The Unrestricted Linear Fractional Assignment Problem

by

Jingxin Guo

B.S., Western Washington University, 2015

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Applied Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Spring 2018

Approval

Name:	**Jingxin Guo**
Degree:	**Master of Science (Mathematics)**
Title:	***The Unrestricted Linear Fractional Assignment Problem***
Examining Committee:	**Chair:** Natalia Kouzniak Senior Lecturer

Abraham Punnen
Senior Supervisor
Professor

George Zhang
Supervisor
Professor
Beedie School of Business

Binay Bhattacharya
Internal Examiner
Professor
Department of Computing Science

Date Defended: April 20, 2018

Abstract

The linear fractional assignment problem (LFAP) is a well-studied combinatorial optimization problem with various applications. It attempts to minimize ratio of two linear functions subject to standard assignment constraints. When the denominator of the objective function is positive, LFAP is solvable in polynomial time. However, when the denominator of the objective function is allowed to take positive and negative values, LFAP is known to be NP-hard. In this thesis, we present two 0-1 programming formulations of LFAP and compare their relative merits using experimental analysis. We also show that LFAP can be solved as a polynomialy bounded sequence of constrained assignment problems. Experimental results using this methodology are also given. Finally, some local search heuristics are developed and analyzed their efficiency using systematic experimental analysis. Our algorithms are able to solve reasonably large size problems within acceptable computational time.

Keywords: Assignment Problem, Fractional Programming, Combinatorial Optimization, Local Search.

Dedication

I dedicate this work to my beloved family.

Acknowledgements

First, I would like to express my sincere gratitude to my senior supervisors Professor Punnen, I cannot imagine finishing the thesis without the support from him. During the past two years, he became my best friend and a role model of mine. He helped me not only academically, he also taught me to become a better person. I can never ask for a better supervisor.

I also want to thank my supervisor Professor Zhang. He is the reason I am here today. He introduced me to the Operations research program at SFU. Also, I would like to thank Ms. Kouzniak for her support and insightful advice. In addition, I would like to thank Professor Bhattacharya to be my internal examiner.

I am also deelp grateful to Professor Kropinski and Professor Mishna. I would like to thank them for supporting me when I was at the most difficult time in graduate school. Without their support, I may not be able to finish the program.

Besides, I want to thank Dr. Wu. He spent a lot of his time teaching me computer science. I am thankful to all his help and patience.

In addition, I would like to thank John Hayfron, Ellen Harris, Michelle Spencer and Pooja Pandey for giving me feedbacks to my thesis.

I also want to thank Xiaorui Li, Jiaojiao Yang, Shanwen Yan and Rimi Sharma for their support. As well as Weixing Cui, Arthur and Joe Ngai for giving me helpful advice.

Last, I would like to thank my family for supporting me in every possible way. I could never ask for a better family.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Linear-fractional Programming

A *Linear fractional programming* (LFP) problem is to

$$\begin{aligned} \text{Minimize} \quad & \frac{\mathbf{c^T x} + \alpha}{\mathbf{d^T x} + \beta} \\ \text{Subject to} \quad & \mathbf{A x} \geq b, \qquad (1.1) \\ & \mathbf{x} \geq 0, \qquad (1.2) \end{aligned}$$

where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{x}$, $\mathbf{b}$ is a column vector of size m, $\mathbf{x}$, $\mathbf{c}$ and $\mathbf{d}$ are column vectors of size n, and α and β are scalars. If $\mathbf{d}$ is the zero vector of size n, then the LFP reduces to the well known linear programming problem [27]. LFP has been studied by many authors and we refer to the book [2] for details. LFP can be used in modeling many real life problems such as those in the financial industry [16] and production planning [23], among others. Many authors have proposed various algorithms to solve the model [1, 9, 20, 24, 32, 41, 42].

The objective function of a LFP have properties very similar to that of a linear program. When $\mathbf{d^T x} + \beta > 0$ for all feasible solution $\mathbf{x}$, if the LFP has an optimal solution, then there exists an optimal solution that corresponds to an extreme point of the polytope defined by (1.1) and (1.2). Further, a local optimum on this polytope is also a global optimum. To validate this, let us now consider some properties of LFP.

A function f is said to be *quasiconvex on domain* X if for all $\mathbf{x^1}, \mathbf{x^2} \in X$ and for all $t \in [0, 1]$

$$f\left[t\mathbf{x^1} + (1-t)\mathbf{x^2}\right] \leq \max\left[f(\mathbf{x^1}), f(\mathbf{x^2})\right]. \qquad (1.3)$$

A function f is *explictly quasiconvex on* X if for all $\mathbf{x^1}, \mathbf{x^2} \in X$ with $f(\mathbf{x^1}) \neq f(\mathbf{x^2})$ and for all $t \in (0,1)$ the inequality (1.3) is strict.

For instance, $f(x) = x^3$ for all $x \in \mathbb{R}$ is quasiconvex as well as explicitly quasiconvex.

A function f is said to be *quasiconcave on* X if for all $\mathbf{x^1}, \mathbf{x^2} \in X$ and for all $t \in [0,1]$

$$f[t\mathbf{x^1} + (1-t)\mathbf{x^2}] \geq min[f(\mathbf{x^1}), f(\mathbf{x^2})] \tag{1.4}$$

A function f is *explicitly quasiconcave on* X if for all $\mathbf{x^1}, \mathbf{x^2} \in X$ with $f(\mathbf{x^1}) \neq f(\mathbf{x^2})$ and for all $t \in (0,1)$ the inequality (1.4) is strict.

For instance, $f(x) = x^{-1}$ for $\{x \in \mathbb{R} | x > 0\}$ is quasiconave as well as explicitly quasiconcave.

Theorem 1.1. *[31] The objective function* $f(x) = \dfrac{\mathbf{c^T x} + \alpha}{\mathbf{d^T x} + \beta}$ *of a LFP problem is both quasiconvex and quasiconcave.*

Proof. To prove that $f(x) = \dfrac{\mathbf{c^T x} + \alpha}{\mathbf{d^T x} + \beta}$ is both quasiconvex and quasiconcave, by definition, we need to prove that:

$$\min[f(\mathbf{x^1}), f(\mathbf{x^2})] \leq f[t\mathbf{x^1} + (1-t)\mathbf{x^2}] \leq \max[f(\mathbf{x^1}), f(\mathbf{x^2})] \tag{1.5}$$

Choose $\mathbf{x^1}, \mathbf{x^2} \in \mathbf{x}$. For all $x^1, x^2 \in x$, where x is the domain of f, designate :

$$\begin{aligned} &\mathbf{x^0} = \alpha\mathbf{x^1} + (1-\alpha)\mathbf{x^2}, \text{ where } 0 < \alpha < 1 \\ &C_i = \mathbf{c^T x^i} + \alpha, \\ &D_i = \mathbf{d^T x^i} + \beta, i = 0, 1, 2. \end{aligned} \tag{1.6}$$

Then,
(1.5) becomes:

$$\min[f(\mathbf{x^1}), f(\mathbf{x^2})] \leq f(\mathbf{x^0}) \leq \max[f(\mathbf{x^1}), f(\mathbf{x^2})] \tag{1.7}$$

which is equivalent to:

$$\min\left[\frac{C_1}{D_1}, \frac{C_2}{D_2}\right] \leq \frac{C_0}{D_0} \leq \max\left[\frac{C_1}{D_1}, \frac{C_2}{D_2}\right] \tag{1.8}$$

Suppose that:

$$\frac{C_1}{D_1} \geq \frac{C_2}{D_2} \tag{1.9}$$

Then (1.8) is equivalent to:

$$\frac{C_2}{D_2} \leq \frac{C_0}{D_0} \leq \frac{C_1}{D_1} \tag{1.10}$$

which is equivalent to:

$$C_1 D_2 - C_2 D_1 \geq 0 \tag{1.11}$$

for $D_1 D_2 > 0$.

Multiply both sides of (1.11) by $(1-\gamma)$ where $\gamma \in (0,1)$, add $\gamma C_1 D_1$,and then subtract $\gamma C_1 D_1$ we get the inequality:

$$C_1[\gamma D_1 + (1-\gamma)D_2] - D_1[\gamma C_1 + (1-\gamma)C_2] \geq 0 \tag{1.12}$$

Because of (1.6), (1.12) is equivalent to:

$$C_1 D_0 - D_1 C_0 \geq 0 \tag{1.13}$$

Suppose $\frac{C_1}{D_1} \geq \frac{C_0}{D_0}$ and multiply both sides by $D_0 D_1 > 0$, the inequality still holds since $D_0 D_1 > 0$ and we get (1.13).

Thus, (1.13) have proven for $D_0 D_1 > 0$, the left hand side inequality of (1.10) is valid. We can prove the right hand side of the inequality (1.10) in a similar way. □

The above proof is taken from [31] and presented here for completion.

Theorem 1.2. *[44] If $f : X \subseteq R^n \to R$ is explicitly quasiconvex on the convex set X, then any local minimum of the function f is also a global minimum of f on X.*

Theorem 1.3. *[32] When $\mathbf{d^T x} + \beta > 0$ for all feasible $\mathbf{x}$, an optimal solution in the LFP is attained at an extreme point of the polyhedral set defining the feasible region and a local minimum is global minimum.*

1.2 LFP as a linear program

When $\mathbf{d^T x} + \beta > 0$, the LFP can be formulated as a *linear program* (LP) as shown by Charnes and Cooper [9]. We discuss this transformation below.
Consider the LFP:

$$\begin{aligned} \text{Minimize} \quad & \frac{\mathbf{c^T x} + \alpha}{\mathbf{d^T x} + \beta} \\ \text{Subject to} \quad & \mathbf{A x} \geq b, \\ & \mathbf{x} \geq 0 \end{aligned}$$

where $\mathbf{d^T x} + \beta > 0$. Let $1/t = \mathbf{d^T x} + \beta$. Then, LFP can be written as:

$$\begin{aligned} \text{Minimize} \quad & t(\mathbf{c^T x} + \alpha) \\ \text{Subject to} \quad & \mathbf{A x} \geq b, \\ & t(\mathbf{d^T x} + \beta) = 1, \\ & \mathbf{x} \geq 0 \end{aligned}$$

which is equivalent to:

$$\begin{aligned} \text{Minimize} \quad & \mathbf{c^T x} t + \alpha t \\ \text{Subject to} \quad & \mathbf{A x} \geq b, \\ & t\mathbf{d^T x} + t\beta = 1, \\ & \mathbf{x} \geq 0, \end{aligned}$$

Let $\mathbf{y} = t\mathbf{x}$, then the problem becomes:

$$\begin{aligned} \text{Minimize} \quad & \mathbf{c^T y} + \alpha t \\ \text{Subject to} \quad & \mathbf{A}\left(\frac{\mathbf{y}}{t}\right) \geq b, \\ & \mathbf{d^T y} + t\beta = 1, \\ & \mathbf{y} \geq 0, t \geq 0 \end{aligned}$$

Which is equivalent to:

$$\begin{aligned} \text{Minimize} \quad & \mathbf{c^T y} + \alpha t \\ \text{Subject to} \quad & \mathbf{Ay} - bt \geq 0, \\ & \mathbf{d^T y} + t\beta = 1, \\ & \mathbf{y} \geq 0, t \geq 0 \end{aligned}$$

The above problem is an LP.

In the above reduction, we assumed that $\mathbf{d^T x} + \beta > 0$. This can be relaxed as follows. Obviously, if $\mathbf{d^T x} + \beta = 0$, for some feasible $\mathbf{x}$, the problem is not well defined. For this reason, we will assume that either $\mathbf{d^T x} + \beta \leq -\epsilon$ or $\mathbf{d^T x} + \beta \geq \epsilon$ for some small $\epsilon > 0$.
Consider the linear fractional programs:

$$\begin{aligned} LFP1: \ \text{Minimize} \quad & \frac{\mathbf{c^T x} + \alpha}{\mathbf{d^T x} + \beta} \\ \text{Subject to} \quad & \mathbf{Ax} \geq b, \\ & \mathbf{d^T x} + \beta \geq \epsilon, \\ & \mathbf{x} \geq 0 \end{aligned}$$

and

$$\begin{aligned} LFP2: \ \text{Minimize} \quad & \frac{\mathbf{c^T x} + \alpha}{\mathbf{d^T x} + \beta} \\ \text{Subject to} \quad & \mathbf{Ax} \geq b, \\ & \mathbf{d^T x} + \beta \leq -\epsilon, \\ & \mathbf{x} \geq 0 \end{aligned}$$

An optimal solution to LFP is the best solution amongest those of LFP1 and LFP2. But for LFP2, $\mathbf{d^T x} + \beta$ is negative.
LFP2 can be reformulated as:

$$\begin{aligned} LFP3: \ \text{Minimize} \quad & \frac{-\mathbf{c^T x} - \alpha}{-\mathbf{d^T x} - \beta} \\ \text{Subject to} \quad & \mathbf{Ax} \geq b, \\ & \mathbf{d^T x} + \beta \leq -\epsilon, \\ & \mathbf{x} \geq 0 \end{aligned}$$

Note that LFP3 is equivalent to LFP2 and $-\mathbf{d^T x} - \beta > 0$. Thus, we can reformulate LFP1 and LFP3 as linear programs as shown earlier. The best solution produced amongst of those of LFP1 and LFP3 gives an optimal solution to LFP; subject to the tolerance limit ϵ. Thus, without loss of generality, we assume $\mathbf{d^T x} + \beta > 0$ for all x. Based on the reduction discussed above, LFP can be solved by solving at most two linear programs. Let us now discuss another algorithm that can be used to solve LFP.
Consider the parametric linear problem:

$$\begin{aligned} LFP(\lambda): \ F(\lambda) = \min_{x \in S} \{(\mathbf{c^T x} + \alpha) - \lambda(\mathbf{d^T x} + \beta)\} \\ \text{Subject to} \quad \mathbf{A x} \geq b, \\ \mathbf{x} \geq 0. \end{aligned}$$

For convenience, we denote $C(x) = \mathbf{c^T x} + \alpha$ and $D(x) = \mathbf{d^T x} + \beta$.

Theorem 1.4. *[2] Let S denotes the set of feasible solutions of LFP and $D(x) > 0$ for all $x \in S$. Then, a feasible solution x^* is an optimal solution to LFP if and only if*

$$F(\lambda^*) = \min_{x \in S} \{C(x) - \lambda^* D(x)\} = 0$$
$$\text{where } \lambda^* = \frac{C(x^*)}{D(x^*)}$$

Proof. Suppose x^* is an optimal solution to LFP. Then,

$$\lambda^* = \frac{C(x^*)}{D(x^*)} \leq \frac{C(x)}{D(x)}, \forall \ x \in S$$

Since $D(x) > 0$, for all x, we have:

$$C(x) - \lambda^* D(x) \geq 0, \forall \ x \in S$$

thus,

$$\min_{x \in S} \{C(x) - \lambda^* D(x)\} = 0, \lambda \in R$$

Conversely, assume $F(\lambda^*) = 0$, we have:

$$C(x) - \lambda^* D(x) \geq C(x^*) - \lambda^* D(x^*) = 0, \forall \ x \in S$$

This means that x^* as optimal solution to LFP. □

Based on the Theorem 1.4, we can develop an algorithm for computing an optimal solution to LFP assuming $D(x) > 0 \ \forall \ x \in S$.
Since $D(x) > 0 \ \forall x \in S$, we have:

$$\frac{\partial F(\lambda)}{\partial \lambda} = -D(x) < 0$$

which means that $F(\lambda)$ is strictly decreasing in λ.

Based on Theorem 1.4, we have the following algorithm to solve LFP.

Algorithm: Newton-LFP

Step 1: Choose a starting solution $x^0 \in S$, compute $\lambda^1 = \dfrac{C(x^0)}{D(x^0)}$, and let $k = 1$;

Step 2: Determine x^k where $\min\limits_{x \in S}\{C(x) - \lambda^k D(x)\} = 0$;

Step 3: If $F(\lambda^k) = 0$ then $x^* = x^k$ is the optimal solution; Stop;

Step 4: Let $\lambda^{k+1} = \dfrac{C(x^k)}{D(x^k)}$; Let $k = k + 1$; Go to Step 1

The above algorithm is called Dinkelbach's algorithm which was proposed by Dinkelbach in 1967 [12]. One can see that it is equivalent to Newton's method for finding the roots of the equation $F(\lambda) = 0$ and hence it is also called Newton's method.

The above proof is taken from [2] and presented here for completion.

Other approaches to solve LFP include simplex method [11] [32], interior point method [35] and binary search method [43].

1.3 Combinatorial Fractional Programming

Let $E = \{1, 2, ..., m\}$ be a finite set and F be a family of feasible solutions. Note that $F \subseteq 2^E$. For each $e \in E$, a cost c_e is defined. Then, the *Linear Combinatorial Optimization Problem* (LCOP) is to

$$\begin{aligned} \text{Minimize} \quad & \sum_{e \in S} c_e \\ \text{Subject to} \quad & S \in F \end{aligned}$$

Further, let d_e for $e \in E$ be another cost defined for elements of E and α and β are real numbers. Then, a *Fractional Combinatorial Optimization Problem* (FCOP) is to

$$\begin{aligned} \text{Minimize} \quad & \frac{\alpha + \sum_{e \in S} c_e}{\beta + \sum_{e \in S} d_e} \\ \text{Subject to} \quad & S \in F \end{aligned}$$

We assume that $\beta + \sum_{e \in S} d_e \neq 0$ for any $S \in F$. An application of the FCOP is to minimize cost-to-time ratio. The problem of the minimum cost-to-time ratio cycle in a graph had been introduced by Dantzig, Blattner and Rao [40] and Lawler [30].

When E is the edge set of a graph $G = (E, V)$ and F is the collection of all spanning trees of G, the resulting FCOP is the *minimum ratio spanning tree problem* [8]. Chandrasekaran [8] stuided the minimum ratio spanning tree problem and proposed an algorithm that computes a minimum ratio spanning tree in time $O(|E|^2 \log |V|)$ time when $\beta + \sum_{e \in S} d_e > 0$. Similarly, when E is the edge set of a balanced complete bipartite graph G and F is the family of all perfect matchings in G, then FCOP becomes the fractional assignment problem [43]. We will talk about the fractional assignment problem in detail later.

Meggido [34] studied FCOP with the restriction that $\beta + \sum_{e \in S} d_e > 0$ for all $S \in F$. An FCOP with the restriction that $\beta + \sum_{e \in S} d_e > 0$ is denoted by $FCOP^+$. He proposed a general algorithm to handle $FCOP^+$. He proved that

Theorem 1.5. *[34] If LCOP is solvable with $O(p(m))$ comparisons and $O(q(m))$ additions, then $FCOP^+$ is solvable in $O(p(m)(q(m) + p(m)))$ time.*

The above result improved the complexity of solving minimum ratio spanning tree problem to $O(|E| \log^2 |V| \log \log |V|)$ and the complexity of solving minimum cycle problem to $O(|E||V|^2 \log |V|)$.

Ealier we mentioned that Newton's method can be used for solving LFP. The same idea can also be used to solve $FCOP^+$. Matsui, Saruwatari and Shigeno [33] studied the method and showed it terminates in a polynomial number of iterations. Here, we denote $\overline{C} = max\{\max_{(i=1,2,\ldots,m)} |c_i|, 1\}$ and $\overline{D} = max\{\max_{(i=1,2,\ldots,m)} |d_i|, 1\}$.

Theorem 1.6. *[33] The number of iterations of Newton's method applied to $FCOP^+$ is $O(\log(m\overline{CD}) \leq O(\log(mM))$ in the worst case when $M = max\{\overline{C}, \overline{D}\}$.*

This Theorem gives a complexity bound for the minumum ratio spanning tree problem of $O(T(n, m) \log(n\overline{CD}))$ time, where n is the number of nodes, m is the number of edges of the given graph and $T(n, m)$ represent the complexity of a stronlgy polynomial time algorithm for the ordinary minimum spanning tree problem. The Theorem also shows that the worst case time bound for the fractional assignment problem on a bipartite graph with $2n$ nodes and m edges is $O(\sqrt{n}m(\log(2n^3\overline{CD}))(\log(m\overline{CD}))) \leq O(\sqrt{n}m(\log(n\overline{CD}))^2)$ when Newton's method is applied.

Radzik [39] studied $FCOP^+$ and showed that the Newton's method, when adapted to solve $FCOP^+$, terminates in a polynomial number of iterations. More specifically, he proved that

Theorem 1.7. *[39] If in an FCOP problem, the coordinates of vectors* $\mathbf{c}$ *and* $\mathbf{d}$ *are integers from* $[-U, U]$*, then Newton's method finds the optimum solution in* $FCOP^+$ *in* $O(\log(mU))$ *iterations.*

This result is similar to that obatained by Matsui, Saruwatari and Shigeno [33] discussed earlier. Further, Radzik showed that

Theorem 1.8. *Newton's method finds an optimal solution to* $FCOP^+$ *in* $O(m^4 \log^2 m)$ *iterations.*

Note the above bound on the number of iterations is stongly polynomial.

Any $S \in F$ can be represented by its incidence vector $\mathbf{x} = \{x_1, x_2, ..., x_m\}$ which is a 0-1 vector with $x_e = 1$ if and only if $e \in S$. Thus, FCOP can be represented as a 0-1 integer program

$$\begin{aligned} \text{Minimize} \quad & \frac{\alpha + \sum_{i=1}^{m} \mathbf{c_j x_j}}{\beta + \sum_{j=1}^{m} \mathbf{d_j x_j}} \\ \text{Subject to} \quad & x \in \hat{F} \end{aligned}$$

where $\hat{F}$ is the collection of all incidence vertors corresponding to elements of F. Sometimes, $\hat{F}$ will have an implicit compact representation. For example, the family of all perfect matchings on a complete balanced bipartite graph $G = (V_1, V_2, E)$ can be written in terms of the incidence vectors as:

$$\begin{aligned} &\sum_{j \in V_1} x_{ij} = 1, i \in V_2, \\ &\sum_{i \in V_2} x_{ij} = 1, j \in V_1, \\ &x_{ij} \in \{0, 1\} \text{ and } (i, j) \in E \end{aligned}$$

Let us now discuss linear fractional programs with integer variables which encompass FCOP, when F has a compact representation in terms of its incidence vectors.

1.4 Integer Fractional Programs

In the definition of LFP, if we put the additional restriction that the variables can take only integer values, we get an *Integer Linear Fractional programming* (ILFP).
An ILFP can be represented as follow:

$$\begin{aligned} \text{Minimize} \quad & \frac{\mathbf{c^T x} + \alpha}{\mathbf{d^T x} + \beta} \\ \text{Subject to} \quad & \mathbf{A x} \geq b, \\ & \mathbf{x} \geq 0, \mathbf{x} \in \mathbb{I}, \end{aligned}$$

where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{b}$ is an m column vector, $\mathbf{c}$, $\mathbf{d}$ and $\mathbf{x}$ are column vectors of size n and α and β are scalars, and $\mathbb{I}$ is the set of all integers.

Using the Charnes and Cooper transformation [9], the ILFP can be represented as

$$\begin{aligned} ILFP1: \ \text{Minimize} \quad & \mathbf{c^T y} + \alpha t \\ \text{Subject to} \quad & \mathbf{A y} - bt \geq 0, \\ & \mathbf{d^T y} + t\beta = 1, \\ & \mathbf{y} \geq 0, t \geq 0, \left(\frac{\mathbf{y}}{t}\right) \in \mathbb{I}, \end{aligned}$$

Note that the constraint $\left(\frac{\mathbf{y}}{t}\right) \in \mathbb{I}$ makes ILFP1 hard to solve. Moreover, such a constraint cannot be used directly within general purpose mixed integer linear programming solvers. For this reason, we have to reformulate it into a different form to be able to use a general purpose solver.

Algorithms for solving ILFP had been proposed by many authors. Robillard [41], Grunspan and Thomas [20] and Anzai [1] developed their algorithms for solving integer fractional programs by using an algorithm proposed by Martos and Isbell-Marlow [24, 32] for LFP. Grunspan and Thomas described a general algorithm for solving the integer linear program by reducing it to a sequence of linear integer problems in 1973 [20]. Granot and Granot [18] developed a cutting plane algorithm to solve both integer fractional programming (IFP) and the mixed integer fractional programming by using Charnes and Cooper's approach for solving associated continuous fractional programs [9].

As noted earlier, when the integer variables are restricted to take values from $\{0, 1\}$, the resulting ILFP is called the *Binary Fractional Linear Programming problem* (BFLP).

BFLP can be formulated as follow:

$$\begin{aligned} \text{Minimize} \quad & \frac{\mathbf{c^T x} + \alpha}{\mathbf{d^T x} + \beta} \\ \text{Subject to} \quad & \mathbf{A x} \geq b, \\ & \mathbf{x} \in \{0, 1\} \end{aligned}$$

Following the transformation of Charnes and Cooper [9] with approporiate modification, we get the following formulation for BFLP:

$$\begin{aligned} BFLP1: \ \text{Minimize} \quad & \mathbf{c^T y} + \alpha t \\ \text{Subject to} \quad & \mathbf{A y} - bt \geq 0, \\ & \mathbf{d^T y} + t\beta = 1, \\ & \mathbf{y} \in \{0, t\}, t \geq 0 \end{aligned} \tag{1.14}$$

Constraint (1.14) enforces y takes values from the discrete set $\{0, t\}$. Such a constraint cannot be used directly for general purpose solvers such as CPLEX or Gurobi. But, it can be handled as follows. We let $y = ty'$ and have the equivalent formulation:

$$\begin{aligned} BFLP2: \ \text{Minimize} \quad & \mathbf{c^T} t\mathbf{y'} + \alpha t \\ \text{Subject to} \quad & \mathbf{A} t\mathbf{y'} - bt \geq 0, \\ & \mathbf{d^T} t\mathbf{y'} + t\beta = 1, \\ & \mathbf{y'} \in \{0, 1\}, t \geq 0 \end{aligned} \tag{1.15}$$

Note that t is a decision variable. Replacing $y't$ with z and adding the McCormick envelope inequalities [17], BFLP2 reduces to

$$\begin{aligned} BFLP3: \ \text{Minimize} \quad & \mathbf{c^T} z + \alpha t \\ \text{Subject to} \quad & \mathbf{A} z - bt \geq 0, \\ & \mathbf{d^T} z + t\beta = 1, \\ & z \geq \mathbf{y'} t, && (1.16) \\ & z \geq t + \mathbf{y'}\bar{t} - \bar{t}, && (1.17) \\ & z \leq t + \mathbf{y'}\underline{t} - \underline{t}, && (1.18) \\ & z \leq \mathbf{y'}\bar{t}, && (1.19) \\ & \mathbf{y'} \in \{0, 1\}, t \geq 0 \\ & \underline{t} \leq t \leq \bar{t} \end{aligned}$$

where $\bar{t}$ and $\underline{t}$ respectively represent lower and upper bounds on t over the set of feasible solutions.

Constraints (1.16) - (1.19) together make sure $z = y't$. Note that $y' \in \{0, 1\}$. When $y' = 0$, constraint (1.17) and (1.18) is equivalent to $t - \bar{t} \leq z \leq t - \underline{t}$. When $y' = 1$, constraint (1.17) and (1.18) is equivalent to $z = t$.

Since all the constraints in BFLP3 can be forced using a MIP solver, it can be solved by a solver theoretically. However, since MIP is a NP-hard problem [14], it cannot be solved efficiently as the problem size gets bigger. Using exact algorithms to solve the problem is usually hard and inefficient, so we often use heuristic algorithms to solve MIP problems.

In 1968, Hammer and Reudeanu [41] proposed an algorithm to solve the Constrained (0, 1) Fractional Programming (CFP) problem [21]. This method was refined by Robillard in 1971. Granot and Granot proposed an Implicit Enumeration method for solving CFP [19] by generating a stronger surrogate constrains for CFP. The algorithm was based on Balas' additive algorithm [3], and Geoffrion's backtracking procedure [15]. Furthermore, Beale showed that method can be used to solve problems using zero-one decision variables, by using a code which allows special ordered sets of along type as defined by Beale and Tomlin [4] without requiring any new algorithms. Beale's method is to group the zero-one variables into multiple-choice set x_{jk} such that $\sum_k x_{jk} = 1, \forall j$. Since the independent zero-one variable can be represented as a set of a set of two element. One is the original zero-one varibale and another one is a slack variable which has no entires in other rows. Thus, we do not require additional constraint when involving zero-one variables.

The assignment problem is one of the fundamental combinatorial optimization problems and it is studied extensively in literature [7].

1.5 Linear Fractional Assignment problem

The *linear fractional assignment problem* (LFAP) is variation of the well known assignment problem. Let $C = (c_{ij})$ and $D = (d_{ij})$ be $n \times n$ matrices. Then LFAP can be defined as:

$$LFAP: \text{Minimize} \quad \frac{\mathbf{C}(\mathbf{x}) + \alpha}{\mathbf{D}(\mathbf{x}) + \beta}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \tag{1.20}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n \tag{1.21}$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j \tag{1.22}$$

We use the notation

$$\mathbf{C}(\mathbf{x}) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}, \text{ and } \quad \mathbf{D}(\mathbf{x}) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij},$$

and represent the set of all $n \times n$ 0-1 matrices satisfying (1.20) and (1.21) by F. Without loss of generality we assume $\alpha = \beta = 0$. For otherwise, define $c'_{ij} = c_{ij} + \frac{\alpha}{n}$
Then

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n} c'_{ij}x_{ij} =& \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} + \frac{\alpha}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} \\ =& \mathbf{C}(\mathbf{x}) + \alpha\end{aligned}$$

Likewise, define $d'_{ij} = d_{ij} + \frac{\beta}{n}$. Then

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d'_{ij}x_{ij} = \mathbf{D}(\mathbf{x}) + \beta$$

When $\mathbf{D}(\mathbf{x}) > 0$ for all $x \in F$ the LFAP can be solved polynomial time [26, 34] and if $D(x)$ is arbitrary, then LFAP is NP-hard [26].

When $\mathbf{D}(\mathbf{x}) + \beta > 0$ then the constraints $x_{ij} \in \{0, 1\}$ can be replaced by $x_{ij} \geq 0$. In this case, LFAP reduces to LFP. Thus, this version of LFAP is a special case of LFP. Also, as noted earlier, LFAP is a special case of FCOP. Meggiddo showed that the LFAP can be solved in $O(min\{n^2 \log n + nm^2, nm^2 \log^2(nC_{max}D_{max})\})$ [34] when $D(x) > 0 \; \forall \; x \in F$. Radzik established an algorithm which has complexity $O((n^2 logn + nm)n^4 log^2 n)$ for the same problem. Thus, both Meggiddo's algorithm and Radzik's algorithm are strongly polynomial. Another method specifically designed for LFAP was proposed by Shigone, Saruwatari and Matsui [43] in 1995 for $D(x) > 0 \; \forall \; x \in F$. As in the case of Radzik's algorithm, this method is based on Newton's method and has complexity of $O(\sqrt{n}m \log(nC_{max}D_{max}))$, where $C_{max} = max\{|c_{ij}| : (i, j) \in E\}$ and $D_{max} = max\{|c_{ij}| : (i, j) \in E\}$. In 2008, Kabadi and Punnen [26] proposed a simplex scheme for the LFAP with complexity of $O(min\{n^3 m \log^2(nC_{max}D_{max})\})$.

To the best of our knowledge, all published works in the literature assumes that $\mathbf{D}(\mathbf{x}) > 0$. In this thesis, we consider the LFAP without any sign restriction on $\mathbf{C}(\mathbf{x})$ or $\mathbf{D}(\mathbf{x})$. One can use the same transformations I used before. However, we are giving better formulations, in term of number of additional constraints and variables.

Chapter 2

LFAP formulations

2.1 Mixed 0-1 linear programming formulations

Let us now discuss mixed 0-1 linear programming formulations of LFAP. First note that if $D(x) = 0$ then the objective function of LFAP denoted $f(x) = \frac{C(x)}{D(x)}$ is undefined. However, testing if such a solution exists is difficult, as shown by the theorem below.

Theorem 2.1. *Verifying if there exists an $x \in F$ such that $D(x) = 0$ is NP- hard.*

Proof. The proof uses a reduction from the *partition problem.* The partition problem is: "Given numbers $a_1, a_2, ..., a_n$ does there exist a subset $S \subseteq \{1, 2, ..., n\}$ such that $\sum_{j \in S} a_j = \sum_{j \in \bar{S}} a_j$?" where $\bar{S} = \{1, ..., n\} - S$
From an instance of partition, we construct an instance of the assignment problem as follows. Construct a bipartite graph $G = (V_1, V_2, E)$ where $V_1 = \{1, 2, ..., 2n\}$, $V_2 = \{1', 2', ..., 2n'\}$.

Define

$$d_{ij} = \begin{cases} a_i & \text{if } j = i', \text{ and } i \leq n \\ -a_i & \text{if } j = n + i', \text{and } i \leq n \\ M & \text{a large number if } i \leq n \text{ and not of the above type} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Suppose there is a partition S, $\bar{S}$, such that $\sum_{i \in S} a_i = \sum_{i \in \bar{S}} a_i$. Then we need to construct an assignment x with $D(x) = 0$, for the D matrix defined in equation 2.1. For $i \in S$, match i with i'. For $i \in \bar{S}$, match i with $n + i'$. Extend this to a perfect matching using edge of value zero. Let x be the resulting assignment. It is easy to see that $D(x) = 0$.

Conversely, assume $D(x) = 0$. Then for $i \leq n$, if i is assigned to i', put i in S and if i is assigned to $n + i'$, put i in $\bar{S}$. It can be verified that $\sum_{i \in S} a_i = \sum_{i \in \bar{S}} a_i$

□

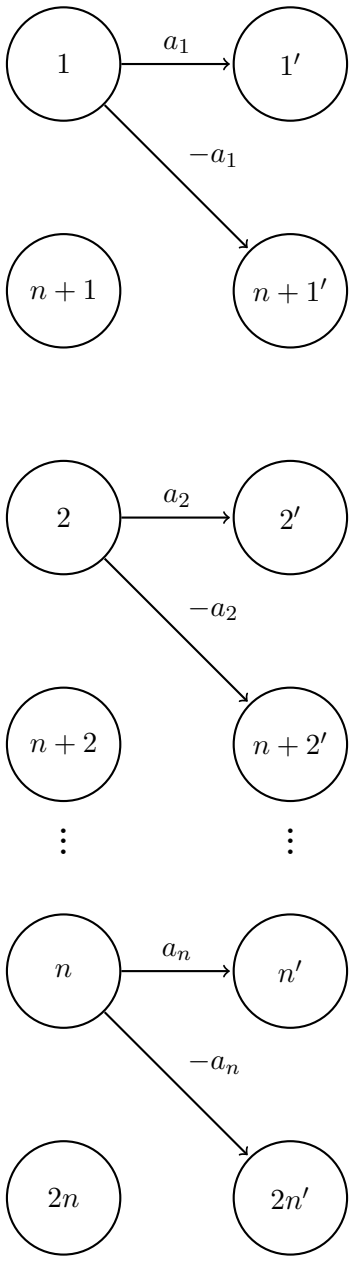


Figure 2.1: Instance of the partition problem described in Theorem 2.1

Let $F^+ = \{x \in F : D(x) > 0\}$ and $F^- = \{x \in F : D(x) < 0\}$. Denote by $LFAP^+$ the variation of LFAP where the family of feasible solutions is restricted to F^+ and $LFAP^-$ the variation of LFAP where the family of feasible solutions is restricted to F^-. LFAP is solvable in (strongly) polynomial time [38, 43] if $D(x) > 0$ for all $x \in F$. However, this simplicity doesn't translate to $LFAP^+$ and $LFAP^-$.

Theorem 2.2. *Both $LFAP^+$ and $LFAP^-$ are NP-hard.*

Proof. The proof for $LFAP^+$ and $LFAP^-$ can be obtained using the same approach. Note that since we assume $D(x) \neq 0$ for all $x \in F$, $F = F^+ \cup F^-$. Thus the best solution amongst the optimal solutions of $LFAP^+$ and $LFAP^-$ solves LFAP. Moreover, any $LFAP^-$ can be reformulated as an $LFAP^+$ by multiplying the numerator and denominator by -1. Thus if $LFAP^+$ can be solved in polynomial time, then LFAP can be solved in polynomial time. Since LFAP is NP-hard [26], the result follows. □

Since we are not interested in solutions x with $D(x) = 0$, we need to exclude this possibility. This can be achieved by introducing the following constraints:

$$D(x) \geq \epsilon - Mz \tag{2.2}$$

$$D(x) \leq -\epsilon + M(1 - z) \tag{2.3}$$

$$z \in \{0, 1\}$$

where M is a very large number and ϵ is a small positive tolerance limit. If the entries in the matrix D are integers, ϵ can be chosen as 1.

2.2 MILP formulation

Integer programming formulations of 0-1 fractional programming problems was studied in [3, 10]. In these works, the denominator in the objective function is assumed to be positive for all feasible solutions. The basis of such formulations is the original idea of Charnes and Cooper [9] with additional refinement in handling the binary variable. Let us first consider a variation of these ideas for LFAP to generate a mixed 0-1 programming formulation.

Let $y = \dfrac{1}{D(x)}$ and $\underline{y}, \bar{y}$ respectively represent the lower and upper bounds on y over the set of feasible solutions. Note that $\underline{y}$ and $\bar{y}$ can be identified by solving at most 4 assignment problems. Then the LFAP can be formulated as a mixed 0-1 quadratic program as follows:

$$LFAP1 : \text{Minimize} \quad f_1(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}y$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \tag{2.4}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n \tag{2.5}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}y = 1 \tag{2.6}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} + Mz \geq \epsilon \tag{2.7}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} + Mz \leq M - \epsilon \tag{2.8}$$

$$x_{ij}, z \in \{0, 1\} \text{ for all } i, j \tag{2.9}$$

$$\underline{y} \leq y \leq \bar{y} \tag{2.10}$$

Constraint (2.4) is to make sure each x_i matches with exactly one x_j. At the same time, constraint (2.5) is to make srue each x_j matches with exactly one x_i. Constraint (2.7) and constraint(2.8) are same as constraint(2.2) and constraint(2.3).

We can linearize the quadratic terms in the above formulation. Note that thses quadratic terms are the product of a binary variable and a bounded continuous variables. This product can be linearized using standard transformations [17] by introducing a new variable y_{ij} to replace $x_{ij}y$ and adding the McCormick envelop inequalities

$$y_{ij} \geq x_{ij}\underline{y} \tag{2.11}$$

$$y_{ij} \geq y + x_{ij}\bar{y} - \bar{y} \tag{2.12}$$

$$y_{ij} \leq y + x_{ij}\underline{y} - \underline{y} \tag{2.13}$$

$$y_{ij} \leq x_{ij}\bar{y} \tag{2.14}$$

Note that McCormick envelop inequalities guarantee that $x_{ij}y = y_{ij}$ for all i,j. Thus, we get the mixed 0-1 linear programming formulation of LFAP as

$$LFAP2: \text{Minimize} \quad f_2(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}y_{ij}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \tag{2.15}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n \tag{2.16}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}y_{ij} = 1 \tag{2.17}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} + Mz \geq \epsilon \tag{2.18}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} + Mz \leq M - \epsilon \tag{2.19}$$

$$y_{ij} \geq x_{ij}\underline{y} \text{ for all } i, j \tag{2.20}$$

$$y_{ij} \geq y + x_{ij}\bar{y} - \bar{y} \text{ for all } i, j \tag{2.21}$$

$$y_{ij} \leq y + x_{ij}\underline{y} - \underline{y} \text{ for all } i, j \tag{2.22}$$

$$y_{ij} \leq x_{ij}\bar{y} \text{ for all } i, j \tag{2.23}$$

$$x_{ij}, z \in \{0, 1\} \text{ for all } i, j \tag{2.24}$$

$$\underline{y} \leq y \leq \bar{y} \tag{2.25}$$

This formulation contains $O(1+n^2)$ binary variables, 1 continuous variable, $2n+3+4n^2$ general constraints and n^2+1 binary restrictions. Let us now consider a different formulation with the assumption that the entries in $C(x)$ and $D(x)$ are integers. Under this assumption, $C(x)$ and $D(x)$ are also integers and hence can be written as the difference of two positive integers. Let $C(x) = u^1 - u^2$ and $D(x) = u^3 - u^4$ where $u^1, u^2, u^3, u^4 \geq 0$. Let $\bar{u}^1$ and $\bar{u}^2$ respectively be upper bounds on u^1 and u^2. Let

$$t_1 = \begin{cases} \lfloor \log_2(\bar{u}^1) \rfloor + 1 \text{ if } \bar{u}^1 > 0, \\ 0 \text{ otherwise} \end{cases} \quad \text{and} \quad t_2 = \begin{cases} \lfloor \log_2(\bar{u}^2) \rfloor + 1 \text{ if } \bar{u}^2 > 0, \\ 0 \text{ otherwise.} \end{cases}$$

We use the convention that in the summation operator, if the lower limit is higher than the upper limit, then the value of the sum is zero. With this understanding, $C(x)$ can be represented as follows:

$$C(x) = \sum_{k=1}^{t_1} 2^{k-1} u_k^1 - \sum_{\ell=1}^{t_2} 2^{\ell-1} u_\ell^2$$

where $u_k^1, k = 1, 2, \ldots, t_1$ and $u_\ell^2, \ell = 1, 2 \ldots t_2$ are binary variables. Good values of $\bar{u}^1$ and $\bar{u}^2$ can be identified as

$$\bar{u}^1 = \begin{cases} z_1 \text{ if } z_1 = \max\{C(x) : x \in F\} \text{ and } z_1 > 0, \\ 0 \text{ otherwise} \end{cases}$$

and

$$\bar{u}^2 = \begin{cases} |z_2| \text{ if } z_2 = \min\{C(x) : x \in F\} \text{ and } z_2 < 0, \\ 0 \text{ otherwise} \end{cases}$$

However, computing these values can be time consuming. We can also set $\bar{u}^1$ and $\bar{u}^2$ to approximations that are easily computable. Valid choices for $\bar{u}^1$ and $\bar{u}^2$, that can be computed in $O(n^2)$ time, are given below:

$$\bar{u}^1 = \begin{cases} z_1 \text{ if } z_1 = \min\left\{\sum_{i=1}^n \max_{1 \le j \le n}\{c_{ij}\}, \sum_{j=1}^n \max_{1 \le i \le n}\{c_{ij}\}\right\} \text{ and } z_1 > 0, \\ 0 \text{ otherwise} \end{cases}$$

and

$$\bar{u}^2 = \begin{cases} |z_2| \text{ if } z_2 = \max\left\{\sum_{i=1}^n \min_{1 \le j \le n}\{c_{ij}\}, \sum_{j=1}^n \min_{1 \le i \le n}\{c_{ij}\}\right\} \text{ and } z_2 < 0, \\ 0 \text{ otherwise} \end{cases}$$

Similarly, let $\bar{u}^3$ and $\bar{u}^4$ respectively be upper bounds on u^3 and u^4. Define

$$t_3 = \begin{cases} \lfloor \log_2(\bar{u}^3) \rfloor + 1 \text{ if } \bar{u}^3 > 0, \\ 0 \text{ otherwise} \end{cases} \quad \text{and} \quad t_4 = \begin{cases} \lfloor \log_2(\bar{u}^4) \rfloor + 1 \text{ if } \bar{u}^4 > 0, \\ 0 \text{ otherwise.} \end{cases}$$

Then $D(x)$ can be represented as

$$D(x) = \sum_{k=1}^{t_3} 2^{k-1} u_k^3 - \sum_{\ell=1}^{t_4} 2^{\ell-1} u_\ell^4$$

where $u_k^3, k = 1, 2, \ldots, t_3$ and $u_\ell^4, \ell = 1, 2 \ldots t_4$ are binary variables. As any integer can be represented as difference of two nonnegative integers. The values of $\bar{u}^3$ and $\bar{u}^4$ can be set as

$$\bar{u}^3 = \begin{cases} w_1 \text{ if } w_1 = \max\{D(x) : x \in F\} \text{ and } w_1 > 0, \\ 0 \text{ otherwise} \end{cases}$$

and

$$\bar{u}^4 = \begin{cases} |w_2| \text{ if } w_2 = \min\{D(x) : x \in F\} \text{ and } w_2 < 0, \\ 0 \text{ otherwise} \end{cases}$$

Good approximations that can be used in place of the above values are given below:

$$\bar{u}^3 = \begin{cases} w_1 \text{ if } w_1 = \min\left\{\sum_{i=1}^n \max_{1\le j\le n}\{d_{ij}\}, \sum_{j=1}^n \max_{1\le i\le n}\{d_{ij}\}\right\} \text{ and } w_1 > 0, \\ 0 \text{ otherwise} \end{cases}$$

and

$$\bar{u}^4 = \begin{cases} |w_2| \text{ if } w_2 = \max\left\{\sum_{i=1}^n \min_{1\le j\le n}\{d_{ij}\}, \sum_{j=1}^n \min_{1\le i\le n}\{d_{ij}\}\right\} \text{ and } w_2 < 0, \\ 0 \text{ otherwise} \end{cases}$$

Using the notations introduced above, we give another mixed 0-1 quadratic programming formulation for LFAP as follows:

$$LFAP3: \text{Minimize} \quad f_3(x) = \sum_{k=1}^{t_1} 2^{k-1} u_k^1 y - \sum_{\ell=1}^{t_2} 2^{\ell-1} u_\ell^2 y$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \tag{2.26}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n \tag{2.27}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} x_{ij} = \sum_{k=1}^{t_3} 2^{k-1} u_k^3 - \sum_{\ell=1}^{t_4} 2^{\ell-1} u_\ell^4 \tag{2.28}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} = \sum_{k=1}^{t_1} 2^{k-1} u_k^1 - \sum_{\ell=1}^{t_2} 2^{\ell-1} u_\ell^2 \tag{2.29}$$

$$\sum_{k=1}^{t_3} 2^{k-1} u_k^3 y - \sum_{\ell=1}^{t_4} 2^{\ell-1} u_\ell^4 y = 1 \tag{2.30}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} x_{ij} + Mz \geq \epsilon \tag{2.31}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} x_{ij} + Mz \leq M - \epsilon \tag{2.32}$$

$$z, x_{ij}, u_k^1, u_\ell^2, u_p^3, u_h^4 \in \{0, 1\} \text{ for all } i, j, k, \ell, p, h \tag{2.33}$$

$$\underline{y} \leq y \leq \bar{y} \tag{2.34}$$

The non-linear terms in the objective function and constraints involve product of binary variables with the continuous variable y. Thus we can linearize these terms using McCormick envelop inequalities. Introducing the variables $z_k^i = u_k^i y, k = 1, 2, \ldots, t_i, i = 1, 2, 3, 4$ and adding the McCormick envelop inequalities to force these definitions, we get our next 0-1 integer linear programming formulation of LFAP as:

$$LFAP4: \text{Minimize } f_4(x) = \sum_{k=1}^{t_1} 2^{k-1} z_k^1 - \sum_{\ell=1}^{t_2} 2^{\ell-1} z_\ell^2$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \tag{2.35}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n \tag{2.36}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} = \sum_{k=1}^{t_1} 2^{k-1} u_k^1 - \sum_{\ell=1}^{t_2} 2^{\ell-1} u_\ell^2 \tag{2.37}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} x_{ij} = \sum_{k=1}^{t_3} 2^{k-1} u_k^3 - \sum_{\ell=1}^{t_4} 2^{\ell-1} u_\ell^4 \tag{2.38}$$

$$\sum_{p=1}^{t_3} 2^{p-1} z_p^3 - \sum_{l=1}^{t_4} 2^{l-1} z_h^4 = 1 \tag{2.39}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} x_{ij} + Mz \geq 1 \tag{2.40}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} x_{ij} + Mz \leq M - 1 \tag{2.41}$$

$$z_k^i \geq u_k^i \underline{y} \text{ for all } k = 1, 2, \ldots, t_i, i = 1, 2, 3, 4 \tag{2.42}$$

$$z_k^i \geq y + u_k^i \bar{y} - \bar{y} \text{ for all } k = 1, 2, \ldots, t_i, i = 1, 2, 3, 4 \tag{2.43}$$

$$z_k^i \leq y + u_k^i \underline{y} - \underline{y} \text{ for all } k = 1, 2, \ldots, t_i, i = 1, 2, 3, 4 \tag{2.44}$$

$$z_k^i \leq u_k^i \bar{y} \text{ for all } k = 1, 2, \ldots, t_i, i = 1, 2, 3, 4 \tag{2.45}$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j = 1, 2, \ldots, n \tag{2.46}$$

$$u_k^i \in \{0, 1\} \text{ for all } k = 1, 2, \ldots, t_i, i = 1, 2, 3, 4 \tag{2.47}$$

$$\underline{z} \leq z_k^i \leq \bar{z} \text{ for all } k = 1, 2, \ldots, t_i, i = 1, 2, 3, 4 \tag{2.48}$$

$$\underline{y} \leq y \leq \bar{y} \tag{2.49}$$

$$\tag{2.50}$$

where $\underline{y} = -M$ and $\bar{y} = M$ for some large values of M. Note that $y = \dfrac{1}{\mathrm{D(x)}}$. Thus, when d_{ij} is integer, we can choose $\underline{y} = -1$, $\bar{y} = 1$,$\underline{z} = -1$ and $\bar{z} = 1$.

Although the representation of integers using binary variables is well known, to the best of our knowledge, it is not used to the context of LFAP.

2.3 Computational Experiments

In this section, we present the results of extensive computational experiments carried out using LFAP2 and LFAP4 formulations. To obtain reasonable satisfied conclusion, we generated nine different groups of instances, solve the instances using both LFAP2 and LFAP4 formulations. The formulations LFAP2 and LFAP4 are solved using solver Gurobi 7.5.1 on a workstation with the following configuration: Intel i7-4790 CPU, 32GB RAM, and 64-bit Windows 7 Enterprise operating system. Gurobi is called from a Python program that prepared the input models and data.

2.3.1 Test Problem Generation

To evaluate the performance of the formulations on LFAP2 and LFAP4 we selected different test intances with different characteristics. For random numbers, we used the function random.randint in Python. The experiments are conducted on a number of instances categorized as follows.

1. Random symmetric problems with fixed range:
   The entries in both matrix C(x) and D(x) are randomly generated using the range $[-50, 50]$.

2. Random symmetric problems with size dependent data:
   The entries in both matrix C(x) and D(x) are randomly generated using the range $\left[\left\lceil -\frac{n^2}{n^{0.5}} \right\rceil, \left\lceil \frac{n^2}{n^{0.5}} \right\rceil\right]$.

3. Random right skewed problems:
   The entries in both matrix C(x) and D(x) are randomly generated using the range $[-5, 50]$. Thus, both C(x) and D(x) are right skewed matrices.

4. Randomly right skewed problems with size dependent data:
   The entries in both matrix C(x) and D(x) are randomly generated using the range $\left[\left\lceil -\frac{n^2}{15n^{0.5}} \right\rceil, \left\lceil \frac{n^2}{n^{0.5}} \right\rceil\right]$. Thus, both C(x) and D(x) are right skewed matrices.

5. Random left skewed problems:
   The entries in both matrix C(x) and D(x) are randomly generated using the range $[-50, 5]$. Thus, both C(x) and D(x) are left skewed matrices.

6. Random left skewed problems with size dependent data:
   The entries in both matrix C(x) and D(x) are randomly generated using the range $\left[\left\lceil -\frac{n^2}{n^{0.5}} \right\rceil, \left\lceil \frac{n^2}{15n^{0.5}} \right\rceil\right]$. Thus, both C(x) and D(x) are left skewed matrices.

7. Negatively correlated problems:
   C(x) is a randomly generated increasing matrix using the range $[-1000, 1000]$. Each row in matrix C(x) is sorted in ascending order. D(x) is a randomly generated decreasing matrix using the range $[-1000, 1000]$. Each row in matrix D(x) is sorted in descending order.

8. Positively correlated problems-increased:
   Both C(x) and D(x) are randomly generated increasing matrices using the range $[-1000, 1000]$, we name it increasing positive correlated problem.

9. Positively correlated problems-decreased:
   Both C(x) and D(x) are randomly generated decreasing matrices using the range $[-1000, 1000]$, we name it decreasing positive correlated problem.

In the test problem we generated, we are not assuming $D(x) \neq 0$. However, in the formulations, we eliminated this possibility. Also, there are many other ways to generate test problems. With limited experiment analysis, we focus this thesis on limited classes of test instances with some relationship between $C(x)$ and $D(x)$ Under each category, we generated instances in size n = 100, 200,...,1000.

2.4 Computational Results

In this section, we present the computational results on all of the nine instance groups for both LFAP2 and LFAP4. In Table 2.1 to Table 2.9, the column 'Obj.Value' represents the objective function value of the problem solved. The column 'Time(s)' represents the time taken to solve the problem in seconds. Under Time(s), '- ' indicates that it reaches the four hour time limit and the corresponding Obj.Value is the best value we obtained within the time limit.

Table 2.1: Results of Symmetric Problems with fixed range

		LFAP2		LFAP4	
Test	n	Obj.Value	Time(s)	Obj.Value	Time(s)
R1	100	-4890	60.83	-4890	19.05
R2	200	-9917	1738.07	-9917	1029.27
R3	300	-14969	3416.07	-14969	196.41
R4	400	-19989	10242.59	-19989	117.47
R5	500	-24994	11374.66	-24994	1163.32
R6	600	-29999	13236.66	-29999	530.19
R7	700	-34997	15360.98	-34997	709.14
R8	800	-40000	1280.45	-40000	780.19
R9	900	-45000	3173.25	-45000	1732.75
R10	1000	-50000	-	-50000	-

Table 2.2: Results of Symmetric Problems with data size dependent range

		LFAP2		LFAP4	
Test	n	Obj.Value	Time(s)	Obj.Value	Time(s)
R1	100	-9767	95.85	-9767	10508.23
R2	200	-55808	2233.17	-27905.5	-
R3	300	-154446	-	-154446	-
R4	400	-153838	-	-317600	-
R5	500	-317527	-	-185360	-
R6	600	-217736	-	-438757	-
R7	700	-1281409	-	-430410	-
R8	800	-1797159	-	-1803220	-
R9	900	-1206280	-	-220117	-
R10	1000	-200413	-	-1048421.67	-

Table 2.3: Results of Left Skewed Problems with fixed range

		LFAP2		LFAP4	
Test	n	Obj.Value	Time(s)	Obj.Value	Time(s)
R1	100	-4669	4.32	-4669	18.80
R2	200	-9671	34.97	-9671	14.80
R3	300	-14768	179.16	-14768	67.50
R4	400	-19830	776.02	-19830	194.48
R5	500	-24841	1785.94	-24841	50.83
R6	600	-29891	4304.35	-29891	9666.82
R7	700	-34927	14827.87	-34927	839.95
R8	800	-39963	7033.47	-39963	165.12
R9	900	-69.15	-	-44994	204.83
R10	1000	-66.09	-	-50000	358.97

Table 2.4: Results of Left Skewed Problems with data size dependent range

		LFAP2		LFAP4	
Test	n	Obj.Value	Time(s)	Obj.Value	Time(s)
R1	100	-90147	3874.47	-90147	32.96
R2	200	-536641	326.28	-536641	929.25
R3	300	-745792	-	-1499082	437.16
R4	400	-1023206	-	-3107025	1744.19
R5	500	-5424025	-	-5468865	5218.83
R6	600	-254578	-	-8647804	4911.91
R7	700	-669357	-	1797418	-
R8	800	-1484738	-	-1272638	-
R9	900	-24001796	-	-23999333	1998.54
R10	1000	-11092	-	-9773293	-

Table 2.5: Results of Right Skewed Problems with fixed range

		LFAP2		LFAP4	
Test	n	Obj.Value	Time(s)	Obj.Value	Time(s)
R1	100	-4632	3.58	-4632	7.53
R2	200	-9739	36.17	-9739	47.78
R3	300	-14752	200.9	-14752	34.62
R4	400	-19807	360.10	-19807	91.38
R5	500	-24861	868.70	-24861	205.24
R6	600	-29884	5201.34	-29884	474.44
R7	700	-34920	4453.90	-34920	953.37
R8	800	-39957	7732.39	-39957	104.47
R9	900	-44982	11423.66	-44982	158.57
R10	1000	-50000	17985.35	- 50000	171.51

Table 2.6: Results of Right Skewed Problems with data size dependent range

		LFAP2		LFAP4	
Test	n	Obj.Value	Time(s)	Obj.Value	Time(s)
R1	100	-89057	9822.31	-89057	112.88
R2	200	-536773	511.92	-536773	141.73
R3	300	-1495795	-	-1497965	184.07
R4	400	-1497696	-	-3111269	1539.31
R5	500	-900910	-	-5453382	-
R6	600	-254578	-	-4329428	2688.44
R7	700	-669357	-	-12751720	-
R8	800	11909	-	-17795050	-
R9	900	-23988703	-	-5990118.5	-
R10	1000	-31262549	-	-5144754	-

Table 2.7: Results of Negative Correlated Problems

		LFAP2		LFAP4	
Test	n	Obj.Value	Time(s)	Obj.Value	Time(s)
R1	100	19.97	-	3.1	4631.19
R2	200	-2.07	-	-2.15	6975.61
R3	300	-0.35	-	-0.55	-
R4	400	-0.85	-	-1.63	-
R5	500	-5.02	-	-89	-
R6	600	0.06	-	-0.84	-
R7	700	1.43	-	1.36	-
R8	800	0.35	-	0.32	-
R9	900	-0.33	-	-19.46	-
R10	1000	-3.10	-	125.22	-

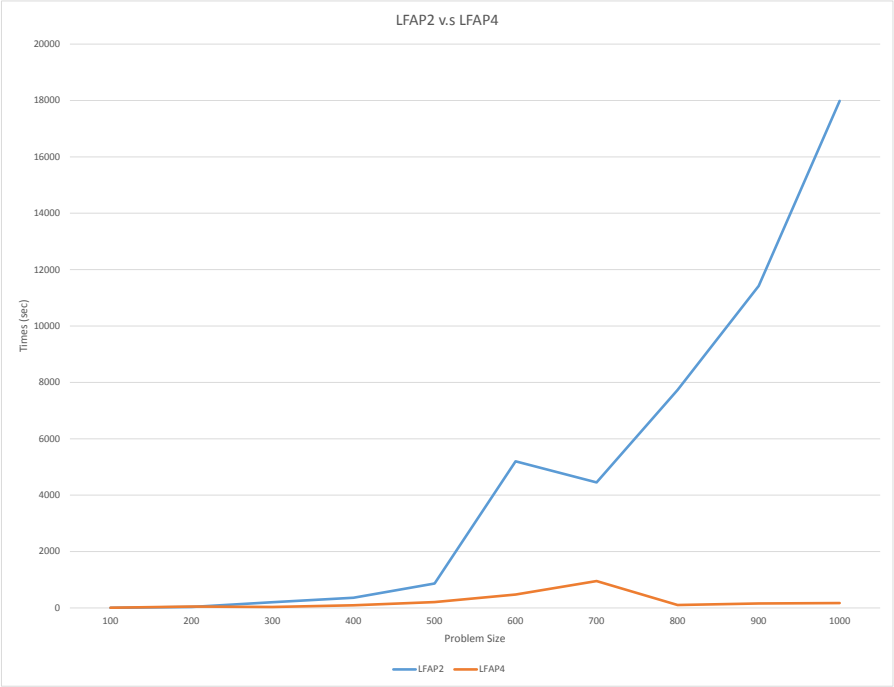
Table 2.8: Results of Positive Correlated Problems-increased matrices

		LFAP2		LFAP4	
Test	n	Obj.Value	Time(s)	Obj.Value	Time(s)
R1	100	0.04	-	0.00	-
R2	200	0.16	-	-0.25	-
R3	300	1.04	-	0.35	-
R4	400	1.6	-	1.55	-
R5	500	1.18	-	1.14	-
R6	600	4.07	-	-418	-
R7	700	-0.38	-	-0.54	-
R8	800	-0.44	-	-0.84	-
R9	900	-0.94	-	-1.94	-
R10	1000	0.31	-	-0.20	-

Table 2.9: Results of Positive Correlated Problems-decreased matrices

		LFAP2		LFAP4	
Test	n	Obj.Value	Time(s)	Obj.Value	Time(s)
R1	100	- 0.94	-	-1.02	-
R2	200	-2	-	-2.53	-
R3	300	-1	-	-0.61	-
R4	400	6.83	-	124.88	-
R5	500	-1134	-	165	
R6	600	-3.42	-	-8.06	-
R7	700	2.40	-	-585	-
R8	800	-1.09	-	-3.2	-
R9	900	-1.39	-	-710	-
R10	1000	0.4	-	-1.07	-

Figure 2.2: Problem Size v.s. Times of Right Skewed Fix Range



From the results presented in the Tables, we can see that as the matrix size n gets bigger, both LFAP2 and LFAP4 require longer computation time to find the optimal solution.

Comparing Table 2.1 with Table 2.2, Table 2.3 with Table 2.4, and Table 2.5 with Table 2.6 respectively, we see that instances with fixed range parameters usually took less computation time compared to the ones with data size dependent range for the same size instances. For the parameter setting, the ones with data size dependent range had wider range compared to the fixed ones for all size instances.

None of the positively correlated problems solved to optimality even for $n = 100$. Positive correlation, although considered as a structured problem, resulted in harder instances. Negative correlation data also produced hard instances.

Also, LFAP4 generally ran faster than LFAP2 when the time limit had not been reached. In some instances, LFAP4 solved the problem within the time limit while LFAP2 does not. In the instances that both LFAP2 and LFAP4 could not solve the problem within the time limit, LFAP4 usually produced better objective values compared to LFAP4.

Thus, LFAP4 appears to be a better model. But note that the model assume integer values for elements of C and D. If the elements of C and D are rational numbers, they can be scaled to integers and apply LFAP4. But such a scaling operation could result in large integers and could result in overflow errors. We did not test LFAP4 model with rational data scaled to integers.

Chapter 3

Newton's Method

3.1 Introduction

In the introduction chapter, we discussed how Newton's method (also called Dinkelbach's algorithm) can be used to solve LFP and $FCOP^{+}$.

Since LFAP is a special case of LFP and $FCOP^{+}$, Newton's method (Dinklebach's method) can be used to solve LFAP when $D(x) > 0$ for $x \in F$.
Recall that Newton's method solves the parametric problem

$$FAP(\lambda): \qquad f(\lambda) = \min_{x \in S}\{(\mathbf{c^T x}) - \lambda(\mathbf{d^T x})\}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \tag{3.1}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n \tag{3.2}$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j \tag{3.3}$$

The validity of Newton's method follows from Theorem 1.4 and applies to LFAP when D(x)>0 for all x.

However, the method is not directly applicable to solve the general version of the LFAP. When D(x) > 0 for all $x \in F$, x^* is an optimal solution for LFAP if and only if $f(\lambda) = c^T x^* - \lambda d^T x^* = \min\{c(x) - \lambda D(x) : x \in F\} = 0$ [12, 25, 29]. (The analogous result is true for more general linear and integer fractional programs.) When $D(x)$ is allowed to take arbitrary values, the condition is only necessary but not sufficient. To see this, consider a 3×3 LFAP with

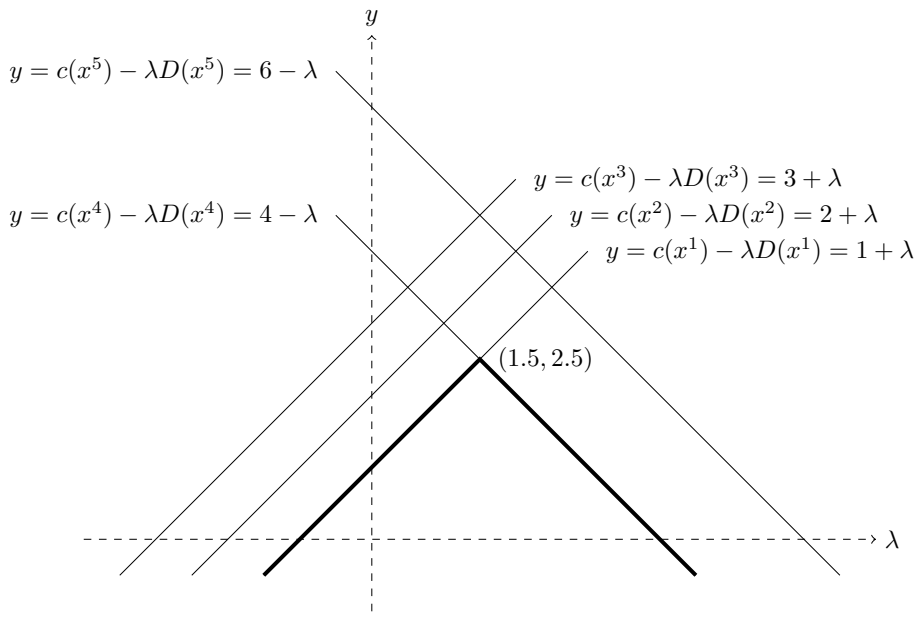


Figure 3.1: The graph of $f(\lambda)$ is shown in thick lines. The straight lines corresponding to x^2 and x^6 are the same.

$$C = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 4 & 1 & 0 \end{pmatrix} \text{ and } D = \begin{pmatrix} -1 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Assignment x	$c(x)$	$D(x)$	Ratio	Line
$x^1 = (1,2,3)$	1	-1	-1	$y = 1 + \lambda$
$x^2 = (2,1,3)$	2	-1	-2	$y = 2 + \lambda$
$x^3 = (1,3,2)$	3	-1	-3	$y = 3 + \lambda$
$x^4 = (3,2,1)$	4	1	4	$y = 4 - \lambda$
$x^5 = (2,3,1)$	6	1	6	$y = 6 - \lambda$
$x^6 = (3,1,2)$	2	-1	-2	$y = 2 + \lambda$

Table 3.1: Assignment and values

There are six possible assignments $x^1 = (1,2,3), x^2 = (2,1,3), x^3 = (1,3,2), x^4 = (3,2,1), x^5 = (2,3,1)$ and $x^6 = (3,1,2)$. Figure 3.1 shows the plot of $c^T(x^i) - \lambda d^T(x^i)$ for $i =, 2, \ldots, 6$ and the lower envelop of these lines represents $f(\lambda)$. Note that x^3 is the unique optimal solution to the resulting LFAP but it doesn't contribute to $f(\lambda)$.

We can however use Newton's method to solve LFAP as two problems of the type LFAP+. Consider the following *constrained fractional assignment problems*:

$$CLFAP1: \text{Minimize} \quad f_1(x) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \tag{3.4}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n \tag{3.5}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \geq \epsilon \tag{3.6}$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j \tag{3.7}$$

$$CLFAP2: \text{Minimize} \quad f(x) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} -c_{ij}x_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{n} -d_{ij}x_{ij}}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \tag{3.8}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n \tag{3.9}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} -d_{ij}x_{ij} \geq \epsilon \tag{3.10}$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j \tag{3.11}$$

Note that the denominator in the objective function of both CLFAP1 and CLFAP2 are non-negative and hence we can apply Newton's method to solve these problems.

Theorem 3.1. *Let x^0 and x^* respectively be optimal solutions of CLFAP1 and CLFAP2. Then, the minimum amongst x^0 and x^* is an optimal solution to LFAP.*

The proof of this theorem follows from Theorem 2.2. CLFAP1 and CLFAP2 belongs to $LFAP^+$ where the family of feasible solution is restricted to F^+. In Theorem 2.2, we have shown that both $LFAP^+$ and $LFAP^-$ are NP-hard. Moreover, the best solution amongest optimal solutions to $LFAP^+$ and $LFAP^-$ solves LFAP. From this, it can be verified that the best solution amongst the solution of CLFAP1 and CLFAP2 gives an optimal solution to LFAP.

Let us now discuss Newton's method specialized to CLFAP1.

Algorithm - Newton's method - CLFAP1

Step 1: Choose an upper bound for λ.

Step 2: Solve the constrained assignment problem

$$\alpha = \min \sum_{i=1}^{n} \sum_{j=1}^{n} (c_{ij} - \lambda d_{ij}) x_{ij}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \geq \epsilon$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j$$

Let x^* be the optimal solution produced.

Step 3: If $-\delta \leq \alpha \leq \delta$, where δ is a prescribed tolerance, output x^* and stop.

Step 4: Update $\lambda = \dfrac{C(x^*)}{D(x^*)}$ and go to step 2.

Theorem 3.2. *Newton's method for CLFAP1 solves the problem in* $O\left(m^4 log^2 m \times \phi(n, C, D)\right)$ *time, where* $\phi(n, C, D)$ *is the complexity of solving a linear constrained assignment problem.*

The proof of this theorem follows from Theorem 1.8.

The algorithm for CLFAP2 is similar to that of CLFAP1 except that in step 2 we solve the problem

$$\alpha = \min \sum_{i=1}^{n} \sum_{j=1}^{n} (-c_{ij} + \lambda d_{ij}) x_{ij}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n \tag{3.12}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n \tag{3.13}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} -d_{ij} x_{ij} \geq \epsilon \tag{3.14}$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j \tag{3.15}$$

Combining CLFAP1 and CLFAP2, we get the following algorithm to solve LFAP, which is based on standard Newton's method applied to CLFAP1 and CLFAP2.

Algorithm - Newton's method for LFAP

Step 1: Solve the assignment problem.

$$\text{Maximize} \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j$$

Let x^* be the optimal solution and z^* be the optimal objective function value. If $z^* \leq 0$ set $z_1 = \infty$ and go to Step 3.

Step 2: Compute $\lambda = \dfrac{C(x^*)}{D(x^*)}$. Starting with this value of λ apply Newton's method to compute an optimal solution x^1 to CLFAP1 with optimal objective function value z^1.

Step 3: Solve the assignment problem

$$\text{Minimize} \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j$$

Let x^0 be the optimal solution produced with optimal objective function value z^0. If $z^0 > 0$ set $z_2 = \infty$ and go to Step 5.

Step 4: Compute $\lambda = \dfrac{C(x^0)}{D(x^0)}$. Starting with this value of λ apply Newton's method to compute an optimal solution x^2 to CLFAP2 with optimal objective function value z^2.

Step 5: Let $obj = \min\{z^1, z^2\}$. If $obj = z_1$ output x^1 else output x^2.

It may be noted that if z_1 or z_2 is ∞, then LFAP is solvable in polynomial time.

3.2 Heuristic Newton's method

In the previous section, we showed that the LFAP can be solved by two application of Newton's method. One for solving CLFAP1 and another for solving CLFAP2. In each of these cases the intermediate problem solved is a constrained assignment problem [45] which is NP-hard. Thus, essentially, we are solving LFAP as a sequence of NP-hard problem, solving which could be time consuming. Thus, we considered the possibility solving the constrained assignment problem by a heuristic. We call the resulting algorithm the Heuristic Newton's method.

There are many heuristic algorithms available to solve a constrained assignment problem [37,47]. Although, the Newton's method is guaranteed to terminate in a polynomial number of iterations [39], the heuristic version need not converge. Depending on the nature of the heuristic used, it is possible for the heuristic Newton's method to cycle and never converge. There are different methods one can adopt to insure finite termination. We simply set a time limit. For solving the constrained assignment problem heuristically, we simply used the Gurobi solver with a time limit.

We formulate heuristic Newton's method by adding a computation time limit of 10 minutes to each iteration of the exact Newton's method which we described in the previous section. Thus, for the iterations which took more than ten minutes to find the global solution, we find the local optimal within the time limit. In that way, we can shorten the computation time. However, it will not guarantee the global optimality of the solution.

3.3 Computational Experiments

In this section, we present the computational results of both the exact Newton's method and the heuristic Newton's method. We use the same nine groups of instances generated before (see in section 2.3.1). All LFAP models are solved using solver Gurobi 7.5.1 on a workstation with the configuration: Intel i7-4790 CPU, 32GB RAM, and 64-bit Windows 7 Enterprise operating system. The algorithms are implemented using Python. Time limit parameter for each instance overall is set to 3 hours. In Table 3.2 to Table 3.10, the column 'Obj.Value' represents the objective value of the problem solved. The column 'Time(s)' represents the time taken to solve the problem. Under Time(s) column, '-' indicates that it reaches the three hour time limit and the corresponding Obj.Value is the best value we can obtain within the time limit. The column 'Iteration(s)' represents the number of iterations it took to get the objective value presented.

Table 3.2: Results of Symmetric Problems with fixed range

		Exact Newton's Method			Heuristic Newton's Method		
Test	n	Obj.Value	Time(s)	Iteration(s)	Obj.Value	Time(s)	Iteration(s)
R1	100	-4890	2.64	3	-4890	2.97	3
R2	200	-9917	8.03	3	-9917	6.19	3
R3	300	-14969	18.03	3	-14969	14.66	3
R4	400	-19989	41.75	3	-19989	38.66	3
R5	500	-24994	65.09	3	-24994	60.97	3
R6	600	-29999	107.44	3	-29999	103.28	3
R7	700	-34997	128.58	3	-34997	125.14	3
R8	800	-40000	196.96	3	-40000	190.54	3
R9	900	-45000	250.83	3	-45000	245.38	3
R10	1000	-50000	294.09	3	-50000	289.64	3

Table 3.3: Results of Symmetric Problems with data size dependent range

		Exact Newton's Method			Heuristic Newton's Method		
Test	n	Obj.Value	Time(s)	Iterations	Obj.Value	Time(s)	Iterations
R1	100	-9767	9.67	3	-9767	4.48	3
R2	200	-55808	36.71	4	-55808	21.00	4
R3	300	-154446	79.00	3	-154446	59.40	3
R4	400	-317604	118.73	3	-317604	117.40	3
R5	500	-556086	732.57	3	-556086	521.56	3
R6	600	-876627	-	100	-876626	-	125
R7	700	-1290663	-	90	-1290663	-	92
R8	800	-1803376	-	66	-1802453	-	23
R9	900	-2421499	-	4	-2421499	-	4
R10	1000	-3153447	-	5	-3153448	-	20

Table 3.4: Results of Left Skewed Problems with fixed range

		Exact Newton's Method			Heuristic Newton's Method		
Test	n	Obj.Value	Time(s)	Iterations	Obj.Value	Time(s)	Iterations
R1	100	-4669	2.95	3	-4669	3.43	3
R2	200	-9671	10.79	3	-9671	7.14	3
R3	300	-14768	38.09	3	-14768	20.25	3
R4	400	-19830	48.02	3	-19830	33.94	3
R5	500	-24841	88.42	3	-24841	64.10	3
R6	600	-29891	144.28	3	-29891	82.95	3
R7	700	-34927	-	3	-34927	712.21	3
R8	800	-39963	-	2	-39963	724.21	3
R9	900	-44994	220.70	3	-44994	217.72	3
R10	1000	-50000	282.77	3	-50000	278.35	3

Table 3.5: Results of Left Skewed Problems with data size dependent range

		Exact Newton's Method			Heuristic Newton's Method		
Test	n	Obj.Value	Time(s)	Iterations	Obj.Value	Time(s)	Iterations
R1	100	-90147	8.87	3	-90147	5.62	3
R2	200	-536641	51.84	3	-536641	25.57	3
R3	300	-1499082	236.50	4	-1499082	171.71	4
R4	400	-3107025	229.55	3	3107025	181.55	3
R5	500	-5462120	-	96	-5462120	-	133
R6	600	-8647804	1215.05	3	-8647804	877.22	3
R7	700	-12761000	-	61	-12761000	-	69
R8	800	-17846723	-	70	-17846723	-	91
R9	900	-23999236	-	25	-23999236	-	27
R10	1000	-31271804	-	3	-15635845	-	25

Table 3.6: Results of Right Skewed Problems with fixed range

		Exact Newton's Method			Heuristic Newton's Method		
Test	n	Obj.Value	Time(s)	Iteration(s)	Obj.Value	Time(s)	Iteration(s)
R1	100	-4632	2.88	3	-4632	3.08	3
R2	200	-9739	8.69	3	-9739	6.67	3
R3	300	-14752	21.63	3	-14752	16.62	3
R4	400	-19807	43.38	3	-19807	38.10	3
R5	500	-24863	72.62	3	-24863	64.96	3
R6	600	-29885	116.53	3	29885	114.01	3
R7	700	-34922	-	3	-34922	721.00	3
R8	800	-39959	185.43	3	-39959	180.34	3
R9	900	-44984	205.31	3	-44984	204.87	3
R10	1000	-50000	321.54	3	-50000	318.09	3

Table 3.7: Results of Right Skewed Problems with data size dependent range

		Exact Newton's Method			Heuristic Newton's Method		
Test	n	Obj.Value	Time(s)	Iterations	Obj.Value	Time(s)	Iterations
R1	100	-89057	6.83	3	-89057	6.66	3
R2	200	-536773	53.18	3	-536773	55.88	3
R3	300	-1497965	102.52	3	-1497965	107.66	3
R4	400	-3111269	627.14	4	-3111269	680.11	3
R5	500	-5464140	612.09	4	-5464140	657.84	4
R6	600	-8658797	-	28	-8658797	-	31
R7	700	-12764225	-	62	-12764225	-	73
R8	800	-947469	-	35	-947469	-	39
R9	900	-23986211	-	6	-1331429	-	23
R10	1000	-31258829	-	4	-583940	-	23

Table 3.8: Results of Negative Correlated Problems

		Exact Newton's Method			Heuristic Newton's Method		
Test	n	Obj.Value	Time(s)	iteration(s)	Obj.Value	Time(s)	iteration(s)
R1	100	-806	3.39	3	-806	3.39	3
R2	200	-2.16	4.07	4	-2.16	4.32	4
R3	300	-0.57	10.22	4	-0.57	10.63	4
R4	400	-1.64	28.47	4	-1.64	28.80	4
R5	500	-617	67.70	3	-617	70.44	3
R6	600	-0.85	63.04	4	-0.85	67.53	4
R7	700	0.40	92.49	3	0.41	78.58	3
R8	800	0.04	99.50	3	0.05	102.12	3
R9	900	-287	3271.35	3	-287	899.21	3
R10	1000	-1125	750.70	3	-1125	744.85	3

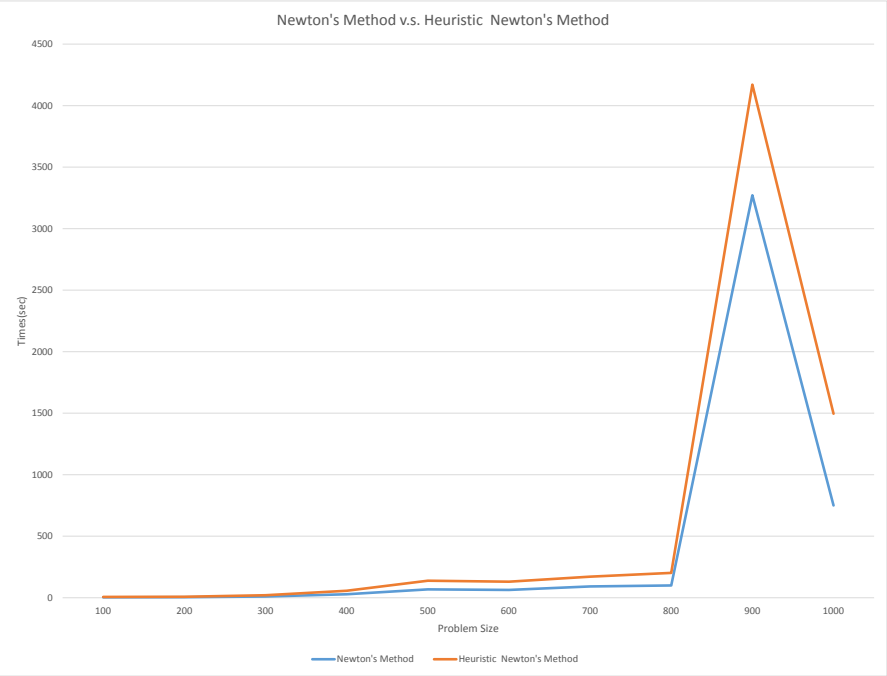
Table 3.9: Results of Positive Correlated Problems-increased matrices

		Exact Newton's Method			Heuristic Newton's Method		
Test	n	Obj.Value	Time(s)	Iterations(s)	Obj.Value	Time(s)	Iterations(s)
R1	100	-0.04	1.50	4	-0.04	0.41	4
R2	200	-0.25	3.45	4	-0.25	3.63	4
R3	300	0.27	6.71	3	0.27	7.23	3
R4	400	1.16	14.44	3	1.16	15.24	3
R5	500	0.67	24.85	3	0.67	26.12	3
R6	600	-418	83.91	3	-418.00	89.39	3
R7	700	-0.54	72.11	4	-0.54	76.10	4
R8	800	-0.87	95.39	4	-0.87	123.72	4
R9	900	-2.31	162.17	4	-2.31	272.02	4
R10	1000	-0.29	119.28	3	-0.29	309.85	3

Table 3.10: Results of Positive Correlated Problems-decreased matrices

		Exact Newton's Method			Heuristic Newton's Method		
Test	n	Obj.Value	Time(s)	Iteration(s)	Obj.Value	Time(s)	Iteration(s)
R1	100	-1.03	0.97	4	-1.03	0.97	4
R2	200	-2.56	4.48	4	-2.56	4.57	4
R3	300	-0.63	11.10	4	-0.63	11.55	4
R4	400	-990	40.72	3	-990	42.38	3
R5	500	-1147	62.82	3	-1147	67.72	3
R6	600	-8.06	63.56	4	-8.06	67.86	4
R7	700	-585	158.65	3	-585	157.32	3
R8	800	-3.20	170.04	4	-3.2	170.24	4
R9	900	-710	302.93	3	-710	312.68	3
R10	1000	-349	379.57	2	-349	390.68	2

Figure 3.2: Problem Size v.s. Times of Negative Correlated Problems



3.4 Experimental Analysis

From the experimental results we presented in Table 3.2-3.10, we can see that the bigger the matrix size n, the longer computation time needed to get the optimal solution. For instances which have been solved within the three hour time limit, the objective value for both methods and the time it took to solve the problem are similar and the number of iterations are the same. For the instance which cannot be solved within the time limit, the number of iterations is less for exact Newton's method compared to the heuristic Newton's method. However, the best result found within the time limit for both problems is very similar. Overall, the efficiency and effectiveness of both methods are similar.

Compared to the mixed integer programming formulations discussed in chapter 2, the adaption of Newton's method discussed in this chapter was more efficient. Note, the method solved positively correlated problems optimally but our MIP models could not solve them.

Chapter 4

Local Search Algorithms

4.1 Introduction

In the previous chapter, we introduced a heuristic method for solving the LFAP using a variation of the Newton's method. In this chapter, we develop some local search based heuristics for solving LFAP. These are important algorithms in the sense that the algorithms take input a feasible solution and tries to find improved solutions.

4.2 Local Search

Let x^0 be a feasible solution for the LFAP and N be a given neighborhood. Then x^0 is a *local optimum* with respect to the neighborhood N if $f(x^0) \leq f(x)$ for all $x \in N$. Recall that f is the objective function of LFAP. Note that $N \subseteq F$ and F is the family of feasible solutions. Similarly, x^0 is a *global optimum* if $f(x^0) \leq f(x)$ for all $x \in F$. A neighborhood N is said to be an *exact neighborhood* if a local optimum with respect to N is also a global optimum. When $D(x) > 0$ for all $x \in F$, the adjacency structure of the assignment polytope [5] defines an exact neighborhood.

A local search algorithm starts with a feasible solution x and explore its neighborhood $N(x)$ for an improving solution. If no such solution is available, the algorithm terminates and x is a locally optimal solution. If an improving solution is found in $N(x)$, the algorithm *moves* to one such solution. In the 'best improvement version' of local search, the algorithm moves to the best improving solution in $N(x)$. In the 'first improvement version' of local search, the algorithm moves to the improving solution that found first. In the first improvement version, the search for improving solution could be faster but the magnitude of improvement could be smaller. In the 'best improvement version', the search for the best improving solution could be time consuming but have the potential for steeper improvement in the objective function values. As a compromise between these two variations, a best im-

proving solution from a pre-defined candidate list is sometimes used. A formal description of a generic local search algorithm is given below.

Algorithm 1: local search for LFAP

```
Require: findMove()
repeat
    mv ← findMove()
    if is NULL then
        break
    else
        makeMove(mv)
    end
until termination condition is met;
```

For the LFAP, we consider a 2-exchange neighborhood to develop local search algorithms. 2-exchange neighborhoods are often being used for solving linear and quadratic combinatorial optimization problems [13]. To the best of our knowledge, this class of neighbors are not studied for fractional combinatorial optimization problem. Any feasible solution x of the LFAP can be represented by a set $X(S)$ of ordered pairs, where $(i,j) \in X(S)$ if and only if $x_{ij} = 1$. For simplicity, in this chapter, we use the notation x to represent the set $X(S)$. For any two distinct elements $(i,j),(k,l)$ in an assignment x, let $\hat{x}$ be the assignment given by $x - \{(i,j),(k,l)\} \cup \{(i,l),(k,j)\}$. The operation of generating $\hat{x}$ from x is called a 2-exchange. Then, the 2-exchange neighborhood of x, denoted by $N(x)$, is the collection of all assignments obtained from x by a 2-exchange operation. It can be verified that the objective function value $f(\hat{x})$ of $\hat{x}$ is given by

$$f(\hat{x}) = \frac{C(x) - c_{ij} - c_{k\ell} + c_{i\ell} + c_{kj}}{D(x) - d_{ij} - d_{k\ell} + d_{i\ell} + d_{kj}} = \frac{C(x) + \Delta C(mv)}{D(x) + \Delta D(mv)}.$$

where $\Delta C(mv) = c_{i\ell} + c_{kj} - c_{ij} - c_{k\ell}$ and $\Delta D(mv) = d_{i\ell} + c_{kj} - d_{ij} - d_{k\ell}$.
The solution x is locally optimal with respect to a 2-exchange neighborhood if $f(x) - f(\hat{x}) \leq 0$ for all $\hat{x} \in N(x)$. Choose $(p,q),(r,s) \in x$ such that

$$\frac{C(x) + \Delta C(mv)}{D(x) + \Delta D(mv)} = \min\left\{ \frac{C(x) + \Delta C(mv)}{D(x) + \Delta D(mv)} : (i,j),(k,\ell) \in x \right\}$$

Then $\hat{x} = x - \{(r,s),(p,q)\} \cup \{(r,q),(p,s)\}$ is the best solution in $N(x)$.
The above process of finding a local search move is described in the following algorithm.

Algorithm 2: findMove

bestMv $\leftarrow NULL$,
bestMvObj $\leftarrow \infty$;
forall $mv(i,j) \in N(x)$*:* **do**
  if $bestMvObj > \dfrac{C + \Delta C(mv)}{D + \Delta D(mv)}$ **then**
    bestMv$\leftarrow mv$
    bestMvObj$\leftarrow \dfrac{C + \Delta C(mv)}{D + \Delta D(mv)}$
  end
end
return bestMv

If the output is Null, then the input of algorithm1 is locally optimal. Given $C(x)$ and $D(x)$, we can compute $\hat{x}$ and $f(\hat{x})$, (where $\hat{x}$ is the solution we move to from x) in $O(n^2)$ time. If $f(x) - f(\hat{x}) > 0$, a local search algorithm replaces x by $\hat{x}$ and the search for an improving 2-exchange solution is continued. The algorithm terminates when a locally optimal solution is reached. The function makeMove means we move from one solution to another wehre mv defines the movement. The local search is presented below.

As mentioned earlier, a local search algorithm terminates when a local optimum is reached. However, this local optimum may be far from a global optimum. One of the strategies to enhance a basic local search algorithm is to initiate the algorithm multiple times with different starting solutions. We then compare the local optimum we get each time and use the best one as our final solution. A new local search can be initiated by selecting a different starting solution and continuing the process for a fixed number of iterations and choosing the overall best solution results in our *multi-start local search algorithm* (MSLS-Algorithm).

However, since the MSLS-algorithm needs to run multiple times to improve the solution, it takes lots of computation time. Given that this strategy may be too random, it cannot guarantee improvement of local search algorithms.

For instance, if the start solution has $C(x)$ and $D(x)$ both positive or negative, algorithm 1 tends to move to the direction where |C(x)| is minimized and |D(x)| is maximized and terminate with such a local solution. However, it is possible that the global solution is with |C(x)| maximized and |D(x)|minimized. In this case, a local search will not produce a good solution.

If there is no huge gap between local optimum and global optimum, the strategy we mentioned above may not be very helpful. To improve it, instead of using a random starting

solution, we could use a local search strategy to find a relatively good starting solution and perform a 2-exchange local search to find local solution.

Our strategy is to find a starting solution with negative objective function value (if exists). There are two cases from which we can obtain solutions with negative objective function values. The first case is when $C(x)$ is positive and $D(x)$ is negative. The second case is when $C(x)$ is negative while $D(x)$ is positive.

For finding positive $C(x)$ value and negative $D(x)$ value, we have 4 kinds of randomly generated starting solution. First is when $C(x)$ is positive while $D(x)$ is negative. For this kind, we terminate the algorithm and use the starting solution as the output. Second is when $C(x)$ and $D(x)$ are both negative. In this case, we try to find moves where values of both $C(x)$ and $D(x)$ are increased while making sure $D(x)$ remains negative. Third is when $C(x)$ and $D(x)$ both positive. Fourth is when $C(x)$ is negative and $D(x)$ is positive. For the third and fourth, we try to move to a new solution where $C(x)$ is increasing while $D(x)$ is decreasing. After finishing running for all potential movements on the candidate list, we output the final solution. The formal description of algorithm for finding a start solution with $C(x)$ positive and $D(x)$ negative can be written as follows:
Note that we replace $findmove()$ in Algorithm 1 to Algorithm 3 to get the whole algorithm. where mv represents the current movement.$\Delta C(mv)$ represents changing value of $C(x)$ when move to mv. $\Delta D(mv)$ represents changing value of $D(x)$ when move to mv. bestMv represent the best movement we have so far. $\Delta C(bestMv)$ represent the the changing value

of $C(x)$ when make the bestMv. $\Delta D(bestMv)$ represent the the changing value of $D(x)$ when make the bestMv.

Algorithm 3: Find Move with positive C(x) and negative D(x)

```
bestMv ← NULL
for mv ∈ N(x) do
    if D(x) < 0: then
        if ΔD(mv) + D > 0: then
            continue
        end
        if ΔC(mv) > 0 and ΔD(mv) > 0 and (ΔC(bestMv) < 0 or
         ΔD(bestMv) < 0)): then
            bestMv ← mv
        else
            if ΔC(bestMv) + ΔD(bestMv) < ΔC(mv) + ΔD(mv) then
                bestMv ← mv
            end
        end
    end
    else if ΔD(bestMv) > ΔD(mv) or (ΔD(bestMv) = ΔD(mv) and
     ΔC(bestMv) < ΔC(mv)) : then
        bestMv ← mv
    end
    return bestMv
end
output: the best solution found so far
```

For the second case is when $C(x)$ is negative while $D(x)$ is positive, we also have 4 kinds of randomly generated starting solution. First is when $C(x)$ is negative while $D(x)$ is positive. For this kind, we terminate the algorithm and use the starting solution as the output. Second is when $C(x)$ and $D(x)$ are both negative. In this case, we try to find a move where values of both $C(x)$ and $D(x)$ are increased while making sure $C(x)$ remains negative. Third is when $C(x)$ and $D(x)$ are both positive. In this case, we try to find a move where values of both $C(x)$ and $D(x)$ are decreasing while making sure $D(x)$ remains negative. $C(x)$ and $D(x)$ both positive. Fourth is when $C(x)$ is positive and $D(x)$ is negative. In this case, we try to move to a point where $C(x)$ is decreasing and $D(x)$ is decreasing. After finishing running for all potential movements on the candidate list, we output the final solution. The formal description of algorithm for finding a start solution

with $C(x)$ negative and $D(x)$ positive can be written as follows:
Note that we replace $findmove()$ in Algorithm 1 to Algorithm 4 to get the whole algorithm.

Algorithm 4: Find Move with negative C(x) and positive D(x)

```
bestMv ← NULL
for mv ∈ N(x) do
    if D(x) > 0: then
        if ΔD(mv) + D < 0: then
            continue
        end
        if ΔC(mv) < 0 and ΔD(mv) < 0 and (ΔC(bestMv) > 0 or
         ΔD(bestMv) > 0)): then
            bestMv ← mv
        else
            if ΔC(bestMv) + ΔD(bestMv) > ΔC(mv) + ΔD(mv) then
                bestMv ← mv
            end
        end
    end
    else if ΔD(bestMv) < ΔD(mv) or (ΔD(bestMv) = ΔD(mv) and
     ΔC(bestMv) > ΔC(mv)) : then
        bestMv ← mv
    end
    return bestMv
end
output: the best solution found so far
```

Based on the above two algorithms, we can have the Modified local search for LFAP.

For the modified local search, we first start with a random solution x. We run Algorithm 3 and see if we can get a new starting solution with postive $C(x)$ value and negative $D(x)$ value. If yes, we output the solution and use it as input to regular Local Search algorithm. If not, we run Algorithm 4 and see if we can get a new starting solution with negative $C(x)$ value and positive $D(x)$. If we can find such solution, we use it as input to the regular Local Search Algorithm. If cannot find such solution, we find a new random solution x and repeat the above process until we can find a solution.

A formal high level description of our modified local search algorithm for LFAP is given below.

Algorithm 5: Modified local search for LFAP

$isNegobj \leftarrow false$
initilize()
localsearch(Find Move with positive C(x) and negative D(x))
if $C(x) > 0$ *and* $D(x) < 0$*:* **then**
  localsearch(findMove);
  $isNegobj \leftarrow true$
end
initialize()
localsearch(Find Move with negative C(x) and positive D(x))
if $C(x) < 0$ *and* $D(x) > 0$*:* **then**
  localsearch(findMove);
  $isNegobj \leftarrow true$
end
if $isNegobj = false$ **then**
  initialize()
  localsearch(findMove)
end
output: *the best solution found so far*

4.3 Experimental Analysis

In this section, we will present results of the computational carried out using the heuristic methods developed in this chapter to solve the LFAP. All algorithms are coded in Java and tested on a workstation with the configuration: Intel i7-4790 CPU, 32GB RAM, and 64-bit Windows 7 Enterprise operating system. The test problems used are the same as those used in the experimental studies in Chapter 2, which include randomly generated sysmetric problems, right skewed problems, left skewed problems, negative correlated problems and positive correlated problems. We used the same instances as generated before for the experiments.

Four types of the experiments have been conducted:

1. Local search Algorithm without repetition. In other words, single start local search algorithm. That means for each experiment, we ran a local search algorithm without repetition with random starting solution and collected the corresponding local solution and time it took.

2. Local search Algorithm with ten restarts. In other words, ten start local search algorithm. Under this type of experiment, for each instance we ran a local search Algorithm with random starting solution independently ten times, then collected the corresponding best local solution, average value of the local solution, average time it took for each run and the average number of iterations for the ten runs.

3. Modified local search Algorithm with ten restarts. For this experiment, we ran a modified local search Algorithm for LFAP Algorithm ten times independently, then collected the corresponding best local solution, average local solution, average time it took for each run and the average number of iterations for the ten runs. For the modified local search Algorithm , we try to find local solution with positive $C(x)$ and negative $D(x)$ or with negative $C(x)$ and positive $D(x)$. If we can find neither case as the starting solution, we use a random starting solution.

4. Modified multistart local search Algorithm with a fifteen minute time limit. As the name suggests, we set a time limit for each experiment and ran the experiment instance as many times as possible to find the best objective value it could get within the time limit. Then collected the best obejctive value archieved for each instance and the total time it took to get the object value.

Table 4.1-4.9 show the computational results of the Local Search Algorithm without repeatation and the computational results of the Local Search Algorithm with ten repeatations. Table 4.10-4.9 show the computational results of the Modified Multistart Local Search Algorithm with ten restarts and the computational results of the Modified Multistart Local Search Algorithm with fifteen minutes time limit. The column 'Obj.Value' represents the best heuristic solution value found before reaching the termination condition. The column 'Time(s)' represents the time taken to solve the instance in seconds. The column 'Iteration(s)' represents the number of iterations it took to get the solution. The column 'Best Obj.Value' represents the best heuristic value it obtained among the result in multiple runs. The column 'Avg Obj.Value' represents the average heuristic solution among the result in multiple runs.

Table 4.1: Results of Symmetric Problems with fixed range

		Local Search Without Repeatation			Local Search With 10 Repeatations			
Test	n	Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-1291	0.00	24	-1410	-1098.6	0.00	20.2
R2	200	-3369	0.02	54	-4053	-3703.6	0.02	65.7
R3	300	-7536	0.13	153	-7941	-7258.8	0.13	143.6
R4	400	-10944	0.37	221	-12158	-11264.5	0.37	224.6
R5	500	-15782	0.84	312	-16116	-15410.7	0.78	300
R6	600	-19677	1.50	377	-20115	-19670.3	1.48	385.5
R7	700	-23698	2.73	491	-24857	-24084.3	2.65	476.2
R8	800	-28951	4.29	544	-28951	-28366.4	4.33	554.3
R9	900	-32319	8.10	638	-33111	-32412.4	8.10	630.8
R10	1000	-37886	14.88	731	-37886	-37241.3	13.12	726.6

Table 4.2: Results of Symmetric Problems with fixed range

		Modified Multistart Local Search with 10 restarts				Modified Multistart Local Search with 15 mins time limit			
Test	n	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-1584	-1194.5	0.02	39.2	-6059	-5809	413.60	8947.1
R2	200	-4150	-3754.4	0.04	134.9	-7729	-7652.2	458.31	25767
R3	300	-12034	-11273.8	10.29	15480.4	-12252	-12003.8	352.30	175263
R4	400	-12112	-11436.9	0.66	447.3	-16012	-15779.8	475.89	29933
R5	500	-16230	-15652	1.26	613.3	-20211	-199125.1	515.08	29644
R6	600	-20492	-20010	2.60	782.8	-22902	-22254	573.32	20515
R7	700	-24712	-24166.7	4.68	934.9	-26684	-26215.6	637.35	21729
R8	800	-28886	-28528.5	10.33	1112.8	-30756	-30202.50	369.16	21513
R9	900	-33486	-32976.4	19.82	1276.1	-34980	-34441.00	493.14	18830
R10	1000	-37618	-37279	35.55	1390.3	-39631	-38677.60	962.90	16677

Table 4.3: Results of Symmetric Problems with data size dependent range

		Local Search Without Repeatation			Local Search With 10 Repeatations			
Test	n	Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-840	0.00	4	-1624	-1186	0.00	10.3
R2	200	-6892	0.01	14	-6892	-4958.6	0.01	13.7
R3	300	-18418	0.02	20	-18418	-12415.5	0.02	17.3
R4	400	-26705	0.03	14	-35749	-23127.2	0.04	19.8
R5	500	-41932	0.08	24	-51904	-38042.1	0.07	22.4
R6	600	–59107	0.11	25	-69036	-52550.9	0.11	23.9
R7	700	-41934	0.10	16	-111085	-79820.4	0.18	30.3
R8	800	-94403	0.26	30	-107567	-83804.3	0.24	27.1
R9	900	-156193	0.83	51	-220579	-123495.8	0.41	32.6
R10	1000	-127597	0.72	30	-174950	-134601.7	0.62	32.8

Table 4.4: Results of Symmetric Problems with data size dependent range

		Modified Multistart Local Search with 10 restarts				Modified Multistart Local Search with 15 mins time limit			
Test	n	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-1782	-1236.2	0.01	20.1	-6059	-5810.5	404.67	11096.5
R2	200	-7610	-5737.8	0.01	29.1	-28446	-27533.6	372.08	25161
R3	300	-19563	- 13488.4	0.02	35.2	-59990	-55807.4	541.60	28478
R4	400	-31131	- 25102.5	0.04	36	-67439	-54547	494.35	20076
R5	500	-44156	-34496.8	0.08	43.7	-81995	-55322.5	504.33	14479
R6	600	73647	-56877.4	0.17	50.1	-102475	-82888.4	401.93	14029
R7	700	-106398	-47750.1	0.33	59.1	-125964	-99230.4	374.13	14276
R8	800	-157691	-94195.1	0.45	60.6	-1478813	-119122.6	529.78	17188
R9	900	-220579	-150889.9	0.88	60.6	-220579	-158973	560.36	15814
R10	1000	-200631	-152572	1.27	65.2	-519269	-157077.6	352.50	16606

Table 4.5: Results of Left Skewed Problems with fixed range

		Local Search Without Repeatation			Local Search With 10 Repeatations			
Test	n	Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-0.26	0.01	116	-0.31	-0.24	0.01	114.8
R2	200	-3460	0.14	371	-3876	-1508.1	0.09	325
R3	300	-89	0.31	419	-164	-120.7	0.30	399.2
R4	400	-250	0.83	486	-352	-262.3	0.73	490.2
R5	500	-469	1.48	607	-525	-393.1	1.25	580.8
R6	600	-439	2.56	662	-569	-493.5	2.29	672.7
R7	700	-517	4.60	771	-718	-620	3.84	782.3
R8	800	-947	8.73	913	-947	-833.5	6.81	874.9
R9	900	-865	14.87	963	-1079	-32549.87	18.24	1518.4
R10	1000	-1247	19.33	1079	-1247	-1134.1	41.71	1061.1

Table 4.6: Results of Left Skewed Problems with fixed range

		Modified Multistart Local Search with 10 restarts				Modified Multistart Local Search with15 mins time limit			
Test	n	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-2709	-2476.6	0.01	148.4	-3853	-3806.9	269.47	17351
R2	200	-5724	-5563.4	0.08	366.4	-8189	-8138.9	301.74	21369
R3	300	-9426	-9036.4	0.34	584.6	-12934	-12842.9	423.96	28614
R4	400	-18391	-13186.1	0.85	778.9	-17549	-17322.3	486.88	34558
R5	500	-17895	-17056.3	1.64	885.8	-21804	-21009.5	565.33	23337
R6	600	-21804	-21310.4	5.44	1106.7	-24826	-24163	541.18	21452
R7	700	-26624	-25830.8	13.25	1287.6	-28614	-28221.1	677.97	23679
R8	800	-30479	- 30133.8	21.43	1676.5	-32460	-32078.3	523.58	18814
R9	900	-35503	- 34707.8	36.10	1528.7	-36839	-36450.7	526.69	21795
R10	1000	-39760	- 39257.8	49.38	1720.5	–42012	-40768.4	637.47	18619

Table 4.7: Results of Left Skewed Problems with data size dependent range

		Local Search Without Repeatation			Local Search With 10 Repeatations			
Test	n	Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-0.07	0.01	108	-0.08	-0.06	0.01	114.8
R2	200	-0.12	0.07	263	-0.16	-0.13	0.07	252.7
R3	300	-0.19	0.46	398	-0.20	-0.18	0.29	402.3
R4	400	-0.24	0.84	559	-0.28	-0.26	0.82	568.8
R5	500	-0.28	1.74	706	-0.42	-0.34	1.69	732.3
R6	600	-0.44	3.57	919	-0.53	-0.43	1.69	732.3
R7	700	-0.65	6.62	1147	-0.65	-0.54	5.21	1075.4
R8	800	-21362	14.44	1389	-96704	-35069.07	10.81	1386.2
R9	900	-37805	18.54	1496	-63108	-32549.87	18.24	1518.4
R10	1000	-53496	37.44	1667	-102794	-61689.95	33.17	1674.3

Table 4.8: Results of Left Skewed Problems with data size dependent range

		Modified Multistart Local Search with 10 restarts				Modified Multistart Local Search with 15 mins time limit			
Test	n	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-49249	-40080	0.01	160.2	-63409	-62480.3	415.02	20477
R2	200	-285189	-252680.65	0.08	313.8	-334084	-327217	417.73	18656.1
R3	300	-794354	-735947.3	0.30	469.3	-858967	-541856.8	562.75	27274
R4	400	-1589148	-1513478.2	0.78	629.7	-1701892	-1654047.9	484.87	17971
R5	500	-2780301	-2572494.8	1.60	792.5	-2929794	-2859745.6	436.27	18280
R6	600	-4372739	-4148791.4	3.06	970	-4427038	-4369890.38	709.04	20992
R7	700	-6445156	-5959532.5	6.36	1113.3	-6451089	-6265127.4	745.66	21018
R8	800	-8909521	-8570011.1	14.65	1274.8	-8862556	-8749544.8	885.74	22085
R9	900	-11936383	-11455318.4	25.34	1439.2	-11989465	-11587678.4	960.93	19650
R10	1000	-15290086	-14771399.3	43.97	1594	–15650460	-15132368.7	1559.05	19585

Table 4.9: Results of Right Skewed Problems with fixed range

		Local Search Without Repeatation			Local Search With 10 Repeatations			
Test	n	Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-0.27	0.01	123	-0.47	-0.28	0.01	117
R2	200	-82	0.08	282	-3946	-1156.2	0.08	316.2
R3	300	-131	0.30	392	-163	-133.3	0.29	316.2
R4	400	-251	0.74	480	-316	-239.4	0.69	476.2
R5	500	-290	1.45	570	-422	-368.2	1.28	578.8
R6	600	-597	3.38	691	-602	-527.4	2.33	669.9
R7	700	-728	5.64	767	-859	-655.2	4.07	789.6
R8	800	-954	9.74	861	-1024	-849.7	7.73	867.3
R9	900	-1145	17.51	979	-1145	-974.5	11.63	970.4
R10	1000	-1266	20.22	1083	-1294	-1187.7	21.30	1052.9

Table 4.10: Results of Right Skewed Problems with fixed range

		Modified Multistart Local Search with 10 restarts				Modified Multistart Local Search with 15 mins time limit			
Test	n	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-2970	–2648.2	0.01	152.3	-3822	-3754	352.71	19059
R2	200	-5896	-5514.1	0.03	359.9	-8251	-8186	344.15	22306
R3	300	-9805	-9200.3	0.16	534.5	-12940	-12865.9	418.95	27391
R4	400	-16848	-16268.9	22.4	15480.4	-17425	-17176.3	569.30	27353
R5	500	-17749	-17300	0.99	978.1	-21398	-20738.5	579.26	25814
R6	600	-21702	-21193.1	1.62	1063.4	-25249	-24302	549.09	21476
R7	700	-26673	-25981.8	8.44	1291.2	-29442	-28529.5	388.79	18263
R8	800	-30565	-30225.7	12.24	1412.1	-33234	-32278.6	732.50	21301
R9	900	-35245	-34761.6	27.12	1756.7	-37037	-36206	185.43	17211
R10	1000	-39832	-39293.2	42.42	1897.4	-41439	-40739.80	532.20	14075

Table 4.11: Results of Right Skewed Problems with data size dependent range

		Local Search Without Repeatation			Local Search With 10 Repeatations			
Test	n	Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-0.06	0.01	102	-0.08	-0.06	0.01	102.7
R2	200	-0.12	0.07	249	-0.13	-0.12	0.07	245.8
R3	300	-0.20	0.35	443	-0.22	-0.19	0.31	405.3
R4	400	-0.25	0.94	588	-0.37	-0.27	0.79	574.9
R5	500	-0.30	2.30	703	-0.39	-0.35	1.83	756.9
R6	600	-0.40	4.75	891	-0.59	-0.43	3.05	901
R7	700	-0.61	8.84	1103	-0.81	-0.60	6.19	1109.4
R8	800	-0.56	11.18	1248	-100132	-17782.82	10.86	1305.1
R9	900	-0.68	21.31	1394	-97247	-37203.23	18.53	1530
R10	1000	-90692	29.22	1695	-120705	-62887.60	31.96	1681.1

Table 4.12: Results of Right Skewed Problems with data size dependent range

		Modified Multistart Local Search with 10 restarts				Modified Multistart Local Search with 15 mins time limit			
Test	n	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-50698	-40129.2	0.01	162.2	-63448	-61862.4	437.03	10000
R2	200	-276980	-263954.9	0.02	311.2	- 327380	-32546.6	408.06	23496
R3	300	-752729	-686431.55	0.10	469.3	-865669	-843593.1	563.13	64040
R4	400	-1601450	-1504792.9	0.25	625.7	-1704412	-1645465.8	408.48	71127
R5	500	-2762451	-2627402.4	0.50	790.2	-2884366	-2785977.9	234.73	54880
R6	600	-4354986	-3931968.65	1.04	946.1	-4571089	-4370460.7	329.29	31085
R7	700	-6290842	-5782424.95	2.13	1128.7	-6509430	-6402371.1	350.78	29300
R8	800	-8680552	-8440357	4.00	1304.9	-9018245	-8698666.2	433.37	28399
R9	900	-11599027	-11301115.3	7.08	1486.2	-11917799	-11529996.8	351.80	36007
R10	1000	-15046710	-14611528.4	10.74	1617.9	-15336168	-14903654.2	253.10	29888

Table 4.13: Results of Negative Correlated Problems

		Local Search Without Repeatation			Local Search With 10 Repeatations			
Test	n	Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	3.28	0.06	82	3.24	3.31	3.31	82.8
R2	200	-2.02	0.04	120	-2.04	-2.03	0.04	119.6
R3	300	-0.54	0.13	148	-0.55	-0.54	0.11	147.5
R4	400	-1.49	0.30	185	-1.50	-1.49	0.26	184.2
R5	500	-8.12	0.45	206	-9.20	-8.48	0.46	208.6
R6	600	-0.79	0.82	252	-0.80	-0.79	0.80	250.2
R7	700	0.57	1.55	332	0.55	0.56	1.50	326.3
R8	800	0.11	2.67	392	0.11	0.11	2.78	377.2
R9	900	-241	3.25	300	-241	-240.5	3.88	306.7
R10	1000	0.63	9.63	458	0.63	0.65	9.38	466.5

Table 4.14: Results of Negative Correlated Problems

		Modified Multistart Local Search with 10 restarts				Modified Multistart Local Search with 15 mins time limit			
Test	n	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-777	-772	0.01	78.3	-786	-785.4	309.03	10600000
R2	200	-2.05	-2.03	0.17	522.4	-2.07	-2.07	543.62	2725009
R3	300	-0.55	-0.55	0.09	1674.3	-0.55	-0.55	412.70	1018020
R4	400	-1.52	-1.51	0.26	2871.6	-1.53	-1.52	451.04	531747
R5	500	-9.76	-8.97	0.69	4300.4	-10.29	-9.98	470.71	317931
R6	600	-0.82	-0.80	1.37	9728.4	-0.82	-0.82	366.85	184881
R7	700	0.51	0.52	886.94	58737.5	0.51	0.52	713.42	198639
R8	800	0.1	0.10	1193.10	57493	0.1	0.10	1269.68	146110
R9	900	-241	-240.6	7.39	675.1	-241	-241	5.97	64225
R10	1000	-978	-970.9	26.73	565.8	-986	-985.1	452.11	48250

Table 4.15: Results of Positive Correlated Problems-increased

		Local Search Without Repeatation			Local Search With 10 Repeatations			
Test	n	Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	0.01	0.01	87	0.00	0.01	0.00	78
R2	200	-0.07	0.05	166	-0.1	-0.03	0.04	127.7
R3	300	0.39	0.14	160	0.38	0.39	0.12	162.8
R4	400	1.17	0.31	171	1.17	1.17	0.25	175.8
R5	500	0.72	0.62	224	0.71	0.71	0.50	225.5
R6	600	-100	0.97	255	-314	-119.7	0.81	245.9
R7	700	-0.37	2.02	360	-0.38	-0.37	1.64	363.3
R8	800	-0.63	3.20	379	-0.64	-0.63	2.72	367.1
R9	900	-1.87	5.49	412	-1.87	-1.86	4.96	413.9
R10	1000	-0.17	16.85	691	-0.17	-0.16	13.83	689.3

Table 4.16: Results of Positive Correlated Problems-increased

		Modified Multistart Local Search with 10 restarts				Modified Multistart Local Search with 15 mins time limit			
Test	n	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	0.00	0.01	0.04	390.2	-0.02	-0.02	592.89	12293927
R2	200	-0.09	-0.05	0.42	1653.7	-0.12	-0.12	407.16	2574864
R3	300	0.38	0.38	2.93	4713.9	0.36	0.37	206.18	940328
R4	400	-16848	-16268.9	22.40	15480.4	1.17	1.17	368.19	470449
R5	500	0.70	0.71	35.45	16033.3	0.7	0.7	502.90	271573
R6	600	-362	-355.6	0.68	330.8	-395	-390.6	713.00	45771
R7	700	-0.42	-0.4	489.28	45100.2	-0.51	-0.51	2227.47	59575
R8	800	-0.7	-0.69	972.94	51103.2	-0.80	-0.8	3209.46	55807
R9	900	-2	-1.99	7.06	48204.6	-2.19	-2.18	662.96	1000
R10	1000	-0.15	-0.14	12.12	88515.7	-0.24	-0.24	1864.93	100000

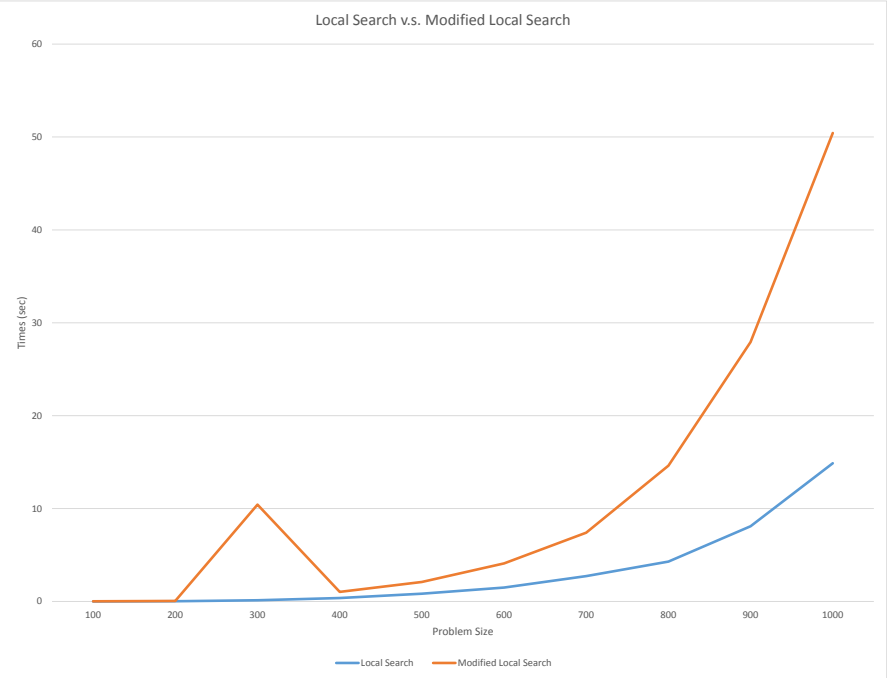
Table 4.17: Results of Positive Correlated Problems-decreased

		Local Search Without Repeatation			Local Search With 10 Repeatations			
Test	n	Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-1.01	0.01	73	-1.01	-1.01	0.01	72.3
R2	200	-2.44	0.05	127	-2.45	-2.44	0.04	126.7
R3	300	-0.49	0.14	158	-0.51	-0.50	0.12	166
R4	400	2.59	0.33	189	2.59	2.60	0.26	188.1
R5	500	2.65	0.73	217	2.61	2.64	0.49	220.5
R6	600	-6.09	1.39	298	-6.09	-5.94	0.92	284.3
R7	700	0.38	1.72	281	0.37	0.38	1.29	283.9
R8	800	-2.43	3.86	368	-2.45	-2.42	2.69	371.8
R9	900	-571	5.15	341	-577	-568.6	3.74	326.3
R10	1000	-0.55	15.26	704	-0.57	-0.55	12.25	699.8

Table 4.18: Results of Positive Correlated Problems-decreased

		Modified Multistart Local Search with 10 restarts				Modified Multistart Local Search with 15 mins time limit			
Test	n	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)	Best Obj.Value	Avg. Obj.Value	Times	Iteration(s)
R1	100	-1.01	-1.01	0.00	209	-1.02	-1.02	534.37	8874797
R2	200	-2.46	-2.45	0.03	868.5	-2.48	-2.48	525.86	2427330
R3	300	-0.52	-0.51	0.08	2174.6	-0.53	-0.53	359.71	1004211
R4	400	-971	-966.2	0.43	292.5	-978	-976.8	398.24	593672
R5	500	-1129	-1125	1.57	337.9	-1136	-1134	525.08	332385
R6	600	-6.33	-6.24	94.49	11362.6	-6.36	-6.31	390.22	169410
R7	700	-572	-566.1	2.47	410.9	-576	-574.3	281.96	89250
R8	800	-2.6	-2.58	723.01	31423.5	-2.62	-2.6	845.94	92590
R9	900	-625	-609.2	15.46	680.1	-625	-621.3	269.37	48693
R10	1000	-349	-209.6	504.95	19429	-349	-314.15	743.31	40659

Figure 4.1: Problem Size v.s. Times of Symmetric Problems with fixed range



From Table 4.1-4.9, we can see that Local Search with ten repetitions finds better heuristic solution compare to Local Search without repetition. Time it takes for Local Search Algorithm to terminate is similar to the time it takes for each local search run in the Local Search with ten repetitions to terminate. From Table 4.10-4.18, we can see that Modified Multistart Local Search with a fifteen minutes time limit gets the best heuristic solution. However, it takes the longest time compared to other local search methods. Modified Multistart Local Search with ten restarts archieved the heuristic solution worse than Modified Multistart Local Search with a 15 minutes time limit but takes a much shorter time.

Overall, Modified Multistart Local Search with fifteen minutes time limit finds the best solution among all the local search algorithms we tested. However, it takes the longest computation time. Modified Multistart Local Search with ten times restart gets the second best solution but it takes longer computation time compare to Local Search with ten repetitions and Local Search without repetition. However, Local Search with ten repetitions and Local Search without repetition bot has significant shorter computation time.

Chapter 5

Conclusion

In this thesis, we studied the unrestricted linear fractional assignment problem (LFAP) which is a generalization of the standard linear fractional assignment problem.

We first present four different mathematical programming formulations of LFAP. Out of these, two are of the type of MILP. We compared these two formulations experimentally using randomly generated data. One of these formulations assume integer data, and this formulation performed better in terms of computational time. The limitation however is restriction to integer data.

We also proposed an exact algorithm to solve LFAP based on Newton's method. In this case, we solve two fractional constrained assignment problems and select the best solution. Note that constrained assignment problem is NP-hard, but in practice, these problems can be solved relatively fast. The resulting algorithm performed better than our algorithms based on MILP formulations. A heuristic version of this algorithm is also developed and compared experimentally with the exact version. The heuristic works almost like the exact algorithm except that a time limit is imposed on the associated constrained assignment problem solved.

Finally, we developed a local search algorithm and two other variations. One of the variation is a multistart version of the local search. The local search uses a simple 2-exchange neighborhood. The other variation attempts to obtain good starting solutions that take advantage of the fractional objective function. Result of extensive experimental study are presented.

We also generated some test instances with various properties. These instances can be used by researches in future to conduct experimental analysis on new algorithms.

Theoretical analysis of heuristics and identify new polynomially solvable special cases of LFAP are interesting and left as topics for future research by interested researchers.

Chapter 6

Bibliography

Bibliography

[1] Y. Anzai. On integer fractional programming. *Journal of the Operations Research Society of Japan*, 17 (1974) 49–66.

[2] E. B. Bajalinov. *Linear-fractional programming theory, methods, applications and software.* Kluwer Academic Publishers, 2004.

[3] E. M. L. Beale. *Fractional programming with zero-one variables.* Springer, 1980

[4] E. M. L. Beale and J. A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. *OR*, 69 (1970) 447-454.

[5] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucuman. Revista A,* 5:147-151, 1946.

[6] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete applied mathematics*, 123 (2002) 155–225.

[7] R. Burkard. *Assignment Problems.* SIAM, 2012

[8] R. Chandrasekaran. Minimal ratio spanning trees. *Networks*, 7 (1977) 335–342.

[9] A. Charnes and W. W. Cooper. Programming with linear fractional functionals. *Naval Research Logistics*, 9 (1962) 181–186.

[10] G. D. H. Claassen. Mixed integer (0–1) fractional programming for decision support in paper production industry. *Omega*, 43 (2014) 21–29.

[11] G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, 1951.

[12] W. Dinkelbach. On nonlinear fractional programming. *Management science*, 13 (1967) 492–498.

[13] R. M. Essaid. Incorporating a Modified Uniform Crossover and 2-Exchange Neighborhood Mechanism in a Discrete Bat Algorithm to Solve the Quadratic Assignment Problem. *Egyptian Informatics Journal*, 18 (2017) 221-232.

[14] M. R. Garey and D. S. Johnson. “strong”np-completeness results: Motivation, examples, and implications. *Journal of the ACM*, 25 (1978) 499–508.

[15] A. M. Geoffrion. Integer programming by implicit enumeration and Balas' method. *SIAM Review*, 9 (1967) 178–190.

[16] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem – part ii. *Operations research*, 11 (1963) 863–888.

[17] F. Glover and E. Woolsey. Converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22 (1974) 180–182.

[18] D. Granot and F. Granot. On integer and mixed integer fractional programming problems. *Annals of Discrete Mathematics*, 1 (1977) 221–231.

[19] D. Granot and F. Granot. On solving fractional (0, 1) programs by implicit enumeration. *INFOR: Information Systems and Operational Research*, 14 (1976) 241–249.

[20] M. Grunspan and M. E. Thomas. Hyperbolic integer programming. *Naval Research Logistics*, 20 (1973) 341–356.

[21] P.L. Hammer and S. Rudeanu. Boolean methods in operations research and related areas. *Springer Science & Business Media*, 2012.

[22] P. Hansen, M. V. P. de Aragão, and C. C. Ribeiro. Hyperbolic 0–1 programming and query optimization in information retrieval. *Mathematical Programming*, 52 (1991) 255–263.

[23] E. Heinen. *Das Zielsystem der Unternehmung: Grundlagen betriebswirtschaftlicher Entscheidungen*. Springer-Verlag, 2013

[24] J.R. Isbell and W.H. Marlow. Attrition games. *Naval Research Logistics*, 3 (1956) 71–94.

[25] R. Jagannathan. On some properties of programming problems in parametric form pertaining to fractional programming. *Management Science*, 12 (1966) 609–615.

[26] S. N. Kabadi and A. P. Punnen. A strongly polynomial simplex method for the linear fractional assignment problem. *Operations Research Letters*, 36 (2008) 402–407.

[27] K. G. Murty, *Linear programming*. John Wiley and Sons, 1983

[28] H. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics*, 2 (1955) 83–97.

[29] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 2001.

[30] Eugene L Lawler. Optimal cycles in doubly weighted directed linear graphs. *Theory of Graphs*, pages 209–232, 1966.

[31] B. Martos. The direct power of adjacent vertex programming methods. *Management Science*, 12 (1965) 241–252.

[32] B. Martos and V. Whinston. Hyperbolic programming. *Naval Research Logistics*, 11 (1964) 135–155.

[33] T. Matsui, Y. Saruwatari, and M Shigeno. *An analysis of dinkelbach's algorithm for 0-1 fractional programming problems*, 1992.

[34] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4 (1979) 414–424.

[35] Y. E. Nesterov and A.S Nemirovskii. An interior-point method for generalized linear-fractional programming. *Mathematical Programming*, 69 (1995) 177–204.

[36] C. H. Papadimitrious and K. Steiglitz *Combinatorial Optimization: Algorithms and Complexity.* Courier Corporation, 1998

[37] A. Punnen, and Y. Aneja. A tabu search algorithm for the resource-constrained assignment problem. *Journal of the Operational Research Society*, 46 (1995) 214-220.

[38] T. Radzik. Complexity in numerical optimization. *World Scientific*, 1993.

[39] T. Radzik. Newton's method for fractional combinatorial optimization. *Foundations of Computer Science, 1992. Proceedings. 33rd Annual Symposium on*, 1992.

[40] S. Raff. Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research*, 10 (1983) 63–211.

[41] P. Robillard. (0, 1) hyperbolic programming problems. *Naval Research Logistics*, 18 (1971) 47–57.

[42] S. C. Sharma and A. Bansal. A integer solution of fractional programming problem. *General Mathematics Notes*, 4 (2011) 1–9.

[43] M. Shigeno, Y. Saruwatari, and T. Matsui. An algorithm for fractional assignment problems. *Discrete Applied Mathematics*, 56 (1995) 333–343.

[44] I. M. Stancu-Minasian. *Fractional programming: theory, methods and applications.* Springer Science & Business Media, 2012.

[45] A. Vijay. A Lagrangean-relaxation method for the constrained assignment problem. *Computers and operations research*, 12 (1985) 97-106.

[46] W. L. Winston, M. Venkataramanan, and J. B. Goldberg. *Introduction to mathematical programming.* Thomson/Brooks/Cole Duxbury; Pacific Grove, CA, 2003.

[47] M. J. Yao, H. S. Soewandi, and S. E. Elmaphraby. Simple heuristics for the two machine openshop problem with blocking. *Journal of the Chinese Institute of Industrial Engineers*, 17 (2000) 537-547.