation as defined in (5.4). All methods were run significantly long with over 10 hours on a SUN SPARC20 workstation in each run.

We have tried various parameter settings and report the best solutions in Table 5.4. *Novel* improves Johnston's solutions consistently. SA results in larger transition bandwidths than Johnston's. EA-Wt has difficulty in improving over Johnston's design across all measures. In particular, the solutions of EA-Wt have larger transition bandwidth. EA-Ct also finds a solution that improves Johnston's design across all measures, although it is not as good as the found found by *Novel*. Finally, DLM obtained a design that improves the reconstruction error while the other metrics are either exactly the same or slightly worse than those of Johnston's.

Considering the facts that *Novel*, SA, EA-Wt, and EA-Ct use continuous coefficients that will need either a complex carry-save adder or a 32-bit multiplier in each tap in their hardware implementations, and that the DLM design uses a maximum of five additions in each tap, the designs obtained by DLM are significant improvements over previous designs.

5.5 Summary

To summarize, using DLM to design multiplierless powers-of-two (PO2) QMF filter banks is unique because it starts from a constrained formulation with the objective of finding a design that improves over a benchmark design. In contrast, existing methods for designing PO2 filter banks can only obtain designs with different trade-offs among the performance metrics and cannot guarantee that

the final design is always better than the benchmark design with respect to all the performance metrics.

In the experimental results, DLM has obtained better PO2 filter banks than other discrete optimization methods. Overall, this approach is effective because it finds designs with very few ONE bits for each filter coefficient, allowing a cost-effective design to be implemented.

Chapter 6

Application 3-Solving Mixed Integer Optimization Problems

In this chapter, we apply DLM to solve mixed integer optimization problems. These problems have both continuous and discrete variables and are formulated as follows:

$$\begin{array}{lll} \text{minimize} & f(x,y) & \\ \text{subject to} & h(x,y) = 0 & x = (x_1, x_2, \ldots, x_{n_1}) \\ & g(x,y) \leq 0 & y = (y_1, y_2, \ldots, y_{n_2}) \end{array} \tag{6.1}$$

where $f(x,y)$ is the objective function, $g(x,y) = [g_1(x,y), \ldots, g_k(x,y)]^T$ is a k-component vector representing inequality constraints, $h(x,y) = [h_1(x,y), \ldots, h_m(x,y)]^T$ is an m-component vector representing equality constraints, x is a vector of real variables, and y is a vector of discrete variables. In the general form, $f(x,y)$, $g(x,y)$ and $h(x,y)$ are nonlinear functions that are either continuous or discrete.

In order to apply DLM to solve the problem defined in (6.1), we define the neighbourhoods in mixed integer space. Thus, the first-order method in discrete space can be carried out. Also, we propose two heuristics, *exponential probing* to handle the precision problem for continuous variables in a mixed integer space, and *relax-and-tighten* to handle hard-to-satisfy equlity constraints, respectively.

6.1 Previous Work

The problem defined in (6.1) is generally hard to solve because existing methods that solve either continuous or discrete problems alone cannot be applied. New methods that are a hybrid of both have been developed. In this section we describe briefly their limitations.

(a) *Generalized Benders Decomposition (GBD)* [13, 17, 2]. This method is used to solve mixed-integer optimization problems under some convexity assumptions. Its basic idea is to generate in each iteration an upper bound on the solution sought by solving a *primal* problem and a lower bound on a *master* problem. The primal problem corresponds to the original optimization problem with fixed discrete variables; the solution of which provides information on upper bounds and Lagrange multipliers associated with the equality and inequality constraints. The master problem is derived via nonlinear duality theory, making use of Lagrange multipliers obtained in the primal problem. Its solution provides information on lower bounds, as well as the set of fixed discrete variables to be used in the next primal problem. As the process iterates, it is shown that the sequence of upper bounds are non-increasing, that the sequence of lower bounds are nondecreasing, and that the sequences converge in a finite number of iterations.

One major disadvantage of GBD-like methods is that they are applicable to only a subclass of the constrained mix-integer optimization problems with some restrictions on the variable space. For example, GBD requires the continuous subspace to be a nonempty and convex set and the objective and constraint functions to be convex.

(b) *Outer Approximation (OA)* [11, 10]. The method is similar to GBD except that the master problem is formulated using primal information and outer linearization. OA has some restrictions on the problems it can solve. It requires the continuous subspace to be a nonempty, compact, and convex set, and the objective and constraint functions to be convex in the continuous subspace.

(c) *Generalized Cross Decomposition (GCD)* [13, 20, 21, 39]. The method iterates between two phases: Phase 1 that solves the primal and dual subproblems, and Phase 2 that solves the master problem. Phase 1 solves the primal subproblem similar to that in GBD and provides an upper bound on the solution of the original optimization problem and Lagrange multipliers for the dual subproblem. The dual subproblem provides a lower bound on the solution of the original problem and supplies solutions to the discrete variables of the primal subproblem. After solving the primal and the dual subproblems, a primal convergence test is applied on the discrete variables, while a

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

(d) *Genetic algorithms (GA)* [31, 55, 41, 27, 41, 8, 4]. The basic idea is to construct a fitness function, usually based on a weighted sum of constraints and penalties, and to apply genetic operators to minimize the penalty function. A general formulation may use a binary representation to represent real variables or a real-value representation to represent variables whose precision and range are not known beforehand. The genetic operators used include crossovers and mutations. Crossovers take two chromosomes and create a new one based on a weighted sum. If the original variable is discrete, the newly created chromosomes is then truncated to discrete. Non-uniform mutations [34] may be used for manipulating floating-point numbers and for genes that are required to be integers. As the search aims at minimizing the penalty function, the overall penalty will gradually decrease as constraint violations are suppressed in the search. The idea of minimizing a penalty function is also used in simulated annealing [56].

The major issue in using penalty functions is that a good solution can be very difficult to find if penalties were not chosen properly. In general, hard-to-satisfy constraints should carry larger penalties than easy-to-satisfy ones. However, one constraint may shift between easy and hard depending on other constraints in the problem. Without the ability to change penalties dynamically, penalty-based methods do not work well.

In short, existing methods are restricted in solving general nonlinear mixed-integer optimization problems. Some Lagrange multiplier-based algorithms, like GBD, OA, and GCD, have restrictions on the functions or the variable space they can handle, and can only be applied to a subclass of the problems. Moreover, the decomposed subproblems generated by these methods are not trivial to solve. Stochastic methods, like GA and SA, can be applied to solve general mixed-integer problems. However, they are not effective in handling hard-to-satisfy constraints because they rely on penalty functions that use a fixed penalty for each constraint. Without the ability to dynamically change the penalties, it is difficult to focus the search on hard-to-satisfy constraints.

To address the drawbacks in existing methods, we propose to apply DLM to solve mixed-integer problems. The use of Lagrange multipliers in DLM allows it to dynamically vary Lagrange

multipliers according to constraint violations. However, continuous variables in the original mixed-integer problem have to be transformed into discrete before DLM can be applied. In the following section, we present two such transformations.

6.2 Neighborhoods in Mixed Integer Space

In this section, we define the neighborhood of real variables so that discrete gradients can be computed and the next point in the trajectory, identified. To this end, we need to restrict the number of neighboring points for a point in the mixed-integer space to be finite so that DLM can be applied. There are two alternatives here.

(a) *Finite-digit approximation of real numbers.* In the first representation, we use a finite number of digits to represent a real number, thereby restricting its number of neighbors. For instance, π, a number that requires infinite precision to represent, can be approximated in three decimal digits (3.14) or six digits (3.14159). The different representations will impact the precision of the final result. By representing a real number using a finite number of digits, we can define a finite set of neighboring points. For example, the neighbors of 1.2 can be defined to be $\{0.2, 2.2, \ldots, 9.2, 1.0, 1.1, 1.3, \ldots, 1.9\}$. The following definition specifies this representation.

Definition 9. Given point x in continuous space represented in k digits $(d_1, d_2, \ldots, d_k)$, the neighborhood of x, $\mathcal{N}_c(x)$, is defined as follows:

$$\begin{aligned}\mathcal{N}_c(x) \;=\; & \bigcup_{t_1 \in S_1} \{(d_1 + t_1) \mod 10, d_2, \ldots, d_k\} \bigcup \bigcup_{t_2 \in S_2} \{d_1, (d_2 + t_2) \mod 10, d_3, \ldots, d_k\} \\ & \bigcup \cdots \bigcup \bigcup_{t_k \in S_k} \{d_1, d_2, \ldots, d_{k-1}, (d_k + t_k) \mod 10\} \end{aligned} \tag{6.2}$$

where $S_1, S_2, \ldots, S_k$ are sets of user-defined signed or unsigned digits.

The sets $S_1, \ldots, S_k$ are user specified. For example, $S_1 = \{-1, +1\}$, $S_2 = \{-2, -1, +1, +2\}$, etc. In practice, we found empirically that more variations should be given to less significant digits and less variations given to more significant digits. That is, set S_k should be larger than or equal to S_{k-1}. This allows the search to be performed more thoroughly in the close vicinity of the given point and less in regions further away.

(b) *Real representation without approximation.* A second alternative is to represent real numbers as they are but define their neighborhoods to be of a finite set. This representation allows DLM to be used in a mixed-integer space without the approximation of real variables. The following definition formalizes this idea.

Definition 10. The neighborhood of point x in continuous representation is defined as follows:

$$\mathcal{N}_c'(x) = \bigcup_{t \in S} \{x + t\} \tag{6.3}$$

where S is a set of real-value numbers.

Combining Definitions 1, 9, and 10, we define the neighborhood in mixed-integer space as follows.

Definition 11. Given point (x, y) in mixed-integer space, where x represents the continuous subspace and y the discrete subspace,

$$\mathcal{N}(x,y) = \begin{cases} \mathcal{N}_c(x) \ \cup \ \mathcal{N}_d(y) & \text{finite digit approximation, or} \\ \mathcal{N}_c'(x) \ \cup \ \mathcal{N}_d(y) & \text{real representation} \end{cases} \tag{6.4}$$

where $\mathcal{N}_c$ and $\mathcal{N}_c'$ are, respectively, the sets of neighboring points in continuous subspace x defined in Definitions 9 and 10, and $\mathcal{N}_d$ is the set of neighboring points in discrete subspace y defined in Definition 1.

Example 2. Consider a mixed-integer space with two dimensions, one discrete and the other continuous. Using the two representations for continuous variables, the neighborhoods of point $(5, 2.3)$ are:

$$\mathcal{N}(5,2.3) = \begin{cases} \{(6,2.3),(4,2.3)\} \bigcup \{(5,1.3),(5,3.3)\} \bigcup \{(5,2.0),(5,2.1), & \\ \qquad (5,2.2),(5,2.4),\ldots,(5,2.9)\} & \text{finite digit rep.} \\ \{(6,2.3),(4,2.3)\} \bigcup \{(5,2.3+2.0),(5,2.3+1.0),(5,2.3+0.5), & \\ \qquad (5,2.3-0.5),(5,2.3-1.0),(5,2.3-2.0)\} & \text{real rep.} \end{cases}$$

Although the two definitions of neighborhoods in mixed-integer space are different, they both try to identify a finite set of neighboring points in continuous space so that DLM can be applied. Since DLM enumerates all neighboring points in finding the gradient to move the search trajectory, gradients can be found by enumeration rather than differentiation, and the objective and constraint functions are not required to be continuous or convex. However, it also means that the size of the neighborhood must be chosen properly to result in high-quality solutions and efficient search. It is not possible to define the neighborhood of x as a uniform sampling of the space around x, as the number of neighboring points grows exponentially with respect to the distance from x. A compromise between the size of the neighborhood and the number of points in it is to have more sample points closer to x and less sample points further away. This strategy permits a search trajectory to traverse in one step to a distant point without enumerating an exponential number of neighboring points.

Our proposed approach of using a fixed precision for each real variable may be limited because it is not possible to reach a solution point whose precision is beyond the precision defined in the neighborhood. For instance, if we use four digits to represent a neighborhood and the local minimum requires eight digits to represent, then it is not possible for a search trajectory to reach the local minimum. The point with limited precision reached by the trajectory can, of course, be used as a starting point by other methods to locate the exact local minimum.

To overcome this issue, we have developed a strategy called *exponential probing* that extends the precision (number of digits) of a real variable without increasing substantially the search complexity. Specifically, at a given point (x, y), if all the neighboring points of continuous variable x_i have been evaluated without finding any improvement in the Lagrangian value, then points in the set $\{x_i + 2^j\delta\}, j = 1, 2, 3, \ldots$ are tried, where δ is a very small positive real number. (In our experiments, setting $\delta = 10^{-10}$ was good for most problems.) Starting at $j = 1$, the procedure computes the Lagrangian value for the updated $x_i^{'}$ position, while keeping constant the other x variables, y and λ, until no improvement is found in the Lagrangian value. This step stops if a new $x_i^{'}$ with better Lagrangian value is found; otherwise, the procedure is repeated using the set $\{x_i - 2^j\delta\}$, $j = 1, 2, 3 \ldots$. This method extends the set of neighboring points by adding some exponentially

1. Set positive real constant c_1 and c_2
2. Set time constraint
3. Generate a starting point (x, y)
4. Set initial value of λ
5. **while** search does not converge or time constraint is not reached **do**
6. update (x, y) to (x', y') only if this will result in $L_m((x', y'), \lambda, \mu) < L_m((x, y), \lambda, \mu)$
7. **if** condition for updating λ is satisfied **then**
8. $\lambda_i := \lambda_i + c_1 \times h_i$
9. $\mu_i := \mu_i + c_2 \times \max(0, g_i)$
10. **end if**
11. **if** condition for adjusting w is satisfied **then**
12. call dynamic weight adaptation
13. **end if**
14. **end while**

Figure 6.1: DLM for mixed-integer optimization

increasing offsets to the original continuous variable. Note that the set of neighboring points probed are not predefined because probing stops whenever no improvement is found.

Exponential probing can lead to finer search and better solutions. As an example, using DLM (described in the next section) without exponential probing to solve the discretized Problem 2.7.1 (from [14]) results in a solution value of -391.383 (best of 100 runs), whereas the best known solution of -394.75 was found using exponential probing.

6.3 DLM for Solving Mixed-Integer Problems

The Lagrangian function in mixed-integer optimization problems is defined similarly as before except that the variable space is mixed. We denote L_m as the Lagrangian function in mixed-integer space:

$$L_m(x, y) = w\, f(x, y) + \lambda^T h(x, y) + \mu^T \max(0, g(x, y)) \tag{6.5}$$

Figure 6.1 shows DLM for solving general nonlinear mixed-integer constrained optimization problems. In Line 3, a starting point is generated in the mixed-integer space. In our experiments,

we generated 100 starting points randomly in the search space and evaluated the average solution quality and the speed of the search. The initial values of all Lagrange multipliers are set to zeroes in Line 4. The search process will continue until the search converges or time is expended, as shown in Line 5. In Line 6, we evaluate all possible neighboring points of (x, y) in order to find improvements in its Lagrangian value. We try the variables of the space, $x_1, \ldots x_{n_1}, y_1, \ldots, y_{n_2}$, in a round-robin fashion, one variable at a time, and compare the Lagrangian values of the neighboring points of the chosen variable. To save time, we apply a greedy strategy rather than a hill climbing strategy, switching from one variable to the next once improvement in the Lagrangian value is found with respect to the first variable. In our experiments, we used Definition 10 to specify the neighborhood of a real variable, since it was more convenient to implement. The size of a neighborhood is problem dependent: more points will be contained in the neighborhood if the variable has a larger range. If no improvement in the Lagrangian value is found, we apply exponential probing described in the last subsection. In Lines 7-9, the Lagrange multipliers are updated when certain conditions are met. Also, if certain conditions for adjusting w in the Lagrangian function are satisfied, the dynamic weight-adaptation strategy is used to balance the relative weight between the objective and the constraints (Lines 11-13).

In mixed problems, equality constraints are sometimes hard to satisfy. The reasons are that equality constraints generally have fewer solutions than inequality constraints, and that a continuous variable in an equality constraint must have high precision in order to satisfy the constraint. For instance, an equality constraint $\sin(x) = 0$ will only have feasible solutions at $x = j\pi$ where j is integral. Hence, the more precision x has, the better the equality constraint can be satisfied.

To apply DLM to solve problems with hard equality constraints, we use a method called *relax-and-tighten*. The key idea is to first relax equality constraints $h(x) = 0$ to inequality constraints $|h(x)| \leq \varepsilon$, where ε is a positive value to be determined. The relaxed problem should be easier to solve by DLM as it has a larger feasible space. After obtaining a feasible point x^* for the relaxed problem, we update ε to $\min(maxVio, \varepsilon)/d$, where d is a positive value larger than 1, and $maxVio$ is the maximal violation of the current solution x^* with respect to the original problem with equality constraints. In this way, the relaxed equality constraints are tightened step by step until all the equality constraints are satisfied.

Intuitively, the step to relax the equality constraints increases the size of the feasible region of the original problem, making it easier to solve. The amount of relaxation, however, must be chosen carefully. If the relaxed feasible region is too large (initial ε is too large), then starting points located in this region may not be useful in locating the final solutions that satisfy the equality constraints. In contrast, if the relaxed feasible region is too small, then the relaxed equality constraints are almost as hard to satisfy as the original equality constraints, making the relaxed problem too difficult to solve. Empirically, we found that setting ε to $MaxVio_0/d$ works well, where d is in the range $[1.08, 1.2]$, and $MaxVio_0$ is the initial maximal violation of the equality constraints. Similarly, the step to tighten the relaxed constraints is equally important. If the step size is too small, then it will take a long time to find a feasible solution that satisfies the equality constraints. In contrast, if the step size is too large, then DLM may not be able find a feasible solution. Experimentally, we found that using the same step size d as in the relaxation step works well.

The step size d used in our experiments was fixed in our experiments. The performance of our relax-and-tighten step can probably be improved by using a dynamic step size in accordance to the difficulty in satisfying the set of equality constraints. We will investigate this alternative in the future.

As an illustration, consider Problem 5.2 in [14] as an example. With 46 variables, 36 equality constraints, and an initial maximum violation ($MaxVio_0$) of 24.0, the problem is quite difficult to solve. If we apply DLM directly to solve a discretized version of the problem from a randomly generated starting point (see next section for the discretization process), the best solution found has a maximal violation of 0.65 to all the constraints after running two hours on a Pentium Pro 200 MHz computer. By applying the *relax and tighten* strategy, a solution with maximal violation of 0.0000075 was found in the same amount of CPU time. Our strategy allowed better solutions with smaller maximal violations be found after each time the constraints were tightened.

6.4 Experimental Results

Since large scale nonlinear mixed-integer benchmarks are not readily available, we construct our benchmarks from continuous ones[14]. Given the best-known solutions for these continuous benchmarks, we set those variables with integral solution values to be discrete. If there is no variable

with integral solution, we choose some variables randomly and make them discrete. When we set variables with continuous solutions to be discrete, the final solution obtained may be different from the original continuous solution. The variables chosen are then discretized into 501 evenly spaced points in their allowed ranges. For instance, the discrete points for a variable in the range between 0 and 10 are $\{j \times 0.02\}$ for all integers $j \in [0, 500]$.

Table 6.1 lists the complete experimental results of applying DLM to solve the constructed mixed-integer optimization problems. The second (resp. third) column shows the total number of constraints (resp. equality) in the original continuous formulations. The fourth (resp. fifth) column shows the total number of variables (resp. variables that have integral solutions) in the original continuous formulations. The number of variables with integral solutions varies greatly from problem to problem. For instance, there are no variables with integral solutions in Problem 4.6, whereas only two variables among the 46 variables in Problem 5.2 have integral solutions. On the other hand, more than a half of the variables in Problem 7.3 have integral solutions.

The seventh and eighth columns show the number of continuous and integral variables in the discretized problem used in our experiments. As described earlier, we first assign *a priori* the number of integral variables in our discretized problem. If this number is larger than the number of variables that have integral solutions (Column 5), then we select randomly some continuous variables and discretize them. If this number is small than the value in Column 5, then we select variables randomly from the set of variables with integral solutions and discretize them.

The ninth column shows the results on the best out of 100 runs from randomly generated starting points. These results should be compared with those in the sixth column that shows the best-known solution for the continuous formulations. Our results shown that DLM was able to find the same solutions as in the continuous case for most problems. Moreover, it was able to improve the solutions for Problems 2.8 and 4.5.1, respectively. However, it was not able to find good solutions for Problems 5.2, 5.4 and 7.2. The search space of these three problems have a lot of local minima, and a discretized search like DLM may not be able to find a good minimum in a continuous space.

Finally, the last two columns show the percentage of starting points in DLM that lead to feasible solutions, and the average computation time per starting point when DLM was run on a Pentium Pro 200 MHz computer with the Linux operating system.

6.5 Summary

In this chapter, we extend DLM to solve nonlinear mixed-integer benchmark problems. Traditionally, Lagrangian methods for solving mixed-integer problems usually require certain convexity conditions and limitations on their search space. By first representing continuous variables using a discrete representation and by developing a search procedure to refine the discretized search space, DLM is able to solve general mixed-integer problems using a Lagrangian approach. We further propose heuristics to handle hard-to-satisfy constraints in the problem. The experimental results show that DLM is able to find the same solution to many of the benchmark problems from randomly generated starting points.

Table 6.1: Experimental results of applying DLM to solve mixed-integer optimization benchmark problems

Problem	Constraints		Variables		Best Orig.	Discretized Problems		Best DLM	Feasibility	Avg.
ID	Total	Equality	Total	Integral	Solution	Continuous	Integral	Solution	Ratio	Time
2.1	11	0	5	5	-17.0	3	2	-17.0	99	0.02
2.2	2	0	6	5	-213.0	3	3	-213.0	97	0.02
2.3	35	0	13	13	-15.0	8	5	-15.0	95	0.01
2.4	17	0	6	6	-11.0	3	3	-11.0	96	0.02
2.5	31	0	10	7	-268.0	5	5	-268.0	99	0.03
2.7.1	50	0	20	15	-394.75	10	10	-394.75	96	3.8
2.7.2	50	0	20	15	-884.75	10	10	-884.75	98	2.8
2.7.3	50	0	20	15	-8695.0	10	10	-8695.0	96	2.4
2.7.4	50	0	20	15	-754.75	10	10	-754.75	97	4.5
2.7.5	50	0	20	14	-4150.4	10	10	-4150.4	95	3.7
2.8	10	10	24	24	15990	12	12	15639	100	4.0
3.1	22	0	8	2	7049.25	4	4	7049.25	100	0.7
3.2	16	0	4	2	-30665.5	2	2	-30665.5	100	1.1
3.3	18	0	6	6	-310	3	3	-310	99	0.5
3.4	9	0	3	3	-4.0	2	1	-4.0	99	0.9
4.3	9	1	4	3	-4.51	2	2	-4.51	98	0.9
4.4	9	1	4	3	-2.217	2	2	-2.217	95	1.3
4.5	15	3	6	5	-11.96	3	3	-13.40	100	1.0
4.6	6	0	2	0	-5.51	1	1	-5.51	97	1.4
4.7	5	1	2	0	-16.74	1	1	-16.74	95	1.2
5.2	86	36	46	2	1.567	23	23	1.92	70	39
5.4	58	26	32	25	1.86	16	16	2.28	74	45
6.2	17	4	5	5	400.0	3	2	400.0	95	3.5
6.3	17	4	5	5	600.0	3	2	600.0	92	3.8
6.4	17	4	5	4	750.0	3	2	750.0	94	4.2
7.2	41	13	16	16	56285.0	8	8	57298.8	80	21.0
7.3	64	19	27	19	46266.0	14	13	46266.0	85	38.0
7.4	90	23	38	14	35920.0	19	19	35920.0	82	44.0

Chapter 7

Conclusions

In this research, we have extended Lagrangian theory that works for continuous problems to solve discrete problems. In continuous space, we have shown the limitations of search methods based on the first-order conditions. These methods cannot find some of the local minima subject to constraints in the solution space. In discrete space, we have proposed a new definition of discrete gradients. Starting from the concept of discrete saddle points, we have proved two first-order conditions that are necessary for discrete local minima satisfying constraints. When all the constraint functions are non-negative, we have further proved the property that there is a one-to-one correspondence between local minima subject to constraints and discrete saddle points. This means that the discrete-space first-order conditions are more powerful than their counter-parts in continuous space. Based on the first-order conditions, we have proposed Discrete Lagrangian method (DLM), a procedure to look for discrete saddle points.

To test DLM, we first applied it to discrete benchmark problems which are transformed from a set of continuous benchmark problems. Using all these discrete problems, we fully test DLM both in the convergence time and final solution quality. Experimental results show that DLM is both good at handling constraints and finding high quality solutions. Morever, the convergence speed is fast for a large range of initial relative weights between the objective and the constraints, which demonstrates the robustness of DLM.

We have applied DLM to design multiplierless powers-of-two (PO2) QMF filter banks. This method is unique because it starts from a constrained formulation with the objective of finding a design that improves over a benchmark design. In contrast, existing methods for designing PO2

filter banks can only obtain designs with different trade-offs among the performance metrics and cannot guarantee that the final design is always better than the benchmark design with respect to all the performance metrics. Expemental results show that DLM has obtained better PO2 filter bank designs than other discrete optimization methods.

We have extended DLM to solve nonlinear mixed-integer problems. Traditionally, Lagrangian methods for solving mixed-integer problems usually require certain convexity conditions and limitations on their search space, whereas penalty-based methods are more general but have difficulties in constraint satisfaction. By first representing continuous variables using a discrete representation and by developing a search procedure to refine the discretized search space, DLM is able to solve general mixed-integer problems using a Lagrangian approach. We have further proposed heuristics to cope with equality constraints in the problem. Since benchmarks in this area are not readily available, we have created them from nonlinear constrained optimization problems. Our results show that DLM was able to find the same solution to many of these problems from randomly generated starting points, a task that is difficult even for nonlinear continuous optimization algorithms.

In short, our work on discrete Lagrangian theory and discrete Lagrangian search represent a significant advance in the state-of-the-art in this area.

Bibliography

[1] 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.

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

[3] A. D. Booth. A signed binary multiplication technique. *Quart. J. Mech. Appl. Math.*, 4:236–240, 1951.

[4] Y. J. Cao and Q. H. Wu. Mechanical design optimization by mixed-variable evolutionary programming. *Proc. 1997 IEEE International Conference on Evolutionary Computation*, pages 443–6, 1997.

[5] R. A. Caruana and B. J. Coffey. Searching for optimal FIR multiplierless digital filters with simulated annealing. *Technical report, Philips Laboratories*, 1988.

[6] 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.

[7] 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.

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

[9] C. D. Creusere and S. K. Mitra. A simple method for designing high-quality prototype filters for m-band pseudo QMF banks. *IEEE Trans. on Signal Processing*, 43(4):1005–1007, April 1995.

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

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

[12] 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.

[13] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization.* Topics In chemical engineering. Oxford University Press, 1995.

[14] 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.

[15] M. R. Garey and D. S. Johnson. *Computers and intractability, a guide to the theory of NP-Completeness.* W.H.Freeman and company, 1979.

[16] R. S. Garfinkel and G. L. Nemhauser. *Integer Programming.* John Wiley & Sons, New York, NY, 1972.

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

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

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

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

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

[22] J. J. Hopfield and D. W. Tank. Neural computation by concentrating information in time. In *Proc. National Academy of Sciences*, volume 84, pages 1896–1900, Washington, D.C., 1987. National Academy of Sciences.

[23] 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.

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

[25] V. K. Jain and R. E. Crochiere. Quadrature mirror filter design in the time domain. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 32(2):353–361, April 1984.

[26] J. D. Johnston. A filter family designed for use in quadrature mirror filter banks. *IEEE Proc. of Int'l Conf. on ASSP*, pages 291–294, 1980.

[27] J. A. Joines and C. R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's. *Proc. of IEEE Int'l Conf. on Evolutionary Computing*, pages 579–584, 1994.

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

[29] 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.

[30] 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.

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

[32] 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.

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

[34] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1994.

[35] 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.

[36] T. Q. Nguyen. Digital filter bank design quadratic-constrained formulation. *IEEE Trans. on Signal Processing*, 43(9):2103–2108, September 1995.

[37] A. V. Oppenheim and R. W. Schafer. *Discrete-time signal processing.* Prentice Hall, 1989.

[38] J. G. Proakis. *Digital Signal Processing: Principles, Algorithms, and Applications.* Maxwell Macmillan, 1992.

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

[40] 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.

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

[42] 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.

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] P. P. Vaidyanathan. *Multirate Systems and Filter Banks*. Printice-Hall Inc., 1993.

[50] B. W. Wah and Y. Shang. A discrete Lagrangian-based global-search method for solving satisfiability problems. *Satisfiability Problem: Theory and Applications*, pages 365–392, 1997.

[51] B. W. Wah, Y. Shang, T. Wang, and T. Yu. Global optimization of qmf filter-bank design using novel. *Proc. Int'l Conf. on Acoustics, Speech and Signal Processing*, 3:2081–2084, 4 1997.

[52] B. W. Wah, Y. Shang, and Z. Wu. Discrete Lagrangian method for optimizing the design of multiplierless QMF filter banks. In *Proc. Int'l Conf. on Application Specific Array Processors*, pages 529–538. IEEE, July 1997.

[53] B. W. Wah, Y. Shang, and Z. Wu. Discrete Lagrangian method for optimizing the design of multiplierless QMF filter banks. *Journal of Global Optimization*, December 1997.

[54] B. W. Wah and T. Wang. Efficient and adaptive lagrange-multiplier methods for nonlinear continuous global optimization. *J. of Global Optimization*, (accepted to appear) 1998.

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

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

Vita

Zhe Wu received his B.E. degree from the Special Class for Gifted Young, University of Science & Technology in 1996. He is expecting a M.S. degree in Computer Science from the University of Illinois at Urbana-Champaign in May, 1998.

His interests include optimization, efficient algorithms design, software engineering and computer networks.