2014

Xqx Based Modeling For General Integer Programming Problems

Vijay P. Ramalingam
University of Mississippi

XQX BASED MODELING FOR GENERAL INTEGER PROGRAMMING PROBLEMS

A Dissertation
presented in partial fulfillment of requirements
for the degree of Doctor of Philosophy
in the School of Business Administration
The University of Mississippi

by

Vijay P. Ramalingam

December 2014

ABSTRACT

We present a new way to model general integer programming (IP) problems with inequality and equality constraints using XQX. We begin with the definition of IP problems followed by their practical applications, and then present the existing XQX based models to handle such problems. We then present our XQX model for general IP problems (including binary IP) with equality and inequality constraints, and also show how this model can be applied to problems with just inequality constraints. We then present the local optima based solution procedure for our XQX model. We also present new theorems and their proofs for our XQX model. Next, we present a detailed literature survey on the 0-1 multidimensional knapsack problem (MDKP) and apply our XQX model using our simple heuristic procedure to solve benchmark problems. The 0-1 MDKP is a binary IP problem with inequality constraints and variables with binary values. We apply our XQX model using a heuristics we developed on 0-1 MDKP problems of various sizes and found that our model can handle any problem sizes and can provide reasonable quality results in reasonable time. Finally, we apply our XQX model developed for general integer programming problems on the general multi-dimensional knapsack problems. The general MDKP is a general IP problem with inequality constraints where the variables are positive integers. We apply our XQX model on GMDKP problems of various sizes and find that it can provide reasonable quality results in reasonable time. We also find that it can handle problems of any size and provide feasible and good quality solutions irrespective of the starting solutions. We conclude with a discussion of some issues related with our XQX model.

DEDICATION

To my late father P.K. Ramalingam.

ACKNOWLEDGEMENTS

I am highly indebted to advisor Prof. Bahram Alidaee for his patience and his belief in me. He has been very supportive and very kind throughout my doctoral studies. His continuous support and encouraging words have been a source of constant inspiration.

I would like to thank my work supervisor Prof. Mustafa Altinakar for his invaluable support in finishing up this work.

I would also like to thank my former supervisor Dr. Thomas Fink who was extremely supportive during my tenure at the National Center for Physical Acoustics.

Thanks to my dissertation committee members, Drs. Dawn Wilkins, Hugh Sloan and Cesar Rego for being accommodative and for their suggestions to improve this work.

Finally, thanks to my family, friends and well wishers who encouraged me to finish this work.

TABLE OF CONTENTS

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGEMENTS iv

LIST OF FIGURES viii

LIST OF TABLES ix

1 INTRODUCTION 1

2 XQX MODELING FOR IP PROBLEMS 4

3 0-1 MULTIDIMENSIONAL KNAPSACK PROBLEM 67

4 GENERAL MULTIDIMENSIONAL KNAPSACK PROBLEM 93

5 CONCLUSIONS 116

6 FUTURE DIRECTIONS 118

BIBLIOGRAPHY 121

VITA 128

LIST OF FIGURES

FIGURE NUMBER Page

2.1 All possible conditions for variable update. 21

2.2 Stencil used for calculating change in function value with respect to any variable. 44

2.3 Change in Δf_x when: $x_i = 0$ and $x_i^\star \leq 0$. 50

2.4 Change in Δf_x when: $x_i = 0$ and $x_i^\star > 0$. 51

2.5 Change in Δf_x when: $x_i > 0$ and $x_i^\star \leq 0$. 52

2.6 Change in Δf_x when: $x_i > 0$ and $x_i^\star > 0$. 53

2.7 Change in Δf_s when: $s_i = 0$ and $s_i^\star \leq 0$. 59

2.8 Change in Δf_s when: $s_i = 0$ and $s_i^\star > 0$. 60

2.9 Change in Δf_s when: $s_i > 0$ and $s_i^\star \leq 0$. 61

2.10 Change in Δf_s when: $s_i > 0$ and $s_i^\star > 0$. 62

LIST OF TABLES

TABLE NUMBER Page

2.1 Penalties for inequality constraints. 8

2.2 Alternate Representations for IP problems. 41

2.3 Updating integer type unknown variable based on critical value. 54

2.4 Updating integer type slack variable based on critical value. 63

2.5 Updating boolean type variable based on partial derivative. 65

3.1 Results for: $m = 5$, $n = 100$ (Chu & Beasley Instances) 86

3.2 Results for: $m = 10$, $n = 100$ (Chu & Beasley Instances) 86

3.3 Results for: $m = 30$, $n = 100$ (Chu & Beasley Instances) 87

3.4 Results for: $m = 5$, $n = 250$ (Chu & Beasley Instances) 87

3.5 Results for: $m = 10$, $n = 250$ (Chu & Beasley Instances) 88

3.6 Results for: $m = 30$, $n = 250$ (Chu & Beasley Instances) 88

3.7 Results for: $m = 5$, $n = 500$ (Chu & Beasley Instances) 89

3.8 Results for: $m = 10$, $n = 500$ (Chu & Beasley Instances) 89

3.9 Results for: $m = 30$, $n = 500$ (Chu & Beasley Instances) 90

3.10 Results for: Other large instances (Glover & Kochenberger Instances) 90

4.1 Results for: $m = 10, n = 100$. (Generated Instances) 104

4.2 Results for: $m = 10, n = 250$. (Generated Instances) 104

4.3 Results for: $m = 10, n = 500$. (Generated Instances) 105

4.4 Results for: $m = 100, n = 100$. (Generated Instances) 105

4.5 Results for: $m = 100, n = 250$. (Generated Instances) 105

4.6 Results for: $m = 100, n = 500$. (Generated Instances) 106
4.7 Results for: $m = 200, n = 100$. (Generated Instances) 106
4.8 Results for: $m = 200, n = 250$. (Generated Instances) 106
4.9 Results for: $m = 200, n = 500$. (Generated Instances) 107
4.10 Results for: Other large instances (Glover & Kochenberger Instances) 107
4.11 Results for: Other large instances with $P1 = 100$, $P2 = 500$, and $P3 = 1000$ (Glover & Kochenberger Instances) 108
4.12 % Difference with CPLEX for other large instances with $P1 = 100$, $P2 = 500$, and $P3 = 1000$ (Glover & Kochenberger Instances) 109
4.13 Results for: Other large instances with $R1 : x_i = 0$, $R2 : x_i \sim U[0, 1]$, and $R3 : x_i \sim U[1, 5]$ (Glover & Kochenberger Instances) 110
4.14 % Difference with CPLEX for other large instances $R : x_i = 0$, $R2 : x_i \sim U[0, 1]$, and $R3 : x_i \sim U[1, 5]$ (Glover & Kochenberger Instances) 111

CHAPTER 1

INTRODUCTION

The objective of this work is to present a new way to model general integer programming (IP) problems, which contains both inequality and equality constraints using XQX. Our new model allows any general IP which contains only inequality constraints, or a combination of inequality and equality constraints to be modeled using XQX. The XQX method or the unconstrained quadratic programming method allows re-casting a problem and this method has been in existence since 1968 (Kochenberger & Glover, 2006). The re-casting method is similar to a Quadratic Lagrangian relaxation method where the constraints are combined with the objective function value to create a single function and this function is used to solve for optimality.

Many real life applications such as train scheduling, airline crew scheduling, production planning, electricity generation planning, telecommunication networks, and many others can be modeled as IP problems. The fundamental procedure (Wolsey, 1998) to model an IP is first to define the necessary variables, then use these variables to define a set of feasible constraints, and finally use these variables to define the objective function. If needed, additional variables or constraints are defined if difficulties arise. To solve difficult problems, the literature has considered additional modeling techniques such as linear relaxations, combinatorial relaxations, Lagrangian relaxations, duality, and XQX to increase the efficiency of the solution procedures. To solve IP models, the solutions procedures available in literature can be broadly classified in to two groups namely exact procedures and heuristic procedures. Methods such as brach & bound, dynamic programming, and cutting plane are some of the

exact solution methods, and these methods have been used effectively to solve small IP problems with a limited number of variables. The branch & bound procedure to solve problems with a large number of variables n creates an exponential growth of 2^n subproblems (Bosch & Trick, 2005), hence making it unsuitable for such problem types. To handle problems with a large number of variables, exact solution procedures are coupled with heuristics such as Annealing, tabu search, genetic algorithm and others to effectively solve large problems within a reasonable amount of time. Readers are referred to Jünger *et al.* (2008), Wolsey (1998), and Chen *et al.* (2011) for details of the above modeling and solution methods since their review is out of our scope.

The XQX based modeling for general IP problems, which forms the basis of this thesis, has been applied to solve many IP problems with binary variables ($x_i \in \{0, 1\}$) and equality constraints, and a very limited number of binary variables with inequality constraints were also considered in the literature (Kochenberger *et al.*, 2004).

In our extensive literature search, the XQX based modeling for IP problems with inequality and equality constraints, or problems with just inequality constraints, does not exist. The current literature on XQX only considers problems with binary variables and equality constraints. Also, the XQX method we propose to model IP problems with binary variables and inequality constraints (binary IP problems) is different from the existing methods available.

1.1 Problem Statement

Our contribution to the existing literature with this work is as follows:

- Model a general IP problem and binary IP problem using XQX. The model developed is very general, which allows inequality, equality, or a combination of both.
- Local optima based solution procedures for the XQX model for general and binary IP problems.

- To apply our XQX model for binary IP problems, we solve benchmark instances of the 0-1 multidimensional knapsack problem using a simple heuristics developed based on XQX model. The variables in this knapsack problem are boolean ($x_i \in \{0, 1\}$), and the constraints can be equalities or inequalities.

- To apply our XQX model for general IP problems, we solve benchmark instances of general multidimensional knapsack problem using a simple heuristics developed based on XQX model. The variables in this knapsack problem are integers ($x_i \in \mathbb{Z}_+$), and the constraints can be equalities or inequalities.

1.2 Overview of the Remainder of Thesis

In Chapter 2, we begin with the definition of the IP problem, provide real world applications of IP problems, and then present the existing XQX based modeling approach for IP problems. We then present our detailed derivations of our XQX model for general and binary IP problems with equality and inequality constraints. Finally, we present our local optima based solution procedures for general and binary IP problems based on the XQX model obtained.

In Chapter 3, we begin with the definition of the 0-1 multidimensional knapsack problem, describe the real life applications for this problem, and provide a detailed literature review on the state of the art algorithms. We then apply our XQX model for binary IP to solve benchmark instances using a basic heuristics developed based on our model, and finally present the results from solving benchmark problems.

In Chapter 4, we define and provide a detailed literature survey for the general multidimensional knapsack problem, including its application in real life. We then apply our XQX model for general IP to solve benchmark instances using a basic heuristics developed based on our model, and finally present the results from solving benchmark problems.

Finally in Chapter 5, we present our conclusions from this work and identify possible extensions.

CHAPTER 2

XQX MODELING FOR IP PROBLEMS

In this chapter, we begin with an overview of IP problems and provide few examples of their application in various industries. We then describe the XQX and neural network method currently available in literature to model IP problems. Detailed derivations of our XQX model for the general and binary IP problems are presented next. Finally we present the solution procedures for the general and binary IP problems.

2.1 Integer Program

The IP problems are a class of constrained optimization problem where the variables or unknowns are required to be integers. The IP problems are among the most important combinatorial optimization models in management science and have many real life applications such as design and implementations of supply chain management. The IP problems are also much harder to solve when compared with a linear program where the unknowns are real instead of integer values. Following are some of the successful integer programming applications in various industries (Chen *et al.*, 2011):

- Airline

  IP model to determine least-cost crew.

  IP algorithm to build flight crew schedules.

  Network optimization model to help reduce delays caused in air traffic.

  IP based system to generate optimal crew recovery solutions.

- Railway

  IP based algorithm for distributing empty cars and planning locomotive use.

  IP model to develop alternative set of scheduling routes.

- Telephone

  IP based optimization tool to improve productivity.

  IP based decision support to design robust fiber optic networks.

- Automobile

  IP model to shorten planning process and establish global procedures.

  Create network tools to reduce logistics cost.

- Energy

  IP and network managing tools to manage a system of reservoirs.

  IP based algorithm to reduce fuel costs for power generation.

One can see that IP problems arise in various walks of life, and also provide several millions of dollars savings when they are modeled and solved. We now proceed to formally define an IP problem and then present the XQX based modeling currently available in literature.

2.2 Definition of IP

The following definition of the IP problem is stated from Wolsey (1998):

"Suppose we have a linear program (a maximization problem)

$$\max\{dx : Ax \leq b, x \geq 0\}, \tag{2.1}$$

where A is an m by n matrix, d an n-dimensional row vector, b an m-dimensional column vector, and x an n-dimensional column vector of variables or unknowns. If all the variables are integers, written as

$$\begin{aligned} \max \quad & dx \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \text{ and integer,} \end{aligned} \tag{2.2}$$

then the above form (otherwise known as canonical form) is called an (Linear) Integer Program. If the variables in (2.2) are restricted to 0 or 1, then it is known as the Binary Integer Program." A standard form of the above IP problem is obtained by adding slack variables, which converts the inequality constraints to equality constraints.

2.3 Existing XQX Based Modeling Methods

The existing XQX based models only consider binary variables with equality constraints. A few inequality constrained IP problems with binary variables have been modeled using penalties, which is different from what we propose. Also, the neural network based model (same as XQX) only considers binary variables with equality constraints, and in some cases the binary variables are treated as continuous between 0 and 1.

The XQX based modeling, also known as the unconstrained quadratic program, refers to the following model after the problem has been transformed (Kochenberger *et al.*, 2004):

$$\text{UQP: min } f(x) = xQx. \tag{2.3}$$

Here Q is an n x n matrix of constants and x is an n-vector of binary variables. The following example from Kochenberger *et al.* (2004) shows how a transformation is performed:

$$\begin{aligned} \min \quad & x_0 = xQx \\ \text{subject to} \quad & Ax = b \\ & x \in \{0,1\}, \end{aligned} \tag{2.4}$$

where the the objective function is recasted as a diagonal Q matrix; x_j^2 equals x_j when x_j is binary. The constraint is then written as:

$$P(Ax-b)^\top(Ax-b), \tag{2.5}$$

where P is a large positive number for the penalty. The term in equation (2.5) is added to the objective function from (2.4), and the following is obtained:

$$\begin{aligned} x_0 &= xQx + P(Ax-b)^\top(Ax-b) \\ &= xQX + xDx + c \\ &= x\hat{Q}x + c, \end{aligned} \tag{2.6}$$

where the D matrix and the constant c result from the matrix multiplication in (2.5). Additional equality constraints in a problem are handled similarly as shown in (2.5) and are added to the objective function. Note that, if the problem is a maximization problem, then the constraints are subtracted from the objective function instead of being added. The tutorial presented by Kochenberger & Glover (2006) considers equality constrained IP problems with binary variables and special cases of inequalities with binary variables. To handle inequalities the authors propose penalty functions, shown in Table 2.1, where each inequality constraint is converted to the equivalent penalty and then added to the objective function (Alidaee *et al.*, 2008). A recent Ph.D. thesis by Wang (2014) presented metaheuristics for unconstrained binary quadratic problems. One problem considered in their work is the maximum clique problem, which is a problem with inequality constraints and binary variables. To obtain XQX transformation, penalty based methods shown in Table

2.1 were applied. Note that the penalty based method only works when the coefficients of variables in the constraints are equal to 1. In other cases, the penalty based transformation does not work. A very recent survey on XQX by Kochenberger *et al.* (2014) suggests the same penalty based transformations given in Table 2.1.

Classical Constraint	Equivalent Penalty
$x + y \leq 1$	$P(xy)$
$x + y \geq 1$	$P(1 - x - y + xy)$

Table 2.1: Penalties for inequality constraints.

2.3.1 Neural Network Based Models

The neural network based method to re-cast the problem is exactly the same as the XQX method (an example problem shown by Smith (1999) is exactly the same when compared to the transformation shown in (2.6)), and the final term in (2.6) is referred to as energy function in neural network terminology. The application of neural networks to provide solutions to NP-hard problems was proposed by Hopfield & Tank (1985) and is known as the discrete Hopfield neural network method. The authors showed that combinatorial optimization problems such as travelling sales man (TSP) can be solved using neural networks, and the lowest value of the energy function corresponds to the best path found for the TSP problem. The review done by Smith (1999) notes that equality constrained IP problems with binary variables have been successfully modeled using the neural network method, and the case of inequalities with binary variables was handled using the continuous (by treating binary variables as real values between 0 and 1) neural network approach. In neural network based procedures, the main method used to convert a constrained problem to an unconstrained problem is the exterior penalty function as stated by Monfared & Etemadi (2006) and Wen *et al.* (2009). A term is added to the objective function to penalize the violation of constraints in the exterior penalty method. The original contribution from Abe

et al. (1992) describes a method to handle inequality constraints for Hopfield neural network. The proposed method handles inequality constraints by multiplying the right hand side value by a variable after which the energy function is calculated. By using Eigen values and an algorithm to determine proper weights for the energy function, the authors solve a small instance of knapsack problem. The work done by Yamamoto *et al.* (1995) shows an asymmetric neural network to solve inequality constrained optimization problems. This method adds one neutron (variable) to each of the already existing neutrons which is then used to create an energy function. This method depends on these additional neurons to converge to zero in order to obtain a feasible solution. A generalized Boltzmann machine concept was proposed by Vaithyanathan *et al.* (1994) to solve multidimensional knapsack problems (MDKP). The authors have shown how to map MDKP into the Boltzmann machine and solve them using probability distribution generated from the states of neurons. The method suggested by Talaván & Yáñez (2006) solves a generalized quadratic knapsack problem using continuous Hopfield network. In this problem the variables or unknowns are booleans, but the slack variables considered are continuous between 0 and 1. In a similar approach, Lee & Hsu (1989) presented a neural network based solution method for MDKP. Again, they have considered the variables to be continuous between 0 and 1, and an additional term to represent the variables was added to the energy function. Ohlsson *et al.* (1993) proposed a neural network based method to solve the MDKP. Their method does not consider any slack variables and the penalty for each constraint is adjusted according to the amount of violation. This method tends to produce final solutions that violate constraints and is of limited practical importance as noted by Fréville (2004).

2.4 Existing Solution Procedures for XQX

To solve the model obtained in (2.6), techniques such as tabu search, genetic algorithm, path relinking, and scatter search, among other methods, have been applied in the literature. Lewis *et al.* (2005) solved their XQX model for uncapacitated task allocation problem using

tabu search based on the use of strategic oscillation. Alidaee *et al.* (2008) also used the tabu search method with strategic oscillation to solve their XQX model for set packing problems. Douiri & Elbernoussi (2012) modeled the maximum independent set problem as XQX and solved using genetic algorithm. Wang *et al.* (2012c) proposed a solution procedure for the XQX model based on path relinking procedure. Wang *et al.* (2012b) presented a memetic based algorithm to solve large instances of the XQX models.

Apart from directly applying heuristics on the XQX model, the literature has also considered applying gradient descent procedure on the model and then applying heuristics to obtain the best solution. For example, the solution method considered by Wang *et al.* (2012a) combines gradient descent with estimation of distribution algorithm and tabu search to solve maximum diversity problems modeled as XQX. The gradient descent method is a first-order optimization algorithm which is used to find the local minimum or local maximum based on the gradient obtained. In gradient descent approach, the partial derivative with respect to each variable is calculated, and the variable that gives the lowest derivative value (highest derivative value in maximization problems) is assigned the value of 1.

To apply gradient descent to the XQX model obtained earlier, consider the expanded form of the $\hat{Q}$ matrix from (2.6),

$$\begin{aligned} x_0 =& x_1(x_1q_{11} + x_2q_{21} + \cdots + x_nq_{n1}) + \\ & x_2(x_1q_{12} + x_2q_{22} + \cdots + x_nq_{n2}) + \\ & \vdots \\ & x_n(x_1q_{1n} + x_2q_{2n} + \cdots + x_nq_{nn}) + c. \end{aligned} \tag{2.7}$$

The partial derivative with respect to each variable is:

$$\begin{aligned}
\frac{\partial x_0}{\partial x_1} &= q_{11} + x_2 q_{21} + \cdots + x_n q_{n1}, \\
\frac{\partial x_0}{\partial x_2} &= x_1 q_{12} + q_{22} + \cdots + x_n q_{n2}, \\
&\vdots \\
\frac{\partial x_0}{\partial x_n} &= x_1 q_{1n} + x_2 q_{2n} + \cdots + q_{nn}.
\end{aligned} \tag{2.8}$$

Note that the term x_n^2 is represented as x_n in (2.8) since the variables are binary.

Starting with an initial solution, the gradient descent approach helps to choose the variable to be added or removed based on the partial derivatives obtained. Even though this provides a feasible solution, the best solution is obtained by coupling this variable selection procedure with tabu search or other heuristic algorithms. To conclude this section, the XQX model and the solution procedures in the existing literature are limited to binary problems with equality constraints. In the next section, we present our XQX model for the general IP problems, and the local optima based solution procedures for the same.

2.5 XQX Model for General IP Problems

In this section we formally present our XQX based model for general IP problems, and this is a new contribution to the existing literature and also the first objective of this work. The proposed XQX based model for general IP problems is presented for problems with both inequality and equality constraints. Formal proof for the XQX model and its solution procedures are presented in this section, and this forms the basis for the rest of this thesis. In section 2.6, we provide detailed derivations of our model using example problem sets.

We define the following to aid the formal presentation of our XQX model:

- n, number of variables
- m_1, number of inequality constraints
- m_2, number of equality constraints

- $m = m_1 + m_2$, total number of constraints
- X, a vector of variables of n integer components
- S, a vector of slack variables of m_1 components
- $f(X, S)$, value of objective function evaluated at variables X and slack variables S (sometimes written as $f(X)$)
- D (also d), a vector of n components
- A_1, an m_1 by n matrix
- A_2, an m_2 by n matrix
- $A = A_1 \cup A_2$
- a_{ij}, the ij-th component of A
- a_j, the j-th column vector of A
- B (also b), a column vector of m numbers
- B_1, a column vector of m_1 numbers
- B_2, a column vector of m_2 numbers
- $\mathbb{I}_k$, a k x k identity matrix
- e_k, k-th column vector of $\mathbb{I}_k$
- $a \cdot b$ (also ab), dot product of two vectors a and b
- L, an n x n symmetric matrix
- K, a finite set

- P, a large enough constant

Suppose we have a general integer program **P1** (a minimization problem), and the objective of this problem is to find the a vector $X = (X_1, \ldots, X_n)$ to satisfy **P1**.

$$\begin{aligned} \textbf{(P1)}\ \text{Min } f(X) = \quad & DX \\ \text{subject to} \quad & A_1X \leq B_1 \\ & A_2X = B_2 \\ & X \in K, \text{ and integer,} \end{aligned} \tag{2.9}$$

where we let $K = \{X_j \colon l_j \leq X_j \leq u_j, \text{ for } j = 1, \ldots, n\}$, where $0 \leq l_j \leq u_j$.

We have the following equivalent problem without inequality constraints by adding slack variables S to the first set of constraints in **P1**.

$$\begin{aligned} \textbf{(P1)}\ \text{Min } f(X) = \quad & DX \\ \text{subject to} \quad & A_1X + S = B_1 \\ & A_2X = B_2 \\ & X \in K, \text{ and integer,} \\ & S \geq 0. \end{aligned} \tag{2.10}$$

Let $Y = \{X \in K, \text{ and integer} \colon A_1X \leq B_1, \text{ and } A_2X = B_2\}$ and $\overline{Y} = K \backslash Y$ complement of Y.

Define a penalty function $L(X)$ by

$$L(X) = \begin{cases} 0, & \text{if } X \in Y. \\ \geq L^* > 0, & \text{if } X \in \overline{Y}. \end{cases} \tag{2.11}$$

Here L^* in the definition of $L(X)$ is a constant number. In the literature, different alternatives for $L(X)$ have been proposed. Two such cases are quadratic penalties which are defined as follows:

$$\begin{aligned} L_1(X) &= \sum_{i=1}^{m_1} \max\left(0, \sum_{j=1}^{n} a_{ij}X_j - b_i\right)^2 + \sum_{i=1}^{m_2}\left(\sum_{j=1}^{n} a_{ij}X_j - b_i\right)^2. \\ L_2(X) &= \sum_{i=1}^{m_1}\left(\sum_{j=1}^{n} a_{ij}X_j + S_i - b_i\right)^2 + \sum_{i=1}^{m_2}\left(\sum_{j=1}^{n} a_{ij}X_j - b_i\right)^2. \end{aligned} \tag{2.12}$$

Lemma 1: Given a vector $(X_1, \ldots, X_n)$, define each slack variable $S_k (k = 1, \ldots, m_1)$ by

$$\begin{aligned} S_k^* &= b_k - \sum_{j=1}^{n} a_{kj}X_j, \text{ and} \\ S_k &= \begin{cases} S_k^*, & \text{if } S_k^* \geq 0, \\ 0, & \text{otherwise,} \end{cases} \end{aligned} \tag{2.13}$$

then, $L_1(X)$ and $L_2(X)$ are the same.

Proof of the lemma is straightforward, and we omit it here. In this research we use the penalty function under the condition where lemma 1 is satisfied.

Given a constant $P \geq 0$, the *quadratic dual problem* of **P1**, namely **P2**, can be defined as follows:

$$\begin{aligned} \textbf{(P2)} \quad & \text{Min } F(X, P) = f(X) + PL(X) \\ & \text{s.t. } X \in K, \text{and integer.} \end{aligned} \tag{2.14}$$

It is well known that under suitable conditions problem **P2** is an exact method for problem **P1**, i.e., solution of **P2** solves problem **P1** (Luenberger, 1973), (Luenberger & Ye, 2008), (Sinclair, 1986), (Li & Sun, 2006), and (Ben Hadj-Alouane & Bean, 1997). Some of the useful results from the literature can be summarized as follows:

(a) Let X^* be the optimal solution to **P1**. Given a constant $P \geq 0$, let $\overline{X}$ be optimal solution to **P2**. We then have
$F(\overline{X}, P) = f(\overline{X}) + PL(\overline{X}) \leq f(X^*)$.

(b) Given a series of constant numbers $P_k \geq 0$ (for $k = 1, 2, \ldots$) such that $P_k < P_{k+1}$, let X_k be the optimal solution for **P2** for P_k. The following relations hold true under this condition:

(1) $F(X_k, P_k) \leq F(X_{k+1}, P_{k+1})$

(2) $L(X_k) \geq L(X_{k+1})$

(3) $f(X_k) \leq f(X_{k+1})$

(4) $f(X_k) \leq F(X_k, P_k) \leq f(X^*)$

(c) Let $\underline{f}$ be a lower bound of $Min_{X \in K} f(X)$ and $\gamma > 0$ be a lower bound of $Min_{X \in K \backslash Y} f(X)$. Suppose $K \backslash Y \neq \varnothing$, then, there exists a P_0 such that for any $P > P_0$, any solution X^* that solves **P2** also solves **P1** and $F(X^*, P) = f(X^*)$. A value of P_0 is given by

$$P_0 = \frac{f(X^*) - \underline{f}}{\gamma}. \tag{2.15}$$

(d) Let $\overline{f}$ be an upper bound of the objective function of **P1**. If constraints $A_1 X + S - B_1$ and $A_2 X - B_2$ are integer-valued on K, then for any $P > P_0 = \overline{f} - \underline{f}$, any solution X^* that solves **P2** also solves **P1** and $F(X^*, P) = f(X^*)$ (where $L(X)$ is defined by quadratic form.)

Note that in (c) the value of P_0 depends on $f(X^*)$, which is the optimal solution of the original problem, and thus it is not easy to find. In practice, however, it is (d) that is being used. Also note that the quadratic penalty defined above is same as

$$L(X) = (A_1 X + S - B_1)^\top (A_1 X + S - B_1) + (A_2 X - B_2)^\top (A_2 X - B_2). \tag{2.16}$$

Let $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ and $B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$, and $\mathbb{I}_{m_1}$ to be an m_1 x m_1 identity matrix. We then have an equivalent optimization problem given in proposition 1.

Proposition 1: Problem (**P2**) can be written in the equivalent form

$$\textbf{(P2)} \quad \text{Min } F(X,P) = DX + P(A_1X + S - B_1)^\top (A_1X + S - B_1) + P(A_2X - B_2)^\top (A_2X - B_2) \tag{2.17a}$$

$$= DX + P \begin{bmatrix} X & S \end{bmatrix} \begin{bmatrix} A^\top A & A_1^\top \\ A_1 & \mathbb{I}_{m_1} \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix} - 2P \begin{bmatrix} (B^\top A) & (B_1) \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix} + PB^\top B \tag{2.17b}$$

$$\text{s.t. } X \in K, \text{and integer},$$

$$S \geq 0.$$

Proof: An equality i in the form $A_1X + S = B_1$ can be written as

$$a_{i1}X_1 + \cdots + a_{ij}X_j + a_{in}X_n + S_i = b_i.$$

This constraint in the objective function appears as

$$P(a_{i1}X_1 + \cdots + a_{in}X_n + S_i - b_i)^2.$$

Carrying the quadratic operation we get

$$P[(a_{i1}^2X_1^2 + \cdots + a_{in}^2X_n^2 + S_i^2 + b_i^2) + (2a_{i1}a_{i2}X_1X_2 + \cdots + 2a_{i1}a_{in}X_1X_n) +$$
$$(2a_{i1}X_1S_i + \cdots + 2a_{in}X_nS_i) - (2a_{i1}b_iX_1 + \cdots + 2a_{in}b_iX_n + 2S_ib_i)].$$

Similar quantity can be given regarding an equality i in the form $A_2X = B_2$. Expanding the quantity over all $i = 1, \ldots, m$, and adding DX to the objective function, we have the desired result. A detailed derivation for this proposition is provided in section 2.6. Corollary 1 follows from proposition 1.

Corollary 1: Problem (**P2**) can be written in the equivalent form

$$\textbf{(P2)} \quad \text{Min } F(X,P) = P \begin{bmatrix} X & S \end{bmatrix} \begin{bmatrix} a_1a_1 & \cdots & a_1a_n & a_1e_1 & \cdots & a_1e_{m_1} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_na_1 & \cdots & a_na_n & a_ne_1 & \cdots & a_ne_{m_1} \\ e_1a_1 & \cdots & e_1a_n & e_1e_1 & \cdots & e_1e_{m_1} \\ \vdots & & \vdots & \vdots & & \vdots \\ e_{m_1}a_1 & \cdots & e_{m_1}a_n & e_{m_1}e_1 & \cdots & e_{m_1}e_{m_1} \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix}$$

$$+ \begin{bmatrix} d_1 - 2Pba_1 & \cdots & d_n - 2Pba_n & -2Pbe_1 & \cdots & -2Pbe_{m_1} \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix} + Pbb \tag{2.18}$$

$$\text{s.t. } X \in K, \text{and integer},$$

$$S \geq 0.$$

Proof: Use of Proposition 1 and applications of dot products.

Let $C = \begin{bmatrix} d_1 - 2Pba_1 & \cdots & d_n - 2Pba_n & -2Pbe_1 & \cdots & -2Pbe_{m_1} \end{bmatrix}$,
and Q the matrix in the problem, then equivalently we can write **P2** in a compact form as follows:

$$\textbf{(P2)} \text{ Min } F(X,P) = CX + P\,[X,S]\,Q\,[X,S] \tag{2.19}$$

$$\text{s.t. } X \in K, \text{and integer},$$

$$S \geq 0.$$

Although the objective function of **P1** includes a linear DX, it should be clear that we can also consider a quadratic objective function as $DX + XLX$ where L is an n x n symmetric matrix. In that case, after carrying P into matrix Q, we can add each ij-th element L_{ij} of L to ij-th element of Q and consider dual problem **P2** to solve **P1**.

Below we present theoretical results that help to implement a solution procedure to **P2**. Since the matrix Q is symmetric, with no loss of generality we can carry penalty P into the matrix and multiply the strict upper triangular part of Q by 2, and concentrate on the upper triangular matrix Q. The following definitions are used in the results.

For $j = 1, \ldots, n$ define:

$$C_j = d_j - 2Pba_j, \tag{2.20a}$$

$$M_j = 2P\left(\sum_{i=1, i\neq j}^{n} a_j a_i X_i + \sum_{i=1}^{m_1} a_j e_i S_i\right), \tag{2.20b}$$

$$q_{jj} = Pa_j a_j \geq 0, \tag{2.20c}$$

$$Z^* = \frac{-(C_j + M_j)}{2q_{jj}}, \tag{2.20d}$$

$$\Delta f^* = \frac{-(C_j + M_j)^2}{4q_{jj}} - (C_j + M_j + q_{jj}X_j)X_j, \tag{2.20e}$$

$$\frac{\partial \Delta f}{\partial Z} = 2q_{jj}Z + M_j + C_j, \tag{2.20f}$$

$$Z_1 = X_j, \tag{2.20g}$$

$$Z_2 = -X_j - \frac{C_j + M_j}{q_{jj}}. \tag{2.20h}$$

For $k = 1, \ldots, m_1$ define:

$$C_{n+k} = -2Pbe_k = -2Pb_{n+k}. \tag{2.20i}$$

Note that in these definitions we have $a_j e_i = a_{ij}$ and $q_{jj} \geq 0$. Below it will be seen that the positivity of element $q_{jj} \geq 0$ has implications in the development of results.

Proposition 2: Given a vector $(X, S) = (X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_{m_1})$, if a component X_j is changed to X_j', the value of the objective function will change by

$$\Delta f = f(X', S) - f(X, S) = (X_j' - X_j)\left[C_j + M_j + q_{jj}(X_j' + X_j)\right]. \tag{2.21}$$

Proof: Substitute $(X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_{m_1})$ and $(X_1, \ldots, X_j^{'}, \ldots, X_n, S_1, \ldots, S_{m_1})$ in the objective function and cancel common terms, we then have the desired results.

Note 1: Given a vector (X, S), when X_j is changed, the amount of change Δf is a quadratic function of one variable, Z, given by

$$\Delta f = (Z - X_j)\left[C_j + M_j + q_{jj}(Z + X_j)\right]. \tag{2.22}$$

The amount of change is $\Delta f = 0$ when $Z_1 = X_j$, or, $Z_2 = -X_j - \frac{C_j + M_j}{q_{jj}}$.

Proposition 3: The quadratic function Δf is convex. For any component j $(j = 1, \ldots, n)$, the critical point is given by Z^*, and the amount of change at Z^* is equal to Δf^*.

Proof: Using note 1 we have Δf, a quadratic function of one variable Z. Convexity of Δf now follows from the fact that $q_{jj} = P a_j a_j \geq 0$. Taking derivative of the quadratic function with respect to Z (i.e., with respect to component j), we have

$$\frac{\partial \Delta f}{\partial Z} = 2q_{jj}Z + M_j + C_j. \tag{2.23a}$$

Solving equation $\dfrac{\partial \Delta f}{\partial Z} = 0$ gives the critical point equal to

$$Z^* = \frac{-(C_j + M_j)}{2q_{jj}}. \tag{2.23b}$$

By substituting $X^{'} = (X_1, \ldots, Z^*, \ldots, X_n, S_1, \ldots, S_{m_1})$ in the quadratic function Δf, we have the amount of change equal to

$$\Delta f^* = \frac{-(C_j + M_j)^2}{4q_{jj}} - (C_j + M_j + q_{jj}X_j)X_j. \tag{2.24}$$

Note 2: Values Z^* and Δf^* depend only on known numbers. However, there is no guarantee that Z^* is integer, and even if it is integer we do not know if it is feasible for the original

problem. Furthermore, we have $Z^* = \frac{Z_1+Z_2}{2}$.

Theorem 1: Given a vector $(X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_{m_1})$, changing a component X_j results in one and only one of the following conditions:

(1) If $X_j = 0$ and $Z_2 \leq 0$, there is no improving value for the component j (Fig.2.1.a, b).

(2) If $X_j = 0$ and $Z_2 > 0$, any integer number $X_j^{'} \in (0, Z_2)$ improves the value of the objective function $F(X, P)$. If Z^* is integer, it is the most improving, otherwise the closest integer to Z^* within the interval is the most improving value (Fig.2.1.c).

(3) If $X_j > 0$ and $Z_2 = X_j$, there is no improving value for the component j (Fig.2.1.d).

(4) If $X_j > 0$ and $X_j < Z_2$, any integer number $X_j^{'} \in (X_j, Z_2)$ improves the value of the objective function $F(X, P)$. If Z^* is integer, it is the most improving, otherwise the closest integer to Z^* within the interval is the most improving value (Fig.2.1.e).

(5) If $X_j > 0$ and $0 \leq Z_2 < X_j$, any integer number $X_j^{'} \in (Z_2, X_j)$ improves the value of objective function $F(X, P)$. If Z^* is integer, it is the most improving, otherwise the closest integer to Z^* within the interval is the most improving value (Fig.2.1.f, g).

(6) If $X_j > 0$ and $Z_2 < 0$, any integer number $X_j^{'} \in [0, X_j)$ improves the value of the objective function $F(X, P)$.

   (a) If $Z^* > 0$ and integer, it is the most improving, otherwise the closest integer to Z^* within the interval is the most improving value (Fig.2.1.h).

   (b) If $Z^* \leq 0$, the most improving value is $X_j^{'} = 0$ (Fig.2.1.i, j).

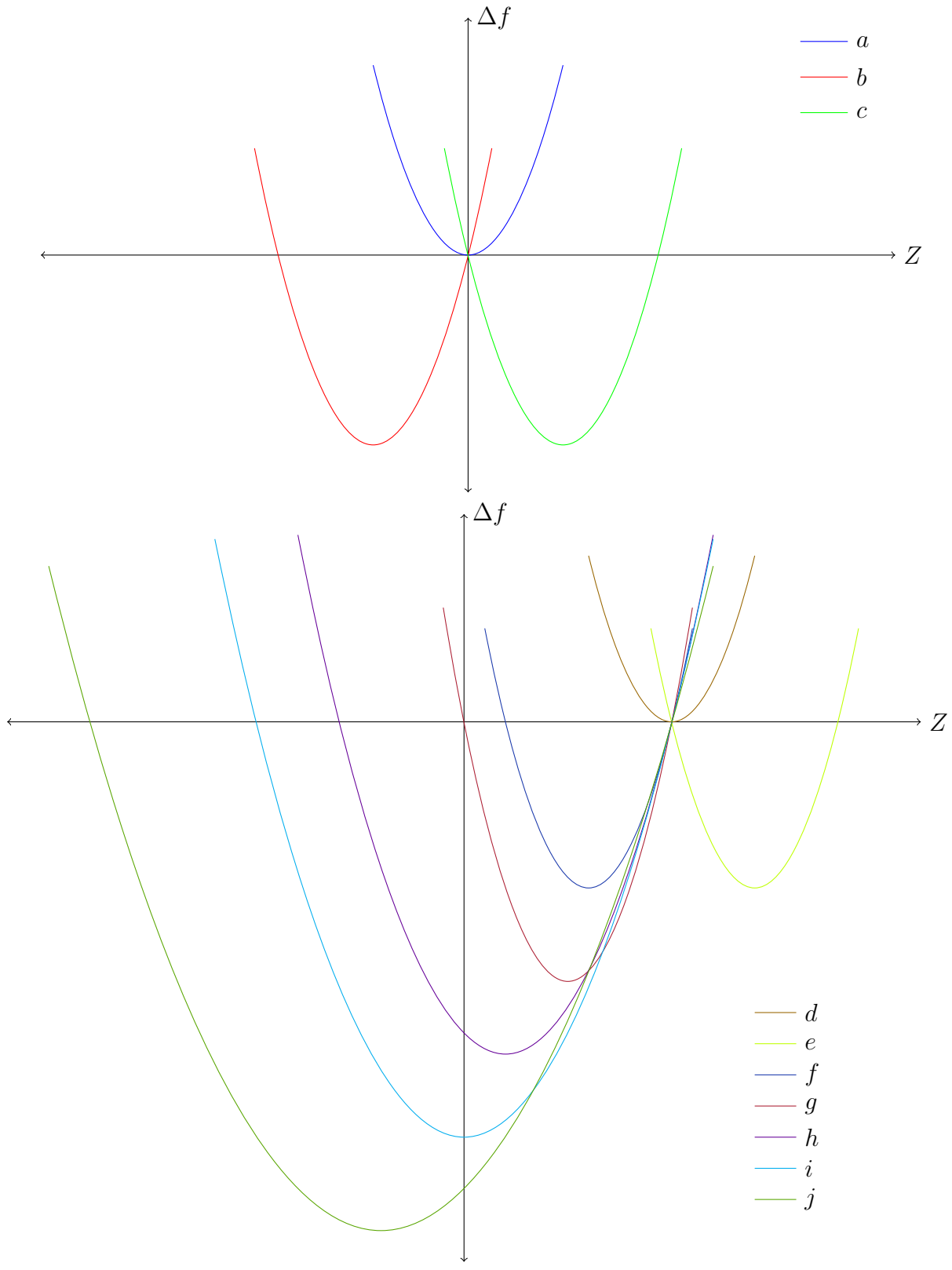

Figure 2.1. All possible conditions for variable update.

Proof: Consider the quadratic function

$$\Delta f = (Z - X_j)\left[C_j + M_j + q_{jj}(Z + X_j)\right]. \tag{2.25}$$

Equating $\Delta f = 0$ leaves two possible situations:

$$Z_1 = X_j \text{ and } Z_2 = -X_j - \frac{C_j + M_j}{q_{jj}}. \tag{2.26}$$

By Proposition 3, Δf is quadratic and convex. Since variables must be non-negative, two possible cases $X_j = 0$ and $X_j > 0$ can be considered.

Case 1, $(X_j = 0)$: Here we have one of the three possible situations (Fig.2.1.*a*-*c*).

Fig.2.1.*a*: If $X_j = Z_2$, changing X_j only increases the value of Δf; thus there is no improving value for the component j.

Fig.2.1.*b*: If $Z_2 < X_j$, all improving values are negative numbers. Since X_j cannot take negative number; thus no improving value exists for component j.

Fig.2.1.*c*: If $X_j < Z_2$, changing X_j within the interval $(0, Z_2)$ improves the value of Δf. Since Z^* is the critical point, it is the most improving. However, there is no guarantee that Z^* is integer. If it is not integer then any integer in this interval is improving, and clearly the closest integer to Z^* is the most improving. If there is no integer in $(0, Z_2)$, there is no improving value for component j.

Case 2, $(X_j > 0)$: Here we have one of the seven possible situations (Fig.2.1.*d*-*j*).

Fig.2.1.*d*: If $X_j = Z_2$, changing X_j only increases the value of Δf; thus there is no improving value for component j.

Fig.2.1.e: If $X_j < Z_2$, changing X_j within the interval (X_j, Z_2) improves the value of Δf. Since Z^* is the critical point, it is the most improving. However, there is no guarantee that Z^* is integer. If it is not integer then any integer in this interval is improving, and clearly the closest integer to Z^* is the most improving. If there is no integer in (X_j, Z_2) there is no improving value for component j.

Fig.2.1.f, g: If $0 \leq Z_2 < X_j$, changing X_j within the interval (Z_2, X_j) improves the value of Δf. Since Z^* is the critical point, it is the most improving. However, there is no guarantee that Z^* is integer. If it is not integer then any integer in this interval is improving and clearly the closest integer to Z^* is the most improving. If there is no integer in (Z_2, X_j) there is no improving value for component j.

Fig.2.1.h, i, j: If $Z_2 < 0$, any value of X_j in the interval (Z_2, X_j) is improving. However, we must have non-negative values for all variables; thus only changing X_j within the interval $[0, X_j)$ that improves the value of Δf. Depending on if $Z^* > 0$ or $Z^* \leq 0$ we have two situations.

Fig.2.1.h: If $Z^* > 0$, since it is the critical point, it is the most improving. However, there is no guarantee that Z^* is integer. If it is not integer then any integer in $[0, X_j)$ is improving, and clearly the closest integer to Z^* is the most improving. If there is no integer number in the interval, there is no improving value for component j.

Fig.2.1.i, j: If $Z^* \leq 0$, clearly $0 \in [0, X_j)$ is an acceptable integer; thus it is improving.

Note 3: In theorem 1 we consistently used Z^* which is the critical value of change in objective function value, Δf, as a result of change in a component X_j. The following proposition emphasizes that Z^* is equal to the critical value of the objective function $F(X, P)$ as a result of change in component X_j.

Proposition 4: Critical value of the objective function f with respect to component X_j is Z^* defined by

$$Z^* = \frac{-(C_j + M_j)}{2q_{jj}}. \tag{2.27}$$

Proof: Taking derivative of $F(X, P)$ with respect to a component X_j we have

$$\frac{\partial f}{\partial X_j} = C_j + M_j + 2q_{jj}X_j. \tag{2.28}$$

In the above equation we have

$$C_j = d_j - 2Pba_j, \tag{2.29a}$$

$$M_j = \sum_{i=1}^{j-1} q_{ij}X_i + \sum_{i=j+1}^{n} q_{ji}X_i + 2P\sum_{k=1}^{m_1} a_{kj}S_k = 2P\left[\sum_{i=1, i\neq j}^{n} a_i a_j X_i + \sum_{k=1}^{m_1} a_{kj}S_k\right], \tag{2.29b}$$

$$q_{ij} = 2Pa_i a_j, \text{ and } q_{jj} = 2Pa_j a_j. \tag{2.29c}$$

Solving $\frac{\partial F(X,P)}{\partial X_j} = 0$ for the unknown component X_j we have the desired result for the critical value Z^*.

Note 4: In theorem 1 we concentrated on changing a component X_j, however all results are applicable to the case when a slack variable S_k for $(k = 1, \ldots, m_1)$ is changed. In this case, results can be simplified as follows. Given a vector $(X_1, \ldots, X_n)$ the optimal value, S^*, for a component S_k is equal to

$$S^* = b_k - \sum_{j=1}^{n} a_{kj}X_j. \tag{2.30}$$

Note that components S_i (for all $i \neq k$) does not contribute in calculation of S^*. Also note that there is no guarantee that $S^* \geq 0$. Thus, it is clear that given $(X_1, \ldots, X_n)$ the optimal

value of S_k $(k = 1, \ldots, m_1)$ needs to be set equal to

$$S_k = \begin{cases} S_k^*, & \text{if } S_k^* \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.31}$$

Theorem 1 can be used to find a local optimal solution when one component at a time can be changed. Results of theorem 1 are further simplified in theorem 2.

Theorem 2: Given a vector $(X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_{m_1})$, a component X_j can be improved if and only if one of the following conditions holds:

(1) If $Z^* \geq 0$ and $X_j < Z_j$, then all integer numbers in the interval (X_j, Z_2) are improving. The most improving value is the closest integer to Z^* (including Z^*).

(2) If $Z^* \geq 0$ and $X_j > Z_2$, then all non-negative integer numbers in the interval (Z_2, X_j) are improving. The most improving is the closest non-negative number close to Z^* (including Z^*).

(3) If $Z^* < 0$, $X_j > Z_2$ and $X_j > 0$, then all integer numbers in the interval $[0, X_j)$ are improving. The most improving value is 0.

Proof: It follows directly from all possible cases included in theorem 1. Improving cases are Fig.2.1.*c*, Fig.2.1.*e*-*j*. If $X_j = Z_2$, there is no improving value for the component j, thus there is no need to consider such case. Condition 1 follows from Fig.2.1.*c*. Condition 2 follows from Fig.2.1.*e*-*i*. Condition 3 follows from Fig.2.1.*j*. Using theorem 2, a necessary and sufficient condition for a locally optimal solution can be given as follows

Theorem 3: A solution $(X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_{m_1})$ is locally optimal for problem **P2** if and only if for each component X_j one of the following holds, and each slack variable S_k

satisfies

$$S_k = \begin{cases} S_k^*, & \text{if } S_k^* \geq 0, \\ 0, & \text{otherwise,} \end{cases} \tag{2.32a}$$

$$\text{where } S^* = b_k - \sum_{j=1}^{n} a_{kj} X_j. \tag{2.32b}$$

(1) $X_j = Z_2$.

(2) $Z_2 < 0 = X_j$.

(3) $X_j < Z_2$ and there are no integer numbers in the interval (X_j, Z_2).

(4) $0 \leq Z_2 \leq X_j$ and there are no integer numbers in the interval (Z_2, X_j).

Proof: A vector $(X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_{m_1})$ is locally optimal if and only if there is no component improving X_j and all slack variables are optimally chosen. Theorem 1 provided 10 possible situations for each component X_j. Out of the 10 possibilities, Fig.2.1.h-j are always improving and thus cannot happen for any component in a local optimal solution. These situations are excluded from occurring for any component j in the theorem. Condition 1 is the case when we have Fig.2.1.a or Fig.2.1.d where both are non-improving for a component j. Condition 2 is the situation where Fig.2.1.b is applicable; thus all improving values are negative. However, we never allow negative number for all X_j and S_k. Applicable figures to condition 3 are Fig.2.1.c and Fig.2.1.e. Both situations can have improving value if and only if there is an integer number in (X_j, Z_2). Applicable figures for condition 4 are Fig.2.1.f and Fig.2.1.g. Both situations can have improving value if and only if there is an integer number in (Z_2, X_j). This covers all possible situations for component j, $(j = 1, \ldots, n)$. Given $(X_1, \ldots, X_n)$ in order to have local optimal solution for **P2**, all values of S_k $(k = 1, \ldots, m_1)$ must be optimally chosen. Based on note 4, all values of S_k $(k = 1, \ldots, m_1)$ are optimally chosen. This completes the proof of the theorem.

Note that after a component X_j is changed to $X_j^{'}$ we update all slack variables accordingly. The proposition 5 proves that for each change in slack variable the objective function value will improve or will not change. Thus, when a component j is changed to improve the objective function, a series of possible improving changes on slack variables are implemented.

Proposition 5: Given a vector $(X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_k, \ldots, S_{m_1})$, assume component X_j is changed to an improving value $X_j^{'}$. Then changing each slack variable S_k to $S_k^{'}$ $(k = 1, \ldots, m_1)$ will not increase the value of objective function where:

$$S_k^{'} = \begin{cases} S_k^*, & \text{if } S_k^* \geq 0, \\ 0, & \text{otherwise,} \end{cases} \tag{2.33a}$$

$$\text{and } S^* = b_k - \sum_{j=1}^{n} a_{kj} X_j. \tag{2.33b}$$

Proof: Assume we have changed component X_j to $X_j^{'}$. Thus for each k $(k = 1, \ldots, m_1)$ there is a change on S_k^*. Now, changing a slack variable S_k to $S_k^{'}$, the amount of change to the objective function is equal to

$$\begin{aligned} \Delta f &= (S_k^{'} - S_k) \left[C_k + M_k + P(S_k^{'} + S_k) \right] \\ &= (S_k^{'} - S_k) \left[-2Pb_k + 2P \sum_{j=1}^{n} a_{kj} X_j + P(S_k^{'} + S_k) \right] \\ &= P(S_k^{'} - S_k) \left[-2 \left(b_k - \sum_{j=1}^{n} a_{kj} X_j \right) + (S_k^{'} + S_k) \right] \\ &= P(S_k^{'} - S_k) \left[-2S_k^* + (S_k^{'} + S_k) \right] \\ &= -P(S_k^{'} - S_k) \left[(S_k^* - S_k^{'}) + (S_k^* - S_k) \right]. \end{aligned} \tag{2.34}$$

Two cases are considered:

Case 1, $(S_k^* > 0)$: Substitute $S_k^{'} = S_k^*$ in Δf then for all $S_k \geq 0$ we have $\Delta f = -P(S_k^{'} -$

$S_k)^2 \leq 0$. If equality holds, there is no change in the objective function; otherwise the objective function is improved.

Case 2, $(S_k^* \leq 0)$: Substitute $S_k^{'} = 0$ in Δf. Now two subcases are considered.

(i) $(S_k = 0)$: Here we have $\Delta f = 0$; thus no change in the objective function.

(ii) $(S_k > 0)$: Here we have $\Delta f = PS_k[2S_k^* - S_k] < 0$ and thus the objective function is improved.

Of course all presented results are applicable to the case when $X_j \in \{0, 1\}$ for $j = 1, \ldots, n$. However, in the case of binary variables and possibility of having general inequalities, the above results can be simplified. This is due to the fact that for each $j = 1, \ldots, n$, we have $X_j^{'} + X_j = 1$ and $X_j^2 = X_j$. In theorem 4 we present the necessary and sufficient condition for local optimality of a solution when one component X_j at a time is changed.

Theorem 4: A vector $(X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_{m_1})$ is locally optimal for problem **P2** if and only if for each component X_j both (a) and (b) hold true.

(a) $X_j = 1 \iff \Delta_j(X) < 0$, and $X_j = 0 \iff \Delta_j(X) \geq 0$,

where,

$$\Delta_j(X) = C_j + M_j + q_{jj}, \tag{2.35a}$$

$$M_j = \sum_{i=1}^{j-1} q_{ij} X_i + \sum_{i=j+1}^{n} q_{ji} X_i + 2P \sum_{k=1}^{m_1} a_{kj} S_k = 2P \left(\sum_{i=1, i \neq j}^{n} a_j a_i X_i + \sum_{k=1}^{m_1} e_j a_k S_k \right), \tag{2.35b}$$

$$C_j = d_j - 2Pba_j, \tag{2.35c}$$

$$q_{jj} = Pa_j a_j. \tag{2.35d}$$

(b) Each component S_k satisfies

$$S_k = \begin{cases} S_k^*, & \text{if } S_k^* \geq 0, \\ 0, & \text{otherwise,} \end{cases} \tag{2.36a}$$

$$\text{where } S^* = b_k - \sum_{j=1}^{n} a_{kj} X_j. \tag{2.36b}$$

Proof: Since the variables are binary, $X_j^{'} + X_j = 1$ and $X_j^2 = X_j$ for $j = 1, \ldots, n$. If X_j is changed to $X_j^{'} = 1 - X_j$, the amount of change is equal to

$$\Delta f = (1 - 2X_j)[C_j + M_j + q_{jj}] = (1 - 2X_j)\Delta_j(X). \tag{2.37}$$

Part (a) follows from Δf.

Given $(X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_{m_1})$, optimal value of S_k $(k = 1, \ldots, m_1)$ is equal to $S^* = b_k - \sum_{j=1}^{n} a_{kj} X_j$.

Part (b) follows from the fact that we must have $S_k \geq 0$.

Proposition 6: Given a vector $(X_1, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_k, \ldots, S_{m_1})$, assume component X_i is changed to an improving value $X_i^{'}$. The slack variable S_k can be updated to $S_k^{'}$ $(k = 1, \ldots, m_1)$ using

$$S_k^{'} = \begin{cases} S_k^* = S_k - a_{ki}(X_i^{'} - X_i), & \text{if } S_k^* \geq 0, \\ 0, & \text{if } S_k^* < 0. \end{cases} \tag{2.38a}$$

Proof: Given a vector $(X_1, \ldots, X_j, \ldots, X_n)$, the value of each slack variable $(S_1, \ldots, S_k, \ldots, S_{m_1})$ is calculated using

$$S_k = b_k - \sum_{j=1}^{n} a_{kj} X_j. \tag{2.39}$$

The value of S_k could be positive or negative above. However, we are interested only in values ≥ 0. Based on note 1, we have the following:

$$S_k = \begin{cases} S_k^*, & \text{if } S_k^* \geq 0, \\ 0, & \text{otherwise,} \end{cases} \tag{2.40a}$$

$$\text{where } S^* = b_k - \sum_{j=1}^{n} a_{kj} X_j. \tag{2.40b}$$

Now if X_i is changed to $X_i^{'}$ we update vector S. If we keep track of vector S^*, it makes the update of vector S easier. Updating to new S^* is as follows:

$$S_k = b_k - \sum_{j=1}^{n} a_{kj} X_j, \tag{2.41a}$$

$$S_k^{'} = b_k - \sum_{j=1}^{n} a_{kj} X_j + a_{ki} X_i - a_{ki} X_i^{'}, \tag{2.41b}$$

$$S_k^{'} = S_k + a_{ki} X_i - a_{ki} X_i^{'}, \tag{2.41c}$$

$$S_k^{'} = S_k - a_{ki}(X_i^{'} - X_i) = \text{new } S_k^*. \tag{2.41d}$$

Again we have:

$$\text{If } S_k^{'} \geq 0 \implies \text{new } S_k = \text{ new } S_k^*. \tag{2.42a}$$

$$\text{If } S_k^{'} < 0 \implies \text{new } S_k = 0. \tag{2.42b}$$

Note 5: In proposition 6, we showed an easy way to update the slack variables S_k, but the same can be achieved by recomputing S_k^* $(k = 1, \ldots, m_1)$ using note 4 with the updated value of $X_i^{'}$. After updating X_i to $X_i^{'}$ and updating slack variables S, all the partial derivatives with respect to variable X needs to be updated to select the next variable update. This can be accomplished as shown in proposition 7.

Proposition 7: Given a vector $(X_1, \ldots, X_i, \ldots, X_j, \ldots, X_n, S_1, \ldots, S_k, \ldots, S_{m_1})$, and an updated vector $\left(X_1, \ldots, X_i^{'}, \ldots, X_j, \ldots, X_n, S_1^{'}, \ldots, S_k^{'}, \ldots, S_{m_1}^{'}\right)$, updating partial derivatives with respect to X can be easily accomplished by keeping track of vector X and vector S.

Proof: From proposition 4 and theorem 4 we have:

Case 1: $\frac{\partial f}{\partial X_j} = C_j + M_j + 2q_{jj}X_j$, when $X \geq 0$ and integer.

Case 2: $\frac{\partial f}{\partial X_j} = C_j + M_j + q_{jj}$, when $X\{0, 1\}$.

After changing X_i to $X_i^{'}$ and updating vector S to $S^{'}$:

For $j = 1, \ldots, n$:

$$C_j = d_j - 2Pba_j, \text{ will not change.} \tag{2.43a}$$

$$q_{jj} = Pa_j a_j \geq 0, \text{will not change.} \tag{2.43b}$$

For $j \neq i$, the new values of M_j are:

$$\begin{aligned} M_j &= M_j + 2P\left(-a_j a_i X_i + a_j a_i X_i^{'}\right) + 2P\left(-\sum_{k=1}^{k=m_1} a_j e_k S_k + \sum_{k=1}^{k=m_1} a_j e_k S_k^{'}\right), \\ &= M_j + 2Pa_j a_i\left(X_i^{'} - X_i\right) + 2P\sum_{k=1}^{k=m_1} a_j e_k\left(S_k^{'} - S_k\right). \end{aligned} \tag{2.44}$$

This is the same as:

$$\begin{aligned} &\text{For } j < i, M_j = M_j + 2Pq_{ji}\left(X_i^{'} - X_i\right) + 2P\sum_{k=1}^{k=m_1} a_{jk}\left(S_k^{'} - S_k\right). \\ &\text{For } j > i, M_j = M_j + 2Pq_{ij}\left(X_i^{'} - X_i\right) + 2P\sum_{k=1}^{k=m_1} a_{jk}\left(S_k^{'} - S_k\right). \end{aligned} \tag{2.45}$$

For $j = i$:

$$M_j = M_j + 2P \sum_{k=1}^{k=m_1} a_{jk} \left(S_k^{'} - S_k \right). \tag{2.46a}$$

$$2q_{jj}X_j = 2q_{jj} \left(X_j^{'} - X_j \right). \tag{2.46b}$$

Note 6: In proposition 6 we updated vector S to $S^{'}$ after updating X_i to $X_i^{'}$. The amount of change on value of f can be calculated using proposition 2 and proposition 5.

2.6 Derivation for General XQX Model

The detailed derivations to model an integer programming problem in XQX form is presented in this section and this complements the theorems provided in the previous section. Consider Example 1 which is a general IP minimization problem with 2 unknowns, 3 inequality constraints, 2 equality constraints, and integer variables.

Example 1

$$\begin{aligned} min\ z = \quad & d_1x_1 + d_2x_2 \\ subject\ to \quad & a_{11}x_1 + a_{12}x_2 \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 \leq b_2 \\ & a_{31}x_1 + a_{32}x_2 \leq b_3 \\ & a_{41}x_1 + a_{42}x_2 = b_4 \\ & a_{51}x_1 + a_{52}x_2 = b_5, \end{aligned} \tag{2.47}$$

where $d_j \in \mathbb{R}$, $a_{ij} \in \mathbb{R}$, $b_i \in \mathbb{R}$, *and* $x_j \in \mathbb{Z}_+$, $i = 1, 2, 3, 4, 5$ *and* $j = 1, 2$, *and,*

$$a_1 = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \\ a_{51} \end{bmatrix}, a_2 = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \\ a_{42} \\ a_{52} \end{bmatrix}, b^{'} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}, A_1 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix},$$
$$A_2 = \begin{bmatrix} a_{41} & a_{42} \\ a_{51} & a_{52} \end{bmatrix}, A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, D = \begin{bmatrix} d_1 & d_2 \end{bmatrix}, X = \begin{bmatrix} x_1 & x_2 \end{bmatrix}, \text{and } S = \begin{bmatrix} s_1 & s_2 & s_3 \end{bmatrix}. \tag{2.48}$$

Example 1 is converted to standard form by adding slack variables:

$$\begin{aligned} \min\ z = \quad & d_1x_1 + d_2x_2 \\ \text{subject to} \quad & a_{11}x_1 + a_{12}x_2 + s_1 = b_1 \\ & a_{21}x_1 + a_{22}x_2 + s_2 = b_2 \\ & a_{31}x_1 + a_{32}x_2 + s_3 = b_3 \\ & a_{41}x_1 + a_{42}x_2 = b_4 \\ & a_{51}x_1 + a_{52}x_2 = b_5, \end{aligned} \tag{2.49}$$

where $s_i \geq 0$. The constraints are then represented similar to (2.5) to obtain the transformation shown in (2.6).

$$\begin{aligned} z = d_1x_1 + d_2x_2 \quad & + P(a_{11}x_1 + a_{12}x_2 + s_1 - b_1)^\top (a_{11}x_1 + a_{12}x_2 + s_1 - b_1) + \\ & P(a_{21}x_1 + a_{22}x_2 + s_2 - b_2)^\top (a_{21}x_1 + a_{22}x_2 + s_2 - b_2) + \\ & P(a_{31}x_1 + a_{32}x_2 + s_3 - b_3)^\top P(a_{31}x_1 + a_{32}x_2 + s_3 - b_3) + \\ & P(a_{41}x_1 + a_{42}x_2 - b_4)^\top P(a_{41}x_1 + a_{42}x_2 - b_4) + \\ & P(a_{51}x_1 + a_{52}x_2 - b_5)^\top P(a_{51}x_1 + a_{52}x_2 - b_5), \end{aligned} \tag{2.50}$$

where P is a large positive number for penalty. Expanding the terms produced by each constraint gives the following:

For constraint 1 we have:

$$\begin{aligned} P(a_{11}x_1 + a_{12}x_2 + s_1 - b_1)^\top (a_{11}x_1 + a_{12}x_2 + s_1 - b_1) = P(a_{11}^2x_1^2 + a_{12}^2x_2^2 + s_1^2 + b_1^2 + \\ a_{11}a_{12}x_1x_2 + a_{11}s_1x_1 - a_{11}b_1x_1 + \\ a_{12}a_{11}x_2x_1 + a_{12}s_1x_2 - a_{12}b_1x_2 + \\ a_{11}s_1x_1 + a_{12}s_1x_2 - b_1s_1 - \\ a_{11}b_1x_1 - a_{12}b_1x_2 - b_1s_1). \end{aligned} \tag{2.51}$$

For constraint 2, we have:

$$\begin{aligned} P(a_{21}x_1 + a_{22}x_2 + s_2 - b_2)^\top (a_{21}x_1 + a_{22}x_2 + s_2 - b_2) = P(a_{21}^2x_1^2 + a_{22}^2x_2^2 + s_2^2 + b_2^2 + \\ a_{21}a_{22}x_1x_2 + a_{21}s_2x_1 - a_{21}b_2x_1 + \\ a_{22}a_{21}x_2x_1 + a_{22}s_2x_2 - a_{22}b_2x_2 + \\ a_{21}s_2x_1 + a_{22}s_2x_2 - b_2s_2 - \\ a_{21}b_2x_1 - a_{22}b_2x_2 - b_2s_2). \end{aligned} \tag{2.52}$$

For constraint 3, we have:

$$\begin{aligned} P(a_{31}x_1 + a_{32}x_2 + s_3 - b_3)^\top (a_{31}x_1 + a_{32}x_2 + s_3 - b_3) = P(a_{31}^2x_1^2 + a_{32}^2x_2^2 + s_3^2 + b_3^2 + \\ a_{31}a_{32}x_1x_2 + a_{31}s_3x_1 - a_{31}b_3x_1 + \\ a_{32}a_{31}x_2x_1 + a_{32}s_3x_2 - a_{32}b_3x_2 + \\ a_{31}s_3x_1 + a_{32}s_3x_2 - b_3s_3 - \\ a_{31}b_3x_1 - a_{32}b_3x_2 - b_3s_3). \end{aligned} \tag{2.53}$$

For constraint 4, we have:

$$\begin{aligned} P(a_{41}x_1 + a_{42}x_2 - b_4)^\top P(a_{41}x_1 + a_{42}x_2 - b_4) &= P(a_{41}^2x_1^2 + a_{42}^2x_2^2 + b_4^2 + \\ & a_{41}a_{42}x_1x_2 - a_{41}b_4x_1 + \\ & a_{42}a_{41}x_2x_1 - a_{42}b_4x_2 - \\ & a_{41}b_4x_1 - a_{42}b_4x_2). \end{aligned} \tag{2.54}$$

Finally, for constraint 5, we have:

$$\begin{aligned} P(a_{51}x_1 + a_{52}x_2 - b_5)^\top P(a_{51}x_1 + a_{52}x_2 - b_5) &= P(a_{51}^2x_1^2 + a_{52}^2x_2^2 + b_5^2 + \\ & a_{51}a_{52}x_1x_2 - a_{51}b_5x_1 + \\ & a_{52}a_{51}x_2x_1 - a_{52}b_5x_2 - \\ & a_{51}b_5x_1 - a_{52}b_5x_2). \end{aligned} \tag{2.55}$$

Substituting (2.51) through (2.55) in (2.50), we obtain the following:

$$z = z_1 + z_2 + z_3 + z_4, \text{where,} \tag{2.56a}$$

$$z_1 = d_1x_1 + d_2x_2, \tag{2.56b}$$

$$\begin{aligned} z_2 = \; & P(a_{11}^2x_1^2 + a_{12}^2x_2^2 + s_1^2 + 2a_{11}a_{12}x_1x_2 + 2a_{11}s_1x_1 + 2a_{12}s_1x_2 + \\ & a_{21}^2x_1^2 + a_{22}^2x_2^2 + s_2^2 + 2a_{21}a_{22}x_1x_2 + 2a_{21}s_2x_1 + 2a_{22}s_2x_2 + \\ & a_{31}^2x_1^2 + a_{32}^2x_2^2 + s_3^2 + 2a_{31}a_{32}x_1x_2 + 2a_{31}s_3x_1 + 2a_{32}s_3x_2 + \\ & a_{41}^2x_1^2 + a_{42}^2x_2^2 + 2a_{41}a_{42}x_1x_2 + \\ & a_{51}^2x_1^2 + a_{52}^2x_2^2 + 2a_{51}a_{52}x_1x_2), \end{aligned} \tag{2.56c}$$

$$\begin{aligned} z_3 = \; & -2P(a_{11}b_1x_1 + a_{12}b_1x_2 + b_1s_1 + \\ & a_{21}b_2x_1 + a_{22}b_2x_2 + b_2s_2 + \\ & a_{31}b_3x_1 + a_{32}b_3x_2 + b_3s_3 + \end{aligned} \tag{2.56d}$$

$$
\begin{aligned}
& a_{41}b_4x_1 + a_{42}b_4x_2 + \\
& a_{51}b_5x_1 + a_{52}b_5x_2), \\
z_4 = \quad & 2P(b_1^2 + b_2^2 + b_3^2 + b_4^2 + b_5^2). \qquad (2.56e)
\end{aligned}
$$

In the above set of equations, z_1 can be represented as $[X \cdot d^\top]$. The term z_4 is a constant term and can be represented by c. The term z_2 can be regrouped to produce the following:

$$z_2 = P(z_{2a} + z_{2b} + z_{2c} + z_{2d} + z_{2e} + z_{2f} + z_{2g} + z_{2h} + z_{2i} + z_{2j} + z_{2k} + z_{2l}), \qquad (2.57a)$$

where,

$$z_{2a} = x_1^2(a_{11}^2 + a_{21}^2 + a_{31}^2 + a_{41}^2 + a_{51}^2), \qquad (2.57b)$$

$$z_{2b} = x_2^2(a_{12}^2 + a_{22}^2 + a_{32}^2 + a_{42}^2 + a_{52}^2), \qquad (2.57c)$$

$$z_{2c} = 2x_1x_2(a_{11}a_{12} + a_{21}a_{22} + a_{31}a_{32} + a_{41}a_{42} + a_{51}a_{52}), \qquad (2.57d)$$

$$z_{2d} = 2s_1x_1(a_{11}), \qquad (2.57e)$$

$$z_{2e} = 2s_1x_2(a_{12}), \qquad (2.57f)$$

$$z_{2f} = 2s_2x_1(a_{21}), \qquad (2.57g)$$

$$z_{2g} = 2s_2x_2(a_{22}), \qquad (2.57h)$$

$$z_{2h} = 2s_3x_1(a_{31}), \qquad (2.57i)$$

$$z_{2i} = 2s_3x_2(a_{32}), \qquad (2.57j)$$

$$z_{2j} = s_1^2, \qquad (2.57k)$$

$$z_{2k} = s_2^2, \qquad (2.57l)$$

$$z_{2l} = s_3^2. \qquad (2.57m)$$

The terms represented by z_{2a}, z_{2b}, and z_{2c} in (2.57) can be written as follows using the definitions in (2.48):

$$z_{2a} = x_1[a_1^\top a_1]x_1, \tag{2.58a}$$

$$z_{2b} = x_2[a_2^\top a_2]x_2, \tag{2.58b}$$

$$z_{2c} = 2x_1[a_1^\top a_2]x_2. \tag{2.58c}$$

Using the equations in (2.57) and (2.58), z_2 can be represented in XQX form as follows:

$$\begin{aligned} z_2 =& P \begin{bmatrix} z_{2a} & \frac{z_{2c}}{2} & \frac{z_{2d}}{2} & \frac{z_{2f}}{2} & \frac{z_{2h}}{2} \\ \frac{z_{2c}}{2} & z_{2b} & \frac{z_{2e}}{2} & \frac{z_{2g}}{2} & \frac{z_{2i}}{2} \\ \frac{z_{2d}}{2} & \frac{z_{2e}}{2} & 1 & 0 & 0 \\ \frac{z_{2f}}{2} & \frac{z_{2g}}{2} & 0 & 1 & 0 \\ \frac{z_{2h}}{2} & \frac{z_{2i}}{2} & 0 & 0 & 1 \end{bmatrix}, \\ =& P \begin{bmatrix} x_1 & x_2 & s_1 & s_2 & s_3 \end{bmatrix} \begin{bmatrix} a_1^\top a_1 & a_1^\top a_2 & a_{11} & a_{21} & a_{31} \\ a_1^\top a_2 & a_2^\top a_2 & a_{12} & a_{22} & a_{32} \\ a_{11} & a_{12} & 1 & 0 & 0 \\ a_{21} & a_{22} & 0 & 1 & 0 \\ a_{31} & a_{32} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}. \end{aligned} \tag{2.59}$$

The XQX matrix from (2.59) can then be generalized as:

$$z_2 = P \begin{bmatrix} X & S \end{bmatrix} \begin{bmatrix} A^\top A & A_1^\top \\ A_1 & \mathbb{I} \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix}. \tag{2.60}$$

The remaining term z_3 represented by (2.56d) can be regrouped as follows:

$$z_3 = -2P(z_{3a} + z_{3b} + z_{3c}). \tag{2.61a}$$

Where,

$$z_{3a} = x_1(a_{11}b_1 + a_{21}b_2 + a_{31}b_3 + a_{41}b_4 + a_{51}b_5), \tag{2.61b}$$

$$z_{3b} = x_2(a_{12}b_1 + a_{22}b_2 + a_{32}b_3 + a_{42}b_4 + a_{52}b_5), \tag{2.61c}$$

$$z_{3c} = b_1s_1 + b_2s_2 + b_3s_3. \tag{2.61d}$$

The terms represented by z_{3a}, z_{3b}, and z_{3c} in (2.61) can be written as follows using the definitions in (2.48):

$$z_{3a} = x_1[b^\top a_1], \tag{2.62a}$$

$$z_{3b} = x_2[b^\top a_2], \tag{2.62b}$$

$$z_{3c} = S[b'^\top]. \tag{2.62c}$$

The term z_3 in (2.62) can be generalized as:

$$z_3 = -2P \begin{bmatrix} z_{3a} & z_{3b} & z_{3c} \end{bmatrix}, \tag{2.63a}$$

$$= -2P \begin{bmatrix} b^\top a_1 & b^\top a_2 & b'^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ S \end{bmatrix}, \tag{2.63b}$$

$$= -2P \begin{bmatrix} b^\top a_1 & b^\top a_2 & b'^\top \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix}. \tag{2.63c}$$

Thus the final form for $z = z_1 + z_2 + z_3 + z_4$ in (2.56a) is:

$$
\begin{aligned}
z &= d_1x_1 + d_2x_2 + P\begin{bmatrix} X & S \end{bmatrix}\begin{bmatrix} A^\top A & A_1^\top \\ A_1 & \mathbb{I} \end{bmatrix}\begin{bmatrix} X \\ S \end{bmatrix} - 2P\begin{bmatrix} b^\top a_1 & b^\top a_2 & b'^\top \end{bmatrix}\begin{bmatrix} X \\ S \end{bmatrix} + c. \\
&= P\begin{bmatrix} X & S \end{bmatrix}\begin{bmatrix} A^\top A & A_1^\top \\ A_1 & \mathbb{I} \end{bmatrix}\begin{bmatrix} X \\ S \end{bmatrix} + \\
&\begin{bmatrix} d_1 - 2P(b^\top a_1) & d_2 - 2P(b^\top a_2) & -2P(b'^\top) \end{bmatrix}\begin{bmatrix} X \\ S \end{bmatrix} + c.
\end{aligned}
\tag{2.64}
$$

The form obtained in (2.64) represents the XQX form for the general integer programming problems. This XQX model is certainly new and does not exist in the literature. This general form can be used to represent a maximization or minimization problem with integer or binary variables, and with inequality and equality constraints. The XQX model for the boolean IP problem is presented next, followed by the XQX model for general IP problem with only inequality constraints.

2.6.1 XQX Model for Binary IP Problems

The XQX form for a binary IP problem is exactly the same as (2.64) since the variables being binary or integer does not make any difference in the derivations presented above. Even though $x^2 = x$ for a binary IP, this was never used in the above derivations. Thus, if Example 1 had binary variables ($x \in \{0, 1\}$) instead of integer variables, the XQX form is still the same as obtained in (2.64).

2.6.2 XQX Model for IP Problems with Inequality Constraints

Consider the problem in Example 2, which is the same as Example 1, but without the equality constraints. We present this new example since the multi-dimensional knapsack problems does not contain any equality constraints.

Example 2

$$\begin{aligned} min\ z = \quad & d_1x_1 + d_2x_2 \\ subject\ to \quad & a_{11}x_1 + a_{12}x_2 \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 \leq b_2 \\ & a_{31}x_1 + a_{32}x_2 \leq b_3, \end{aligned} \tag{2.65}$$

where $d_j \in \mathbb{R}$, $a_{ij} \in \mathbb{R}$, $b_i \in \mathbb{R}$, and $x_j \in \mathbb{Z}_+$, $i = 1, 2, 3$ and $j = 1, 2$. The standard form after adding slack variables to Example 2 is:

$$\begin{aligned} \text{min } z = \quad & d_1x_1 + d_2x_2 \\ \text{subject to} \quad & a_{11}x_1 + a_{12}x_2 + s_1 = b_1 \\ & a_{21}x_1 + a_{22}x_2 + s_2 = b_2 \\ & a_{31}x_1 + a_{32}x_2 + s_3 = b_3, \end{aligned} \tag{2.66}$$

where $s_i \geq 0$. Let

$$\begin{gathered} a_1^{'} = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix}, a_2^{'} = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}, b^{'} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}, A_1 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}, \\ D = \begin{bmatrix} d_1 & d_2 \end{bmatrix}, X = \begin{bmatrix} x_1 & x_2 \end{bmatrix}, \text{and } S = \begin{bmatrix} s_1 & s_2 & s_3 \end{bmatrix}. \end{gathered} \tag{2.67}$$

Using the procedure presented in section 2.5, the XQX form for this problem is as follows:

$$z = d_1x_1 + d_2x_2 + P \begin{bmatrix} X & S \end{bmatrix} \begin{bmatrix} A_1^\top A_1 & A_1^\top \\ A_1 & \mathbb{I} \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix} - 2P \begin{bmatrix} b'^\top a_1^{'} & b'^\top a_2^{'} & b'^\top \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix} + c. \tag{2.68a}$$

$$= P \begin{bmatrix} X & S \end{bmatrix} \begin{bmatrix} A_1^\top A_1 & A_1^\top \\ A_1 & \mathbb{I} \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix} + \tag{2.68b}$$

$$\begin{bmatrix} d_1 - 2P(b'^\top a_1') & d_2 - 2P(b'^\top a_2') & -2P(b'^\top) \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix} + c.$$

This XQX form is the same as the general form obtained in (2.64) and only difference being that the A matrix, which represented the coefficients of variables for the inequality and equality constraints in the general form, is replaced by A_1 matrix which represents the variable's coefficients for the inequality constraints. The column matrix b, which contained the right hand side coefficients for both equality and inequality constraints, is replaced by b'. Also note that the column matrix a_1' and a_2' represent the coefficients from the three inequality constraints.

2.6.3 Alternate Representations of Min and Max IP problems

Our solution method (presented in the next section) for the general XQX form in (2.64) is based on gradient descent method, and a given IP problem is transformed to an equivalent alternate representation as shown in Table 2.2.

Given Problem	Equivalent Alternate Representation
$\max z = \{dx : Ax \leq b, x \in \mathbb{Z}_+\}$	$\min -z = \{-dx : Ax \leq b, x \in \mathbb{Z}_+\}$
$\min z = \{dx : Ax \geq b, x \in \mathbb{Z}_+\}$	$\min z = \{dx : -Ax \leq -b, x \in \mathbb{Z}_+\}$

Table 2.2: Alternate Representations for IP problems.

2.7 Solution Procedures for XQX Model

The solution procedure presented here is also new and does not exist in the literature, and this is the second objective of this work. In this section we present the local optima based solution procedure for our XQX model, which is based on gradient descent method described in section 2.4. We will first present the solution procedure for the general IP XQX

model followed by the solution procedure of the XQX model for binary IP problems. We will continue to use the problem in Example 1 and its corresponding XQX model and show the general solution procedure for any problem size. The solution procedures presented here will be used in our the heuristic procedure to solve the benchmark problems.

The general form of our XQX model from (2.64) is:

$$\begin{aligned} z = P \begin{bmatrix} X & S \end{bmatrix} \begin{bmatrix} A^\top A & A_1^\top \\ A_1 & \mathbb{I} \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix} + \\ \begin{bmatrix} d_1 - 2P(b^\top a_1) & d_2 - 2P(b^\top a_2) & -2P(b'^\top) \end{bmatrix} \begin{bmatrix} X \\ S \end{bmatrix} + c. \end{aligned} \tag{2.69}$$

The expanded XQX form for the above (from (2.59) and (2.63)) with respect to Example 1 is:

$$\begin{aligned} z \;\; = \begin{bmatrix} x_1 & x_2 & s_1 & s_2 & s_3 \end{bmatrix} \begin{bmatrix} P(a_1^\top a_1) & P(a_1^\top a_2) & Pa_{11} & Pa_{21} & Pa_{31} \\ P(a_1^\top a_2) & P(a_2^\top a_2) & Pa_{12} & Pa_{22} & Pa_{32} \\ Pa_{11} & Pa_{12} & P & 0 & 0 \\ Pa_{21} & Pa_{22} & 0 & P & 0 \\ Pa_{31} & Pa_{32} & 0 & 0 & P \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} + \\ \begin{bmatrix} d_1 - 2P(b^\top a_1) & d_2 - 2P(b^\top a_2) & -2Pb_1 & -2Pb_2 & -2Pb_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} + c. \end{aligned} \tag{2.70}$$

To show the solution procedure in a general way we will use the following form to

represent our XQX model:

$$f(x,s) = \begin{bmatrix} x_1 & x_2 & s_1 & s_2 & s_3 \end{bmatrix} \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} & q_{15} \\ q_{21} & q_{22} & q_{23} & q_{24} & q_{25} \\ q_{31} & q_{32} & q_{33} & q_{34} & q_{35} \\ q_{41} & q_{42} & q_{43} & q_{44} & q_{45} \\ q_{51} & q_{52} & q_{53} & q_{54} & q_{55} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} + \begin{bmatrix} r_1 & r_2 & r_3 & r_4 & r_5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} + c. \tag{2.71}$$

As mentioned earlier, the lower the value of $f(x,s)$ is, the better the solution is for our minimization problem. With the general matrices shown in (2.71), we will now present the solution procedure for the general and binary IP problem in the following sections. Also, for a general minimization problem with n variable and m constraints, we will use the stencil shown in Figure 2.2 to calculate ∂f with respect to any variable.

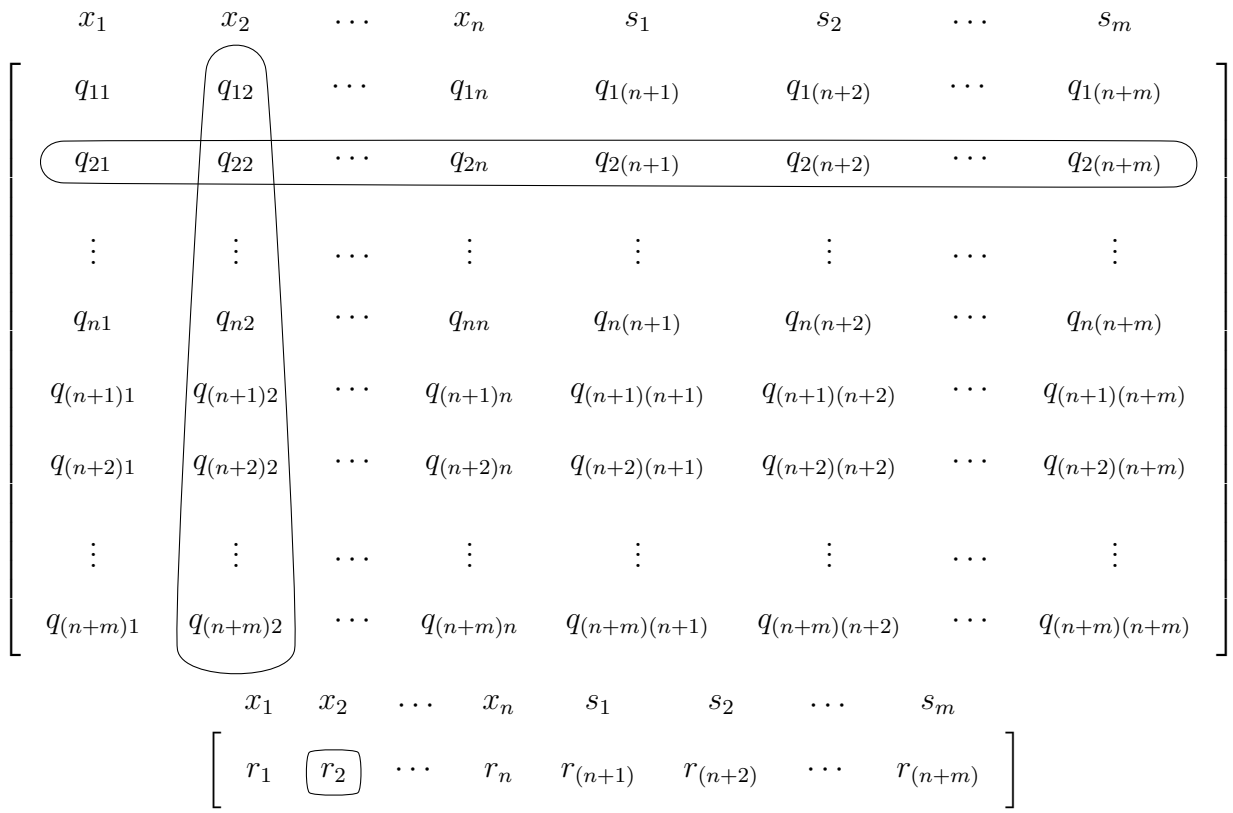

Figure 2.2. Stencil used for calculating change in function value with respect to any variable.

2.8 Solution Procedure for General IP Problem with Inequality Constraints

Consider the problem in Example 1 with 2 integer type unknown variables (x_1 and x_2) and 3 slack variables (s_1, s_2 and s_3), which are also integer type. The solution procedure and the update rules for the integer type unknown variables (x) are presented first, followed by the solution procedure and the update rules for integer type slack variables (s).

2.8.1 Integer Type Unknown Variables

The critical point of $f(x, s)$ based on the variable x_1 is obtained as follows:

$$f(x_1) = x_1(q_{11}x_1 + q_{12}x_2 + q_{21}x_2 + q_{13}s_1 + q_{31}s_1 + q_{14}s_2 + q_{41}s_2 + q_{15}s_3 + q_{51}s_3 + r_1) + c. \tag{2.72}$$

The partial derivative with respect to the unknown variable x_1 is:

$$\frac{\partial f}{\partial x_1} = 2q_{11}x_1 + q_{12}x_2 + q_{21}x_2 + q_{13}s_1 + q_{31}s_1 + q_{14}s_2 + q_{41}s_2 + q_{15}s_3 + q_{51}s_3 + r_1. \tag{2.73}$$

Since $q_{12} = q_{21}, q_{13} = q_{31}, q_{14} = q_{41}, \text{and } q_{15} = q_{51}$:

$$\frac{\partial f}{\partial x_1} = 2q_{11}x_1 + 2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1. \tag{2.74}$$

The critical point $x_1^\star$ for the variable x_1 is obtained by equating the partial derivative to zero:

$$\begin{aligned}
&\frac{\partial f}{\partial x_1} = 0 \implies \\
&2q_{11}x_1 + 2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1 = 0, \\
&2q_{11}x_1 = -(2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1), \\
&x_1 = \frac{2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1}{-2q_{11}}, \\
&\text{or, } x_1^\star = \frac{2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1}{-2q_{11}}.
\end{aligned} \tag{2.75}$$

Substituting the original values from (2.70) in (2.75), we obtain:

$$x_1^\star = \frac{2P(a_1^\top a_2)x_2 + 2Pa_{11}s_1 + 2Pa_{21}s_2 + 2Pa_{31}s_3 + (d_1 - 2P(b^\top a_1))}{-2P(a_1^\top a_1)}, \tag{2.76}$$

which is a critical point at which the function $f(x, s)$ is at relative extremum (relative minimum for our minimization problem).

The change in $f(x,s)$ due to change in x_1 is calculated as follows:

$$f(x_1) = x_1(q_{11}x_1 + q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1) + c. \tag{2.77}$$

If x_1 changes to $x_1^{'}$, then $f(x_1^{'})$ is given by:

$$f(x_1^{'}) = x_1^{'}(q_{11}x_1^{'} + q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1) + c. \tag{2.78}$$

The difference between (2.78) and (2.77) gives the change in value of function when x_1 changes to $x_1^{'}$. Therefore,

$$\begin{aligned}
\Delta f_{x_1^{'}} &= f(x_1^{'}) - f(x_1), \\
&= (x_1^{'} - x_1)(q_{11}(x_1^{'} + x_1) + q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1), \\
&= q_{11}x_1^{'2} - q_{11}x_1^2 + (q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1)x_1^{'} - \\
&\quad (q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1)x_1.
\end{aligned} \tag{2.79}$$

Since $q_{12} = q_{21}, q_{13} = q_{31}, q_{14} = q_{41}$, and $q_{15} = q_{51}$:

$$\begin{aligned}
\Delta f_{x_1^{'}} = q_{11}x_1^{'2} - q_{11}x_1^2 + \\
(2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)x_1^{'} - \\
(2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)x_1.
\end{aligned} \tag{2.80}$$

Note that the partial derivative of (2.80) with respect to $x_1^{'}$ is the same as (2.74):

$$\frac{\partial \Delta f_{x_1^{'}}}{\partial x_1^{'}} = 2q_{11}x_1^{'} + 2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1. \tag{2.81}$$

If the value of x_1 is updated to $x_1^\star$, we obtain Δf due to change in x_1 to $x_1^\star$ as follows:

$$\begin{aligned}
\Delta f_{x_1^\star} &= q_{11}x_1^{\star 2} - q_{11}x_1^2 + \\
&\quad (2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)x_1^\star - (2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)x_1, \\
&= \frac{(2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)^2}{4q_{11}} - q_{11}x_1^2 + \\
&\quad \frac{(2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)^2}{-2q_{11}} - (2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)x_1, \\
&= \frac{(2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)^2}{-4q_{11}} - (q_{11}x_1 + 2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)x_1, \\
&= -[q_{11}x_1^{\star 2} + (q_{11}x_1 + 2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)x_1].
\end{aligned} \tag{2.82}$$

Substituting the original values from (2.70) in (2.82), we obtain:

$$\begin{aligned}
\Delta f_{x_1^\star} &= -x_1^{\star 2}P(a_1^\top a_1) - \\
&\quad x_1(P(a_1^\top a_1)x_1 + 2P(a_1^\top a_2)x_2 + 2Pa_{11}s_1 + 2Pa_{21}s_2 + 2Pa_{31}s_3 + (d_1 - 2P(b^\top a_1))).
\end{aligned} \tag{2.83}$$

Using the similar approach, the value of $x_2^\star$ and Δf_{x_2} is:

$$x_2^\star = \frac{2q_{21}x_1 + 2q_{23}s_1 + 2q_{24}s_2 + 2q_{25}s_3 + r_2}{-2q_{22}}, \tag{2.84a}$$

$$\Delta f_{x_2^\star} = -[q_{22}x_2^{\star 2} + (q_{22}x_2 + 2q_{21}x_1 + 2q_{23}s_1 + 2q_{24}s_2 + 2q_{25}s_3 + r_2)x_2]. \tag{2.84b}$$

The general form for $x_i^\star$, $\Delta f_{x_i^\star}$, and $\Delta f_{x_1^{'}}$ for a problem with n integer variables and m inequality constraints shown in Figure 2.2 is:

$$x_i^\star = \frac{2\sum_{k=1}^{i-1} q_{ik}x_k + 2\sum_{k=i+1}^{n} q_{ki}x_k + 2\sum_{k=1}^{m} q_{i(n+k)}s_k + r_i}{-2q_{ii}}, \tag{2.85a}$$

$$\Delta f_{x_i^\star} = -q_{ii}x_i^{\star 2} - (q_{ii}x_i + 2\sum_{k=1}^{i-1} q_{ik}x_k + 2\sum_{k=i+1}^{n} q_{ki}x_k + 2\sum_{k=1}^{m} q_{i(n+k)}s_k + r_i)x_i, \tag{2.85b}$$

$$\Delta f_{x_i^{'}} = (x_i^{'} - x_i)(q_{ii}(x_i^{'} + x_i) + 2\sum_{k=1}^{i-1} q_{ik}x_k + 2\sum_{k=i+1}^{n} q_{ki}x_k + 2\sum_{k=1}^{m} q_{i(n+k)}s_k + r_i). \tag{2.85c}$$

Substituting the original values from (2.70) in (2.85) we obtain:

$$x_i^\star = \frac{2P\sum_{k=1}^{i-1}(a_k^\top a_i)x_k + 2P\sum_{k=i+1}^{n}(a_i^\top a_k)x_k + 2P\sum_{k=1}^{m} a_{ki}s_k + (d_i - 2P(b^\top a_i))}{-2P(a_i^\top a_i)},$$

(2.86a)

$$\Delta f_{x_i^\star} = -P(a_i^\top a_i)x_i^{\star 2} - (P(a_i^\top a_i)x_i + 2P\sum_{k=1}^{i-1}(a_k^\top a_i)x_k + 2P\sum_{k=i+1}^{n}(a_i^\top a_k)x_k + 2P\sum_{k=1}^{m} a_{ki}s_k + (d_i - 2P(b^\top a_i)))x_i,$$

(2.86b)

$$\Delta f_{x_i'} = (x_i' - x_i)(P(a_i^\top a_i)(x_i' + x_i)) + (x_i' - x_i)(2P\sum_{k=1}^{i-1}(a_k^\top a_i)x_k + 2P\sum_{k=i+1}^{n}(a_i^\top a_k)x_k + 2P\sum_{k=1}^{m} a_{ki}s_k + (d_i - 2P(b^\top a_i))).$$

(2.86c)

Updating Integer Type Unknown Variables

In the previous section, we calculated the value of $x_i^\star$ which is the critical value of x_i at which the function $f(x, s)$ equals zero, we then calculated $\Delta f_{x_i'}$ when the variable changes from x_i to x_i', and finally we calculated $\Delta f_{x_i^\star}$ which shows the change in $f(x, s)$ if the variable changes from x_i to $x_i^\star$. Based on this information we will show that updating variable x_i to a new value is based up on x_i and $x_i^\star$, and using graphical representations we will show the feasible and infeasible regions for the new value of x_i.

Recall from the previous section that if x_1 changes to x_1' then $\Delta f_{x_i'}$ is given by:

$$\begin{aligned}\Delta f_{x_1'} &= f(x_1') - f(x_1), \\ &= (x_1' - x_1)(q_{11}(x_1' + x_1) + q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1),\end{aligned}$$

(2.87)

In (2.87) the value of $\Delta f_{x_1'}$ is equal to zero when one of the following condition is met:

$$\Delta f_{x_1^{'}} = 0 \implies \tag{2.88a}$$

$$x_1^{'} - x_1 = 0, \tag{2.88b}$$

$$q_{11}(x_1^{'} + x_1) + q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1 = 0. \tag{2.88c}$$

From (2.88b), if $x_1^{'} - x_1 = 0$ then the variable is not changing, since $x_1^{'} = x_1$, and $\Delta f_{x_1^{'}}$ is always 0 in this case. But, from (2.88c), we obtain the following:

$$\begin{aligned}
&q_{11}(x_1^{'} + x_1) + q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1 = 0,\\
&q_{11}(x_1^{'}) = -(q_{11}(x_1) + q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1),\\
&x_1^{'} = \frac{-(q_{11}(x_1) + q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1)}{q_{11}},\\
&x_1^{'} = \frac{-(q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3 + r_1)}{q_{11}} - x_1,\\
&x_1^{'} = \frac{-(2q_{12}x_2 + 2q_{13}s_1 + 2q_{14}s_2 + 2q_{15}s_3 + r_1)}{q_{11}} - x_1,\\
&x_1^{'} = 2x_1^{\star} - x_1.
\end{aligned} \tag{2.89}$$

From (2.89), if the value of variable is changed from x_1 to $x_1^{'}$ where $x_1^{'} = 2x_1^{\star} - x_1$, then $\Delta f_{x_1^{'}} = 0$.

To generalize our findings, $x_i^{\star}$ represents the critical value at which the function $f(x, s)$ is at relative extremum (relative minimum for our minimization problem). The value of $\Delta f_{x_i^{'}} = 0$ when:

- $x_i^{'} = x_i$ (new value equals old value).
- $x_i^{'} = 2x_i^{\star} - x_i$ (new value = $2x_i^{\star}$ - old value).

The variable update procedure is simply based on x_i and $x_i^\star$, and the four possible combinations of these two parameters give the following cases. Note that the figures shown in these cases are parabolic since Δf_{x_i} is quadratic in nature.

- Case 1: $x_i = 0$ and $x_i^\star \leq 0$

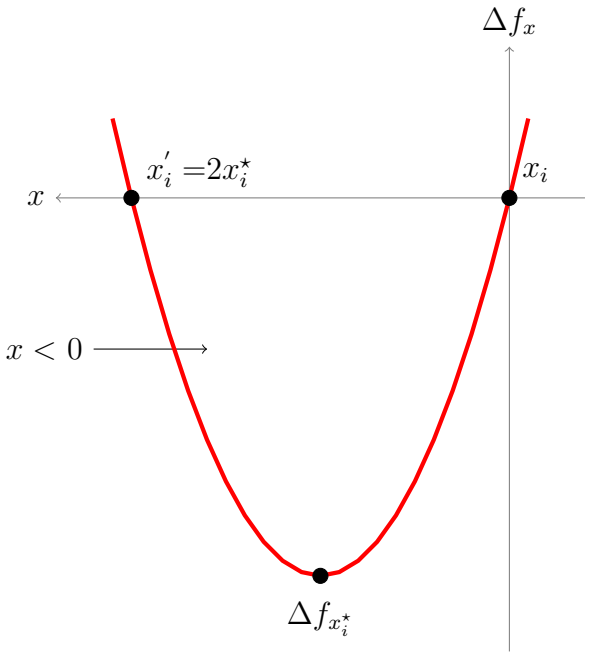

Figure 2.3. Change in Δf_x when: $x_i = 0$ and $x_i^\star \leq 0$.

Figure 2.3 represents the change in Δf_x when $x_i = 0$ and $x_i^\star \leq 0$. Based on this figure, the best new value for x_i is $x_i^\star$ where $\Delta f_{x_i^\star}$ is obtained, but since $x_i^\star < 0$ the new value of x_i is kept at 0.

- Case 2: $x_i = 0$ and $x_i^\star > 0$

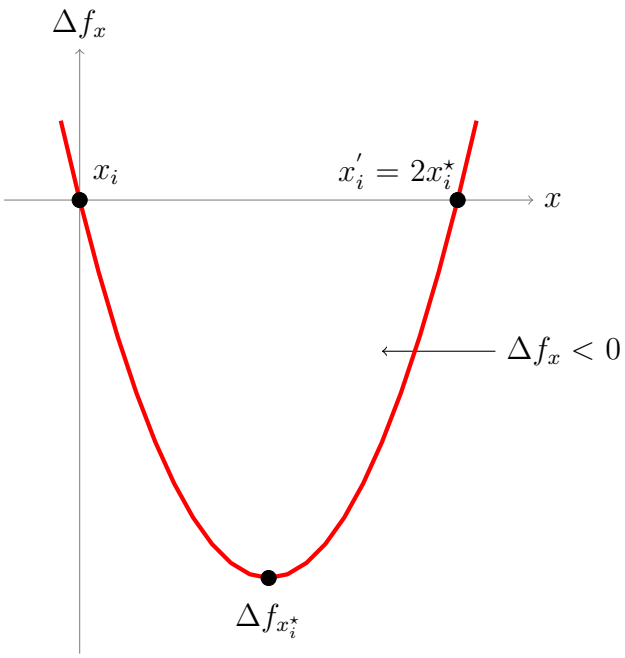

Figure 2.4. Change in Δf_x when: $x_i = 0$ and $x_i^\star > 0$.

Figure 2.4 represents the change in Δf_x when $x_i = 0$ and $x_i^\star > 0$. Based on this, the best new value for x_i is $x_i^\star$, since maximum absolute value $\Delta f_{x_i^\star}$ is obtained at $x_i^\star$. If $x_i^\star$ is not integer then the best integer value based on Δf_x that is closer to $x_i^\star$ is chosen. Also, in an attempt to escape local optimality of the objective function, x_i can be updated to an integer value of x, where $x_i < x \leq 2x_i^\star$.

- Case 3: $x_i > 0$ and $x_i^\star \leq 0$

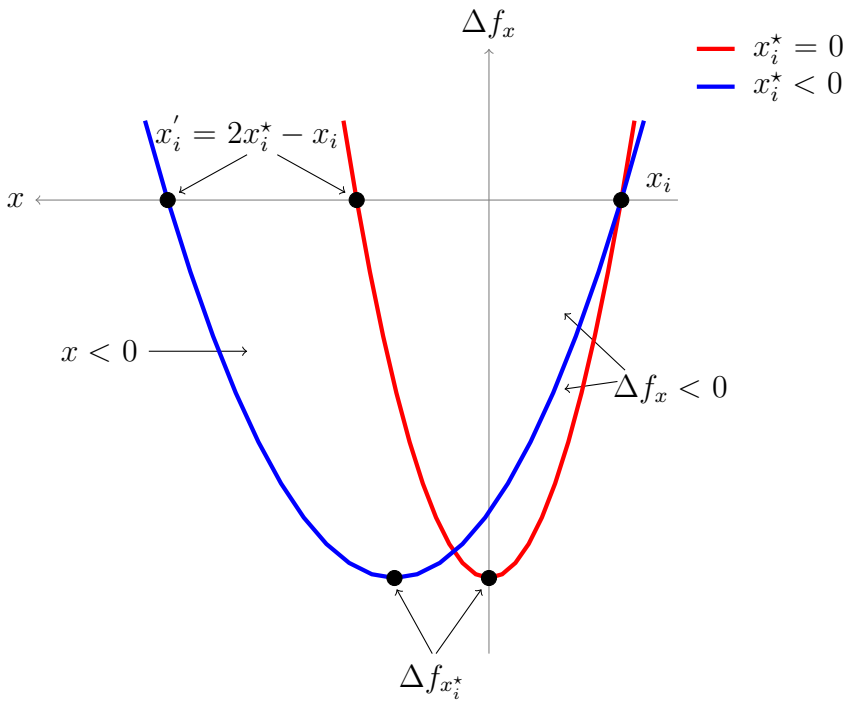

Figure 2.5. Change in Δf_x when: $x_i > 0$ and $x_i^\star \leq 0$.

Figure 2.5 represents two different curves based on the value of $x_i^\star$.

If $x_i^\star = 0$, represented by the red curve, the best new value of x_i is $x_i^\star$ since the maximum absolute value $\Delta f_{x_i^\star}$ is obtained at $x_i^\star$. A new value for x_i can also be updated to an integer value of x, where $0 \leq x < x_i$ to escape local optimality of the objective function value.

If $x_i^\star < 0$, represented by the blue curve, the best new value of x_i is $x_i^\star$ since the maximum absolute value $\Delta f_{x_i^\star}$ is obtained at $x_i^\star$. Since $x_i^\star < 0$, the new value of x_i is updated to 0, or to an integer value of x where $0 \leq x < x_i$.

- Case 4: $x_i > 0$ and $x_i^\star > 0$

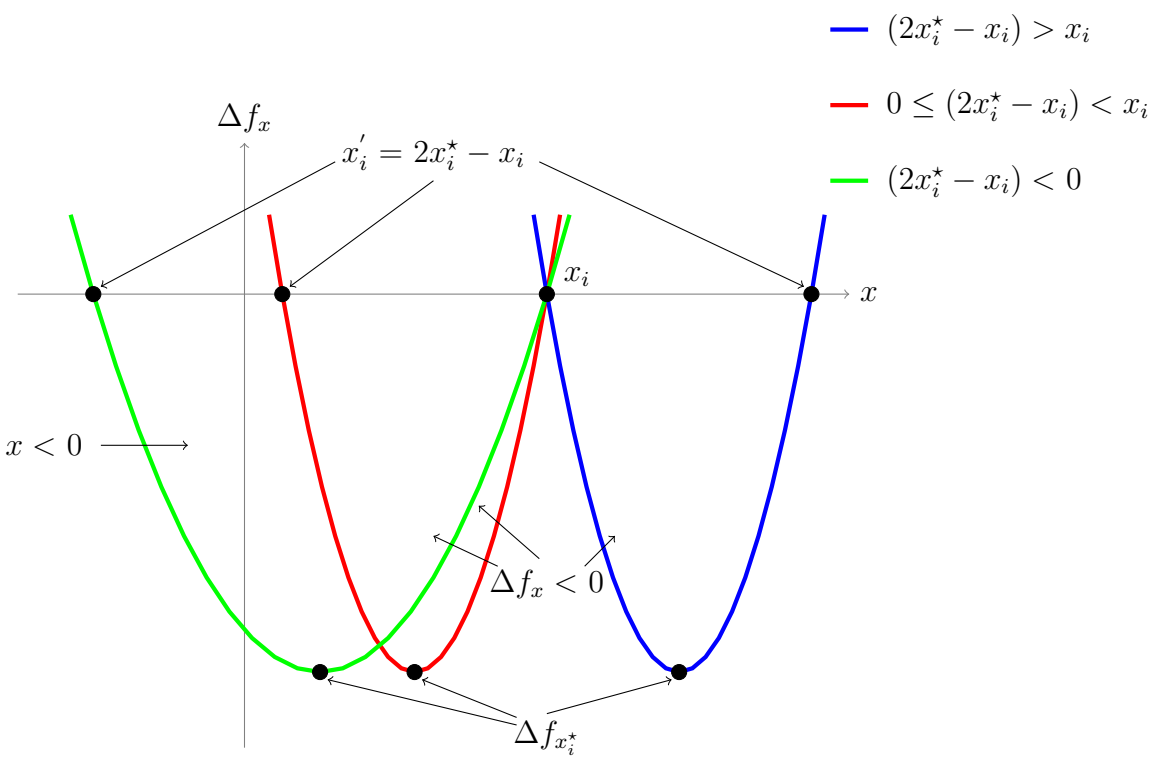

Figure 2.6. Change in Δf_x when: $x_i > 0$ and $x_i^\star > 0$.

Figure 2.6 represents three different curves based on the value of $2x_i^\star - x_i$.

If $(2x_i^\star - x_i) > x_i$, represented by blue curve, the best new value for x_i is $x_i^\star$ since maximum absolute value $\Delta f_{x_i^\star}$ is obtained at $x_i^\star$. If $x_i^\star$ is not integer then the best integer value based on Δf_x that is closer to $x_i^\star$ is chosen. Also, in an attempt to escape local optimality of the objective function, x_i can be updated to an integer value of x, where $x_i < x \leq 2x_i^\star - x_i$.

If $0 \leq (2x_i^\star - x_i) < x_i$, represented by red curve, the best new value for x_i is $x_i^\star$ since maximum absolute value $\Delta f_{x_i^\star}$ is obtained at $x_i^\star$. If $x_i^\star$ is not integer then the best integer value based on Δf_x that is closer to $x_i^\star$ is chosen. Also, in an attempt to escape

local optimality of the objective function, x_i can be updated to an integer value of x, where $2x_i^\star - x_i \leq x < x_i$.

If $(2x_i^\star - x_i) < 0$, represented by green curve, the best new value for x_i is $x_i^\star$ since maximum absolute value $\Delta f_{x_i^\star}$ is obtained at $x_i^\star$. If $x_i^\star$ is not integer then the best integer value based on Δf_x that is closer to $x_i^\star$ is chosen. Also, in an attempt to escape local optimality of the objective function, x_i can be updated to an integer value of x, where $0 \leq x < x_i$.

In Table 2.3, we summarize the update rules from the four cases. Here x_i represents the current value of the variable and $x_i^{'}$ represents the new value of the same variable. Note that an integer value is always selected for the new value of x_i.

Case	x_i	$x_i^\star$	$(2x_i^\star - x_i)$	Best $x_i^{'}$	Acceptable $x_i^{'}$
1	0	≤ 0	≤ 0	0	0
2	0	> 0	> 0	$x_i^\star$	$x_i < x \leq 2x_i^\star$
3	> 0	0	< 0	0	$0 \leq x < x_i$
	> 0	< 0	< 0	0	$0 \leq x < x_i$
4	> 0	> 0	$> x_i$	$x_i^\star$	$x_i < x \leq 2x_i^\star - x_i$
	> 0	> 0	≥ 0 & $< x_i$	$x_i^\star$	$2x_i^\star - x_i \leq x < x_i$
	> 0	> 0	< 0	$x_i^\star$	$0 \leq x < x_i$

Table 2.3: Updating integer type unknown variable based on critical value.

2.8.2 Slack Variables

The critical point of $f(x, s)$ based on the variable s_1 is obtained as follows:

$$f(s_1) = s_1(q_{33}s_1 + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3) + c. \tag{2.90}$$

The partial derivative with respect to the slack variable s_1 is:

$$\begin{aligned}\frac{\partial f}{\partial s_1} &= 2q_{33}s_1 + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3, \\ &= 2q_{33}s_1 + 2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3.\end{aligned} \tag{2.91}$$

The critical point $s_1^\star$ for the variable s_1 is obtained by equating the partial derivative to zero:

$$\begin{aligned}&\frac{\partial f}{\partial s_1} = 0 \implies \\ &2q_{33}s_1 + 2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3 = 0, \\ &2q_{33}s_1 = -(2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3), \\ &s_1 = \frac{2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3}{-2q_{33}}, \\ &\text{or, } s_1^\star = \frac{2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3}{-2q_{33}}.\end{aligned} \tag{2.92}$$

Substituting the original values from (2.70) in (2.92) we obtain:

$$\begin{aligned}s_1^\star &= \frac{2Pa_{11}x_1 + 2Pa_{12}x_2 - 2Pb_1}{-2P}, \\ &= b_1 - (a_{11}x_1 + a_{12}x_2).\end{aligned} \tag{2.93}$$

Note that $s_1^\star$ simply represents the first constraint from Example 1, and $s_1^\star$ represents a critical point at which the function $f(x, s)$ is at relative extremum (relative minimum for our minimization problem).

The change in $f(x, s)$ due to change in s_1 is calculated as follows:

$$f(s_1) = s_1(q_{33}s_1 + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3) + c. \tag{2.94}$$

If s_1 changes to $s_1^{'}$, then $f(s_1^{'})$ is given by:

$$f(s_1^{'}) = s_1^{'}(q_{33}s_1 + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3) + c. \tag{2.95}$$

The difference between (2.95) and (2.94) gives the change in value of function when s_1 changes to $s_1^{'}$. Therefore,

$$\begin{aligned}
\Delta f_{s_1^{'}} &= f(s_1^{'}) - f(s_1), \\
&= (s_1^{'} - s_1)(q_{33}(s_1^{'} + s_1) + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3), \\
&= q_{33}s_1^{'2} - q_{33}s_1^2 + (q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3)s_1^{'} - \\
&\quad (q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3)s_1.
\end{aligned} \tag{2.96}$$

Since $q_{13} = q_{31}, q_{23} = q_{32}, q_{34} = q_{43}$, and $q_{35} = q_{53}$:

$$\begin{aligned}
\Delta f_{s_1^{'}} = q_{33}s_1^{'2} - q_{33}s_1^2 + \\
(2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)s_1^{'} - \\
(2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3)s_1.
\end{aligned} \tag{2.97}$$

Note that the partial derivative of (2.97) with respect to $s_1^{'}$ is the same as (2.92):

$$\frac{\partial \Delta f_{s_1^{'}}}{\partial s_1^{'}} = 2q_{33}s_1^{'} + 2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3. \tag{2.98}$$

If the value of s_1 is updated to $s_1^\star$, we obtain Δf due to change in s_1 to $s_1^\star$ as follows:

$$\begin{aligned}
\Delta f_{s_1^\star} &= q_{33}s_1^{\star 2} - q_{33}s_1^2 + \\
&\quad (2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)s_1^\star - (2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)s_1, \\
&= \frac{(2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)^2}{4q_{33}} - q_{33}s_1^2 + \\
&\quad \frac{(2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)^2}{-2q_{33}} - (2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)s_1, \\
&= \frac{(2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)^2}{-4q_{33}} - (q_{33}s_1 + 2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)s_1, \\
&= -[q_{33}s_1^{\star 2} + (q_{33}s_1 + 2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)s_1].
\end{aligned} \tag{2.99}$$

Substituting the original values from (2.70) in (2.99), we obtain:

$$\Delta f_{s_1^\star} = -s_1^{\star 2}P - s_1(s_1P + 2P(a11)x_1 + 2P(a_{12})x_2 - 2P(b_1)). \tag{2.100}$$

The general form for $s_i^\star$, $\Delta f_{s_1^\star}$ and $\Delta f_{s_1^{'}}$ for a problem with n boolean variables and m inequality constraints shown in Figure 2.2 is:

$$s_i^\star = \frac{2\sum_{k=1}^{n} q_{(k)(n+i)}x_k + 2\sum_{k=1}^{i-1} q_{(n+k)(n+i)}s_k + 2\sum_{k=i+1}^{m} q_{(n+k)(n+i)}s_k + r_{n+i}}{-2q_{(n+i)(n+i)}}, \quad (2.101a)$$

$$\Delta f_{s_i^\star} = -q_{(n+i)(n+i)}s_i^{\star 2} - s_i(q_{(n+i)(n+i)}s_i + 2\sum_{k=1}^{n} q_{(k)(n+i)}x_k + 2\sum_{k=1}^{i-1} q_{(n+k)(n+i)}s_k + 2\sum_{k=i+1}^{m} q_{(n+k)(n+i)}s_k + r_{n+i}), \quad (2.101b)$$

$$\Delta f_{s_i^{'}} = (s_i^{'} - s_i)(q_{(n+i)(n+i)}(s_i^{'} + s_i)) + (s_i^{'} - s_i)(2\sum_{k=1}^{n} q_{(k)(n+i)}x_k + 2\sum_{k=1}^{i-1} q_{(n+k)(n+i)}s_k + 2\sum_{k=i+1}^{m} q_{(n+k)(n+i)}s_k + r_{n+i}). \quad (2.101c)$$

For the general form above, substituting the original values from (2.70) in (2.101), we obtain:

$$s_i^\star = b_i - (\sum_{k=1}^{n} a_{ik}x_k), \quad (2.102a)$$

$$\Delta f_{s_i^\star} = -s_i^{*2}P - s_i^2 P + 2Ps_i(b_i - (\sum_{k=1}^{n} a_{ik}x_k)), \quad (2.102b)$$

$$\Delta f_{s_i^{'}} = (s_i^{'} - s_i)(P(s_i^{'} + s_i) - 2P(b_i - (\sum_{k=1}^{n} a_{ik}x_k))). \quad (2.102c)$$

Updating Slack Variables

The update procedure for slack variables will be similar to the one presented for integer type unknown variables in Section 2.8.1 without the integer restriction, since slack variables can be ≥ 0. Updating a slack variable from s_i to $s_i^{'}$ depends only on s_i to $s_i^\star$, and this is shown using graphical representations in this section.

From 2.96 the value of Δf when s_1 changes to $s_1^{'}$ is given by:

$$\begin{aligned}\Delta f_{s_1^{'}} &= f(s_1^{'}) - f(s_1), \\ &= (s_1^{'} - s_1)(q_{33}(s_1^{'} + s_1) + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3).\end{aligned} \tag{2.103}$$

In (2.103) the value of $\Delta f_{s_1^{'}}$ is equal to zero when one of the following condition is met:

$$\Delta f_{s_1^{'}} = 0 \implies \tag{2.104a}$$

$$s_1^{'} - s_1 = 0, \tag{2.104b}$$

$$q_{33}(s_1^{'} + s_1) + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3 = 0. \tag{2.104c}$$

In the first case above, since $s_1^{'} = s_1$, the value of $\Delta f_{s_1^{'}}$ is always 0. But from the second case, we obtain the following:

$$\begin{aligned}&q_{33}(s_1^{'} + s_1) + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3 = 0, \\ &q_{33}s_1^{'} = -(q_{33}s_1 + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3), \\ &s_1^{'} = \frac{-(q_{33}s_1 + q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3)}{q_{33}}, \\ &s_1^{'} = \frac{-(q_{13}x_1 + q_{31}x_1 + q_{23}x_2 + q_{32}x_2 + q_{43}s_2 + q_{34}s_2 + q_{53}s_3 + q_{35}s_3 + r_3)}{q_{33}} - s_1, \\ &s_1^{'} = \frac{-(2q_{13}x_1 + 2q_{23}x_2 + 2q_{43}s_2 + 2q_{53}s_3 + r_3)}{q_{33}} - s_1, \\ &s_1^{'} = 2s_1^{\star} - s_1.\end{aligned} \tag{2.105}$$

From (2.105), if the value of slack variable is changed from s_1 to $s_1^{'}$ where $s_1^{'} = 2s_1^{\star} - s_1$, then $\Delta f_{s_1^{'}} = 0$.

To generalize our findings, $s_i^{\star}$ represents the critical value at which the function $f(x, s)$ is at relative extremum (relative minimum for our minimization problem). The value of $\Delta f_{s_i^{'}} = 0$ when:

- $s_i^{'} = s_i$ (new value equals old value).

- $s_i^{'} = 2s_i^{\star} - s_i$ (new value = $2s_i^{\star}$ - old value).

Based on the values of s_i and $s_i^{\star}$ we can obtain the four possible combinations for which the varible update procedure is shown pictorially.

- Case 1: $s_i = 0$ and $s_i^{\star} \leq 0$

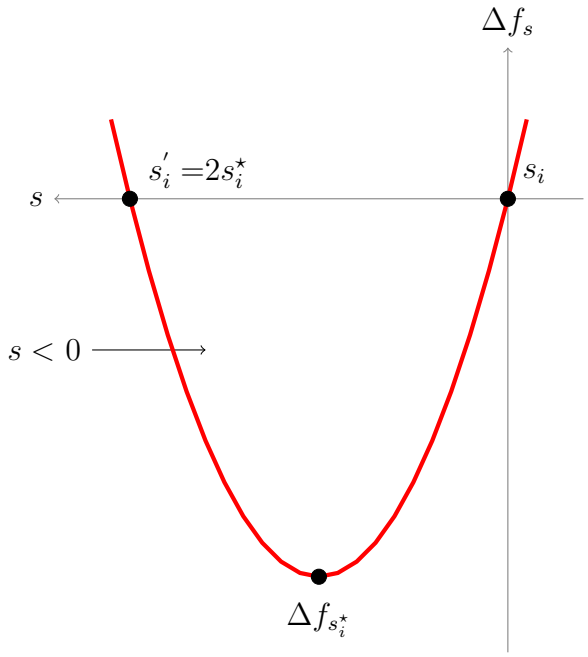

Figure 2.7. Change in Δf_s when: $s_i = 0$ and $s_i^{\star} \leq 0$.

Figure 2.7 represents the change in Δf_s when $s_i = 0$ and $s_i^{\star} \leq 0$. Based on this figure the best new value for s_i is $s_i^{\star}$ where $\Delta f_{s_i^{\star}}$ is obtained, but since $s_i^{\star} < 0$, the new value of s_i is kept at 0. This is the case in which the constraint related with the variable s_i is currently under violation due to the values assigned to the unknown variables x.

- Case 2: $s_i = 0$ and $s_i^{\star} > 0$

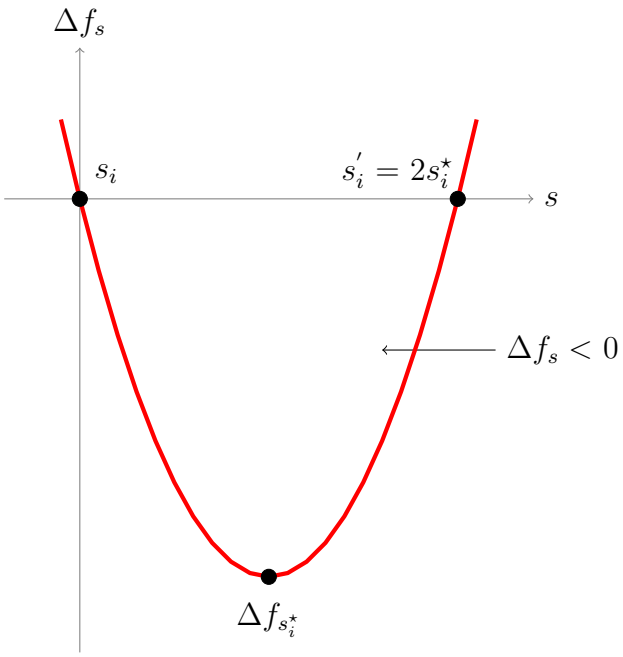

Figure 2.8. Change in Δf_s when: $s_i = 0$ and $s_i^{\star} > 0$.

Figure 2.8 represents the change in Δf_s when $s_i = 0$ and $s_i^{\star} > 0$. Based on this, the best new value for s_i is $s_i^{\star}$ since maximum absolute value $\Delta f_{s_i^{\star}}$ is obtained at $s_i^{\star}$. If $s_i^{'} > s_i^{\star}$ then a constraint violation occurs, hence the range $s_i < s \leq s_i^{\star}$ will be given more importance in our heuristic procedure. Also, in an attempt to escape local optimality of the objective function, s_i can be updated to a value of s, where $s_i < s \leq 2s_i^{\star}$.

- Case 3: $s_i > 0$ and $s_i^\star \leq 0$

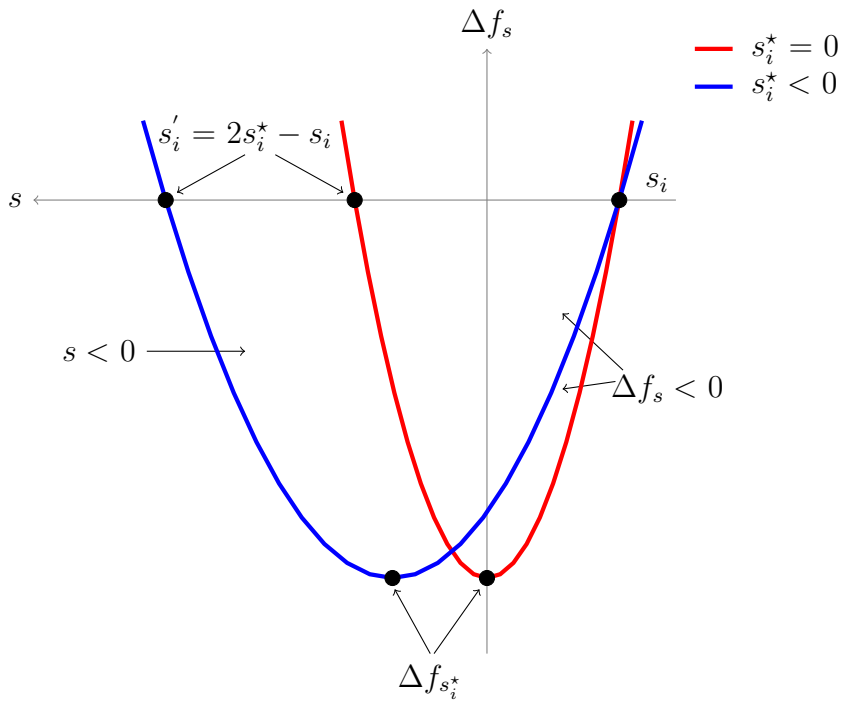

Figure 2.9. Change in Δf_s when: $s_i > 0$ and $s_i^\star \leq 0$.

Figure 2.9 represents two different curves based on the value of $s_i^\star$. In both the conditions represented here, the corresponding constraint will be violated if $s_i^{'} > 0$.

If $s_i^\star = 0$, represented by the red curve, the best new value of s_i is $s_i^\star$ since the maximum absolute value $\Delta f_{s_i^\star}$ is obtained at $s_i^\star$. A new value for s_i can also be updated to a value of s, where $0 \leq s < s_i$ to escape local optimality of the objective function value.

If $s_i^\star < 0$, represented by the blue curve, the best new value of s_i is $s_i^\star$ since the maximum absolute value $\Delta f_{s_i^\star}$ is obtained at $s_i^\star$. Since $s_i^\star < 0$, the new value of s_i is updated to 0, or to a value of s where $0 \leq s < s_i$.

- Case 4: $s_i > 0$ and $s_i^\star > 0$

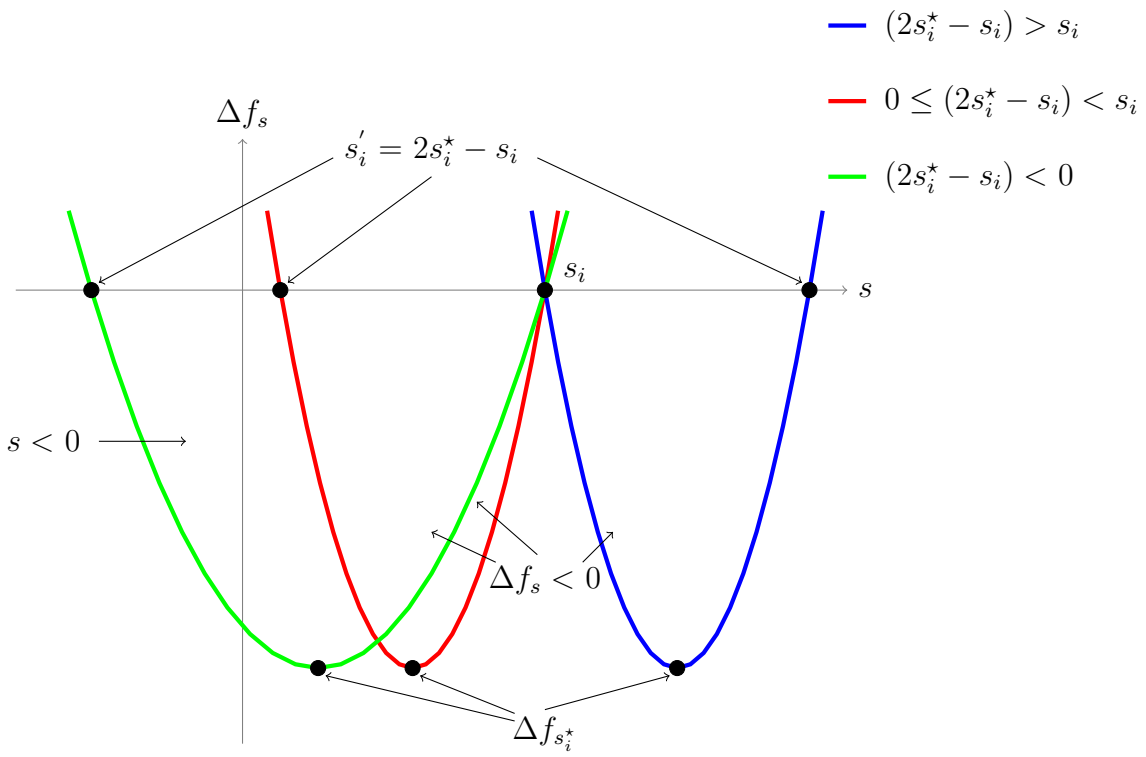

Figure 2.10. Change in Δf_s when: $s_i > 0$ and $s_i^\star > 0$.

Figure 2.10 represents three different curves based on the value of $2s_i^\star - s_i$.

If $(2s_i^\star - s_i) > s_i$, represented by blue curve, the best new value for s_i is $s_i^\star$ since maximum absolute value $\Delta f_{s_i^\star}$ is obtained at $s_i^\star$. If $s_i^{'} > s_i^\star$ then a constraint violation occurs, hence the range $s_i < s \leq s_i^\star$ will be given more importance in our heuristic procedure. Also, in an attempt to escape local optimality of the objective function, s_i can be updated to a value of s, where $s_i < s \leq 2s_i^\star - s_i$.

If $0 \leq (2s_i^\star - s_i) < s_i$, represented by red curve, the best new value for s_i is $s_i^\star$ since maximum absolute value $\Delta f_{s_i^\star}$ is obtained at $s_i^\star$. Also, in an attempt to escape local optimality of the objective function, s_i can be updated to a value of s, where

$2s_i^\star - s_i \leq s < s_i$.

If $(2s_i^\star - s_i) < 0$, represented by green curve, the best new value for s_i is $s_i^\star$ since maximum absolute value $\Delta f_{s_i^\star}$ is obtained at $s_i^\star$. Also, in an attempt to escape local optimality of the objective function, s_i can be updated to a value of s, where $0 \leq s < s_i$.

In Table 2.4, we summarize the update rules from the four cases. s_i represents the current value of the variable and $s_i^{'}$ represents the new value of the same variable. Note that integer restriction is not applied for s_i.

Case	s_i	$s_i^\star$	$(2s_i^\star - s_i)$	Best $s_i^{'}$	Acceptable $s_i^{'}$
1	0	≤ 0	≤ 0	0	0
2	0	> 0	> 0	$s_i^\star$	$s_i < s \leq 2s_i^\star$
3	> 0	0	< 0	0	$0 \leq s < s_i$
	> 0	< 0	< 0	0	$0 \leq s < s_i$
4	> 0	> 0	$> s_i$	$s_i^\star$	$s_i < s \leq 2s_i^\star - s_i$
	> 0	> 0	≥ 0 & $< s_i$	$s_i^\star$	$2s_i^\star - s_i \leq s < s_i$
	> 0	> 0	< 0	$s_i^\star$	$0 \leq s < s_i$

Table 2.4: Updating integer type slack variable based on critical value.

2.8.3 Solution Procedure for Binary IP Problem with Inequality Constraints

We apply the same gradient descent procedure used for the General IP problem with integer unknowns. In gradient descent approach, we determine the change in the value of a function $f(x)$ with respect to a variable x_i. Based on this change, the variable x_i is either changed from 0 to 1 or vice versa for a boolean type variable. Consider Example 1 with binary unknowns instead of integer unknowns. In this problem, we have 2 boolean type variables (x_1 and x_2) and 3 slack variables (s_1, s_2 and s_3) that are integers. The partial derivatives obtained with respect to the unknown variables here will be different from the ones obtained for integer type since $x_i^2 \neq x_i$ in the general case.

$$f(x_1) = x_1(q_{11}x_1 + q_{12}x_2 + q_{21}x_2 + q_{13}s_1 + q_{31}s_1 + q_{14}s_2 + q_{41}s_2 + q_{15}s_3 + q_{51}s_3 + r_1) + c. \tag{2.106}$$

Note that $f(x_1)$ was obtained using (2.71). Since x_i is boolean, $x_i^2 = x_i$, the partial derivative is:

$$\frac{\partial f}{\partial x_1} = (q_{11} + r_1) + q_{12}x_2 + q_{13}s_1 + q_{14}s_2 + q_{15}s_3 + q_{21}x_2 + q_{31}s_1 + q_{41}s_2 + q_{51}s_3. \tag{2.107}$$

Substituting the original values from (2.70) in (2.107), we obtain:

$$\frac{\partial f}{\partial x_1} = (P(a_1^\top a_1) + (d_1 - 2P(b^\top a_1))) + 2P(a_1^\top a_2)x_2 + 2Pa_{11}s_1 + 2Pa_{21}s_2 + 2Pa_{31}s_3. \tag{2.108}$$

Note that $q_{12} = q_{21}, q_{13} = q_{31}, q_{14} = q_{41}, \text{and } q_{15} = q_{51}$ was used to obtain (2.108). Similarly we obtain the partial derivative with respect to x_2:

$$\frac{\partial f}{\partial x_2} = (P(a_2^\top a_2) + (d_2 - 2P(b^\top a_2))) + 2P(a_1^\top a_2)x_1 + 2Pa_{12}s_1 + 2Pa_{22}s_2 + 2Pa_{32}s_3. \tag{2.109}$$

The general form of the partial derivative with respect to variable x_i for a problem with n boolean variables and m constraints shown in Figure 2.2 is:

$$\frac{\partial f}{\partial x_i} = (q_{ii} + r_i) + (\sum_{k=1}^{i-1} q_{ik}x_k + \sum_{k=1}^{i-1} q_{ki}x_k + \sum_{k=i+1}^{n} q_{ik}x_k + \sum_{k=i+1}^{n} q_{ki}x_k + \sum_{k=n+1}^{m} q_{ik}s_{(k-n)} + \sum_{k=n+1}^{m} q_{ki}s_{(k-n)}). \tag{2.110}$$

Substituting the actual values of the matrices obtained from (2.64) in the above general

form will produce:

$$\frac{\partial f}{\partial x_i} = (P(a_i^\top a_i) + (d_i - 2P(b^\top a_i))) + 2P\sum_{k=1}^{i-1}(a_k^\top a_i)x_k + 2P\sum_{k=i+1}^{n}(a_i^\top a_k)x_k + 2P\sum_{k=n+1}^{m} a_{ki}s_{(k-n)}. \tag{2.111}$$

The possible ways to update boolean variable x_i based on the partial derivative are shown in Table 2.5. If the partial derivative is < 0, the function is decreasing in value and the value x_i is changed to 1, as shown in rows 1 & 2 in the table. If the partial derivative is ≥ 0, the function is increasing and hence the value of x_i is changed to 0, as shown in rows 3 & 4.

Old value x_i	Value obtained for $\frac{\partial f}{\partial x_i}$	New value of x_i
0	< 0	1
1	< 0	1
0	≥ 0	0
1	≥ 0	0

Table 2.5: Updating boolean type variable based on partial derivative.

The partial derivatives for the slack variables in the binary IP case are exactly the same as the general IP case shown in Section 2.8.3, since it simply represent the constraints in the given problem. Hence, we will use the update procedure shown in Table 2.4 to update the slack variables in the boolean IP case.

2.9 Summary

In this chapter, we formally stated the IP problems from literature, presented the existing XQX models for IP problems currently used in literature, and presented existing solution procedures to solve the XQX models from literature. We then presented our XQX model for IP problems with inequality and equaity constraints and the local optima based solution procedure for the same. Our XQX model and the solution procedures are new contributions to the existing literature and are two of the four objectives of this work. To show the

application of our XQX model and our solution procedure, we consider 0-1 multidimensional knapsack problem (an instance of binary IP problem) in the next chapter. The definition of 0-1 MDKP, existing heuristic and exact methods to solve this problem, and the application of our XQX model and our solution procedure using a heuristic procedure to solve benchmark cases are presented next.

CHAPTER 3

0-1 MULTIDIMENSIONAL KNAPSACK PROBLEM

To show the application of our XQX model from section 2.5 we consider 0-1 multidimensional knapsack problem (0-1 MDKP) which is an instance of binary IP problem. The application of our XQX model to binary IP is the third objective of this thesis. The 0-1 MDKP problem is a strongly NP-hard problem and one of the most challenging in the class of knapsack problems (Mansini & Speranza, 2012). In this chapter, first we define one dimensional knapsack problem, then present the 0-1 MDKP, then describe the mathematical formulation for these problems from literature, and then present various real life applications for this problem. We then provide a detailed literature survey which includes the state-of-the art algorithms for solving 0-1 MDKP. Finally, we apply our XQX model and solution procedures on 0-1 MDKP using a basic heuristics that we have developed and present the results obtained from solving benchmark problems.

3.1 Definition

The one dimensional knapsack is defined by the following formulation (Kellerer *et al.*, 2004):

$$\begin{aligned} \text{maximise} \quad & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} r_j x_j \leq b, \\ & x_j \in \{0,1\},\ j = 1, \ldots, n, \end{aligned} \tag{3.1}$$

where p_j, r_j, and b are assumed to be positive integer values. In the above single resource constrained problem we are given n items each with profit p_j and cost r_j, and the objective of the problem is to select a subset of n items such that the profit is maximized and the cost doesn't exceed the capacity b.

The 0-1 MDKP includes multiple resource constraints instead and is defined by the following formulation (Kellerer *et al.*, 2004):

$$\begin{aligned} \text{maximise} \quad & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} r_{ij} x_j \leq b_i, \ i = 1, \ldots, m, \\ & x_j \in \{0, 1\}, \ j = 1, \ldots, n, \end{aligned} \tag{3.2}$$

where p_j, r_{ij}, and b_i are assumed to be positive integer values. Given n items, with each having profits p_j and m resources with capacities b_j, the goal is to select a subset of items with maximum total profit when each item j consumes r_{ij} from each of the i resources. This problem has many practical applications such as capital budgeting, project selection, cargo loading, and cutting stock problems (Chu & Beasley, 1998). It is to be noted that there are many different kinds of knapsack problems, and a 0-1 multidimensional knapsack problem is different from 0-1 multiple knapsack problem (Lin, 1998). We make this note since many papers in the literature solve a MDKP problem, but it is referred to as a multiple knapsack problem; Fidanova (2005) is one such paper.

3.2 Applications

The motivation for the work done by Weingartner & Ness (1967) is based on a capital budgeting problem. Capital budgeting problem is the process of making decisions about the financial desirability of a projects. Since capital budgeting decisions impact the firm for several years, they must be carefully planned. A bad decision can have a significant

effect on the firm's future operations, and the capital budgeting is a common issue faced by firms as suggested by Gow & Reichelstein (2007). We are given j different projects each with profit p_j. Project j is selected if $x_j = 1$, and each project j consumes r_{ij} of resource i where the total number of available resources is m. The m resources correspond to the various needs of the project such as man power, machines, inventory. Thus, the objective is to find a subset of the n projects such that the total profit is maximized without violating any resource constraints, and this problem was modeled as 0-1 MDKP.

Cargo loading problem on a ship or airplane is another application of 0-1 MDKP (Shih, 1979), and this problem has its importance in supply chain management. Consider a company shipping several packages where each package has a profit p_j, weight w_j, and volume v_j. Also, there is a weight limit l_1 and volume limit l_2 for the airplane or ship. The objective of this problem is to maximize the profits for the company by selecting a subset of packages while satisfying the weight and volume constraints.

The design of computer and database location in a wide-area network by Gavish & Pirkul (1986) was formulated as an integer programming problem, and the relaxed version of their formulation is a 0-1MDKP with three constraints. The objective of the problem is to determine the number of computers required at a location on a leased telecommunications link in order to process and store all jobs efficiently. The three constraints are: total processing requirements at a location should be less than maximum processing capacity considered, total storage requirement should be less than maximum capacity considered, and total telecommunication capacity must be less than maximum telecommunication capacity.

The scheduling of design process in a product development cycle was decomposed in to a series of 0-1 MDKP by Belhe & Kusiak (1997). A design process consists of several stages ranging from consumer requirements to product adoption, and reducing the product development cycle is an important goal for many industries when multiple concurrent design projects compete for the limited available resources. The authors considered five different phases, namely, identify customer demands, preliminary design, detailed design, build pro-

totype, and testing and evaluation to be executed serially for product development. Due to the dynamic nature of the resources considered, the product development was modeled as a MDKP problem to determine the best eligible activity at time t, and this information is used to model and solve at time $t+1$. The model consists of n activities and m resources where each activity x_j if scheduled has a profit of p_j. Each activity x_j requires many resources and consumes r_{ij} of resource type R_i.

Shelf space allocation problem was modeled as 0-1 MDKP with an added policy constraint by Yang (2001). Shelf space is an important resource in retail management and a well managed shelf space increases sales and profit, improves return on inventory, and improves customer satisfaction. The shelf space allocation problem consists of n products to be displayed in m shelves and each product has a profit p_j when displayed in shelf k. Each shelf m has a total length of T_i, and each product has a length of a_i. The additional policy constraint dictates a lower and upper bound for number of occurrences of each product in each shelf.

3.3 Literature Review

In this section, we present various existing methods that solve MDKP. Several methods, which include exact methods; such as branch & bound; exact methods with heuristics, dynamic programming, genetic algorithm, tabu search , memetic algorithm based, and others, are presented in this section. Even though there are other algorithms and heuristics that were not considered, the presented literature summarizes the important approaches in terms of breadth and performance. A comprehensive survey of different methods to solve MDKP can be found in Chu & Beasley (1998), Fréville (2004), (Kellerer *et al.*, 2004), and Varnamkhasti (2012). The exact methods presented here provided high quality solutions using very little time for small problems, but for larger problems, heuristics based methods, which consume more computational time to solve, provided the best solutions.

Balas (1965) proposed an additive algorithm to solve inequality constrained linear problems with 0-1 variables. The solution method sets all n variables to 0 and systematically assigns certain variables to 1 without going through all 2^n combinations. The author has solved problems with up to 15 variables and up to 12 constraints using hand calculations on an average of about 2 hours per problem.

Senju & Toyoda (1968) proposed efficient gradient method to find approximate solutions for MDKP problems. This method initially starts with all variables equal to 1, and then for each variable an effective gradient is calculated. To calculate the effective gradient, first the variables contributions to the constraint violation are calculated, then the ratio between the profit and the contribution to constraint is calculated to obtain the effective gradient. The effective gradients for all variables are then ordered in ascending order, and the variables with the lowest ratio are made equal to 0. When a feasible solution is found, the effective gradient method is repeated using the new constraint violation contributions based on the updated variables. The time complexity for this algorithm is $O(mn^2)$.

Hillier (1969) presented a 3 phase heuristic procedure for integer linear programming problems having only inequality constraints. In phase 1, optimum non integer solution using simplex algorithm and another near by solution whose rounded integer solution that does not violate any constraints are found. Then, in phase 2, the algorithm searches the line segment that connects these two points for a better solution. Finally, in phase 3, an attempt is made to improve the feasible solution found in phase 2. Different methods were also used in phases 2 and 3.

Zanakis (1977) compared the methods proposed by Senju & Toyoda (1968), Hillier (1969), and Kochenberger *et al.* (1974) where he concluded that one does not dominate the other computationally, and more over the three different heuristic approaches considered only small problem sizes.

Shih (1979) developed the first linear programming based branch & bound method to solve MDKP. In their method, a m- constraint knapsack problem is treated as m single

constraint knapsack problem (solving each single constraint problem separately) to estimate an upper bound and branching node. They applied this method to randomly generated problems with up to 90 variables and 5 constraints and obtained solutions to all problems with a superior solution time when compared to Balas (1965).

Loulou & Michaelides (1979) presented a greedy based heuristics for the MDKP. This method is similar to Senju & Toyoda (1968), but starts with all variables equal to zero, and is based on pseudo-utility function to select variables. The pseudo-utility is defined using multiple ways that include the consumption of resources, the amount of remaining resources, and the potential demand for the resource if a variable is selected. Few real-world problems were considered, and other problems with sizes up to $n = 45$ and $m = 330$ were solved. The method found high quality solutions for small problems and up to 20% deviation for larger problems.

Balas & Martin (1980) proposed a pivot and complement heuristics to solve MDKP problems. The method consists of two phases, pivot and complement for the search phase, and the improvement phase. In pivot and complement phase, the relaxed problem is solved using simplex and then it attempts to include the slack variables in to basis at minimal cost. The algorithm considered three type of pivots: type 1 that maintains feasibility and exchanges a slack variable in to basis; type 2 that maintains feasibility and leaves the number of basic variables unchanged after exchanges; and type 3 where the feasibility is maintained when an exchange is performed. The variables are then complemented ($x_j = 1 - x_j^{'}$) to improve the current solution. In the improvement phase, variable fixing and rounding of fractional solutions are performed based on solution improvement. The time complexity of this heuristics is $O(m+n)^4$ and was applied to some real world problems and problems with up to 200 constraints and 900 variables. The solutions to the various problems were on an average within 0.15% of the optimum.

Magazine & Oguz (1984) proposed an heuristic algorithm for MDKP by combining Lagrange multipliers method of (Everett, 1963) with the efficient gradient method of Senju &

Toyoda (1968). Initially all multipliers are set to 0 and all variables set to 1. Then, during each iteration, the constraint with largest ratio between the remaining and the total capacity is determined. The multiplier associated with this constraint is then made large enough such that one variable can become 0. This process is repeated until a feasible solution is found, after which a solution improvement process is applied. The time complexity of the proposed heuristics is $O(mn^2)$, and randomly generated problems with m up to 1000 and n up to 1000 were solved. For the randomly generated problems, the heuristics produced high quality results for large problems when compared with Senju & Toyoda (1968) and Kochenberger *et al.* (1974), where the later was unable to produce any results. For small and medium size problems, Kochenberger *et al.* (1974) produced high quality results in spite of being the slowest.

Gavish & Pirkul (1985) developed a branch & bound method that has been noted as the most efficient algorithm of such type by Kellerer *et al.* (2004). In this algorithm, lagrangean, surrogate and composite based relaxations were used to find a high quality solution. Using the solutions obtained, a sensitivity analysis is performed to reduce the number of variables by fixing some of the variables. Finally, a branch & bound procedure is applied to the reduced problem and solved for optimality. The authors report that this method is significantly faster and produced the best bounds overall when compared to the exact method by Shih (1979), heuristic based method by Loulou & Michaelides (1979), and commercial integer programming software Scionic/VM. A total of 240 randomly generated problems with up to 500 variables and 3 constraints, and up to 300 variables and 5 constraints were considered.

Pirkul (1987) proposed a surrogate constrained based heuristic procedure for MDKP. The method first solves m single constrained knapsack problems, and the constraint that produces the lowest objective value is marked as surrogate 1. From the values of variables in surrogate 1, the most violated constraint is determined and marked as surrogate 2. Iteratively the set of surrogate multipliers are obtained until no further changes occur in the objective value. The procedure then calculates the ratio between the objective function coef-

ficients and the coefficients of the surrogate resource constraint. The values of the variables are updated using variable fixing, which is based on the violations and the ratios obtained. The method obtained results within 1% of the optimal for 98% of the problems, with m up to 20 and n up to 200, and performed better when compared to Loulou & Michaelides (1979) and Balas & Martin (1980).

Lee & Guignard (1988) proposed a parametric based heuristic approach by combining the pivot and complement method of Balas & Martin (1980) and a modified efficient gradient method of Senju & Toyoda (1968). Problems with up $m = 5$ and $n = 20$ and $m = 6$ and $n = 600$ were solved within an average of 0.34% from optimum. Their heuristics produced better results when compared to Magazine & Oguz (1984), but is outperformed by Balas & Martin (1980).

Fréville & Plateau (1994) proposed a surrogate constraint based heuristic algorithm similar to Pirkul (1987), but with different schemes to fix variables and an additional oscillation assignment technique. This algorithm provides sharp lower and upper bounds on the optimal value and provides an efficient reduction scheme for the problem. Their procedure provided better results when compared to Pirkul (1987) in spite of consuming more computational time. Randomly generated problems with up to $m = 30$ and $n = 500$ were solved using this method with this heuristic algorithm whose time complexity of $O(mn^2)$.

Glover & Kochenberger (1996) proposed a tabu search method based on critical events to solve MDKP. The tabu procedure is based on short term and long term memory obtained by using a strategic oscillation between constructive and destructive phases to obtain intensification and diversification. The constructive phase moves from a feasible solution to an infeasible, and the destructive goes from infeasible to feasible. Surrogate constraints are formed based on the current solution, which guides whether the variable is selected or not in the two phases. Their method was tested on 57 standard test cases for which optimal solutions were obtained for all, and the authors also note that no other method has been able to provide optimal solutions for all the 57 problems. Additionally, 24 randomly generated

problems with m uo to 25 and n up to 500 were solved using this tabu search approach. For 23 of the 24 problems, generated solutions were superior to those given by branch & bound algorithm.

Chu & Beasley (1998) proposed a genetic algorithm based heuristics for the MDKP. In their algorithm, a repair operator based on greedy approach was used in addition to the cross-over and mutation operations, in order to satisfy the constraints. To obtain the pseudo-utility for each variable, they applied surrogate relaxation for which the weights were computed by solving the dual of the problem. By combining CPLEX to solve the dual, and their problem specific genetic algorithm, they solved small real-world problems with m = 2-30 and n = 6-105 to optimality, and showed superior performance when compared with other genetic algorithm based approach proposed by Thiel & Voss (1994), tabu based approach proposed by Løkketangen & Glover (1996), and simulated annealing based approach proposed by Drexl (1988). They also solved large problems with m = 5, 10, and 30, and n = 100, 250, and 500, and showed superiority over other heuristic based methods proposed by Magazine & Oguz (1984), Volgenant & Zoon (1990), and Pirkul (1987) in terms of computation time and quality of solutions. The time complexity for this algorithm is $O(mn)$.

Vasquez & Hao (2001) proposed a hybrid method that combines linear programming and tabu search to solve MDKP. Their method initially solves the relaxed MDKP using simplex with an additional constraint $\sum_{j=1}^{n} x_j = k$ to find the fractional solutions for attractive values of k. Then a tabu search is used to search around the fractional solutions to obtain high quality solutions. Their method was tested on several benchmark problems and outperformed all methods. Their method improved all the solutions found by Chu & Beasley (1998), and, in spite of high computational time (up to 3 days), their method found improved solutions to large problems with m up to 100 and n up to 2500 obtained from Glover & Kochenberger (1996).

Osorio *et al.* (2002) proposed an algorithm based on surrogate, cutting planes, and branch & bound for the MDKP problems. The algorithm first solves the relaxed problem to

obtain optional solution. Based on this solution, and by generating surrogate constraints, the algorithm produces several logical cuts (additional constraints) for the MDKP. Then a branch & bound method is applied to solve the MDKP with the added cuts. Computational results show better quality solutions obtained in less time when compared to the leading commercial software CPLEX.

Gabrel & Minoux (2002) proposed an exact separation scheme to identify the most violated extended cover inequalities for MDKP. Their method is based on finding the ratio between the left and right hand sides of the constraint where the maximum ratio provides the best extended cover. Using an iterative algorithm, they solved randomly generated problems with up to 180 variables and 60 constraints and showed much lower computational times compared to the Cplex 6.5 commercial software.

Vasquez & Vimont (2005) presented a hybrid method with variable fixing, which was based on Vasquez & Hao (2001). They presented comparisons of results between Chu & Beasley (1998), Vasquez & Hao (2001) and Vasquez & Vimont (2005) and showed improved results using their original approach but failed to show any improvements from their latest approach using variable fixing. Their method was able to improve many lower bounds for the bench mark problem sets, and the combination of fixing variable, linear relaxation, and tabu search improved the results from their previous work. Up to 33 hours of wall time was used during the execution of their latest hybrid tabu search method.

Fréville & Hanafi (2005) presented a review of different methods available to solve the MDKP. They concluded that the enhanced versions of tabu search methods provide the best near-optimal results to date for the most difficult problems in spite of their high computational time, and the exact methods with low computational times face difficulties when the number of constraints go higher.

Kaparis & Letchford (2008) proposed lifted cover inequalities based on Gabrel & Minoux (2002) to obtain better upper bounds for MDKP. To obtain a lifted cover, first a minimal cover is obtained, then a lifting coefficient is obtained for each variable, sequentially, by

fixing the variable to either 0 or 1 to create an effective cutting plane. The upper bounds obtained by this method dominates all the other available upper bounds for the MDKP problems in literature.

Balev *et al.* (2008) proposed a dynamic programming based procedure for the MDKP. Their method first solves the relaxed versions of the problem to compute a number of upper bounds by adjusting the variables with respect to constraint feasibility. A series of lower bound is also produced using dynamic programming and the two bounds are compared to fix certain variables and to reduce the problem size. Computational results show the superior performance over the leading commercial software CPLEX interns of solutions quality and the time required to solve.

Vimont *et al.* (2008) proposed a variable fixing heuristics that combines the hybrid method by Vasquez & Hao (2001), hybrid method with variable fixing by Vasquez & Vimont (2005), and an enumeration technique for the sub problems. Their method obtained the best lower bounds for the benchmark problem set but failed to obtain optimum values in most cases due to non-optimal variable fixing.

Puchinger *et al.* (2010) proposed a core concept for solving MDKP and produced high quality results at significantly shorter times. The method first solves the relaxed version of the MDKP, based on this a core concept is defined. A core concept is effectively reducing the problem size by variable fixing, and the core is solved using a meta heuristic called memetic algorithm. The memetic algorithm is based on Chu & Beasley (1998) but includes probabilistic concepts and local improvement concepts. The authors note that the best results for the benchmark problems are provided by Vasquez & Hao (2001) and Vasquez & Vimont (2005).

Boussier *et al.* (2010) presented a multi-level search strategy for MDKP that combines resolution search, branch & bound, and variable fixing. Initially a relaxed problem is solved using simplex, to which an iterative resolution search (inverse of branch & bound method) and branch and bound methods are applied by fixing some variables. Their computational

results show improvements to many solutions obtained by Chu & Beasley (1998) and Vasquez & Vimont (2005), but the time required to obtain the solutions may reach up to 148 hours in some cases. The authors duly note the huge time requirement and state that the algorithm is not really suited for the instances solved.

Angelelli *et al.* (2010) proposed a heuristic procedure based on kernel search for the MDKP. An initial set of promising candidates are obtained by solving the relaxed problem, and this forms the kernel. Then, a sequence of subproblems which are restricted to the present kernel (variables in kernel) is then solved with a few more added variables. The kernel is updated using the better solutions found by an ILP solver, and, in the end of the iterative procedure, the best solution for the MDKP is represented by the kernel. Benchmark problems of size $n = 250$ and $m = 30$, and $n = 500$ and $m = 30$ were solved by setting a time limit of up to 2 hours. The solutions to few problems in the first set ($n = 250$) were improved when compared to Boussier *et al.* (2010).

Mansini & Speranza (2012) proposed an exact algorithm for MDKP based on variable fixing and branch & bound method. Their algorithm solves subproblems (considering fewer variables) by iteratively fixing variables, and then branch & bound method is applied for the sub problem. The initial solutions for their sub problems are obtained either by greedy method or from the kernel search method (Angelelli *et al.*, 2010). The algorithm was tested on various benchmark cases, was shown to be better than the most recent branch & bound method (Vimont *et al.*, 2008), and was able to find new optimal solutions in 5 hours for problems size $m = 30$ and $n = 250$.

3.4 Application of XQX Model for 0-1 MDKP

In this section, we present the application of our XQX model for 0-1 MDKP using a basic heuristics that we developed. The purpose of this heuristic is to show that the XQX model and the solution procedures presented in the previous chapter can be used to solve benchmark problems irrespective of its size. The MDKP problem considered in this chapter

is a binary IP problem with inequality constraints, and we presented our XQX model for such a problem type in section 2.6.2. The variable update procedure for the binary variables was discussed in section 2.8.3, and the update rules for the slack variables were presented in section 2.8.2 for the same problem type. The heuristic incorporates all these methods, and Algorithm 1 shows the overall procedure.

Note that the MDKP problem is a *max* problem type and the XQX model presented in the previous chapter used a *min* problem type. To be consistent with our XQX model, we have converted the *max* problem type to *min* using the following convention:

$$\min\ f(x) = -\max\ (-f(x)). \tag{3.3}$$

The Algorithm was coded using Python programming language and uses a combination of serial and parallel processing during execution. The parallel processing was obtained using PyOpenCL (Klöckner *et al.*, 2012) for the computation of $\frac{\partial f}{\partial x}$ and s^*. PyOpenCL is an open source Python wrapper for OpenCL (Open Computing Language), which allows Python based codes to take advantage of heterogeneous computing resources. OpenCL is an industry standard for heterogeneous computing framework, which is a combination of CPU, GPU, and other processors. Codes with OpenCL modules can make use of the multi-core CPU, or multi-core GPU, or their combination to obtain parallel execution. In our parallel implementation, we have developed a very basic version that does not involve advanced techniques to obtain higher speedups; also the heuristics procedure presented here is very basic. Our goal in this thesis is not to create a heuristic that solves problems to optimality, but to show the application of developed XQX model and the solution procedures.

In Algorithm 1, lines 1 through 4 represent the initialization phase. In line 1, we first read the problem instance from a file; the number of variables, number of constraints, objective function, and the constrains are then stored in appropriate variables. In line 2, we compute the XQX matrix for the problem instance based on 2.68. In line 3, all the variables are

initialized to 0, and in line 4, the initial $\frac{\partial f}{\partial x}$ are calculated using parallel processing.

Algorithm 1: Heuristic for Binary IP

```
1  READ problem instance
2  COMPUTE Q matrix
3  x_i ← 0, s_i ← 0, s_i^* ← 0
4  COMPUTE ∂f/∂x_i
5  while maximum time not reached do
6      if all constraints satisfied then
7          Best x_i ← 1 if x_i == 0
8          COMPUTE s_i^*, constraint violation, s_i, ∂f/∂x_i
9      else
10         while constraints violated do
11             Worst x_i ← 0
12             COMPUTE s_i^*, constraint violation, s_i, ∂f/∂x_i
13     if all constraints satisfied then
14         COMPUTE objective function value
15         if better solution found then
16             UPDATE previous best x and s^*
17             UPDATE current best x and s^*
18     else
19         if random variable == true then
20             Randomly select x_i ← 0 if x_i == 1, or x_i ← 1 if x_i == 0
21             COMPUTE s_i^*, constraint violation, s_i, ∂f/∂x_i
22             FORCE all constraints satisfied
23     if best solution not updated in p iterations then
24         current x, s ← previous best x, s^* OR current x, s ← best x, s^*
25         Randomly select x_i ← 1 if x_i == 0 from current solution
26         COMPUTE s_i^*, constraint violation, s_i, ∂f/∂x_i
27 return best solution
```

The Algorithm then enters the main loop of execution from lines 5 through 26 where the variable update procedures are implemented to solve the problem. We will present a sample code execution description after we present the details of this loop. In line 6, if all the constraints are satisfied, the code then sorts the computed $\frac{\partial f}{\partial x}$, and the best x_i (lowest $\frac{\partial f}{\partial x_i}$) which is 0, is changed to 1. A status is also updated to the selected variable so that

it is not changed back to 0 in the very next iteration, and before changing the variable's value from 0 to 1, the status is checked to make sure that it can be changed in that specific iteration. In line 13, if one or more constraint is violated, the code then sorts the computed $\frac{\partial f}{\partial x}$, but this time the worst x_i (highest $\frac{\partial f}{\partial x_i}$), which has a value of 1, is changed to 0. Similar status checks and updates are performed here. Under line 13, the procedure changes the variable value to 0 until all the constraints are satisfied, which is obtained by calculating the s^*. Note that as soon as a variable x_i is updated, the new values of s^*, constant violations, s_i, and $\frac{\partial f}{\partial x_i}$ are calculated, and this procedure is adopted throughout the code execution. In the variable update procedure for binary variables from 2.8.3, we showed Table 2.5 which used positive or negative values of $\frac{\partial f}{\partial x_i}$ to determine its corresponding update. The sorting procedure implemented also works in the same way where the variable with lowest partial derivative is assigned 1 and the one with highest is assigned 0.

After the updating procedures under line 6 and line 13 are performed, the objective function value is calculated in line 14 if all the constraints are satisfied. If a better solution is found, the array containing the best solution and the previous best solution are updated. For example, if a best solution is found in iteration k, then previous best solution array is updated to the best solution found in iteration $k-i$, and the best solution array is updated to the current best solution obtained in iteration k. Note that the value of s_i is updated to s_i^* in the best solution since s^* represents the actual constraint violation, as shown in 2.93. If the constraints are still violated in line 18, based on the random variable calculated, the current solution set is randomly updated based on the random variable obtained in line 19. If the random variable is true in line 19, then another random variable is calculated to decide between random addition or random deletion. In random addition, one variable x_i from the set with a value of 0 and with an acceptable status is selected randomly and its value is changed to 1, along with status update. In random deletion, a similar step is performed where a variable x_i with value 1 is changed to 0. At the end of this operation, we

update a variable that forces the value of constraint violation to 0 irrespective of the actual calculated constraint violation.

If the best solution has not been updated after certain number of iterations, the current solution set is updated to either the best solution set or the previous best solution set under line 23. If previous best was chosen in iteration j, then best solution set is chosen the next time and so on. After updating the current solution set, the random addition procedure described earlier is performed, which is again based on a random $true$ or $false$ value, and then the new values of s^*, constant violations, s_i, and $\frac{\partial f}{\partial x_i}$ are calculated. If random variable is $true$, then the random addition is performed, or else the solution set is not changed.

3.4.1 Example Code Execution

To visualize a sample code execution, assume that the procedure is beginning its 100th iteration, and also assume best and previous best solution exist where the best solution was found in 82^{nd} iteration. At 100^{th} iteration, in line 6, assume that all the constraints are satisfied. The code then adds one more variable by determining the best variable to add and then the values for slack variables and partial derivatives are calculated in line 8. After adding 1 more variable, assume that one or more constraint has been violated and the code enters in to line 18. Here, let us also assume that a random addition takes place, which makes the constraint violation even higher; also the constraint violation variable is forced to 0 at the end. At this stage too many variables have a value of 1, and at the end of 100^{th} iteration the procedure is forced to assume that all constraints are satisfied.

Now, the procedure enters 101^{st} iteration with all constraints satisfied (since it was forced earlier). The procedure again adds a best variable under line 6, and then the constraints are further violated. The procedure will then enter line 18 since the constraints are violated. Now, assume that the random variable calculated in line 19 is false, and due to this the procedure does not update the current solution (infeasible solution) set and it simply goes to the next iteration.

In the next iteration, 102^{nd}, using the results from previously calculated constraint violations, the worst variables will be removed until all the constraints have been satisfied; as shown in line 10. At the end of this procedure, the code will enter line 13 since all constraints are satisfied. Now assume that after calculating the objective function value a better solution was found. The code will then update the previous best solution to the best solution found in 82^{nd} iteration, and the best solution is updated to the current best solution found in 102^{nd} iteration. If a new solution was not found in 102^{nd} iteration, the code then reaches line 23. Assume that the value of p is 20 in line 23, and since the last best solution was found in 82^{nd} iteration, and since we are currently in 102^{nd} iteration, it is time to change the current solution set with previous best or the best solution with or without random addition. The procedure then continues to execute as shown in the pseudocode until the maximum time has been reached and then returns the best solution found.

In neural networks, the output of one variable depends upon the output of all other variables in system. Similarly here, the output of one variable depends upon all the other binary variables and the slack variables, and this can be clearly seen from 2.101a and 2.111. In our heuristics, we have tried to mimic the same behavior by using the update rules for the binary and slack variables, which in turn changes the contribution of one variable based on the value of others. We have also used randomness to escape local optima.

3.4.2 Time Complexity of XQX based Heuristic for Binary IP

In this section, we provide the time complexity associated with the different operations in a iteration for the heuristic presented above. The computation of $\frac{\partial f}{\partial x_i}$ for each unknown variable (x_i) involves all the unknowns except x_i, slack variables, and a constant number. Hence, the time complexity of this operation is $\mathcal{O}(n(n-1+m))$. The calculation of s_i^* also involves $\mathcal{O}(n(n+m))$ operations, but it can be reduced to $\mathcal{O}(mn)$ operations based on 2.30. To find the best or worst x_i, we have used the quick sort algorithm, and its time complexity is $\mathcal{O}(n(logn))$. Updating s_i involves a total of m operations, and checking the

constraint violation involves m operations in the worst case. Hence, we have $\mathcal{O}(m)$ for for updating $s_i$and $\mathcal{O}(m)$ for checking constraint violation in the worst case. Calculating the new objective function value has a complexity of $\mathcal{O}(n)$ since the slack variables do not play a role in this calculation based on Proposition 5 (2.33). The worst time complexity in this heuristics is the computation of $\frac{\partial f}{\partial x_i}$, but this can be reduced based on the Proposition 7 (2.43). Also note that the while loop in line 10 will not iterate m times. The worst case is m times, but this definitely does not happen. In the worst case scenario, with m operations in line 10, the worst time complexity would be $\mathcal{O}(mn(n+m))$. For other operations in the heuristics the time complexities are much less than the worst case reported.

3.5 Solutions for Benchmark Problems

In this section, we present the results from solving some of the benchmark problems available for 0-1 MDKP problem, and problems were obtained from `http://www.cs.nott.ac.uk/~jqd/mkp/index.html`. The heuristic procedure explained in the previous section was used to obtain solutions for these benchmark problems. The penalty value P was set to 100, and this value was chosen arbitrarily.

The solutions obtained from our heuristics are compared to the best solutions obtained by the most recent exact algorithm proposed by Mansini & Speranza (2012), the large instances are compared against the hybrid method with variable fixing and tabu search developed by Vasquez & Vimont (2005), and the largest with Vasquez & Hao (2001) . It is also to be noted that the best methods available in literature to solve 0-1 MDKP problems, both exact and heuristics based, always include a linear relaxation ($0 \leq x \leq 1$ instead of $x \in \{0, 1\}$) in their procedures. Also, Kellerer *et al.* (2004) states that the d-dimensional knapsack problems are particularly difficult instances of integer programming because the constraint matrix is usually dense, and, due to this, it has been a favorite playground for experiments with heuristic procedures such as tabu search and genetic algorithm to develop advanced techniques.

The size of the benchmark problem ranges from $m = 5$ to $m = 500$ and $n = 100$ to $n = 2500$. There are a total of 281 problems from which we are presenting the results for 92 instances. Each instance was executed on an Intel 2.93 GHz machine with 8 cores, and the CPU occupancy during the execution of an instance was about 20% for $n \leq 500$ and about 40% for $n \geq 1000$. All the instances were solved twice each for a period of 1800 seconds. We report the best solution obtained from those two runs for each reported instance in the following tables. In the tables given below, column 1 gives the best known solution for the instance, column 2 shows the best solution obtained by our method, column 3 shows the % gap between the best known and our solution, column 4 shows the time required to obtain our solution, column 5 gives the best solution from CORAL (Mansini & Speranza, 2012), or from HFTABU (Vasquez & Vimont, 2005), or from HTABU (Vasquez & Hao, 2001), column 6 gives their % gap, and column 7 gives their time required.

The results shown in tables 3.1 to 3.9 are solutions for the problem sets created by Chu & Beasley (1998), and Table 3.10 shows the solutions for the problem sets created by Glover & Kochenberger (1996). The Chu & Beasley (1998) source problem sets were constructed using the procedure suggested by Fréville & Plateau (1994). For each problem set, the w_{ij} are integer numbers uniformly drawn in $[0, 1000]$, and for each combination (n, m) the constraint capacities are computed as $c_i = \alpha \sum_{j=1}^{n} w_{ij}$, where α is the tightness factor. The tightness factors used in the problem sets are 0.25, 0.5 and 0.75 to create these problem sets. In our tables 3.1 to 3.9 the first three rows correspond to problems with tightness factor 0.25, the next three with 0.5, and the last three with 0.75. We could not find the problem construction procedure for Glover & Kochenberger (1996) sets in our literature search.

No.	Best Known Z	XQX	% Gap	Time(sec)	CORAL	% Gap	Time(sec)
1	23534	23468	0.28	1791	23534	0.0	4.59
2	23991	23922	0.288	1527	23991	0.0	1.95
3	24216	24161	0.227	89	24216	0.0	1.22
4	42545	42339	0.484	1406	42545	0.0	0.81
5	45090	44953	0.304	794	45090	0.0	3.14
6	42218	42005	0.505	850	42218	0.0	1.92
7	61091	61091	**0.0**	1051	61091	0.0	0.98
8	58959	58921	0.064	908	58959	0.0	2.06
9	61520	61459	0.099	739	61520	0.0	1.58

Table 3.1: Results for: $m = 5$, $n = 100$ (Chu & Beasley Instances)

No.	Best Known Z	XQX	% Gap	Time(sec)	CORAL	% Gap	Time(sec)
1	23064	22904	0.694	680	23064	0.0	140
2	22635	22394	1.065	1399	22635	0.0	11.23
3	22511	22359	0.675	1089	22511	0.0	18.39
4	45624	45311	0.686	1201	45624	0.0	55
5	43574	43342	0.532	1113	43574	0.0	126
6	42970	42692	0.647	1739	42970	0.0	43
7	57375	57137	0.415	1074	57375	0.0	5
8	56377	56197	0.319	1421	56377	0.0	60
9	60205	59947	0.429	558	60205	0.0	18

Table 3.2: Results for: $m = 10$, $n = 100$ (Chu & Beasley Instances)

No.	Best Known Z	XQX	% Gap	Time(sec)	CORAL	% Gap	Time(sec)
1	21946	21576	1.686	1612	21946	0.0	1925
2	21464	21044	1.957	730	21464	0.0	2456
3	21397	21006	1.827	1099	21397	0.0	2093
4	41041	40659	0.931	499	41041	0.0	1985
5	40889	40450	1.074	1717	40889	0.0	1952
6	41700	41226	1.137	1231	41700	0.0	1937
7	57494	57494	**0.0**	369	57494	0.0	1207
8	60011	59594	0.695	1168	60011	0.0	1899
9	58132	57938	0.334	823	58132	0.0	1736

Table 3.3: Results for: $m = 30$, $n = 100$ (Chu & Beasley Instances)

No.	Best Known Z	XQX	% Gap	Time(sec)	CORAL	% Gap	Time(sec)
1	59463	58903	0.942	1495	59463	0.0	129
2	61472	60905	0.922	1511	61472	0.0	137
3	61885	61276	0.984	1566	61885	0.0	76
4	109109	108059	0.962	770	109109	0.0	111
5	109841	109237	0.55	1607	109841	0.0	99
6	109383	108226	1.058	1502	109383	0.0	110
7	149665	148845	0.548	1595	149665	0.0	116
8	149334	148400	0.625	1258	149334	0.0	155
9	152130	151676	0.298	1416	152130	0.0	88

Table 3.4: Results for: $m = 5$, $n = 250$ (Chu & Beasley Instances)

No.	Best Known Z	XQX	% Gap	Time(sec)	CORAL	% Gap	Time(sec)
1	58781	57656	1.914	1657	58781	0.0	3870
2	61000	59933	1.749	1643	61000	0.0	4364
3	59387	58355	1.738	1576	59387	0.0	3612
4	110913	109062	1.669	1730	110913	0.0	4907
5	108932	107250	1.544	1549	108932	0.0	4085
6	110845	109094	1.58	1022	110845	0.0	7193
7	153578	152480	0.715	1682	153578	0.0	4230
8	149160	148596	0.378	1327	149160	0.0	3776
9	149704	148808	0.599	1258	149704	0.0	3649

Table 3.5: Results for: $m = 10$, $n = 250$ (Chu & Beasley Instances)

No.	Best Known Z	XQX	% Gap	Time(sec)	CORAL	% Gap	Time(sec)
1	56357	53890	4.377	1785	56357	0.0	18000
2	56457	54463	3.532	1316	56457	0.0	18000
3	56447	54378	3.665	1407	56447	0.0	18000
4	107770	105484	2.121	1763	107770	0.0	18000
5	106442	103976	2.317	1570	106442	0.0	18000
6	104032	101381	2.548	1514	104032	0.0	18000
7	149958	147692	1.511	1680	149958	0.0	18000
8	148574	146556	1.358	1711	148574	0.0	18000
9	149570	147401	1.45	1791	149570	0.0	18000

Table 3.6: Results for: $m = 30$, $n = 250$ (Chu & Beasley Instances)

No.	Best Known Z	XQX	% Gap	Time(sec)	CORAL	% Gap	Time(sec)
1	120148	118190	1.63	1699	120148	0.0	839
2	117879	115793	1.77	696	117879	0.0	190
3	121586	119288	1.89	1713	121586	0.0	766
4	218428	216226	1.008	1233	218428	0.0	655
5	221202	218661	1.149	1777	221202	0.0	146
6	218215	215746	1.131	1240	218215	0.0	288
7	295828	293069	0.933	1542	295828	0.0	71
8	308086	305788	0.746	1129	308086	0.0	393
9	300342	297549	0.93	1717	300342	0.0	327

Table 3.7: Results for: $m = 5$, $n = 500$ (Chu & Beasley Instances)

No.	Best Known Z	XQX	% Gap	Time(sec)	CORAL	% Gap	Time(sec)
1	119249	114748	3.774	1567	119232	0.00014	18000
2	118829	114184	3.909	1601	118825	0.00003	18000
3	116530	113304	2.768	1515	116509	0.00018	18000
4	217377	211947	2.498	1051	217377	0.0	18000
5	213873	207514	2.973	1741	213859	0.00006	18000
6	220899	214299	2.988	1353	220899	0.0	18000
7	304374	300470	1.283	1647	304374	0.0	18000
8	296478	292505	1.34	1054	296478	0.0	18000
9	301359	296330	1.669	1707	301359	0.0	18000

Table 3.8: Results for: $m = 10$, $n = 500$ (Chu & Beasley Instances)

No.	Best Known Z	XQX	% Gap	Time(sec)	HFTABU	% Gap	Time(sec)
1	115741	109502	5.39	1665	115741	0.0	616
2	114181	108707	4.794	1798	114181	0.0	110873
3	117116	111123	5.117	1684	117116	0.0	121248
4	218104	210620	3.431	1731	218104	0.0	96952
5	214648	206472	3.809	1787	214648	0.0	167224
6	215890	207742	3.774	1600	215890	0.0	117102
7	301675	295611	2.01	1621	301675	0.0	143430
8	300055	294294	1.92	1516	300055	0.0	218994
9	302032	296525	1.823	1617	302032	0.0	156377

Table 3.9: Results for: $m = 30$, $n = 500$ (Chu & Beasley Instances)

No.	n, m	Best Known Z	XQX	% Gap	Time(sec)	HTABU	% Gap	Time(sec)
1	100, 15	3766	3730	1.218	1735	3766	0.0	259200
2	100, 25	3958	3922	0.91	453	3958	0.0	259200
3	150, 25	5656	5593	1.114	1493	5656	0.0	259200
4	150, 50	5767	5680	1.509	1331	5767	0.0	259200
5	200, 25	7560	7463	1.283	1721	7560	0.0	259200
6	200, 50	7677	7572	1.368	1768	7677	0.0	259200
7	500, 25	19220	18964	1.332	1779	19220	0.0	259200
8	500, 50	18806	18458	1.85	1620	18806	0.0	259200
9	1500, 25	58087	56838	2.15	1772	58087	0.0	259200
10	1500, 50	57295	56075	2.129	1033	57295	0.0	259200
11	2500, 100	95237	93607	1.712	1374	95237	0.0	259200

Table 3.10: Results for: Other large instances (Glover & Kochenberger Instances)

3.5.1 Results Discussion

In tables 3.1 to 3.8 we compared our results with the results obtained by Mansini & Speranza (2012) (CORAL). It is clear that CORAL performs very well for small problem sets by computing optimal results in short time, since it is a complex method which combines different features. But as the problem size gets larger, CORAL does use more time and faces difficulty. It is also evident from Mansini & Speranza (2012) that the commercial solver CPLEX gives similar performance when compared with CORAL for small problem sets, but as the problem sets gets larger, ($m \geq 10, n \geq 500$) CPLEX faces difficulty. To be specific, for problems with ($m \geq 10, n \geq 250$), with infeasible starting solutions, CPLEX provided "infeasible" solutions even after 5 hours or even 10 hours of computational time. Comparing our results with CORAL, we were able to match the optimal results for two cases, and in one case, XQX out-performed CORAL.

It is also evident from Mansini & Speranza (2012) that they fail to show any results for problem sets with ($m >= 30$, $n >= 500$), and, due to this, we have compared our solutions for the large problem sets with the results obtained by Vasquez & Vimont (2005) (HFTABU) and Vasquez & Hao (2001) (HTABU) in tables 3.9 and 3.10. In fact, Mansini & Speranza (2012) clearly states that CORAL cannot handle large problems.

For the large problems sets, the best results were provided by Vasquez & Vimont (2005) (HFTABU) and Vasquez & Hao (2001) (HTABU). It is to be noted that HFTABU results shown in Table 3.9 took more than 1 day of computational time, and in certain cases, up to 2.53 days to obtain the optimal value. We are able to get within 2% of the optimal solution in 0.5 hours for a few cases. Similar conclusion can be drawn from Table 3.10 where the results are obtained after 3 days of computation, and we are able to get within 2% of the optimal solution value in 0.5 hours of computation time for most of the cases.

The conclusions we would like to draw from the results are as follows. The proposed XQX model was able to handle small, medium and very large problem sizes. It was able to produce reasonable solution quality, especially for large problem sets, in a very reasonable

time. This XQX model is new and it does not exist currently in literature, and if our XQX model is included in a sophisticated heuristic model, it can provide better solutions for any problem size.

3.6 Summary

In this chapter, we defined the 0-1 MDKP based on existing literature, presented real life applications of 0-1 MDKP, and described several existing methods from literature that can solve small and large MDKP. We applied our XQX model for 0-1 MDKP using our simple heuristics and demonstrated that our model can be applied to any problem size. In the next chapter, we present the general multidimensional knapsack problem, describe its mathematical formulation for this problem, and then present a detailed literature survey about the existing methods used to solve such a problem. The general MDKP is a problem with integer variables and inequality constraints, and our XQX model developed in section 2.6.2 for general IP and the solution procedures presented in sections 2.8.1 and 2.8.2 will be applied to solve benchmark problems using our heuristic procedure. The application of our general XQX model to general IP problem (general MDKP) will be the fourth and final objective of this thesis.

CHAPTER 4

GENERAL MULTIDIMENSIONAL KNAPSACK PROBLEM

In this chapter, we apply our general XQX model from section 2.5 to general multidimensional knapsack problem (GMDKP). The application of our general XQX model to GDMKP is the fourth and final objective of this thesis. The GMDKP is a strong NP-hard problem and is similar to 0-1 MDKP from the previous chapter, but all the variables in GMDKP are integers.

4.1 Definition

The GMDKP problem is a multiple resource constraint integer type problem and is defined by the following formulation (Kochenberger *et al.*, 1974):

$$\begin{aligned} \text{maximise} \quad & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} r_{ij} x_j \leq b_i, \ i = 1, \ldots, m, \\ & x_j \in \mathbb{Z}_+, \ j = 1, \ldots, n, \end{aligned} \tag{4.1}$$

where p_j, r_{ij}, and b_i are assumed to be positive integer values. Given n items with each having profits p_j and m resources with capacities b_j, the goal is to select a subset of items with maximum total profit when each item j consumes r_{ij} from each of the i resource.

4.2 Literature Review

In our extensive review of literature we found only three published papers which deal with GMDKP, and the solution approaches for GMDKP problems are scarce (Akçay *et al.*, 2007).

Kochenberger *et al.* (1974) proposed a heuristic for general integer programming problem based on the efficient gradient method by Senju & Toyoda (1968). This heuristic starts with all variables set to 0, calculates the efficient gradient for all variables, and the variable with highest gradient is updated to a new value (incremented by 1) if the update does not violate any constraints. If one or more constraints are violated, the next best efficient gradient is chosen. This heuristics was used to solve 26 problems with up to 50 variables and 12 constraints, or 15 variables and 50 constraints, and found optimal solutions to 86% of the problems.

Pirkul & Narasimhan (1986) proposed a branch & bound based algorithm to solve general multidimensional knapsack problem. This algorithm combines surrogate relaxation, variable fixing and sensitivity analysis with their branch & bound procedure to solve randomly generated problem instances with up to 5 constraints and 200 variables. The surrogate relaxation is used to obtain bounds, then variables were fixed according to the bounds obtained, after which sensitivity analysis was performed to improve the bounds. When compared with CPLEX, their algorithm was able to obtain optimal solutions for all problems.

Akçay *et al.* (2007) proposed a greedy-like algorithm called PECH which was shown to be superior when compared with the other two methods, (Kochenberger *et al.*, 1974) & (Pirkul & Narasimhan, 1986). This algorithm considers effective capacity for each item, which is defined as the maximum copies of item j that can be accepted for remaining available capacity for one knapsack. PECH starts with all variables equal to 0 and updates their values based on the reward obtained with respect to effective capacity. The variables are incremented by one or more than one unit at any given update. Randomly generated problem sets with number of variables up to 400 and number of constraints up to 200 were

created to test their method. Note that, in the version of GDMKP generated and solved by PECH, the variables are bounded by a positive integer.

4.3 Application of XQX Model for GMDKP

In this section, we present the application of our XQX model for GMDKP using another basic heuristics that we developed, and this heuristics is similar to the one presented for solving 0-1 MDKP problems. The purpose of this heuristics is to show that the XQX model and the solution procedures presented earlier can be used to solve benchmark problems irrespective of its size. The GMDKP problem is an IP problem with inequality constraints, and we presented our XQX model for such a problem type in section 2.6.2, and the variable update rules were presented in section 2.8.1 and section 2.8.2. The heuristics incorporates all these methods, and Algorithm 2 shows the overall procedure. Note that the GMDKP problem is a *max* problem type and it was converted to a *min* problem type using 3.3.

This Algorithm was coded using Python programming language and uses a combination of serial and parallel processing during execution (similar to the 0-1 MDKP heuristics presented in the previous chapter). The parallel processing was obtained using PyOpenCL for the computation of $\frac{\partial f}{\partial x}$, s^*, and the minimum remaining capacity.

In Algorithm 2, lines 1 through 3 represent the initialization phase. In line 1, we read the problem instance and then compute the XQX matrix for the problem. In line 2, we initialize the unknowns (x) to 0, or a random number between 0 and 1, or a random number between 1 and 5. The initialization of x based on these three cases will be addressed appropriately when the benchmark results are presented. In line 3, we compute the constraint violation and s.

The Algorithm then enters the effective capacity based variable updates from lines 4 through 10. The method PECH suggested by Akçay *et al.* (2007) is used to update the variables, and this method works as follows. First, the effective capacity of all the variables are calculated. In this calculation, for each variable, we compute its remaining capacity

in each constraint and then take the minimum capacity for each variable. The minimum remaining capacity for each variable is then multiplied with its corresponding profit value, and this gives the effective capacity for each variable.

The greedy update in line 5 chooses the variable with highest effective capacity (variable that gives the highest profit based on its minimum remaining capacity) and then updates this variable to $x_i + max(1, int(0.05 * EC_i))$. Note that the 5% was suggested as the best update by Akçay *et al.* (2007). The other values of 20% and 10% in other parts of the Algorithm were chosen arbitrarily. In line 6, the remaining capacity in each constraint (or slack variables s) is updated along with the constraint violation and objective function value. In lines 7 and 8, the best and the previous best solutions are updated if a better solution is found. Note that the concept of best and previous best solution was described in the previous chapter. In line 10, if the solution fails to improve, the Algorithm terminates PECH based updates and enters the main loop in line 12.

In the main loop represented in lines 12 through 46, the XQX based variable update method and the PECH based updates are integrated to improve all the solutions found in lines 4 through 10. In lines 13 to 18, if all the constraints are satisfied, the best greedy improving variable based on PECH is updated, after which the constraint violation, s^*, s, and $\frac{\partial f}{\partial x}$ are calculated in line 15. In line 16, based on a random variable generated between 1 and 2, the best improving x based on the $\frac{\partial f}{\partial x}$ is selected, and a random integer number generated between 1 and 25 is added to its current value. Note that, instead of updating the variable to x^*, we are updating it randomly. The constraint violation, s^*, s, and $\frac{\partial f}{\partial x}$ are then updated in line 18. A status is also updated so that any variable that is increased in value is not increased again for the next few iterations.

In lines 19 to 21, if one or more constraints are violated, the worst improving $\frac{\partial f}{\partial x}$ is chosen, and the corresponding value of the variable is updated to x^*. This x^* calculation is based on the core methods described earlier in section 2.8.1. The Algorithm then computes the objective function value in line 22, and the best and previous best solutions are updated

Algorithm 2: Heuristic for General IP

READ problem instance, COMPUTE Q matrix
$x_i \leftarrow 0$ or $x_i \leftarrow U[0,1]$, or $x_i \leftarrow U[1,5]$
COMPUTE s_i^*, constraint violation
while *all constraints satisfied* **do**
    COMPUTE effective capacities (EC), Greedy $x_i \leftarrow x_i + max(1, int(0.05 * EC_i))$
    COMPUTE s_i^*, constraint violation, objective function value
    if *better solution found* **then**
        UPDATE current & previous best x and s^*
    if *solution not improving* **then**
        break
COMPUTE $\frac{\partial f}{\partial x_i}$
while *maximum time not reached* **do**
    if *all constraints satisfied* **then**
        COMPUTE effective capacities (EC), Greedy $x_i \leftarrow x_i + max(1, int(0.2 * EC_i))$
        COMPUTE s_i^*, constraint violation, s_i, $\frac{\partial f}{\partial x_i}$
        for $i = 1$ *to U[1,2]* **do**
            XQX best $x_i \leftarrow x_i$+ U[1,25]
            COMPUTE s_i^*, constraint violation, s_i, $\frac{\partial f}{\partial x_i}$
    while *constraint(s) violated* **do**
        XQX Worst $x_i \leftarrow x_i^*$
        COMPUTE s_i^*, constraint violation, s_i, $\frac{\partial f}{\partial x_i}$
    COMPUTE objective function value
    if *better solution found* **then**
        UPDATE current & previous best x and s^*
    for $i = 1$ *to U[1,2]* **do**
        XQX Worst $x_i \leftarrow max(0, x_i - 1)$
        COMPUTE s_i^*, constraint violation, s_i, $\frac{\partial f}{\partial x_i}$
    for $i = 1$ *to U[1,2]* **do**
        COMPUTE EC, Greedy $x_i \leftarrow x_i + max(1, int(0.2 * EC_i))$
        COMPUTE s_i^*, constraint violation, s_i, $\frac{\partial f}{\partial x_i}$
    if *all constraints satisfied* **then**
        COMPUTE objective function value
        if *better solution found* **then**
            UPDATE current & previous best x and s^*
        if *random variable == true* **then**
            Randomly select XQX best or worst $x_i \leftarrow max(0, x_i^{'})$
            COMPUTE s_i^*, constraint violation, s_i, $\frac{\partial f}{\partial x_i}$
    if *best solution not updated in p iterations* **then**
        current $x, s \leftarrow$ previous best x, s^* OR current $x, s \leftarrow$ best x, s^*
        for $i = 0$ *to U[0,2]* **do**
            COMPUTE EC
            Randomly select between highest EC_i and lowest EC_i
            Highest EC_i $x_i \leftarrow max(0, x_i - 1)$ or Lowest EC_i $x_i \leftarrow x_i + int(0.1 * x_i)$
            COMPUTE s_i^*, constraint violation, s_i, $\frac{\partial f}{\partial x_i}$
        FORCE all constraints satisfied
return best solution

in line 24 if a better solution is found. The status which was set when increasing the value of a variable is ignored when the value is decreased to x^*, and a new status is set after decreasing the value.

In lines 25 through 27, the worst improving $\frac{\partial f}{\partial x}$ is chosen, and the corresponding value of the variable is updated to $max(0, x_i - 1)$, after which the constraint violation, s^*, s, and $\frac{\partial f}{\partial x}$ are updated. The random number generated in line 25 determines the number of updates performed. In lines 28 through 30, we compute the effective capacity (based on PECH) and update the best improving variable. The constraint violation, s^*, s, and $\frac{\partial f}{\partial x}$ are updated after performing PECH based updates. Again, the random number generated in line 28 determines the number of updates performed. Note that the status set previously for any variable is respected to allow a simple diversification in line 26 and 29. In lines 31 through 34, the current and the previous best solutions are updated if an updated solution is found after calculating the objective function value. If not, based on a random variable generated, the best improving or the worst improving x_i based on $\frac{\partial f}{\partial x}$ is randomly selected and updated to $max(0, x_i^{'})$. The value of $x_i^{'}$ is computed using $2x_i^{\star} - x_i$ from section 2.8.1.

In lines 38 through 45, if the solution has not improved in p iterations, the current best or previous best solution is modified based on the random number generated in line 40. In line 42, we decide randomly between the highest remaining capacity or the lowest remaining capacity. Then in line 43, the variable with the highest remaining capacity is decremented by 1 or the variable with the lowest remaining capacity is incremented by 10%. In line 44, the constraint violation, s^*, s, and $\frac{\partial f}{\partial x}$ are updated, and the number of modifications performed is based on the random number generated in line 40. At the end of these operations, irrespective of the constraint violations, the Algorithm is forced to assume that all constraints have ben satisfied in line 45.

Finally, in line 46, the best solution found is returned after the time limit has been reached. In the next section, we present a sample code execution of this Algorithm, which will help to see how the XQX and PECH methods work together to find solutions.

4.3.1 Example Code Execution

To visualize a sample code execution, assume that the procedure is just starting and the variables are initialized in line 2. The code then calculates the remaining capacities under each constraint (slack variables) and computes the constraint violation based on the selected values for x. Now, assume that all the constrains are satisfied after initialization, and we now enter the effective capacity based variable update in line 4. Here, the code finds the effective capacity of each variable in each iteration and adds one variable at a time. After a certain number of iterations (within a few seconds), the PECH based update will fail to improve the solution since PECH does not have any repair mechanisms to lessen the value of a variable. Also, PECH fails to improve when x_i is updated to $x_i + max(1, int(0.05 * EC_i))$, where sub-optimal quantities are added, which in-turn exhausts the available knapsack capacities.

Now, assume that the code has finished 100 iterations by the time it breaks out of the PECH based updates, and then enters the main loop in line 12 after computing the values of $\frac{\partial f}{\partial x}$, s^*, s in the previous step. Also assume that the current and previous best solutions were found during the PECH updates. Since all the constraints are satisfied at this time, the code tries to increment the best variable to $x_i + max(1, int(0.2 * EC_i))$ based on effective capacity in line 14. Since this update creates a constraint violation, it does not update any variable. The code then updates the variable using the best improving $\frac{\partial f}{\partial x}$ to $x_i + U[1, 25]$; where $U[1, 25]$ randomly generates an integer between 1 and 25. The idea here is to randomly update the best improving variable as this will trigger the least improving variables to reduce in strength in line 19. After this update, the values of s^*, s, and $\frac{\partial f}{\partial x}$ are updated, and this process is repeated if the random number generated in line 25 is greater than 1. At the end of this operation, assume that one or more constraints has been violated and the code now enters line 19 where the least improving variables are updated to the local optimal value. Under the while loop in line 19, the least improving $\frac{\partial f}{\partial x}$ is selected and the corresponding variable is updated to x^*. This operation is performed until all the constraints are satisfied. In line 22, we compute the objective function value

(since all constraints are satisfied) and update the current and previous best solution if an improvement is found. After this, the code tries to update the variable corresponding to the least improving $\frac{\partial f}{\partial x}$ to $x_i = max(0, x_i - 1)$. The max criteria is used so that the variables do not become negative. The code then tries to update the best variable using the PECH method using $x_i + max(1, int(0.2 * EC_i))$. The idea here is to force further reduction for the least improving variables, and this may allow the PECH based update to increment the greedy based best improving variable. Now, assume that the PECH based updates in line 29 found an improved solution. The current and the previous best solutions are then updated in line 33. If an updated solution is not found, the code then modifies the current solution set based on the random variable generated in line 35. If the random variable is true, then the modification occurs, otherwise no changes are made. In the modification process, the variable corresponding to the best improving or least improving $\frac{\partial f}{\partial x}$ is selected randomly and it is updated to $max(0, x^{'})$; the value of $x^{'}$ is derived from $2x_i^{\star} - x_i$. This operation may or may not violate the constraints, and, based on this, the next iteration either starts from line 13 or from line 19.

At the beginning of the next iteration, the constraints are either violated due to the operations in line 36, or the constraints are not violated. For the first case, the code enters into line 19 and the least improving variables are updated to its local optimal value. In the second case, the code enters into line 13 and it tries to update the variables based on available capacity and the random updates based on the best improving partial derivative.

Now, assume that the code has performed p iterations and the solution has not improved. At this stage, the current best or the previous best solutions is chosen randomly as the current solution set for this iteration. Based on the random number generated in line 40, the current solution set is then modified based on either the highest remaining capacity or the lowest remaining capacity. The variable with the lowest remaining capacity is updated to $x_i + int(0.1 * x_i)$, or the variable with the highest remaining capacity is updated to $max(0, x_i - 1)$. The idea here is to increase the strength of a variable that plays a major

part in the solution, or decrease the value of a variable that doesn't play a major role. At the end of these operations, the code is forced to assume that all constraints are satisfied, and in the next iteration, the code will enter line 16 for a random update with the current best or previous best solution (with or without modifications). The code continues in this fashion until the end time has been reached and returns the best solution at the end.

The main idea behind this Algorithm is to integrate the new XQX based variable update procedures with the existing PECH method. The random update of a variable in line 17 forces the constraint violation, and this helps the XQX based method to reduce the value of the least improving variable to its local optimal value. Similarly, the changes in variable values made in line 25 allow the PECH based method to update the best variable without violating any constraints. In this heuristics, we have implemented all the ideas presented in Chapter 2, and have also integrated our XQX based method with an existing method in literature.

4.3.2 Time Complexity of XQX based Heuristic for General IP

In this section, we provide the time complexity associated with the different operations in a iteration for the heuristic presented above. The computation of $\frac{\partial f}{\partial x_i}$ for each unknown variable (x_i) involves all the unknowns, slack variables, and a constant number. Hence, the time complexity of this operation is $\mathcal{O}(n(n+m))$. The calculation of s_i^* also involves $\mathcal{O}(n(n+m))$ operations, but it can be reduced to $\mathcal{O}(mn)$ operations based on 2.30. The calculation of x_i^* involves $\mathcal{O}(n+m)$ operations for one variable. Note that the x_i^* calculation is performed only when required, and not for all variables. The remaining capacity for each variable involves $\mathcal{O}(nm)$ operations, and the effective capacity involves $\mathcal{O}(n)$ operations. To find the best or worst x_i, we have used the quick sort algorithm, and its time complexity is $\mathcal{O}(n(logn))$. Updating s_i involves a total of m operations, and checking the constraint violation involves m operations in the worst case. Hence, we have $\mathcal{O}(m)$ for for updating s_i, and $\mathcal{O}(m)$ for checking constraint violation in the worst case. Calculating the new objective

function value has a complexity of $\mathcal{O}(n)$ since the slack variables do not play a role in this calculation based on Proposition 5 (2.33). The worst time complexity in this heuristics is the computation of $\frac{\partial f}{\partial x_i}$, but this can be reduced based on the Proposition 7 (2.43). Also note that the while loop in line 19 will not iterate m times. The worst case is m times, but this definitely will not happen. In the worst case scenario, with m operations in line 19, the worst time complexity would be $\mathcal{O}(mn(n+m))$. For other operations in the heuristics the time complexities are much less than the worst case reported.

4.4 Solutions for Benchmark Problems

In this section, we present the results from solving some benchmark problems. Since there wasn't any standard set of benchmark problem sets, we have created problems similar to the method used by Akçay *et al.* (2007). The generated set consists of a total of 45 problems of sizes ranging from $m = 10$ to $m = 200$ and $n = 100$ to $n = 500$, and for each combination of m and n, 5 problems were created. The method to create these problem sets are as follows. The coefficients for each variable in each constraint (r_{ij}) was randomly generated from $U[1, 100]$. The profit associated with each variable (p_j) was also randomly generated from $U[1, 100]$. To obtain the right hand side (b_i), first the values of x_j were randomly generated from $U[1, 200]$. Then, we use the slackness ratio $S_i = \frac{b_i}{\sum r_{ij} x_j}$ proposed by Zanakis (1977) to obtain the values for b_i. The different values for S_i were obtained randomly from $U[0.2, 0.4]$, $U[0.4, 0.6]$, or $U[0.6, 0.8]$. For each instance, after obtaining the values for r_{ij} and c_j, a value for S_i was generated, and by using the slackness ratio equation we obtain the values for the b_i by substituting the different values. Note that the values of r_{ij}, c_j and b_j are integers.

We have also considered some large problems of size ranging from $m = 15$ to $m = 100$ and $n = 100$ to $n = 2500$. There are a total of 11 such problems, and these were created by Glover & Kochenberger (1996). Note that, these exact same sets were solved as 0-1 type in the previous chapter, but here we solve them as general integer problems.

We have executed all the 56 problems on a dual core machine (2.88 GHz) with 8GB of RAM. To compare our results, we have done the following. For all the problem sets we are comparing the results obtained from our heuristics with CPLEX and PECH (Akçay *et al.*, 2007). We have coded the PECH algorithm, which is a single threaded algorithm and is also the first part of our XQX based heuristics. The CPLEX is a standard solver used by many in the literature to compare results. It is a multi-threaded program and used about 100% of the CPU resources to solve these 56 problems. On the other had, our XQX algorithm is single threaded with multi-threaded implementations for certain computations as mentioned earlier. The CPU occupancy of our heuristics was between 50% and 75%. Note that we have used multi-threading computation to calculate the remaining capacities for PECH, due to which the execution times are much faster than a completely single threaded PECH.

The first 45 instances were solved 2 times for 15 minutes each, and the larger 11 instances were solved 2 times for 30 minutes each by XQX based heuristics. The CPLEX runtime was set to 1 hour for the first 45 instances, and it returned the best solution found in 1 hour of wall time. For the 11 larger instances, CPLEX was allowed to run beyond 1 day, and the execution was terminated arbitrarily after reaching more than 1 day for certain instances. Due to the arbitrary termination criteria, the CPLEX times reported for the 11 problem sets are not exactly the same. The CPLEX version 12.6 was used, and it was executed using its auto algorithm mode.

In all the tables shown below (except two), the first column shows the problem instance or the problem size. In column 2, the solution obtained by our XQX based heuristics is given, followed by its time required to obtain the solution in column 3. In column 4, the solution obtained by our PECH implementation (Akçay *et al.*, 2007) is given, followed by its time required to obtain the solution in column 5. In column 6, the solution obtained by CPLEX is given, followed by its time required to obtain the solution in column 7. In column 8, we report the %gap between XQX and the CPLEX solution, and in column 9 we report the %gap between PECH and the CPLEX solution.

Note that, we can assume the CPLEX solution to be optimal if: It returned a solution in less than 1 hour for the 45 instances, or less than 1 day for the 11 large instances.

4.4.1 Results for $x_i = 0$ and $P = 100$

In this section, we present the results for the 56 instances in tables 4.1 to 4.10. These instances were solved with the penalty value $P = 100$, and all the unknowns (x_i) were initialized to 0.

No.	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
1	15756.0	92	14328.0	1	15793.0	0.2	0.234	9.276
2	16120.0	93	15003.0	1	16125.0	0.2	0.031	6.958
3	20793.0	834	19617.0	1	20950.0	2	0.749	6.363
4	29730.0	582	25413.0	1	29852.0	0.2	0.409	14.87
5	31201.0	186	28771.0	1	31220.0	0.3	0.061	7.844

Table 4.1: Results for: $m = 10, n = 100$. (Generated Instances)

No.	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
1	42422.0	667	35951.0	2	42595.0	2	0.406	15.598
2	62473.0	663	59267.0	2	63386.0	5	1.44	6.498
3	50112.0	85	50021.0	2	50119.0	0.3	0.014	0.196
4	87254.0	282	85026.0	2	87401.0	0.8	0.168	2.717
5	84859.0	836	82934.0	2	85533.0	4	0.788	3.039

Table 4.2: Results for: $m = 10, n = 250$. (Generated Instances)

No.	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
1	99509.0	890	87378.0	4	104416.0	5	4.699	16.317
2	115719.0	726	113538.0	3	116268.0	6	0.472	2.348
3	90865.0	859	89112.0	3	93587.0	26	2.909	4.782
4	138671.0	895	136826.0	4	142109.0	0.5	2.419	3.718
5	162300.0	849	160908.0	3	166676.0	1	2.625	3.461

Table 4.3: Results for: $m = 10, n = 500$. (Generated Instances)

No.	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
1	10677.0	139	10133.0	2	10870.0	99	1.776	6.78
2	16125.0	600	14824.0	2	16326.0	2814	1.231	9.2
3	16594.0	830	15819.0	2	16742.0	1017	0.884	5.513
4	20874.0	888	19570.0	2	21154.0	1472	1.324	7.488
5	21876.0	429	21213.0	2	22004.0	3600	0.582	3.595

Table 4.4: Results for: $m = 100, n = 100$. (Generated Instances)

No.	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
1	30030.0	266	27850.0	3	30751.0	3600	2.345	9.434
2	29556.0	711	28724.0	3	30355.0	3600	2.632	5.373
3	55461.0	881	53051.0	4	56948.0	3600	2.611	6.843
4	55272.0	533	52273.0	4	56904.0	3600	2.868	8.138
5	56077.0	575	52407.0	4	57905.0	3600	3.157	9.495

Table 4.5: Results for: $m = 100, n = 250$. (Generated Instances)

No.	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
1	115355.0	766	108211.0	7	120906.0	3600	4.591	10.5
2	91353.0	880	87740.0	6	94338.0	3600	3.164	6.994
3	89123.0	835	86142.0	6	92762.0	3600	3.923	7.137
4	89689.0	883	87540.0	6	92675.0	3600	3.222	5.541
5	88610.0	882	83433.0	6	91605.0	3600	3.269	8.921

Table 4.6: Results for: $m = 100, n = 500$. (Generated Instances)

No.	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
1	10350.0	661	9621.0	3	10440.0	3600	0.862	7.845
2	14949.0	718	13956.0	4	15114.0	3600	1.092	7.662
3	10178.0	755	9446.0	3	10302.0	2374	1.204	8.309
4	19443.0	200	18775.0	4	19728.0	3600	1.445	4.831
5	19897.0	692	17999.0	4	20143.0	3600	1.221	10.644

Table 4.7: Results for: $m = 200, n = 100$. (Generated Instances)

No.	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
1	26557.0	736	25541.0	5	27376.0	3600	2.992	6.703
2	28340.0	827	27338.0	5	29158.0	3600	2.805	6.242
3	27078.0	825	25124.0	5	27845.0	3600	2.755	9.772
4	52279.0	750	49969.0	6	53896.0	3600	3.0	7.286
5	52823.0	457	49816.0	6	54409.0	3600	2.915	8.442

Table 4.8: Results for: $m = 200, n = 250$. (Generated Instances)

No.	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
1	58495.0	858	54730.0	8	61056.0	3600	4.195	10.361
2	86371.0	883	83198.0	8	88583.0	3600	2.497	6.079
3	114582.0	897	110237.0	9	118139.0	3600	3.011	6.689
4	113903.0	899	107150.0	9	118881.0	3600	4.187	9.868
5	84417.0	811	77788.0	8	88553.0	3600	4.671	12.157

Table 4.9: Results for: $m = 200, n = 500$. (Generated Instances)

n, m	XQX (Q) Z	Time (sec)	PECH (E) Z	Time (sec)	CPLEX (C) Z	Time (sec)	Q vs. C % Gap	E vs. C % Gap
100, 15	3931.0	952	3824.0	1	3953.0	9	0.557	3.263
100, 25	3994.0	31	3966.0	1	4114.0	32	2.917	3.597
150, 25	5773.0	275	5594.0	1	5879.0	2329	1.803	4.848
150, 50	5713.0	537	5638.0	1	5872.0	113695	2.708	3.985
200, 25	7659.0	136	7438.0	1	7886.0	57387	2.879	5.681
200, 50	7637.0	423	7537.0	1	7839.0	117550	2.577	3.853
500, 25	19996.0	1779	19222.0	2	20536.0	90885	2.63	6.399
500, 50	18703.0	1274	18416.0	4	19391.0	122165	3.548	5.028
1500, 25	59530.0	1306	59078.0	2	62518.0	97624	4.779	5.502
1500, 50	57807.0	1484	57655.0	3	60095.0	91797	3.807	4.06
2500, 100	94757.0	1762	93259.0	5	98031.0	122105	3.34	4.868

Table 4.10: Results for: Other large instances (Glover & Kochenberger Instances)

4.4.2 Results for $x_i = 0$ with $P = 100$, $P = 500$, and $P = 1000$

In this section, we present the results for the 11 large instances in tables 4.11 and 4.12. These instances were solved with the penalty values $P = 100$, $P = 500$, and $P = 1000$ and all the unknowns (x_i) were initialized to 0.

n, m	XQX (P1) Z	Time (sec)	XQX (P2) Z	Time (sec)	XQX (P3) Z	Time (sec)	CPLEX (C) Z	Time (sec)
100, 15	3931.0	952	3938.0	432	3918.0	222	3953.0	9
100, 25	3994.0	31	3972.0	27	4002.0	399	4114.0	32
150, 25	5773.0	275	5787.0	1565	5723.0	168	5879.0	2329
150, 50	5713.0	537	5699.0	1074	5680.0	610	5872.0	113695
200, 25	7659.0	136	7589.0	983	7661.0	337	7886.0	57387
200, 50	7637.0	423	7621.0	725	7577.0	1702	7839.0	117550
500, 25	19996.0	1779	20085.0	629	19890.0	1639	20536.0	90885
500, 50	18703.0	1274	18585.0	1698	18543.0	778	19391.0	122165
1500, 25	59530.0	1306	59680.0	1483	59838.0	1615	62518.0	97624
1500, 50	57807.0	1484	57848.0	1769	57798.0	1629	60095.0	91797
2500, 100	94757.0	1762	93965.0	1724	94079.0	1793	98031.0	122105

Table 4.11: Results for: Other large instances with $P1 = 100$, $P2 = 500$, and $P3 = 1000$ (Glover & Kochenberger Instances)

n, m	P1 vs. C %Gap	P2 vs. C %Gap	P3 vs. C %Gap
100, 15	0.557	0.379	0.885
100, 25	2.917	3.452	2.722
150, 25	1.803	1.565	2.654
150, 50	2.708	2.946	3.27
200, 25	2.879	3.766	2.853
200, 50	2.577	2.781	3.342
500, 25	2.63	2.196	3.146
500, 50	3.548	4.157	4.373
1500, 25	4.779	4.539	4.287
1500, 50	3.807	3.739	3.822
2500, 100	3.34	4.148	4.031

Table 4.12: % Difference with CPLEX for other large instances with $P1 = 100$, $P2 = 500$, and $P3 = 1000$ (Glover & Kochenberger Instances)

4.4.3 Results for $P = 100$ with $x_i = 0$, $x_i \sim U[0, 1]$, and $x_i \sim U[1, 5]$

In this section, we present the results for the 11 large instances in tables 4.13 and 4.14. These instances were solved with the penalty value $P = 100$, and three different initial values for the unknowns were used. $x_i = 0$: all unknown initialized to 0. $x_i \sim U[0, 1]$: all unknown variables were initialized to random integer number generated in the range [0,1]. $x_i \sim U[1, 5]$: all unknown variables were initialized to random integer number generated in the range [1,5].

n, m	XQX (R1) Z	Time (sec)	XQX (R2) Z	Time (sec)	XQX (R3) Z	Time (sec)	CPLEX (C) Z	Time (sec)
100, 15	3931.0	952	3929.0	95	3936.0	1406	3953.0	9
100, 25	3994.0	31	4046.0	128	4043.0	186	4114.0	32
150, 25	5773.0	275	5764.0	1362	5773.0	1459	5879.0	2329
150, 50	5713.0	537	5704.0	508	5713.0	1534	5872.0	113695
200, 25	7659.0	136	7718.0	109	7720.0	941	7886.0	57387
200, 50	7637.0	423	7656.0	1141	7579.0	1146	7839.0	117550
500, 25	19996.0	1779	19876.0	1790	19901.0	1564	20536.0	90885
500, 50	18703.0	1274	18666.0	1263	18712.0	818	19391.0	122165
1500, 25	59530.0	1306	59503.0	1605	59140.0	1259	62518.0	97624
1500, 50	57807.0	1484	57673.0	1619	57654.0	1692	60095.0	91797
2500, 100	94757.0	1762	93806.0	1465	93165.0	1756	98031.0	122105

Table 4.13: Results for: Other large instances with $R1 : x_i = 0$, $R2 : x_i \sim U[0, 1]$, and $R3 : x_i \sim U[1, 5]$ (Glover & Kochenberger Instances)

n, m	R1 vs. C %Gap	R2 vs. C %Gap	R3 vs. C %Gap
100, 15	0.557	0.607	0.43
100, 25	2.917	1.653	1.726
150, 25	1.803	1.956	1.803
150, 50	2.708	2.861	2.708
200, 25	2.879	2.13	2.105
200, 50	2.577	2.334	3.317
500, 25	2.63	3.214	3.092
500, 50	3.548	3.739	3.502
1500, 25	4.779	4.823	5.403
1500, 50	3.807	4.03	4.062
2500, 100	3.34	4.31	4.964

Table 4.14: % Difference with CPLEX for other large instances $R : x_i = 0$, $R2 : x_i \sim U[0, 1]$, and $R3 : x_i \sim U[1, 5]$ (Glover & Kochenberger Instances)

4.4.4 Results Discussion

The results obtained using XQX based heuristics improved all the solutions found by PECH in tables 4.1 to 4.9. In the smallest problem set with $m = 10$ and $n = 100$, CPLEX provides the best results at a very short time. For the same problem set, the XQX based heuristics was able to provide high quality results at a reasonable time, and it was able to significantly improve the results from PECH. Similar conclusions can be drawn for the all problem sets with $m = 10$. As the number of constraints increases, for example when $m = 100$, CPLEX tends to take more time (1 hour) in certain cases. For one such case, the XQX based heuristics was able to provide within 0.58% of the result obtained by CPLEX in about 430 seconds. Again, the XQX based heuristics was able to improve all the PECH

based results significantly for problems with $m = 100$. As the problem size grows larger, $m \geq 100$ and $n \geq 250$, CPLEX takes more time. All the results reported by CPLEX for these problem sizes are obtained at 1 hour, which means that they are not the optimal solutions. For these problem sets, the XQX based heuristics was able to obtain within 0.862% of the result obtained by CPLEX in about 661 seconds in one case. Overall, for the 45 problem instances, the XQX based heuristics was able to improve all the PECH based solutions. When compared with CPLEX, the XQX based heuristics was able to obtain within 4.67% of the result obtained by CPLEX in 811 seconds for the largest problem size within these 45 sets. For the same problem, PECH obtained within 12.157% of the CPLEX solution in 8 seconds. Note that the results reported by our PECH implementation cannot improve beyond what is reported in the tables. This is due to the fact that it does not have any repair mechanisms to alter the strength of any variable. Overall, the XQX based heuristics improved the PECH solution between 15.1% and 0.182% for these problem sets.

In the results reported in table 4.10 for large problem sets, CPLEX runs for a much longer time ($>$ 1 day) even for a small instance such as $m = 50$ and $n = 150$. Note that the CPLEX solutions obtained after 1 day in several of these cases are not optimal, but the best solution found at that time. There is a good possibility that CPLEX may run out of memory or take a much longer time to obtain the optimal solutions for most of problem sets here. The results reported by PECH are of high quality within a very short time. It appears as if PECH can obtain high quality solutions for very large problem sets when compared with its trend from tables 4.1 to 4.9. The XQX based heuristics was also able to provide high quality solutions and improved all the solutions found by PECH; between 3.769% and 0.253% better. In one case, the result obtained by XQX was within 2.63% of the solution found by CPLEX, and in this specific case, CPLEX returned the solution after 1.05 days, where as XQX returned the solution in about 0.5 hours. For all the large problems, XQX was able to find within 4% of the solution returned by CPLEX in most of the cases.

The results presented in table 4.11 show the solutions obtained by solving the 11 large

problem sets with different penalty values and initial $x_i = 0$. Note that the penalty value is a part of the XQX model derived previously. In table 4.12, the % difference between the solutions obtained from CPLEX and by using various penalty values are shown. Note that the solutions from CPLEX presented in table 4.11 are the same as the CPLEX solutions from table 4.10. For the largest problem, the penalty value $P = 100$ provides the best solution when compared with others. For the smallest problem, the penalty value of $P = 500$ provides the best results. The penalty value $P = 1000$ seems to be the worst case since it provided the best solution for 3 problems. The penalty value $P = 100$ provided the best results for 4 problems, similar to the penalty value $P = 500$, which provided the best results for 4 problems. From these results, it is difficult to see which penalty value is a suitable number.

The results presented in table 4.13 show the solutions obtained by solving the 11 large problem sets with different initial values for x_i and $P = 100$. In table 4.12, the % difference between the solutions obtained from CPLEX and by using various initial values are shown. Note that the results from the XQX heuristics with different initial values for x_i are not compared with different initial values for PECH. The results obtained when using $x_i \sim U[0, 1]$ affects the performance of the XQX heuristics since it was able to find the best solutions in only 2 problems The results obtained when using $x_i \sim U[1, 5]$ were able to find the best solutions in 5 problems, followed by $x_i = 0$, which found the best solutions for 6 problems. Also, the solutions with initial values $x_i \sim U[1, 5]$ take more time on an average when compared with the solutions obtained with initial values $x_i = 0$. For the largest problem, the initial value of $x = 0$ found the best results, and for the smallest problem, the initial value of $x_i \sim U[1, 5]$ found the best result. It can also be seen that irrespective of the initial values, the XQX based heuristics was ale to provide almost the same quality results. From the results obtained from CPLEX for the largest problem set, 96% of x_i had a value of 0 and only the rest 4% contributed to the solution. The XQX based heuristics was still able to remove most of the least attractive variables and provide good quality solutions in

the case of $x_i \sim U[1,5]$. Even though we are showing any results from PECH with an initial value of $x_i \sim U[1,5]$, it is fair to say that PECH will perform poorly under such conditions since there are no repair operators. One possible conclusion from these results could be to use initial values $x_i = 0$ for large problems and to use initial values $x_i \sim U[1,5]$ for small problems.

To conclude this discussion, the proposed XQX model was able to handle small, medium and very large problem sizes. It was able to improve all the solutions found by PECH and was able to provide good quality solutions when compared with CPLEX. The solutions for very large problems were also better from XQX heuristics when compared with PECH. Within a reasonable amount of time, XQX heuristics was able to obtain good quality solutions when compared with the results obtained from CPLEX after 1 day. The heuristics we presented integrates the proposed XQX based method and PECH. By integrating our XQX model with other sophisticated heuristic algorithm we can obtain even better solutions for any problem size, and obtain better performance than CPLEX.

4.4.5 Feasibility Issues for XQX based Heuristics

The results presented above from the XQX based heuristics are all feasible solutions. The PECH based initialization used in the initial stage of the heuristics does not provide infeasible solutions. In the XQX based heuristics, the random updates to variables ($U[1,25]$) can make the solution in that iteration infeasible, but a repair operator is applied immediately afterwards to lower the values of least attractive variables. The repair operator is based on the core methods described in the XQX model earlier, and it serves two purposes. One, it is able to lower the values of least attractive variables, and two, it continues to do so until all the constraints are satisfied. Hence, the repair operator, which is based on x^*, helps to maintain feasibility.

4.5 Summary

In this chapter, we defined the GMDKP problem based on existing literature and presented a detailed literature review for the same. Due to the non-availability of benchmark problems, we generated certain problems based on existing methods and applied our XQX model for general integer problems using a basic heuristics we developed. The XQX based heuristics was able to handle problems of any size and it was able to find reasonable quality solutions within reasonable time. It also maintained feasibility for all solutions, and it outperformed the existing procedure in literature called PECH, in all cases.

CHAPTER 5

CONCLUSIONS

This thesis developed a XQX model for general integer programming problems in which the constraints are equality, inequality, or a combination of both.

First, we derived the XQX model for general integer programming problems of any size. Next, we showed how this model is applicable for problems with only binary unknowns. We then defined the procedures to update an unknown integer variable to a new value. We then defined several cases for variable updates based on the current value of the integer variable, including the procedures to update the slack variables after updating the unknowns. We also presented new theorems and their proofs to support our XQX model.

Next, we applied our XQX model to the 0-1 multidimensional knapsack problem (0-1 MDKP) using a heuristics that we developed. We found that the XQX based heuristics was able to handle problems of any size. We found that the XQX based heuristics for the 0-1 MDKP was also able to provide reasonable quality results at a reasonable time when compared with the other methods from literature. We also found that the heuristics was able to provide good quality results in a short time for very large instances.

Finally, we applied our XQX model to the general multidimensional knapsack problem (GMDKP) using another heuristics we developed. We integrated an existing procedure called PECH from literature with our heuristics and solved several problems of various sizes. We found that the XQX based heuristics for GMDKP was able to improve all the results when compared with the existing procedure in literature called PECH. We also found that this heuristics was able to always provide feasible solutions irrespective of the starting solutions

and the procedures implemented. The results from our heuristics when compared with a standard solver also fared well. We found that it is difficult to assign a specific penalty value P for the XQX model. We also found that certain initial solutions may not be the best choice.

CHAPTER 6

FUTURE DIRECTIONS

We believe that the work presented here will provide a new direction for XQX based models in operations management and supply chain management. Operations management is referred to the planning, scheduling, and controlling activities within a firm, which transforms inputs to finished goods and services (Bozarth & Handfield, 2008). Where as in supply chain management, a network of manufacturers and service providers work together to move the goods from raw materials to the end user, and they are linked via the physical, information and monetary flow. There are a number of activities within any supply chain such as, planning activities to plan demand requirements against available resources, sourcing activities to schedule the procurement of goods, production activities to produce the goods and services, delivery activities to move good, and return activities to process the excess and damaged goods. The active management of these activities ensures a firm to achieve competitive advantage and at the same time maximize customer value. In all the activities mentioned above, modeling plays an important role and especially due to the global nature of supply chains, modeling and solving these large-scale, real-world, complex supply chains are very challenging. The main objective in modeling these activities is to either maximize or minimize a specific goal; broadly speaking, maximizing profit or minimizing cost. There also several different resources constraints associated with each activity, which are either equality, inequality or a combination of both.

The following are some of the areas in supply chain where our XQX based model can be used to solve some challenging problems. Managing quality is an important aspect for

any firm producing goods and services. A quality product should conform to the standards required, provide basic performance, must be free from defects, should be reliable and durable among other factors. At the same time, such a product should be manufactured by maintaining low cost. For example, efficient design and manufacturing of printed circuit boards. Project management is another important part of any business process where the technologies, skills, and tools are integrated to achieve project requirement. For example, assigning resources to several stages of the project or controlling the overall duration of the project based on critical paths and process times. Manufacturing process such as production line, continuous flow process, jobshop, batch manufacturing, fixed position layout, hybrid manufacturing, and linking several manufacturing processes across the supply chain is another area where optimal conditions can be obtained by efficient solution methods to the optimization models. Managing capacity to produce output by a specific time in another important issue. Lead and lag capacity should be planned considering the cost, raw material availability and machine downtime. Logistics also plays a vital role in controlling the flow and the reverse flow of goods. Optimal goods delivery over many modes of transportation. Optimal location and types of warehouses for goods handling are important issues. Managing inventory to satisfy the needs of upstream and downstream firms in a supply chain, and to minimize "Bullwhip effect".

The heuristics presented in this work to solve the 0-1 MDKP and GMDKP problems does not integrate any advanced metaheuristics available in the literature. The XQX model and its solution procedures can be implemented using the following metaheuristics such as simulated annealing, advanced tabu search procedures, greedy randomized adaptive search procedure (GRASP), variable neighborhood search, genetic algorithm, estimation of distribution algorithm, scatter search and path relinking, ant colony optimization, particle swarm optimization, mementic algorithm and other procedures. We have provided directions in this work to show that a variable can also be updated to an integer value within the parabola

while improving the objective function function value. Depending on the problem modeled, it will be worth exploring the effects of $x_j^\star$ based updates vs. updating the variable to an integer value within the parabola, while improving the objective function value. Note that the $x_j^\star$ based updates are only good for that specific local solution, and the value of a variable previously updated to its $x_j^\star$ may need an update.

The penalty value P also plays an important role in the XQX model. The convergence and quality of the solution depends upon this value, but since this is a problem dependent value, it is very hard to find an appropriate P value. In the solution procedure we presented earlier, a constant value of P is used. This could be a dynamic value for a problem instance depending upon the solution improvement rate. Finding an appropriate P value based on the number of constraints, or the number of variables, or the coefficients of the objective function, or the coefficients of the constraints is also an area worth exploring.

BIBLIOGRAPHY

BIBLIOGRAPHY

Abe, S.; Kawakami, J.; & Hirasawa, K. (1992) "Solving Inequality Constrained Combinatorial Optimization Problems by the Hopfield Neural Networks." *Neural Networks*, Vol. 5(4), pp. 663–670.

Akçay, Y.; Li, H.; & Xu, S. H. (2007) "Greedy algorithm for the general multidimensional knapsack problem." *Annals of Operations Research*, Vol. 150(1), pp. 17–29.

Alidaee, B.; Kochenberger, G.; Lewis, K.; Lewis, M.; & Wang, H. (2008) "A new approach for modeling and solving set packing problems." *European Journal of Operational Research*, Vol. 186(2), pp. 504–512.

Angelelli, E.; Mansini, R.; & Grazia Speranza, M. (2010) "Kernel search: A general heuristic for the multi-dimensional knapsack problem." *Computers & Operations Research*, Vol. 37(11), pp. 2017–2026.

Balas, E. (1965) "An additive algorithm for solving linear programs with zero-one variables." *Operations Research*, Vol. pages 517–549.

Balas, E. & Martin, C. H. (1980) "Pivot and complement–A heuristic for 0-1 programming." *Management Science*, Vol. 26(1), pp. 86–96.

Balev, S.; Yanev, N.; Fréville, A.; & Andonov, R. (2008) "A dynamic programming based reduction procedure for the multidimensional 0–1 knapsack problem." *European Journal of Operational Research*, Vol. 186(1), pp. 63–76.

Belhe, U. & Kusiak, A. (1997) "Dynamic scheduling of design activities with resource constraints." *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, Vol. 27(1), pp. 105–111.

Ben Hadj-Alouane, A. & Bean, J. C. (1997) "A genetic algorithm for the multiple-choice integer program." *Operations research*, Vol. 45(1), pp. 92–101.

Bosch, R. & Trick, M. (2005) "Integer programming." In *Search Methodologies*, pages 69–95. Springer-Verlag, Heidelberg, Germany.

Boussier, S.; Vasquez, M.; Vimont, Y.; Hanafi, S.; & Michelon, P. (2010) "A multi-level search strategy for the 0–1 Multidimensional Knapsack Problem." *Discrete Applied Mathematics*, Vol. 158(2), pp. 97–109.

Bozarth, C. C. & Handfield, R. B. (2008) *Introduction to operations and supply chain management*. Prentice Hall.

Chen, D.; Batson, R.; & Dang, Y. (2011) *Applied Integer Programming: Modeling and Solution*. John Wiley & Sons, Inc., Hoboken, N.J.

Chu, P. & Beasley, J. (1998) "A Genetic Algorithm for the Multidimensional Knapsack Problem." *Journal of Heuristics*, Vol. 4(1), pp. 63–86.

Douiri, S. M. & Elbernoussi, S. (2012) "An unconstrained binary quadratic programming for the maximum independent set problem." *Nonlinear Analysis*, Vol. 17(4), pp. 410–417.

Drexl, A. (1988) "A simulated annealing approach to the multiconstraint zero-one knapsack problem." *Computing*, Vol. 40(1), pp. 1–8.

Everett, H. (1963) "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources." *Operations Research*, Vol. 11(3), pp. 399–417.

Fidanova, S. (2005) "Heuristics for multiple knapsack problem." *IADIS AC*, Vol. pages 255–260.

Fréville, A. (2004) "The multidimensional 0–1 knapsack problem: An overview." *European Journal of Operational Research*, Vol. 155(1), pp. 1–21.

Fréville, A. & Hanafi, S. (2005) "The multidimensional 0-1 knapsack problem - Bounds and computational aspects." *Annals of Operations Research*, Vol. 139(1), pp. 195–227.

Fréville, A. & Plateau, G. (1994) "An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem." *Discrete Applied Mathematics*, Vol. 49(1), pp. 189–212.

Gabrel, V. & Minoux, M. (2002) "A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems." *Operations Research Letters*, Vol. 30(4), pp. 252–264.

Gavish, B. & Pirkul, H. (1985) "Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality." *Mathematical Programming*, Vol. 31(1), pp. 78–105.

Gavish, B. & Pirkul, H. (1986) "Computer and database location in distributed computer systems." *Computers, IEEE Transactions on*, Vol. 100(7), pp. 583–590.

Glover, F. & Kochenberger, G. A. (1996) "Critical event tabu search for multidimensional knapsack problems." In *Meta-Heuristics*, pages 407–427. Springer-Verlag, Heidelberg, Germany.

Gow, I. & Reichelstein, S. (2007) "Capital Budgeting: The Role of Cost Allocations." *Operations Research Proceedings 2006*, Vol. pages 115–121.

Hillier, F. S. (1969) "Efficient heuristic procedures for integer linear programming with an interior." *Operations Research*, Vol. 17(4), pp. 600–637.

Hopfield, J. J. & Tank, D. W. (1985) "Neural computation of decisions in optimization problems." *Biological cybernetics*, Vol. 52(3), pp. 141–152.

Jünger, M.; Liebling, T.; & Naddef, D. (2008) *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer-Verlag, Heidelberg, Germany.

Kaparis, K. & Letchford, A. N. (2008) "Local and global lifted cover inequalities for the 0–1 multidimensional knapsack problem." *European Journal of Operational Research*, Vol. 186(1), pp. 91–103.

Kellerer, H.; Pferschy, U.; & Pisinger, D. (2004) *Knapsack Problems*. Springer-Verlag, Heidelberg, Germany.

Klöckner, A.; Pinto, N.; Lee, Y.; Catanzaro, B.; Ivanov, P.; & Fasih, A. (2012) "PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation." *Parallel Computing*, Vol. 38(3), pp. 157–174.

Kochenberger, G. & Glover, F. (2006) "A unified framework for modeling and solving combinatorial optimization problems: A tutorial." *Multiscale Optimization Methods and Applications*, Vol. pages 101–124.

Kochenberger, G.; Glover, F.; Alidaee, B.; & Rego, C. (2004) "A unified modeling and solution framework for combinatorial optimization problems." *OR Spectrum*, Vol. 26(2), pp. 237–250.

Kochenberger, G.; Hao, J.-K.; Glover, F.; Lewis, M.; Lü, Z.; Wang, H.; & Wang, Y. (2014) "The unconstrained binary quadratic programming problem: a survey." *Journal of Combinatorial Optimization*, Vol. pages 1–24.

Kochenberger, G. A.; McCarl, B. A.; & Paul Wyman, F. (1974) "A heuristic for general integer programming." *Decision Sciences*, Vol. 5(1), pp. 36–44.

Lee, H.-M. & Hsu, C.-C. (1989) "Neural network processing through energy minimization with learning ability to the multiconstraint zero-one knapsack problem." In *Tools for Artificial Intelligence, 1989. Architectures, Languages and Algorithms, IEEE International Workshop on*, pages 548–555.

Lee, J. S. & Guignard, M. (1988) "Note – An Approximate Algorithm for Multidimensional Zero-One Knapsack ProblemsA Parametric Approach." *Management Science*, Vol. 34(3), pp. 402–410.

Lewis, M.; Alidaee, B.; & Kochenberger, G. (2005) "Using xQx to model and solve the uncapacitated task allocation problem." *Operations research letters*, Vol. 33(2), pp. 176–182.

Li, D. & Sun, X. (2006) *Nonlinear integer programming.*, volume 84 Springer.

Lin, E. Y.-H. (1998) "A Biblographical Survey on Some Wellknown Non-Standard Knapsack Problems." *Infor-Information Systems and Operational Research*, Vol. 36(4), pp. 274–317.

Løkketangen, A. & Glover, F. (1996) "Probabilistic move selection in tabu search for zero-one mixed integer programming problems." In *Meta-Heuristics*, pages 467–487. Springer-Verlag, Heidelberg, Germany.

Loulou, R. & Michaelides, E. (1979) "New greedy-like heuristics for the multidimensional 0-1 knapsack problem." *Operations Research*, Vol. 27(6), pp. 1101–1114.

Luenberger, D. G. (1973) *Introduction to linear and nonlinear programming.*, volume 28 Addison-Wesley Reading, MA.

Luenberger, D. G. & Ye, Y. (2008) *Linear and nonlinear programming.*, volume 116 Springer.

Magazine, M. & Oguz, O. (1984) "A heuristic algorithm for the multidimensional zero-one knapsack problem." *European Journal of Operational Research*, Vol. 16(3), pp. 319–326.

Mansini, R. & Speranza, M. G. (2012) "Coral: An exact algorithm for the multidimensional knapsack problem." *INFORMS Journal on Computing*, Vol. 24(3), pp. 399–415.

Monfared, M. & Etemadi, M. (2006) "The impact of energy function structure on solving generalized assignment problem using Hopfield neural network." *European Journal of Operational Research*, Vol. 168(2), pp. 645 – 654.

Ohlsson, M.; Peterson, C.; & Söderberg, B. (1993) "Neural networks for optimization problems with inequality constraints: the knapsack problem." *Neural Computation*, Vol. 5(2), pp. 331–339.

Osorio, M. A.; Glover, F.; & Hammer, P. (2002) "Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions." *Annals of Operations Research*, Vol. 117(1–4), pp. 71–93.

Pirkul, H. (1987) "A heuristic solution procedure for the multiconstraint zero-one knapsack problem." *Naval Research Logistics (NRL)*, Vol. 34(2), pp. 161–172.

Pirkul, H. & Narasimhan, S. (1986) "Efficient algorithms for the multiconstraint general knapsack problem." *IIE transactions*, Vol. 18(2), pp. 195–203.

Puchinger, J.; Raidl, G. R.; & Pferschy, U. (2010) "The multidimensional knapsack problem: Structure and algorithms." *INFORMS Journal on Computing*, Vol. 22(2), pp. 250–265.

Senju, S. & Toyoda, Y. (1968) "An approach to linear programming with 0–1 variables." *Management Science*, Vol. 15(4), pp. B196–B207.

Shih, W. (1979) "A branch and bound method for the multiconstraint zero-one knapsack problem." *Journal of the Operational Research Society*, Vol. pages 369–378.

Sinclair, M. (1986) "An exact penalty function approach for nonlinear integer programming problems." *European journal of operational research*, Vol. 27(1), pp. 50–56.

Smith, K. A. (1999) "Neural networks for combinatorial optimization: a review of more than a decade of research." *INFORMS Journal on Computing*, Vol. 11(1), pp. 15–34.

Talaván, P. M. & Yáñez, J. (2006) "The generalized quadratic knapsack problem. A neuronal network approach." *Neural Networks*, Vol. 19(4), pp. 416–428.

Thiel, J. & Voss, S. (1994) "Some experiences on solving multiconstraint zero-one knapsack problems with genetic algorithms." *INFOR-Information Systems and Operational Research*, Vol. 32(4), pp. 226–242.

Vaithyanathan, S.; Ogmen, H.; & Ignizio, J. (1994) "Generalized Boltzmann Machines for Multidimensional Knapsack Problems." In *Proceedings of the Artificial Neural Networks in Engineering,1994.*, pages 1079–1084.

Varnamkhasti, M. J. (2012) "Overview of the Algorithms for Solving the Multidimensional Knapsack Problems." *Advanced Studies in Biology*, Vol. 4(1), pp. 37–47.

Vasquez, M. & Hao, J.-K. (2001) "A hybrid approach for the 0-1 multidimensional knapsack problem." In *International Joint Conference on Artificial Intelligence*, pages 328–333. LAWRENCE ERLBAUM ASSOCIATES LTD.

Vasquez, M. & Vimont, Y. (2005) "Improved results on the 0–1 multidimensional knapsack problem." *European Journal of Operational Research*, Vol. 165(1), pp. 70–81.

Vimont, Y.; Boussier, S.; & Vasquez, M. (2008) "Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem." *Journal of Combinatorial Optimization*, Vol. 15(2), pp. 165–178.

Volgenant, A. & Zoon, J. (1990) "An improved heuristic for multidimensional 0-1 knapsack problems." *Journal of the Operational Research Society*, Vol. pages 963–970.

Wang, J.; Zhou, Y.; Cai, Y.; & Yin, J. (2012a) "Learnable tabu search guided by estimation of distribution for maximum diversity problems." *Soft Computing*, Vol. 16(4), pp. 711–728.

Wang, Y. (2014) *Metaheuristics for large binary quadratic optimization and its applications*. PhD thesis, Université dAngers.

Wang, Y.; Lü, Z.; Glover, F.; & Hao, J.-K. (2012b) "A multilevel algorithm for large unconstrained binary quadratic optimization." In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 395–408. Springer.

Wang, Y.; Lü, Z.; Glover, F.; & Hao, J.-K. (2012c) "Path relinking for unconstrained binary quadratic programming." *European Journal of Operational Research*, Vol. 223, pp. 595–604.

Weingartner, H. M. & Ness, D. N. (1967) "Methods for the solution of the multidimensional 0/1 knapsack problem." *Operations Research*, Vol. 15(1), pp. 83–103.

Wen, U.-P.; Lan, K.-M.; & Shih, H.-S. (2009) "A review of Hopfield neural networks for solving mathematical programming problems." *European Journal of Operational Research*, Vol. 198(3), pp. 675–687.

Wolsey, L. (1998) *Integer Programming.* John Wiley & Sons, Inc., New York, N.Y.

Yamamoto, A.; Ohta, M.; Ueda, H.; Ogihara, A.; & Fukunaga, K. (1995) "Asymmetric Neural Network and its Application to Knapsack Problem." *IEICE Trans. Fundamentals*, Vol. E78–A(3), pp. 330–305.

Yang, M.-H. (2001) "An efficient algorithm to allocate shelf space." *European Journal of Operational Research*, Vol. 131(1), pp. 107–118.

Zanakis, S. H. (1977) "Heuristic 0-1 linear programming: An experimental comparison of three methods." *Management Science*, Vol. 24(1), pp. 91–104.

VITA

Vijay P. Ramalingam completed his Bachelor of Engineering degree in Electronics and Communications Engineering from University of Madras, Tamilnadu, India in May 2000.

In August 2001, he joined the Center for Wireless Communication in the University of Mississippi to pursue his MS degree in Engineering Science. He completed his Master's degree in December 2005.

During his Master's he was working at the Ford Center for the Performing arts as Technical assistant. After completing his Master's he was working full time at the National Center for Physical Acoustics as Research & Development Engineer until Jan 2009.

He then joined the National Center for Computional Hydroscience and Engineering in Jan 2009 and is currently employed there as Research Software Developer.

He started his Doctoral program in School of Business with emphasis on POM in Fall 2007 and completed the same in Dec 2014.