The Multiplicative Assignment Problem

by

Jingbo Tian

B.Sc., Beijing Electronic Science and Technology Institute, 2012

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

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Spring 2017

Approval

Name: **Jingbo Tian**

Degree: **Master of Science (Mathematics)**

Title: ***The Multiplicative Assignment Problem***

Examining Committee: **Chair:** Randall Pyke
Senior Lecturer

Abraham Punnen
Senior Supervisor
Professor
Department of Mathematics

George Zhang
Supervisor
Professor
Beedie School of Business

Ramesh Krishnamurti
Internal Examiner
Professor
Department of Computing Science

Date Defended: July 6, 2017

Abstract

The quadratic assignment problem (QAP) is an extensively studied combinatorial optimization problem. The special case of QAP where the cost matrix is of rank one is called the multiplicative assignment problem (MAP). MAP is not well studied in literature, particularly in terms of experimental analysis of algorithms. In this thesis we present some mixed integer linear programming formulations and compare their selective strength using experimental analysis. We also present exact and heuristic algorithms to solve MAP. Our heuristic algorithms include improvements in existing FPTAs, as well as local search and tabu search enhancements. Results of extensive experimental analyses are also reported.

Keywords: Quadratic assignment; multiplicative assignment; linearization; constrained assignment; heuristics; local search; tabu search; CPLEX; C++

Dedication

I dedicate this work to my family and the one who will become my family.

Acknowledgements

I am deeply appreciative of the both academic and financial support from my supervisor, Dr. Abraham Punnen. Without his profound empathy and benevolence, this thesis would not be possible.

I am also much obliged to everyone that gives me help along the way. Amongst them, thanks go to Dr. Yang Wang for her assistance to me on local search and tabu search. I also thank Dr. Aihua Yin and Wilson Lin for assistance during early stages of this work. Especially, I am highly grateful to Dr. George Zhang for partial financial support and academic supervision during early stages of my graduate study.

It has been a long journey. That fact only makes these people greater.

Table of Contents

List of Tables

Chapter 1

Introduction

1.1 The Quadratic Assignment Problem

The quadratic assignment problem (QAP) is one of the most well-studied combinatorial optimization problems. The problem was first introduced by Koopmans and Beckmann in 1957 to model the facility location problem [52]. Since then a large number of real-world problems that can be mathematically modelled by the QAP have been identified in literature. In addition to its usefulness in a wide range of realistic contexts, QAPs are also of theoretical importance in that many significant and interesting combinatorial optimization problems can be formulated as QAPs, such as travelling salesman problem. QAPs still stand as a very hard problem from a computational perspective, both in theory and in practice. Theoretically, QAPs are NP-hard and NP-hard to approximate with a constant performance ratio. In practice, QAP instances of size greater than 20 are normally intractable. It is exceptional considering large-scale instances of some other well-known combinatorial problems are practically solvable. For example, travelling salesman instances of size in thousands can be solved at a reasonable time in practice. Due to the above reasons, the QAP have been an active research topic for over the past 60 years, and it still attracts considerable attention from academia as well as practitioners.

An integer programming formulation of the problem can be given as follows

$$\text{Minimize } \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} q_{ijkl}x_{ij}x_{kl}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \; i = 1, ..., n, \tag{1.1}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \; j = 1, ..., n, \tag{1.2}$$

$$x_{ij} \in \{0, 1\}, \; i, j = 1, ..., n. \tag{1.3}$$

The problem can also be viewed as a permutation problem, i.e. let $N = \{1, 2, ..., n\}$ and $\mathcal{F}$ be the family of all permutations of N. Then the QAP can be written as

$$\underset{\pi\in\mathcal{F}}{\text{Minimize}} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij\pi(i)\pi(j)}$$

The four dimensional array $Q = (q_{ijkl})$ completely represents an instance of the QAP. The array Q can also be viewed as an $n^2 \times n^2$ matrix and hence we refer to Q as the cost matrix associated with the QAP.

The QAP defined above is sometimes called *Lawler QAP* named after Lawler [55], who originally proposed this general version of the model in 1963.

A special case of the QAP, known as $Koopmans - Beckmann\ QAP$ [52] introduced in 1957 is perhaps the most well studied version of the problem. In this case two $n \times n$ real matrices $A = (a_{ij})$ and $B = (b_{ij})$ are given and we want to find a permutation $\pi \in \mathcal{F}$ such that $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{\pi(i)\pi(j)}$ is minimized.

An integer programming formulation of $Koopmans - Beckmann\ QAP$ can be obtained by replacing $q_{ijkl} = a_{ik}b_{jl}$ in the QAP.

The QAP model generalizes some other known optimization problems. For example, the well known the Traveling Salesman Problem (TSP) can be represented as a $Koopmans - Beckmann\ QAP$ [70]. An instance of the TSP can be described as finding a minimum weight Hamiltonian cycle in a complete graph on n nodes. The transformation between the TSP and the QAP can be done by choosing the matrix A as the distance matrix in the graph on which the TSP is defined and the second matrix as the $n \times n$ adjacency matrix $H_n = (h_{ij})$ of a directed Hamiltonian cycle [21].

To see another application of the QAP, let us consider the facility location problem. We are given n facilities and n locations. Let a_{ij} be the flow of materials moving from facility i to facility j, b_{ij} represents the distance from location i to location j. The cost of placing facility $\pi(i)$ at location i and facility $\pi(j)$ at location j is $a_{\pi(i)\pi(j)}b_{ij}$. The goal is to assign facilities to locations so that the overall cost is minimum. This clearly corresponds to *Koopmans – Beckmann QAP*. Concrete applications of the QAP in facility location include campus planning [26] and the design of a hospital layout [29].

Let us consider the hospital layout problem as a specific practical example of QAPs. Alwalid Elshafei studies a real hospital, the Ahmed Mather Hospital, which is located in Cairo, Egypt [29]. The problem can be succinctly stated as to locate hospital departments in order to minimize the total distance travelled by patients in total. The parameters are the yearly flow f_{ik} between each pair of departments (i and k) and the distance d_{jq} between each pair of locations (j and q). Each department needs to occupy a location and each location can only house a department. Let the binary decision variable y_{ij} denote whether locating department i to location j. An integer programming formulation of the problem is presented as follows:

$$\text{Minimize } \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{q=1}^{n} f_{ik} d_{jq} y_{ij} y_{kq}$$

Subject to

$$\sum_{j=1}^{n} y_{ij} = 1, \ i = 1, ..., n, \tag{1.4}$$

$$\sum_{i=1}^{n} y_{ij} = 1, \ j = 1, ..., n, \tag{1.5}$$

$$y_{ij} \in \{0, 1\}, \ i, j = 1, ..., n. \tag{1.6}$$

For more applications of the model we refer to [17].

The QAP is NP-hard since it contains NP-hard problems such as the TSP as a special case. In fact, it is possible to show that finding a $(1+\epsilon)$ approximate solution to the QAP in polynomial time is also a difficult task for any $\epsilon > 0$, when $P = NP$. More formally,

Theorem 1. *[71] The quadratic assignment problem is NP-hard. Further, for any $\epsilon > 0$, the existence of a polynomial time $1 + \epsilon -$ approximation algorithm for the QAP implies $P = NP$.*

Various approaches are used in literature to solve the QAP. In view of Theorem 1, such algorithms are normally of enumerative type. This include:

- Branch and bound algorithms [17, 34, 35, 53, 55, 61]
- Branch and cut algorithms [17, 48, 62]
- Cutting plane algorithms [7–9, 11, 12, 17]
- and a Combination of these approaches [17]

While exact algorithms provide guaranteed optimal solutions, their applicability is limited since solving large scale problems using such algorithms are difficult, if not impossible. A more practical approach is to solve the problem by heuristics, which produce near optimal solutions. Most of the standard heuristic paradigms have been tested in the context of the QAP. These include:

- Construction methods [15, 17, 35, 60]
- Local search [1, 17]
- Tabu search [10, 17, 22, 37–39, 73, 74]
- Ant systems [17, 27, 28, 33, 57]
- Genetic algorithms [6, 17, 25, 31, 40, 45, 75], among others.

When the underlying cost matrix is specially structured, the QAP may be solvable in polynomial time. This is yet another major research area related to the QAP.

Perhaps the simplest polynomially solvable special case is when Q is a diagonal matrix. In this case, the QAP reduces to the standard linear assignment problem. Erdogan [30] identified a much larger class of the QAP that can be solved as a linear assignment problem. This leads to the concept of linearizable instances of the QAP. Kabadi and Punnen [47] characterized all such instances. For various special cases of the QAP that can be solved in polynomial time, we refer to [20].

1.2 The Multiplicative Assignment Problem

The multiplicative assignment problem (MAP) is a special case of the QAP. Let A and B be the two coefficient matrices as defined before, and $C = (c_{ij})_{n \times n}$ be another $n \times n$ matrix. For any $n \times n$ binary matrix $X = (x_{ij})_{n \times n}$, define

$$A(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_{ij},\ B(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} x_{ij},\ \text{and } C(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}.$$

Then the MAP can be formulated as a 0-1 integer program problem

$$\text{Minimize } A(X)B(X) + C(X) \tag{1.7}$$

$$\text{Subject to}$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ i = 1, ..., n, \tag{1.8}$$

$$\sum_{i=1}^{n} x_{ij} = 1,\ j = 1, ..., n, \tag{1.9}$$

$$x_{ij} \in \{0,1\}, \; i,j = 1,...,n. \tag{1.10}$$

Define

$$q_{ijkl} = \begin{cases} a_{ij}b_{kl} & \text{if } (i,j) \neq (k,l), \\ a_{ij}b_{ij} + c_{ij} & \text{if } (i,j) = (k,l). \end{cases}$$

then the resulting QAP is the same as the MAP. To the best of our knowledge, the MAP is not well studied in the literature. Punnen [65] showed that the MAP is NP-hard. Linearizable instances of the MAP have been characterized by Punnen and Kabadi [68].

When C is the zero matrix, the resulting MAP is called homogeneous MAP which is denoted by MAP(H). If $A(x) \geq 0$ and $B(x) \geq 0$ for all $x \in \mathcal{F}$, the resulting MAP is called non-negative MAP and is denoted by MAP^+. The homogeneous version of MAP^+ is called non-negative homogeneous MAP and is denoted by $MAP^+(H)$. $MAP^+(H)$ can be solved using a fully polynomial approximation scheme (FPTAS) as established by [41]. These different variations are crucial in the applicability of some of the algorithms.

Since the multiplicative assignment problem (MAP) is a special case of the QAP, all the algorithms studied for the QAP can be used to solve the MAP.

1.3 Combinatorial Optimization with Product Objective

The MAP is also a special case of the general combinatorial optimization problem with product objective function, (COPP) [41]. Thus the general results available for COPP are applicable to the MAP as well.

Let $\mathcal{E} = \{1, 2, ..., n\}$ be a ground set and $\mathcal{F}$ be a family of subsets of $\mathcal{E}$. For each $e \in \mathcal{E}$, a cost c_e and a weight d_e are given. Then the COPP is to

$$\begin{aligned} &\text{Minimize } \Big(\sum_{e\in\mathcal{S}} c_e\Big)\Big(\sum_{e\in\mathcal{S}} d_e\Big) \\ &\text{Subject to} \\ &\qquad \mathcal{S} \in \mathcal{F} \end{aligned} \tag{1.11}$$

For each solution $\mathcal{S} \in \mathcal{F}$, we can assign a 0-1 vector $\mathbf{x} = (x_1, ..., x_n)$ such that

$$x_i = \begin{cases} 1 & \text{if } i \in \mathcal{S}, \\ 0 & \text{if } i \notin \mathcal{S}. \end{cases}$$

The $\mathbf{x}$ is called the *characteristic vector* of $\mathcal{S}$.

Also, if we consider a characteristic vector as a solution point, let $F(\mathbf{x})$ be the convex hull of all characteristic vectors of elements of $\mathcal{F}$. Then the COPP can be written as

$$COPP1: \quad \text{Minimize } (\sum c_e^T x_e)(\sum d_e^T x_e)$$
$$\text{Subject to}$$
$$\mathbf{x} \in F(\mathbf{x}) \tag{1.12}$$

If $c_e, d_e \geq 0$, then the resulting $COPP1$ is denoted by $COPP1^+$.

When $\mathcal{E}$ is the edge set of a complete bipartite graph G and $\mathcal{F}$ is the collection of all perfect matchings in G, then the COPP reduces to MAP(H).

Other examples of the COPP include the minimum product spanning tree problem where $\mathcal{F}$ is selected as spanning trees of a graph G [41], minimum product s-t cut problem where $\mathcal{F}$ is selected as all s-t cuts in a graph G [41], shortest path problem where $\mathcal{F}$ is selected as all paths between node s and t, etc.

For more details on the COPP, we refer to the papers [41, 59, 65, 76]. Let us now review some of the results discussed in [41] regarding the COPP, the one relevant to our study.

Theorem 2 (Goyal, Genc-Kaya and Ravi [41], 2008)**.** *$COPP1^+$ can be solved by a polynomial time $1 + \epsilon$ polynomial approximation algorithm, say A, for any $\epsilon > 0$, whenever the polytope P representing $F(\mathbf{x})$ is available. Furthermore, $\mathcal{A}$ returns a solution that is an extreme point of $F(\mathbf{x})$.*

Consider the parametric problem $\Pi(B)$ defined as

$$\text{Minimize } (\sum c_e^T x_e)$$
$$\text{Subject to}$$
$$\sum d_e^T x_e \leq B \tag{1.13}$$
$$\mathbf{x} \in F(\mathbf{x}) \tag{1.14}$$

where B is a given parameter.

Theorem 3 (Konno and Kuno [51], 1992)**.** *The function $f_1(\mathbf{x}) = (\sum_{i=1}^n c_i x_i)(\sum_{i=1}^n d_i x_i)$ is quasi-concave when $\sum_{i=1}^n c_i x_i \geq 0, \forall \mathbf{x} \in F(\mathbf{x})$ and $\sum_{i=1}^n d_i x_i \geq 0, \forall x \in F(\mathbf{x})$.*

Proof. Let $\boldsymbol{x^1}$, $\boldsymbol{x^2} \in F(\mathbf{x})$. The theorem is equivalent to showing

$$[\boldsymbol{c^t}(\lambda \boldsymbol{x^1} + (1-\lambda)\boldsymbol{x^2})][\boldsymbol{d^t}(\lambda \boldsymbol{x^1} + (1-\lambda)\boldsymbol{x^2})] \geq min\{(\boldsymbol{c^t x^1})(\boldsymbol{d^t x^1}), (\boldsymbol{c^t x^2})(\boldsymbol{d^t x^2})\} \tag{1.15}$$

for all $\lambda \in [0,1]$. Let us denote $c_i = \boldsymbol{c^t x^i}, d_i = \boldsymbol{d^t x^i}, i = 1,2$, and assume without loss of generality that

$$c_1 d_1 \le c_2 d_2. \tag{1.16}$$

We need to show that

$$\varphi(\lambda) \equiv [\lambda c_1 + (1-\lambda)c_2][\lambda d_1 + (1-\lambda)d_2] - c_1 d_1 \ge 0$$

for all $\lambda \in [0,1]$. Algebra shows that

$$\varphi = (1-\lambda)[(c_2 d_2 - c_1 d_1) - \lambda(c_1 d_1 + c_2 d_2 - c_1 d_2 - c_2 d_1)].$$

Because $1 - \lambda \ge 0$, it suffices to prove that

$$\psi(\lambda) \equiv (c_2 d_2 - c_1 d_1) - \lambda(c_1 d_1 + c_2 d_2 - c_1 d_2 - c_2 d_1) \ge 0,$$

for all $\lambda \in [0,1]$, which is equivalent to show that

$$\psi(0) = c_2 d_2 - c_1 d_1 \ge 0, \tag{1.17}$$

$$\psi(1) = -2c_1 d_1 + c_1 d_2 + c_2 d_1 \ge 0. \tag{1.18}$$

because $\psi(\lambda)$ is a linear function of λ. (1.17) holds by assumption (1.16). Now we prove (1.18) by contradiction. We assume that $\psi(1) < 0$. Then the following inequality must hold:

$$c_1 d_2 + c_2 d_1 < 2c_1 d_1 \le 2c_2 d_2. \tag{1.19}$$

Because $\sum_{i=1}^n c_i x_i \ge 0, \forall \mathbf{x} \in F(\mathbf{x})$ and $\sum_{i=1}^n d_i x_i \ge 0, \forall x \in F(\mathbf{x})$, c_i, d_i are all nonnegative. Thus (1.19) implies that $c_1 > 0$, otherwise $c_1 d_2 + c_2 d_1 = c_2 d_1 > 0 = 2c_1 d_1$. Due to (1.19), we have

$$d_2 + \frac{c_2}{c_1} d_1 < 2d_1 \le 2\frac{c_2}{c_1} d_2.$$

Let us denote $\alpha = \frac{c_2}{c_1}$, we have

$$d_2 \ge (1/\alpha) d_1 \tag{1.20}$$

Because $d_2 + \alpha d_1 < 2d_1$ and (1.20), we have

$$(1/\alpha) d_1 + \alpha d_1 < 2d_1,$$

which is equivalent to

$$(1/\alpha) + \alpha < 2. \tag{1.21}$$

(1.21) is obviously a contradiction since $\alpha + 1/\alpha \geq 2$ for all $\alpha \geq 0$. Thus $\psi(1) \geq 0$ is proved. $\square$

The above proof was taken from [51].

Theorem 4 (Bertsekas, Nedic and Ozdaglar [14], 2003)**.** *The minimum of a quasi-concave function over a compact convex set is attained at an extreme point of the set.*

Thus there always exists an optimal solution for $COPP1$ which is an extreme point of F(x).

For fixed B, an optimal solution to $COPP1$ does not need to be a feasible solution to $\Pi(B)$. Let $\tilde{\mathbf{x}}$ be an optimal solution $\Pi(B)$. Then $\tilde{\mathbf{x}}$ could be either an extreme point of $\mathrm{F}(\mathbf{x})$ or an interior point of $\mathrm{F}(\mathbf{x})$. If $\tilde{x}$ is an interior point of F(x), it is possible to find an extreme point $\hat{\mathbf{x}}$ of $\mathrm{F}(\mathbf{x})$ such that $f_1(\hat{\mathbf{x}}) \leq f_1(\tilde{\mathbf{x}})$. Further,

Lemma 1. *Let $\tilde{x}(B)$ be a basic optimal solution for some $B > 0$. There exists an extreme point $x \in extr(P)$ such that*

$$(\mathbf{c}^T\mathbf{x}) \cdot (\mathbf{d}^T\mathbf{x}) \leq (\mathbf{c}^T\tilde{\mathbf{x}}(B)) \cdot B$$

where $\mathbf{c} = (c_1, ..., c_n)$ and $\mathbf{d} = (d_1, ..., d_n)$. The proof of the lemma can be found in [41].

The algorithm of [41] solves a sequence of problem of the type $\Pi(B)$, for different values of B, identify an extreme point solution, if the result is a fractional solution, and choose the overall best solution. The values of B are selected depending on the designed accuracy of the solution to be produced.

The readers who are interested in more details about the algorithm are directed to [41].

Mittal and Schulz [59] proposed FPTAS for another class of problem, that includes $COPP1^+$. We omit the details and an interested reader is referred to [59]. But the condition imposed is not known to be applicable for MAP. $COPP1^+$ can also be solved using the approach of [76].

1.4 Contributions of the Thesis

In this thesis we study the MAP. To the best of our knowledge, the MAP is not studied systematically from an experimental point of view. We develop exact and heuristic algorithms for solving the MAP. Extensive experimental results are provided. The thesis is arranged as follows:
In Chapter 2, we adapt directly some of the known linearization formulations of the QAP,

and apply them to the MAP [2, 32, 49, 55] and study experimentally the relative merits of these formulation in the context of the MAP. We then give two new formulations, exploiting the special structure of the MAP. Experimental results with these linearization are also given.

In Chapter 3, we focus on an exact algorithm and study the theoretical foundation underlying the algorithm. Experimental results are also given.

In Chapter 4, we study heuristic algorithms applied on the MAP. To start with, we provide the basic swap-based local search algorithms. On the basis of it, tabu search is studied and applied. Extensive experiments with all algorithms discussed are run and related results are provided.

Concluding remarks are presented in Chapter 5. Experimental outcomes are tabulated and presented in an appendix.

Chapter 2

Linearizations

2.1 Introduction

Recall that the MAP is formulated as an integer program as follows.

$$\text{Minimize } \left(\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_{ij}\right)\left(\sum_{k=1}^{n}\sum_{l=1}^{n} b_{kl}x_{kl}\right) + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\begin{aligned}
&\text{Subject to}\\
&\qquad \sum_{j=1}^{n} x_{ij} = 1,\ \ i = 1, ..., n,\\
&\qquad \sum_{i=1}^{n} x_{ij} = 1,\ \ j = 1, ..., n,\\
&\qquad x_{ij} \in \{0, 1\},\ \ i, j = 1, ..., n.
\end{aligned}$$

Simplifying the objective function and using the fact that $x_{ij}^2 = x_{ij}$, we get the quadratic assignment problem

$$MAP1\ \ \text{Minimize } \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} q_{ijkl}x_{ij}x_{kl} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\begin{aligned}
&\text{Subject to}\\
&\qquad \sum_{j=1}^{n} x_{ij} = 1,\ \ i = 1, ..., n,\\
&\qquad \sum_{i=1}^{n} x_{ij} = 1,\ \ j = 1, ..., n,\\
&\qquad x_{ij} \in \{0, 1\},\ \ i, j = 1, ..., n.
\end{aligned}$$

where $q_{ijkl} = a_{ij}b_{kl}$.

This formulation is exactly the Lawler QAP [55], with the special structure that $q_{ijkl} = a_{ij}b_{kl}$. Thus the matrix $Q = (q_{ijkl})_{n^2 \times n^2}$ is of rank one [19].

In this chapter we first study reformulating the above binary quadratic program into integer linear programs using various methods and perform experimental analysis to assess the efficacy of these formulations.

Considering the close similarity between Lawler QAP and the above formulation, we first adopt well-known linearizations of Lawler QAP to our special case. We then introduce two new formulations that exploit the problem structure of the MAP.

Experimental analysis will be conducted to assess the efficiency of these formulations with the following objectives:

- Investigate the strength of the LP relaxations
- Effect of solving the problem using CPLEX as an exact algorithm
- Effect of using CPLEX as a heuristic algorithm with specified time limit

Although comparative studies of various linearization of Lawler QAP as well as of the Koopman-Beckman QAP instances are known [16, 17], these linearizations are not studied in the context of rank one special case, which is precisely MAP. Also comparisons with CPLEX in the form of time restricted heuristics are also not known. In addition, we also have two linearizations which exploit the structure of the MAP that were not studied in the past.

2.2 Traditional Linearizations

We first consider a linearization introduced by Lawler [55] for the general QAP, stated it in the context of MAP by simply replacing q_{ijkl} by $a_{ij}b_{kl}$ in the objective function. The formulation is given below:

$$MILP1 \quad \text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij}b_{kl}y_{ijkl} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2.1}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \; i = 1, ..., n, \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \; j = 1, ..., n, \tag{2.3}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} y_{ijkl} = n^2, \; i, j, k, l = 1, ..., n, \tag{2.4}$$

$$x_{ij} + x_{kl} - 2y_{ijkl} \geq 0, \;\; i,j,k,l = 1,2,...,n, \tag{2.5}$$

$$y_{ijkl} \in \{0,1\}, \;\; i,j,k,l = 1,2,...,n, \tag{2.6}$$

$$x_{ij} \in \{0,1\}, \;\; i,j = 1,2,...,n. \tag{2.7}$$

Note that it contains $n^4 + 2n^2$ variables and $n^4 + 2n^2 + 1$ constraints.

Now we demonstrate the validity the Lawler linearization. Firstly, we show that $y_{ijkl} = 1$ if and only if $x_{ij} = 1$ and $x_{kl} = 1$. If $y_{ijkl} = 1$, then according to the constraint (2.5) we have

$$x_{ij} + x_{kl} \geq 2. \tag{2.8}$$

And due to the binary constraint (2.7), we know that

$$x_{ij} \leq 1. \tag{2.9}$$

Combining (2.8) and (2.9), we conclude that $x_{ij} = 1$ and $x_{kl} = 1$.

Now we will prove the other direction of the statement. According to the binary constraint (2.6), we have

$$y_{ijkl} \leq 1. \tag{2.10}$$

From the constraint (2.5) and (2.7), we know that $y_{ijkl} = 1$ only if $x_{ij} = 1$ and $x_{kl} = 1$.

According to the constraint (2.4), there need to have n^2 out of n^4 y_{ijkl} being one. From (2.2) (or (2.3)) it follows that there are n pairs i, j for which $x_{ij} = 1$. Therefore there are n^2 quadruples i, j, k, l such that $x_{ij} = 1$ and $x_{kl} = 1$. Since $x_{ij} = 1$ and $x_{kl} = 1$ are necessary for $y_{ijkl} = 1$, and there are exactly n^2 when $x_{ij} = 1$ and $x_{kl} = 1$, we can conclude that $x_{ij} = 1$ and $x_{kl} = 1$ sufficiently lead to $y_{ijkl} = 1$.

Now we show that $x_{ij} = 0$ or $x_{kl} = 0$ leads to $y_{ijkl} = 0$.

According to constraint (2.5), if $x_{ij} = 0$, we have

$$x_{kl} \geq 2y_{ijkl}$$

And due to constraint (2.7), we know that

$$2y_{ijkl} \leq 1$$

Because of (2.6), y_{ijkl} must be 0.

In a similar way, we can prove that $x_{kl} = 0$ leads to $y_{ijkl} = 0$.

Therefore, we have demonstrated that $x_{ij} = 0$ or $x_{kl} = 0$ leads to $y_{ijkl} = 0$.

Kaufmann and Broeckx [49] also introduced a linearization for the general QAP. We adapt this model to MAP, by substituting $q_{ijkl} = a_{ij}b_{kl}$.

Let

$$w_{ij} = a_{ij}x_{ij}\sum_{k=1}^{n}\sum_{l=1}^{n} b_{kl}x_{kl}$$

Note that

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} q_{ijkl}x_{ij}x_{kl} &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_{ij}\sum_{k=1}^{n}\sum_{l=1}^{n} b_{kl}x_{kl} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\end{aligned}$$

Then they introduce n^2 constants $d_{ij} = \sum_{k=1}^{n}\sum_{l=1}^{n} c_{ijkl}, \forall i, j$. Taken together, they present the linearized equivalent of MAP1

$$MILP2 \quad \text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

Subject to

$$\begin{aligned}&\sum_{j=1}^{n} x_{ij} = 1, \;\; i = 1, ..., n, \\ &\sum_{i=1}^{n} x_{ij} = 1, \;\; j = 1, ..., n, \\ &d_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij}b_{kl}x_{kl} - w_{ij} \leq d_{ij}, \;\; i, j = 1, 2, ..., n, \\ &w_{ij} \geq 0, \;\; i, j = 1, 2, ..., n, \\ &x_{ij} \in \{0, 1\}, \;\; i, j = 1, 2, ..., n.\end{aligned}$$

The validity of this linearization follows from [49]. Note that it contains n^2 real variables, n^2 binary variables and $n^2 + 2n$ constraints.

Frieze and Yadegar introduced another linearization [32] by replacing the products $x_{ij}x_{kl}$ of the binary variables by continuous variables y_{ijkl}. We can adapt their strategy by setting $y_{ijkl} = x_{ij}x_{kl}$ and deriving the following mixed integer linear programming formulation for MAP1.

$$MILP3 \quad \text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij}b_{kl}y_{ijkl} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \ \ i = 1, ..., n,$$
$$\sum_{i=1}^{n} x_{ij} = 1, \ \ j = 1, ..., n,$$
$$\sum_{i=1}^{n} y_{ijkl} = x_{kl}, \ \ j, k, l = 1, ..., n,$$
$$\sum_{j=1}^{n} y_{ijkl} = x_{kl}, \ \ i, k, l = 1, ..., n,$$
$$\sum_{k=1}^{n} y_{ijkl} = x_{ij}, \ \ i, j, l = 1, ..., n,$$
$$\sum_{l=1}^{n} y_{ijkl} = x_{ij}, \ \ i, j, k = 1, ..., n,$$
$$y_{ijij} = x_{ij}, \ \ i, j = 1, ..., n,$$
$$0 \leq y_{ijkl} \leq 1, \ \ i, j, k, l = 1, ..., n,$$
$$x_{ij} \in \{0, 1\}, \ \ i, j = 1, 2, ..., n.$$

This mixed integer program includes n^4 real variables, $2n^2$ binary variables and $n^4 + 4n^3 + n^2 + 2n$ constraints. The correctness of this program is given in [32].

Adams and Johnson [2] introduced another similar linearization:

$$MILP4 \ \ \text{Minimize} \ \ \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij} b_{kl} y_{ijkl} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \ \ i = 1, ..., n,$$
$$\sum_{i=1}^{n} x_{ij} = 1, \ \ j = 1, ..., n,$$
$$\sum_{i=1}^{n} y_{ijkl} = x_{kl}, \ \ j, k, l = 1, ..., n,$$
$$\sum_{j=1}^{n} y_{ijkl} = x_{kl}, \ \ i, k, l = 1, ..., n,$$
$$y_{ijkl} = y_{klij}, \ \ i, j, k, l = 1, ..., n,$$
$$y_{ijkl} \geq 0, \ \ i, j, k, l = 1, ..., n,$$
$$x_{ij} \in \{0, 1\}, \ \ i, j = 1, 2, ..., n.$$

The correctness of this linearization is shown in [2] and formulation contains n^2 binary variables, n^4 continuous variables and $n^4 + 2n^3 + 2n$ constraints without counting nonnegativity constraints.

Also the well known reformulation-linearization technique (RLT) [72] could be readily modified to obtain a formulation for MAP as.

$$
\begin{aligned}
MILP5: \quad & \text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij}b_{kl}y_{ijkl} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
& \text{Subject to} \\
& \qquad \sum_{j=1}^{n} x_{ij} = 1, \ i = 1, ..., n, \\
& \qquad \sum_{i=1}^{n} x_{ij} = 1, \ j = 1, ..., n, \\
& \qquad y_{ijkl} \leq x_{ij}, \ i, j, k, l = 1, ..., n, \\
& \qquad y_{ijkl} \leq x_{kl}, \ i, j, k, l = 1, ..., n, \\
& \qquad y_{ijkl} \leq x_{ij} + x_{kl} - 1, \ i, j, k, l = 1, ..., n, \\
& \qquad 0 \leq y_{ijkl} \leq 1, \ i, j, k, l = 1, ..., n, \\
& \qquad x_{ij} \in \{0, 1\}, \ i, j = 1, ..., n.
\end{aligned}
$$

For correctness of the formulation we refer to [72] and this linearized program has n^4 real variables and n^2 binary variables and $3n^4 + 2n$ constraints.

2.3 New Linearizations[1]

In this section we present two new linearizations designed to exploit the structure of MAP.

Our first linearization discussed below is inspired by [36, 48]. As defined in Chapter 1, $A = (a_{ij})_{n\times n}$ and $B = (b_{ij})_{n\times n}$ and $C = (c_{ij})_{n\times n}$ are three $n \times n$ matrices. For any $n \times n$ binary matrix $X = (x_{ij})_{n\times n}$, define

$$A(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_{ij}, B(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}x_{ij}, \text{ and } C(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}.$$

Let u be the optimal objective function value of the assignment problem

[1] The models presented in this section, as well as the theoretical results and algorithms in Chapter 3 are provided by my supervisor Dr. Abraham Punnen. My contribution regarding these algorithms are primarily experimental analysis.

$$
\begin{aligned}
&\text{Maximize} \quad B(X) \\
&\text{Subject to} \\
&\qquad \sum_{j=1}^{n} x_{ij} = 1, \;\; i = 1, ..., n, \\
&\qquad \sum_{i=1}^{n} x_{ij} = 1, \;\; j = 1, ..., n, \\
&\qquad x_{ij} \in \{0, 1\}, \;\; i, j = 1, ..., n.
\end{aligned}
$$

and l be that of

$$
\begin{aligned}
&\text{Minimize} \quad B(X) \\
&\text{Subject to} \\
&\qquad \sum_{j=1}^{n} x_{ij} = 1, \;\; i = 1, ..., n, \\
&\qquad \sum_{i=1}^{n} x_{ij} = 1, \;\; j = 1, ..., n, \\
&\qquad x_{ij} \in \{0, 1\}, \;\; i, j = 1, ..., n.
\end{aligned}
$$

Then $l \leq B(X) \leq u$ for any feasible solution of MAP. Similarly let u^0 and l^0 respectively be the optimal objective function value of the following assignment problems

$$
\begin{aligned}
&\text{Maximize} \quad A(X) \\
&\text{Subject to} \\
&\qquad \sum_{j=1}^{n} x_{ij} = 1, \;\; i = 1, ..., n, \\
&\qquad \sum_{i=1}^{n} x_{ij} = 1, \;\; j = 1, ..., n, \\
&\qquad x_{ij} \in \{0, 1\}, \;\; i, j = 1, ..., n.
\end{aligned}
$$

and

$$
\begin{aligned}
&\text{Minimize} \quad A(X) \\
&\text{Subject to}
\end{aligned}
$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ \ i = 1, ..., n,$$
$$\sum_{i=1}^{n} x_{ij} = 1,\ \ j = 1, ..., n,$$
$$x_{ij} \in \{0, 1\},\ \ i, j = 1, ..., n.$$

Then $l^0 \leq A(X) \leq u^0$ for any feasible solution of MAP.

Let $y = B(X)$. Then MAP can be written as:

$$\text{Maximize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_{ij} y + C(X)$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1,\ \ i = 1, ..., n,$$
$$\sum_{i=1}^{n} x_{ij} = 1,\ \ j = 1, ..., n,$$
$$B(X) = y,$$
$$x_{ij} \in \{0, 1\},\ \ i, j = 1, ..., n.$$

where y is a continuous variable such that $l \leq y \leq u$, for appropriate values of l and u. Then the quadratic term $x_{ij} y$ can be replaced by a new variable z_{ij} along with constants:

$$z_{ij} - u x_{ij} \leq 0,\ \ i, j = 1, ..., n,$$
$$z_{ij} - l x_{ij} \geq 0,\ \ i, j = 1, ..., n,$$
$$y - z_{ij} + u x_{ij} \leq u,\ \ i, j = 1, ..., n,$$
$$y - z_{ij} + l x_{ij} \geq l,\ \ i, j = 1, ..., n.$$

These constraints ensure that $z_{ij} = y$ if and only if $x_{ij} = 1$. Thus MAP can be written as the mixed integer linear program:

$$MILP6: \quad \text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} z_{ij} + C(X)$$

Subject to

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

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

$$\sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}x_{ij} - y = 0, \ \ i, j = 1, ..., n, \tag{2.11}$$

$$z_{ij} - ux_{ij} \leq 0, \ \ i, j = 1, ..., n, \tag{2.12}$$

$$z_{ij} - lx_{ij} \geq 0, \ \ i, j = 1, ..., n, \tag{2.13}$$

$$y - z_{ij} + ux_{ij} \leq u, \ \ i, j = 1, ..., n, \tag{2.14}$$

$$y - z_{ij} + lx_{ij} \geq l, \ \ i, j = 1, ..., n, \tag{2.15}$$

$$x_{ij} \in \{0, 1\}, \ \ i, j = 1, ..., n.$$

Our next linearization uses binary expansion the value of $B(X)$ using the well-known transformation [63]. This idea has been used by many authors in integer programming and integer quadratic programming. For example, [43].

We assume that $A(X), B(X) \geq 0$ and integer.

Now let us prove the validity by showing that $z_{ij} = y$ if and only if $x_{ij} = 1$.

If $z_{ij} = y$, constraints (2.12), (2.13), (2.14) and (2.15) now become

$$y - ux_{ij} \leq 0, \ \ i, j = 1, ..., n, \tag{2.16}$$

$$y - lx_{ij} \geq 0, \ \ i, j = 1, ..., n, \tag{2.17}$$

$$x_{ij} \leq 1, \ \ i, j = 1, ..., n, \tag{2.18}$$

$$x_{ij} \geq 1, \ \ i, j = 1, ..., n. \tag{2.19}$$

According to (2.18) and (2.19), we conclude that $x_{ij} = 1$.

On the other hand, if $x_{ij} = 1$, constraints (2.12), (2.13), (2.14) and (2.15) now become

$$z_{ij} \leq u, \ \ i, j = 1, ..., n, \tag{2.20}$$

$$z_{ij} \geq l, \ \ i, j = 1, ..., n, \tag{2.21}$$

$$y \leq z_{ij}, \ \ i, j = 1, ..., n, \tag{2.22}$$

$$y \geq z_{ij}, \ \ i, j = 1, ..., n. \tag{2.23}$$

According to (2.22) and (2.23), we conclude that $z_{ij} = y$.

The mixed integer program MILP6 contains $2n^2$ variables $5n^2 + 2n$ constraints without counting the binary constraints.

Let $y = B(X)$ and $z = A(X)$. Then MAP can be written as:

$$
\begin{aligned}
\text{Minimize} \ \ & yz + C(X) \\
\text{Subject to} \ \ & \\
& \sum_{j=1}^{n} x_{ij} = 1, \ \ i = 1, ..., n, \\
& \sum_{i=1}^{n} x_{ij} = 1, \ \ j = 1, ..., n, \\
& B(X) = y, \\
& A(X) = z, \\
& x_{ij} \in \{0, 1\}, \ \ i, j = 1, ..., n.
\end{aligned}
$$

Note that $0 \leq l \leq y \leq u$. Let $\alpha = \lfloor \log_2(u - l) \rfloor + 1$. Then y can be written as $y = l + \sum_{k=1}^{\alpha} 2^{k-1} v_k$ where $v_k \in 0, 1$. Eliminating y from the constraint in the above MAP, we get

$$
\begin{aligned}
\text{Minimize} \ \ & lz + \sum_{k=1}^{\alpha} 2^{k-1} v_k z + C(X) \\
\text{Subject to} \ \ & \\
& \sum_{j=1}^{n} x_{ij} = 1, \ \ i = 1, ..., n, \\
& \sum_{i=1}^{n} x_{ij} = 1, \ \ j = 1, ..., n, \\
& B(X) = \sum_{k=1}^{\alpha} 2^{k-1} v_k, \\
& A(X) = z, \\
& x_{ij} \in \{0, 1\}, \ \ i, j = 1, ..., n.
\end{aligned}
$$

Let l^0, u^0 be lower and upper bounds on $A(X)$. As discussed earlier, the product zv_k can be linearized using the following constraints

$$w_k - u^0 v_k \leq 0, \ \ k = 1, ..., \alpha, \tag{2.24}$$

$$w_k - l^0 v_k \geq 0, \ \ k = 1, ..., \alpha, \tag{2.25}$$

$$z - w_k + u^0 v_k \leq u^0, \ \ k = 1, ..., \alpha, \tag{2.26}$$

$$z - w_k + l^0 v_k \geq l^0, \ \ k = 1, ..., \alpha. \tag{2.27}$$

Thus we have the MILP formulation as follows.

$$
\begin{aligned}
MILP7: \quad & \text{Minimize} \;\; lz + \sum_{k=1}^{n} 2^{k-1} w_k + C(X) \\
& \text{Subject to} \\
& \qquad\qquad \sum_{j=1}^{n} x_{ij} = 1, \;\; i = 1, ..., n, \\
& \qquad\qquad \sum_{i=1}^{n} x_{ij} = 1, \;\; j = 1, ..., n, \\
& \qquad\qquad B(X) = l + \sum_{k=1}^{\alpha} 2^{k-1} v_k, \\
& \qquad\qquad A(X) = z, \\
& \qquad\qquad w_k - u^0 v_k \leq 0, \;\; k = 1, ..., \alpha, \\
& \qquad\qquad w_k - l^0 v_k \geq 0, \;\; k = 1, ..., \alpha, \\
& \qquad\qquad z - w_k + u^0 v_k \leq u^0, \;\; k = 1, ..., \alpha, \\
& \qquad\qquad z - w_k + l^0 v_k \geq l^0, \;\; k = 1, ..., \alpha, \\
& \qquad\qquad x_{ij} \in \{0, 1\}, \;\; i, j = 1, ..., n, \\
& \qquad\qquad v_k \in \{0, 1\}, \;\; k = 1, ..., \alpha.
\end{aligned}
$$

Note that MILP7 includes $2n + \log_2 (u - l) + 1$ variables and $4\alpha + 2n + 2$ constraints.

2.4 Experimental Analysis

This section is dedicated to experimental evaluation of the previously proposed mixed integer linear program (MILP) formulations. The experiments are conducted on a number of instances categorized into the following groups.

1. The first group of instances we test is pseudo-random problems, in which all parameters are pseudo-random numbers. The pseudo-random numbers used in instances are generated using the built-in C++ functions *rand*(). The instance size n varies from 10 to 55.

   (a) **Randomly generated problems**: The entries in parameter matrix A range from 0 to 100, B from 100 to 200, and C from 200 to 1000.

   (b) **Positively correlated random problems**: The entries in parameter matrix A range from 0 to 100, B from 100 to 200, and C from 200 to 1000. As the name suggests, the matrices A and B positively correlated. That is to say, each row in matrices A and B is sorted in ascending order.

(c) **Negatively correlated random problems**: The entries in parameter matrix A range from 0 to 100, B from 100 to 200, and C from 200 to 1000. As the name suggests, the matrices A and B negatively correlated. That is to say, each row in matrix A is sorted in ascending order and each row in matrix B is sorted in descending order.

2. The second group of instances we test is pseudo-random homogeneous problems. The only difference between this group of problems and the first group is that all the problems in this group do not have matrix C. Precisely, objective function 1.7 becomes $A(X)B(X)$ in the homogeneous cases. We also test three different subgroups of pseudo-random homogeneous problems as for pseudo-random problems. We test the problems of size n ranging from 10 to 55.

3. The third group of instances we test is pseudo-random small problems. The only difference between this group of problems and the first group is that all the problems in this group do have smaller matrix C. More specifically, all the entries in matrix C in this group of problems range from 0 to 200, as opposed to 200 to 1000 in group 1. We test the problems of size n ranging from 10 to 55.

4. The fourth group of instances is the instances from the Quadratic Assignment Problem Library (QAPLIB) [18]. We do not test all the instances from QAPLIB. We focus on testing the most difficult ones of them. As in the experimental protocol of their paper [13], Una Benlic and Jin-Kao Hao states, "Among the 135 instances, 101 instances (including all the real-life instance) can be considered as easy since BLS (and many other state-of-art QAP methods) can solve them to optimality in every singe trial within a very short computation time (often less than a second)." Considering that they tested on the QAP and we test on the MAP, and the MAP are computationally easier than the QAP, we test on the following four groups of QAPLIB problems [44]: J. Skorin-Kapov, E.D. Taillard, U.W. Thonemann and A. Bolte, and M.R. Wilhelm and T.L. Ward.

All MILP models are solved using the commercial MILP solver CPLEX [23] on a workstation with the configuration: Intel i7-4790 CPU, 32GB RAM, and 64-bit Windows 7 Enterprise operating system. The models are implemented using C++ and Concert Technology [46] and complied using the IDE visual studio 2010. Test results are presented in Table A.1 - A.21.

The tables can be classified as follows.

1. Table A.1 - A.3 show the experimental results of relaxed versions of the six classic linearization formulations on the first group of instances indicated above. Relaxed linearization formulations are linearization formulations without integer constraints.

2. Table A.4 - A.6 show the experimental results of integer versions of the six classic linearization formulations on the first group of instances indicated above.
3. Table A.7 - A.9 show the experimental results of the two new linearization formulations on the first group of instances indicated above.
4. Table A.10 - A.12 show the experimental results of the six classic linearization formulations on the second group of instances indicated above, namely, homogeneous pseudo random problems.
5. Table A.13 - A.15 show the experimental results of the two new linearization formulations on the second group of instances indicated above, namely, homogeneous pseudo random problems.
6. Table A.16 - A.18 show the experimental results of the six classic linearization formulations on the third group of instances indicated above, namely, small C pseudo random problems.
7. Table A.19 - A.21 show the experimental results of the two classic linearization formulations on the third group of instances indicated above, namely, small C pseudo random problems.

The results for the linearizations are as follows.

From the experimental results, we notice the following.

1. In tackling the MAP, the computational efficiency, which is of the seven linearizations in test are in the following ascending order. Please note that computational efficiency is measured as the reciprocal of computational time. The less time a method requires, the more efficient it is.

   $$\begin{aligned} &Lawler < RLT < \text{Kaufmann and Broeckx (KB)} \\ &\qquad < \text{Frieze and Yadegar (FY)} < \text{Adams and Johnson (AJ)} < MILP6 < MILP7 \end{aligned}$$

2. The two new linearizations are considerably more efficient than the five traditional ones in solving the MAP. These results are expected due to the fact that the two new linearizations are specially designed to solve the MAP.
3. Relaxed versions of the linearizations are easier to solve than the ones for Lawler, Kaufmann and Broeckx (KB), RLT and the two new linearizations. In contrast, Frieze and Yadegar (FY) and Adams and Johnson (AJ) seem to cope with integer MAPs more efficiently.

4. The quality of solutions to relaxed linearizations varies from linearization to linearization. Frieze and Yadegar (FY) and Adams and Johnson (AJ) relaxed linearizations produce exact integer optimal solutions. The two new linearizations produce solutions close to integer optimal solutions. Between them, MILP7 produces better quality solutions than MILP6 does. And the other three linearizations Lawler, KB and RLT give very low quality solutions.

5. The MAP with positively and negatively correlated matrices are harder to solve than the MAP with regular randomly generated matrices in terms of execution time for all seven linearizations in test.

6. The linearizations are slightly more effective in solving the homogeneous random MAP (matrix C is zero) than in solving regular random MAP. There is little difference between the homogeneous negatively and positively correlated MAP and the regular negatively and positively correlated MAP in terms of execution time.

7. When the problem size grows beyond a certain threshold, MILP6 cannot solve the MAP instances with small c_{ij}. Particularly, one noticeable fact seen from the experimental results is that MILP6 cannot solve the MAP with size greater than 35 within one hour time limit. On the contrary, MILP7 solves the MAP with small C value more efficiently than the regular MAP in terms of execution time.

Chapter 3

The MAP as Constrained Assignment Problem

3.1 Exact Algorithms for the Homogeneous Multiplicative Assignment Problem

Recall that the MAP is formulated as an integer program as follows.

$$\text{Minimize } \left(\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_{ij}\right)\left(\sum_{k=1}^{n}\sum_{l=1}^{n} b_{kl}x_{kl}\right) + \left(\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}\right)$$

Subject to

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

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

$$x_{ij} \in \{0, 1\},\ \ i, j = 1, ..., n. \tag{3.3}$$

The *homogeneous multiplicative assignment problem* (HMAP) is defined based on MAP. When $c_{ij} = 0$ for all $i, j = 1, 2, ..., n$ and $a_{ij}, b_{ij} \geq 0$ for all $i, j = 1, 2, ..., n$, the MAP is called the HMAP.

We first focus on the HMAP by developing some algorithms to solve it and then expand our algorithms to solve the general MAP. It is easy to see that an assignment with $A(X) = 0$ or $B(X) = 0$ is an optimal solution to the HMAP and it is easy to test if an assignment can make $A(X) = 0$ or $B(X) = 0$. Thus, without loss of generality, we assume $A(X) > 0$ and $B(X) > 0$ in the following discussion of the algorithms for the HMAP.

We denote the family of feasible solutions to the MAP by $\mathbb{F}$, namely, $\mathbb{F} = \{X \in \{0, 1\}^{n\times n}$ where X satisfies (3.1) and (3.2).

Consider the constrained assignment problem,

$$CAP(k): \quad \text{Minimize} \quad A(x) \tag{3.4}$$
$$\text{Subject to}$$
$$X \in \mathbb{F},$$
$$\sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}x_{ij} \leq k. \tag{3.5}$$

where k is a constant. The CAP(k) is known to be NP-hard [56]. Several exact and heuristic algorithms have been made to solve this problem [3,42,50,66] and its continuous relaxation is the well-studied linear programming problem, which can be solved in polynomial time [24]. We develop our first algorithm for the HMAP by solving a sequence of CAP(k) problems.

Let X' be any feasible solution to CAP(k). Consider the family of assignments $\mathbb{F}(X') = \{X \in \mathbb{F} : B(X') \leq B(X) \leq k\}$.

Theorem 5. *If X' is an optimal solution to the CAP(k) then $A(X')B(X') \leq A(X)B(X)$ for all $X \in \mathbb{F}(X')$.*

Proof. Since X' is an optimal solution to the CAP(k), we have $A(X') \leq A(X)$ for all $X \in \mathbb{F}$ satisfying $B(x) \leq k$, which includes $X \in \mathbb{F}(X')$. By the definition of $\mathbb{F}(X')$, $B(X') \leq B(X)$. Since $B(X) > 0$, we have $A(X')B(X') \leq A(X)B(X)$ for all $X \in \mathbb{F}(X')$. □

For later use, we hereby define u and l to be the largest and smallest values of $B(X)$ for $X \in \mathbb{F}$. It is easy to obtain u and l by solving respectively assignment problems with maximization and minimization objective functions as follows.

$$\text{Maximize} \quad B(x)$$
$$\text{Subject to}$$
$$X \in \mathbb{F}.$$

and

$$\text{Minimize} \quad B(x)$$
$$\text{Subject to}$$
$$X \in \mathbb{F}.$$

Linear assignment problems can be formulated as weighted bipartite matching problems. It is known that bipartite matching problems can be reduced to network flows problems, to which polynomial algorithms are known. Therefore, linear assignment problems are polynomially solvable.

In light of u and l, repeated applications of Theorem 5 lead to the following algorithm for the HMAP.

Algorithm 1: Iterated Exact CAP Algorithm

Compute l and u and let X^0 and X' respectively be the optimal solutions that produced l and u;

$k \leftarrow B(X') - 1, sol \leftarrow X', obj \leftarrow A(X')B(X')$;

if $l = 0$ **then**

  X^0 is optimal. STOP.

end

while $k \geq l$ **do**

  Let X^* be an optimal solution to CAP(k); $temp \leftarrow A(X^*)B(X^*)$;

  if $obj > temp$ **then**

    $obj \leftarrow temp$, $sol \leftarrow X^*$;

  end

  $k \leftarrow B(X^*) - 1$;

end

Output sol and obj;

Solving large scale CAP(k) problems could be time consuming. One way to improve the efficiency of the iterated exact CAP algorithm is to identify conditions that allow early detection of optimal solutions and conditions that allow steeper decrease in the value of k. Identifying such conditions could lead to reduced number of iterations and thereby improving the overall performance of the algorithm.

Suppose the total number of iterations of the exact CAP algorithm is p and let X^i be the solution generated in iteration i, for $i = 1, 2, ..., p$. Note that

$$A(X^1) \leq A(X^2) \leq \cdots \leq A(X^p) \tag{3.6}$$

and

$$B(X^1) > B(X^2) > \cdots > B(X^p) \tag{3.7}$$

The correctness of the above two inequalities can be easily verified by noticing the fact that in each iteration k decreases. That means the solution X to CAP(k), which is confined by the constraint $B(X) \leq k$, is related to less B(X), thus (3.7) is verified. As to (3.6), A(X) increases with more iterations because the constraint gets stricter as k decreases, which means the scope for possible solutions to A(X) shrinks, and thus A(X) gets larger or does not change.

Let q be an integer between 1 and p and define the set $R(q) = 1, 2, ..., q$. Choose an index $q_r \in R(q)$ such that

$$A(X^{q_r})B(X^{q_r}) = min\{A(X^i)B(X^i) : i \in R(q)\} \tag{3.8}$$

Let $R'(q) = \{i : A(X^i)B(X^i) = A(X^{q_r})B(X^{q_r})\}$ and $\gamma \leq min\{B(X^i) : i \in R'(q)\}$ be a real number.

Theorem 6. *If $l > 0$ and $\frac{A(X^{q_r})B(X^{q_r})}{\gamma} \leq A(X^q)$ then X^{q_r} is an optimal solution to the HMAP.*

Proof. Prove by contradiction. Suppose X^{q_r} is not an optimal solution. Then there exists a $k \in R(p) - R(q)$ such that X^k is optimal. Thus, $A(X^k)B(X^k) < A(X^{q_r})B(X^{q_r})$. i.e.

$$A(X^k) < \frac{A(X^{q_r})B(X^{q_r})}{B(X^k)} \leq \frac{A(X^{q_r})B(X^{q_r})}{\gamma} \leq A(X^q). \tag{3.9}$$

According to (3.6), this implies $k \in R(q)$, which contradicts the premise that X^{q_r} is not an optimal solution. □

Theorem 6 provides a sufficient condition for optimality. Let us now consider another property that can be used to decrease the value of k more rapidly in the exact CAP algorithm.

Theorem 7. *If $l > 0$ and X^{q_r} is not an optimal solution to the HMAP, then there exists a $j \in R(p) - R(q)$ such that $B(X^j) < \frac{A(X^{q_r})B(X^{q_r})}{A(X^q)}$.*

Proof. Since X^{q_r} is not an optimal solution to the HMAP there exists a $j \in R(p) - R(q)$ such that X^j is optimal. Thus $A(X^j)B(X^j) < A(X^{q_r})B(X^{q_r})$. i.e.

$$B(X^j) < \frac{A(X^{q_r})B(X^{q_r})}{A(X^j)} \tag{3.10}$$

Since $j \in R(p) - R(q)$ we have

$$A(X^j) \geq A(X^q) \tag{3.11}$$

From (3.10) and (3.11), we can conclude $B(X^j) < \frac{A(X^{qr})B(X^{qr})}{A(X^q)}$. □

Taking advantage of Theorems 6 and 7, we can now present a modified version of the exact CAP algorithm named *modified exact CAP algorithm.*

Algorithm 2: Modified Exact CAP Algorithm

```
Compute l and u and let X^0 and X' respectively be the optimal solutions that
produced l and u.
k ← B(X'), sol ← X', obj ← A(X')B(X');
if l = 0 then
|  X^0 is optimal. STOP.
end
while k ≥ l do
|  Let X* be an optimal solution to CAP(k); temp ← A(X*)B(X*);
|  if obj > temp then
|  |  obj ← temp, sol ← X*;
|  end
|  if obj/l ≤ A(X*) then
|  |  Output sol and obj and STOP;
|  end
|  k ← min{B(X*) − 1, obj/A(X*) − 1};
end
Output sol and obj;
```

It may be noted conditions similar to those in Theorem 6 and 7 are used by many authors in various contexts [4, 5, 54, 58, 64, 67, 69].

There are only two nuances between the iterated exact CAP algorithm and the modified exact CAP algorithm, namely, the latter has an additional termination condition and a refined updating scheme for k. In spite of the simple structures, these two alterations are powerful in improving the algorithm's efficiency, as demonstrated by the computational experiments in Section 3.4.

3.2 Exact Algorithms for the General MAP

We have discussed how to solve the HMAP, or the MAP with additional restrictive conditions that $A(X), B(X) \geq 0$ and C is a zero matrix. Now we discuss how we can relax these assumptions and expand our algorithm to the general MAP.

Firstly, we notice that the iterated exact CAP algorithm works for any $A(X)$ since Theorem 5 is also valid in this case. Now let us focus on the case when $B(X)$ is allowed to be negative. In that case, the algorithm runs as before, until the value of k first becomes negative. When that happens, we save the best solution obtained up to this point as X^+, as well as its objective function value obj^+. Then set $k = 0$ and begin the second phase of the algorithm by solving CAP(k) as a maximization problem instead of a minimization problem, while retaining everything else unchanged. Let the optimal solution in the second phase be X^-. The overall best solution, or the better solution, out of the two solutions obtained in the two phases will be the optimal solution to the whole problem. Namely, let the overall optimal solution be X^* and $X^* = \ min\ \{obj(X^-), obj(X^+)\}$. It is easy to verify the correctness of this modified algorithm. The correctness of this algorithm is essentially in the property that when $B(X) > 0$, the minimum of $A(X)B(X)$ occurs when $A(X)$ is minimum and when $B(X) < 0$, the minimum of $A(X)B(X)$ occurs when $A(X)$ is maximum.

For the case when C is nonzero, Theorem 5 could be invalid and we need to try every value of k between l and u, which means making inequality constraint (3.5) equality to construct CAP'(k) and applying algorithm 1 on the following altered $CAP'(k)$.

$$
\begin{aligned}
CAP'(k): \quad & \text{Minimize} \quad A(x) \\
& \text{Subject to} \\
& \qquad X \in \mathbb{F}, \\
& \qquad \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} x_{ij} = k. \qquad (3.12)
\end{aligned}
$$

However, the $CAP'(k)$ does not guarantee the validity of the inequality chain (3.6), so the algorithm 2 does not work for the general MAP.

3.3 The Relaxed CAP Algorithm

Now we consider another algorithm to solve the HMAP. Suppose the entries in A and B are nonnegative integers. The only difference between the algorithm we are considering and the iterated exact CAP algorithm (Algorithm 1) is that we solve the LP relaxation of CAP(k) in each iteration instead of CAP(k). Unlike the iterated exact CAP algorithm, this approach will not work for negative A and B. The reason is that if $A(X)$ and $B(X)$ are nonnegative, the objective function of HMAP is quasi-concave and hence there exists an optimal solution at the extreme point of the polytope $\mathbb{P}$, which is the convex hull of solutions belonging to $\mathbb{F}$, except that now $0 \le x_{ij} \le 1, i, j = 1, 2, ..., n$. Thus solving the continuous relaxation is effectively equivalent to solving the HMAP. This property in general may not hold if $A(X)$ or $B(x)$ takes negative values. Denote the continuous relaxation of the HMAP by CHMAP and the continuous relaxation of CAP(k) by CCAP(k).

Let $\bar{x}$ be a feasible solution to the CCAP(k). Consider the family of assignments $\mathbb{P}(\bar{x}) = \{X \in \mathbb{P} : B(\bar{X}) \leq B(X) \leq k\}$.

Theorem 8. *If X^0 is an optimal solution to the CCAP(k) then $A(X^0)B(X^0) \leq A(X)B(X)$ for all $X \in \mathbb{P}(X^0)$.*

Proof. The proof of this theorem is similar to that of Theorem 5 and hence skipped. □

It may be noted that x^0 may be non-integral. However, in this case we can always find an integer solution (assignment) with the same or better objective function value.

Like the way that Theorem 5 leads to the iterated exact CAP algorithm, Theorem 8 leads to the following algorithm for HMAP.

Algorithm 3: Iterated Relaxed CAP Algorithm

```
Compute l and u and let X' be the optimal solution that produced u;
k ← B(X'), sol ← X', obj ← A(X')B(X');
while k ≥ l do
    Let X* be an optimal solution to CCAP(k);
    If X* is not an assignment, find an assignment X^0 with the same or better
    objective function value and set X* ← X^0;
    if obj > A(X*)B(X*) then
        obj ← A(X*)B(X*), sol ← X*;
    end
    decrease k;
end
Output sol and obj;
```

The operation ′decrease k′ can be implemented in many ways and sometimes it depends on how we solve CCAP(k). If X^* is a basic feasible solution with characteristic interval $[\underline{k}, \bar{k}]$ then k can be decreased to $\lfloor \underline{k} \rfloor$. If we do not want to restrict on the LP solver or want to make the additional effort in computing the characteristic interval for X^*, we can use a simple updating scheme for k as follows:

$$\textbf{if} \quad \lfloor B(X^*) \rfloor = B(X^*) \quad \textbf{then} \quad k \leftarrow B(X^*) - 1 \quad \textbf{else} \quad k \leftarrow \lfloor B(X^*) \rfloor \tag{3.13}$$

It is not hard to see that this updating scheme is not very efficient. It normally decreases k by one in each iteration and crawl through the interval $[l, u]$. However, we can easily improve the algorithm by deriving properties similar to that in Theorems 6 and 7 and refining the algorithm accordingly. This approach leads to the following *modified relaxed CAP algorithm*.

Algorithm 4: Modified Relaxed CAP Algorithm

Compute l and u and let X' be the optimal solution that produced u;

$k \leftarrow B(X'), sol \leftarrow X', obj \leftarrow A(X')B(X')$;

while $k \geq l$ **do**

  Let X^* be the optimal solution to CCAP(k);

  If X^* is not an assignment, find an assignment X^0 with the same or better objective function value and set $X^* \leftarrow X^0$;

  if $obj > A(X^*)B(X^*)$ **then**

    $obj \leftarrow temp$ and $sol \leftarrow X^*$;

  end

  if $\frac{obj}{l} \leq A(X^*)$ **then**

    Output sol, obj and STOP;

  end

  if $\lfloor B(X^*) \rfloor = B(X^*)$ **then**

    $k \leftarrow min\{B(X^*) - 1, \frac{obj}{A(X^*)} - 1\}$;

  else

    $k \leftarrow min\{\lfloor B(X^*) \rfloor, \frac{obj}{A(X^*)} - 1\}$;

  end

end

Output sol and obj;

3.4 Experimental Analysis

In this section we present results of experimental analysis that have been carried out using the algorithms with the constrained assignment problems developed in the previous sections. All the algorithms are implemented using the commercial optimizer CPLEX [23] on a workstation with the configuration: Intel i7-4790 CPU, 32GB RAM, and 64-bit Windows 7 Enterprise operating system The models are implemented using C++ and Concert Technology [46] and complied using the integrated development environment visual studio 2010.

As the experiments on linearizations in Chapter 2, the following three classes of test problems are generated: pseudo-random problems, negatively correlated problems, and positively correlated problems.

Parameter setting for each of the classes of problems is also same with that of the linearization experiments in Chapter 2.

The experimental results for the CAP algorithms can be found in appendix A.

Several noteworthy observations can be drawn from the above experimental results.

1. According to Table A.22 - A.24, relaxed versions of CAP algorithms are more efficient than original ones in solving all the three types (random, negatively correlated, and positively correlated) of MAPs. More specifically, relaxed iterated CAP is considerably more efficient than iterated CAP in solving MAPs. However, relaxed modified CAP is only insignificantly faster.

2. According to Table A.22 - A.30, different CAP algorithms solve different types of MAPs with differing efficiency. To be specific, iterated CAP solves MAPs with special matrix structures (negatively or positively correlated) faster than original randomly generated MAPs. Relaxed iterated CAP solves all three types of MAPs with about same efficiency. In contrast, modified CAP solves randomly generated MAPs faster than the MAP with special structures. Relaxed modified CAP has about same efficiency in solving all three types of MAPs.

3. According to Table A.22 - A.30, the execution time by all CAPs on regular size MAPs, homogeneous MAPs and small C MAPs are about same. The difference of matrix C does not cause difference in execution time among the CAP algorithms.

Chapter 4

Solving the MAP by Local Search

4.1 Introduction

The methods discussed in the previous chapters are exact algorithms. As the name suggests, exact algorithms find an optimal solution. However, in many cases, it is not necessary, or preferable, to find exact optimal solutions by investing significant time and computing resources.

In these cases, we apply heuristic algorithms in lieu of exact algorithms. Heuristic algorithms are usually more efficient in terms of computational time than exact algorithms, which means the needed computational resources of heuristic algorithms are usually less than that of exact algorithms. However, instead of giving exact solution as exact algorithms do, heuristic algorithms find approximate solutions.

In this chapter, the types of heuristic methods we use on the MAP is local search and tabu search. In short, local search algorithms refer to the group of methods used to solve problems by iteratively moving from one candidate solution to another in the space of candidate solutions. Local search algorithms terminate when certain pre-set conditions are satisfied, such as a satisfactory solution has been identified or the pre-determined time bound has elapsed.

It is convenient to clarify some terms. In this chapter, the process of moving from one solution to another is referred to as a *move*. The space of of candidate solutions is used under the name *search space*. The conditions that mark the end of the algorithm are called *termination conditions*.

The layout of the remainder of the chapter is as follows. In Section 2, we first introduce the kind of local search algorithm that we use to solve the MAP, including the search space, moves, and the termination conditions. Then in Section 3, we discuss several variants of the local search method. Section 4 is devoted to *tabu search*, which is a way to enhance the local search heuristic algorithm. Subsequently, Section 5 describes the numerical experiments that implement the algorithms we introduced in the former sections and shows the results of these experiments. Section 6 analyzes the results of local search algorithms in comparison with the approaches used in previous chapters. Section 7 concludes this chapter with a discussion of further possible modifications and variants of the heuristic algorithm used in this chapter.

4.2 Local Search

An assignment X can be represented as a permutation $\pi = (\pi(1), ..., \pi(n))$ such that $x_{i\pi(i)} = 1, i = 1, ..., n$ and $x_{ij} = 0$ otherwise. In this chapter we will use the permutation representation of an assignment. We first make some notations.

$$a(\pi) = \sum_{i=1}^{n} a_{i\pi(i)} \tag{4.1}$$

$$b(\pi) = \sum_{i=1}^{n} b_{i\pi(i)} \tag{4.2}$$

$$c(\pi) = \sum_{i=1}^{n} c_{i\pi(i)} \tag{4.3}$$

With the newly introduced notations, the MAP can be written as:

$$\begin{aligned} &\text{Minimize } a(\pi)b(\pi) + c(\pi) \\ &\text{Subject to } \pi \in P_n \end{aligned} \tag{4.4}$$

where P_n is the set of all permutations of 1,2, ..., n. For permutation based optimization problems, one of the most well studied solution neighbourhood is the *swap neighbourhood*, denoted by $N(\pi)$. Given a permutation π and two indices i and j, where $1 \leq i, j \leq n, i \neq j$.

The *swap operation* denoted by $swap(\pi : i, j)$ generates the permutation $\pi^{ij} \in P_n$ such that

$$\pi^{ij}(t) = \begin{cases} \pi(t), & \text{if } t \neq i, j, \\ \pi(j), & \text{if } t = i, \\ \pi(i), & \text{if } t = j. \end{cases}$$

The swap neighbourhood $N(\pi)$ is the set of permutations in P_n that can be obtained from π using a swap operation. Given $a(\pi), b(\pi)$ and $c(\pi)$, we can compute $a(\pi^{ij})$, $b(\pi^{ij})$, $c(\pi^{ij})$ in constant time using the formulae:

$$a(\pi^{ij}) = a(\pi) - a_{i\pi(i)} - a_{j\pi(j)} + a_{i\pi(j)} + a_{j\pi(i)} \tag{4.5}$$

$$b(\pi^{ij}) = b(\pi) - b_{i\pi(i)} - b_{j\pi(j)} + b_{i\pi(j)} + b_{j\pi(i)} \tag{4.6}$$

$$c(\pi^{ij}) = c(\pi) - c_{i\pi(i)} - c_{j\pi(j)} + c_{i\pi(j)} + c_{j\pi(i)} \tag{4.7}$$

Given a permutation, the best solution π^{rs} in $N(\pi)$ can be identified in $O(n^2)$ time by exhaustively evaluating

$$a(\pi^{rs})b(\pi^{rs}) + c(\pi^{rs}) = min\{a(\pi^{ij})b(\pi^{ij}) + c(\pi^{ij}) : i = 1, ..., n, j = 1, ..., n, i \neq j\} \tag{4.8}$$

On the basis of the preceding discussion, we present the Swap Neighborhood Search Algorithm below.

Algorithm 5: Swap Neighbourhood Search

Data: the problem size n and A, B, C as $n \times n$ matrices, an assignment π

Result: an assignment *best.sol* and the corresponding objective function value *best.obj*

```
best.sol ← π;
Initialize best.obj ← a(π) · b(π) + c(π);
for i = 1 ,..., n do
    for j = 1 ,..., n do
        if best.obj > a(π^ij) · b(π^ij) + c(π^ij) then
            best.obj ← a(π^ij) · b(π^ij) + c(π^ij);
            best.sol ← π^ij;
        end
    end
end
Return best.obj and best.sol.
```

We can modify Algorithm 5 as a subroutine SWAP(π), where π is a permutation. The subroutine SWAP(π) finds a best permutation starting with π, returning the optimal objective function value and the corresponding best assignment.

Using the subroutine SWAP(π), we can develop a local search algorithm. The basic idea is that each iteration finds the best solution in the swap neighborhood of the current solution, which will be used as the starting solution in the next search iteration. The iterative search continues until the termination condition is met that no further improvement in the swap neighborhood is possible.

The Local Search Algorithm is presented below.

Algorithm 6: Local Search

Data: the problem size n and A, B, C as $n \times n$ matrices, an assignment π

Result: an assignment $best.sofar.sol$ and the corresponding objective function value $best.sofar.obj$

$best.sofar.sol \leftarrow \pi$;

$local.optimum \leftarrow false$;

Initialize $best.sofar.obj \leftarrow a(\pi) \cdot b(\pi) + c(\pi)$;

while $local.optimum = false$ **do**

  $inter.obj \leftarrow SWAP(best.sofar.sol).best.obj$;

  $inter.sol \leftarrow SWAP(best.sofar.sol).best.sol$;

  if $inter.obj \leftarrow best.sofar.obj$ **then**

    $best.sofar.obj \leftarrow inter.obj$;

    $best.sofar.sol \leftarrow inter.sol$;

  else

    Return $best.sofar.obj$ and $best.sofar.sol$.

    $local.optimum \leftarrow true$

  end

end

4.3 Variants of Local Search

It is easy to see that the effectiveness and efficiency of the swap local search algorithm partly depend on the initial permutation. Thus we test this swap algorithm on different initial solutions for comparative purposes.

We start with a very simple and straightforward initial permutation, $1, 2, ..., n$, where n is the size of the MAP. We refer to this algorithm as One Fixed Initial Permutation Swap

Local Search Algorithm. The detailed algorithm is as follows.

Algorithm 7: One Fixed Initial Assignment Swap Local Search

Data: the problem size n and A, B, C as $n \times n$ matrices
Result: an assignment $best.sofar.sol$ and the corresponding objective function value $best.sofar.obj$

Set the initial permutation π as $\pi(i) = i$;
$best.sofar.sol \leftarrow \pi$;
$local.optimum \leftarrow false$;
Initialize $best.sofar.obj \leftarrow a(\pi) \cdot b(\pi) + c(\pi)$;
while $local.optimum = false$ **do**
    $inter.obj \leftarrow SWAP(best.sofar.sol).best.obj$;
    $inter.sol \leftarrow SWAP(best.sofar.sol).best.sol$;
    if $inter.obj \leftarrow best.sofar.obj$ **then**
        $best.sofar.obj \leftarrow inter.obj$;
        $best.sofar.sol \leftarrow inter.sol$;
    end
    else
        Return $best.sofar.obj$ and $best.sofar.sol$.
        $local.optimum \leftarrow true$
    end
end

Based on Algorithm 7, we develop another algorithm with different initial assignment setting. In lieu of just one initial assignment, we randomly generate ten initial permutations on each of which we run Algorithm 6 to produce ten candidate solutions. We pick the best one to be the final solution. The detailed algorithm is as follows. In the following algorithm, we use Algorithm 6 as a subroutine under the name of $LocalSearch(\pi)$, where π is an assignment.

Algorithm 8: Ten Starts Swap Local Search

Data: the problem size n and A, B, C as $n \times n$ matrices

Result: an assignment *best.sol* and the corresponding objective function value *best.obj*

best.obj $\leftarrow \infty$;

for k = *1 ,..., 10* **do**

  Randomly generate a permutation π;

  $temp.sol \leftarrow LocalSearch(\pi).best.sofar.sol$;

  $temp.obj \leftarrow LocalSearch(\pi).best.sofar.obj$;

  if *temp.obj* $<$ *best.obj* **then**

    $best.obj \leftarrow temp.obj$;

    $best.sol \leftarrow temp.sol$;

  end

end

Return *best.obj* and *best.sol*.

Another line of thought is to use the solutions to specially constructed linear assignment problems as the initial permutations on which to run $LocalSearch(\pi)$. Some of the specially designed linear assignment problems we use to generate the initial permutations are the ones with objective functions $A+C$, $B+C$, $A+B+C$. The detailed algorithms are as follows.

Algorithm 9: Special Initial Solution A+C Swap Local Search Algorithm

Data: the problem size n and A, B, C as $n \times n$ matrices

Result: an assignment *best.sol* and the corresponding objective function value *best.obj*

Solve the linear assignment problem $A + C$ to get the solution π;

$best.sol \leftarrow \pi$;

$best.obj \leftarrow A(\pi) \cdot B(\pi) + C(\pi)$;

$best.sol \leftarrow LocalSearch(best.sol).best.sofar.sol$;

$best.obj \leftarrow LocalSearch(best.obj).best.sofar.obj$;

Return *best.obj* and *best.sol*.

Algorithm 10: Special Initial Solution B+C Swap Local Search Algorithm

Data: the problem size n and A, B, C as $n \times n$ matrices

Result: an assignment *best.sol* and the corresponding objective function value *best.obj*

Solve the linear assignment problem $B + C$ to get the solution π;

$best.sol \leftarrow \pi$;

$best.obj \leftarrow A(\pi) \cdot B(\pi) + C(\pi)$;

$best.sol \leftarrow LocalSearch(best.sol).best.sofar.sol$;

$best.obj \leftarrow LocalSearch(best.obj).best.sofar.obj$;

Return *best.obj* and *best.sol*.

Algorithm 11: Special Initial Solution A+B+C Swap Local Search Algorithm

Data: the problem size n and A, B, C as $n \times n$ matrices

Result: an assignment *best.sol* and the corresponding objective function value *best.obj*

Solve the linear assignment problem $A + B + C$ to get the solution π;

$best.sol \leftarrow \pi$;

$best.obj \leftarrow A(\pi) \cdot B(\pi) + C(\pi)$;

$best.sol \leftarrow LocalSearch(best.sol).best.sofar.sol$;

$best.obj \leftarrow LocalSearch(best.obj).best.sofar.obj$;

Return *best.obj* and *best.sol*.

Furthermore, we test a group of matrix-guided local search algorithms. In the context of solving the MAP, matrix-guided local search algorithms differ from the original local search algorithms in the initial solutions. In matrix-guided local search algorithms, the initial solutions are got by calculating some especially constructed linear assignment problems involving some parameters computed based on the MAP problem. Particularly, we test the so-called A-guided and B-guided local search algorithms. The detailed algorithms are as follows.

Algorithm 12: Special Initial Solution A-Matrix Guided Swap Local Search Algorithm

Data: the problem size n and A, B, C as $n \times n$ matrices

Result: an assignment *best.sol* and the corresponding objective function value *best.obj*

Solve the A-Matrix Guided linear assignment problem to get the solution π;

$best.sol \leftarrow \pi$;

$best.obj \leftarrow A(\pi) \cdot B(\pi) + C(\pi)$;

$best.sol \leftarrow LocalSearch(best.sol).best.sofar.sol$;

$best.obj \leftarrow LocalSearch(best.obj).best.sofar.obj$;

Return *best.obj* and *best.sol*.

Algorithm 13: Special Initial Solution B-Matrix Guided Swap Local Search Algorithm

Data: the problem size n and A, B, C as $n \times n$ matrices

Result: an assignment *best.sol* and the corresponding objective function value *best.obj*

Solve the B-Matrix Guided linear assignment problem to get the solution π;

$best.sol \leftarrow \pi$;

$best.obj \leftarrow A(\pi) \cdot B(\pi) + C(\pi)$;

$best.sol \leftarrow LocalSearch(best.sol).best.sofar.sol$;

$best.obj \leftarrow LocalSearch(best.obj).best.sofar.obj$;

Return *best.obj* and *best.sol*.

4.4 Tabu Search

Tabu search is a metaheuristic strategy to solve combinatorial optimization problems that has a broad range of usefulness. Tabu search is reputed for being able to find good solutions in comparatively short period of time. As Fred Glover wrote in the initial paper on Tabu Search, "Latest research and computational comparisons ... disclosed the ability of tabu search to obtain high quality solutions with modest computational effort, generally dominating alternative methods tested" [37].

Tabu search is powerful in that it can drastically strengthen the effectiveness and efficiency of many existing local search algorithms. The main problem that tabu search aims at dealing with is that many local search algorithms tend to find and terminate with poor local optimums, not being able to reach global optimums. The underlying rationale of tabu search is to forbid certain moves so that the local search can jump out of local optimums. As such, the chances of landing a global optimum increase.

Tabu search is a memory-based strategy. To be more specific, the history of local search is utilized in creating tabu list, which contains all forbidden moves in each step. The composition of tabu list is dynamic, changing according to the current state and search history. The utilization of memory is twofold: recency-based and frequency-based. Recency-based memory method constructs the tabu list on the basis of how recent the moves have been implemented in search history. The guiding principle of recency-based memory method is the most recent moves should be tabu. On the other hand, frequency-based memory approach makes the tabu list on the basis of how frequent the moves have been implemented in search history. The guiding principle of frequency-based approach is the most frequently implemented moves should be tabu.

The idea of *neighbourhood* of a solution is that the set of all possible moves from the solution. What tabu search does is to pre-screen the neighbourhood of the current solution and take out the unpromising moves, which are the moves prohibited by tabu list, in the neighbourhood. Then the best of all un-tabu moves is selected and a transition to a new solution is performed.

Tabu moves, i.e. moves on the tabu list, are not fixed. They are *adaptive* in the sense that tabu list updates as the local search progresses. Furthermore, tabu moves can be overridden if an aspiration criterion is met. Aspiration criteria exist to allow for flexibility in tabu search. The gist of aspiration criteria is that if a tabu move has sufficiently attractive prospect, it should be made an exception. Namely, the tabu status of this move can be overridden under this circumstance.

One of the primary practical challenges in designing and using tabu search is to create a balance between intensification and diversification in search procedure. Intensification refers to strategies that encourage moves visiting historically proven quality solutions. Conversely, diversification is the idea that attempts to explore dissimilar neighbourhoods historically seldom visited.

There are three parameters controlling the trade-off between the quality of solutions and computation time. Maximum restart number indicates the number of restarts the local search will execute. Maximum iteration number defines the maximum number of moves each attempt can make. Tabu tenure has control over how the tabu list is created, which, in turn, has an influence over neighbourhood diversification of tabu search. To be more specific, tabu search will be less diverse with a large tabu list. Since a large tabu list means a number of previous moves will be tabu, thus less new moves will be made compared to a smaller tabu list. Small tabu list, however, lead to cycling. Thus, a proper balance in the size of tabu list is important.

As noted before, different parameter settings give rise to different trade-offs. Specifically, the greater the maximum restart number is, the more computation time it will need. However, with greater number of restarts, the chance of good local optimum also increases. Likewise, greater maximum iteration number supposedly increases the quality of the solution at the cost of longer computation time. The same trade-off comes into play for tabu tenure size.

In this case, we tried different parameters to comparatively assess the effectiveness and efficiency. As a series of experiments show, a good but not necessarily the best combination of these parameters is: maximum restart number being 20, maximum iteration number being 100000, and tabu tenure (size of tabu list) being 10.

We keep track of moves in Tabu List. Tabu List is a $n \times n$ integer matrix. Whenever a swap is exercised, the entries in corresponding rows and columns increment by one. Tabu List is used in determining if a move is tabu by the following formula. We calculate the number $t = tabu.List[i][j] + tabu.tenure + k$, where k is a random number between 0 and 9, and $iter$ is the current iteration number. We define the following tabu rule.

$$\text{swap(i, j) is} \begin{cases} non - tabu, & \text{if } t < iter, \\ \quad tabu, & \text{if } otherwise. \end{cases}$$

Algorithm 14: Identify the Best Swap

Data: problem size n and A, B, C as $n \times n$ matrices, the permutation π, *bst.cost*

Result: next move and its cost

$bst.swap.cost \leftarrow \infty$, $tabu.bst.swap.cost \leftarrow \infty$, $num \leftarrow 0$;

```
for i: 1 to n do
    for j: 1 to n do
        calculate a(π^ij), b(π^ij)andc(π^ij) according to formulas 4.5 to 4.7;
        cost.post.swap ← a(π^ij)b(π^ij) + c(π^ij);
        if swap(i, j) is non-tabu then
            num++;
            if cost.post.swap < bst.swap.cost then
                let cost.post.swap be bst.swap.cost;
                record the permutation to be bst.pi;
                generate a random number randnum.one;
            end
            else if cost.post.swap = bst.swap.cost then
                generate a random number randnum.two;
                if randnum.two > randnum.one then
                    et cost.post.swap be bst.swap.cost;
                    record the permutation to be bst.pi;
                end
            end
        end
        else
            if cost.post.swap < tabu.bst.swap.cost then
                let cost.post.swap be tabu.bst.swap.cost;
                record the permutation to be tabu.bst.pi;
                generate a random number randnum.one;
            end
            else if cost.post.swap = tabu.bst.swap.cost then
                generate a random number randnum.two;
                if randnum.two > randnum.one then
                    let cost.post.swap be tabu.bst.swap.cost;
                    record the permutation to be tabu.bst.pi;
                end
            end
        end
    end
end
```

if *(num = 0) or ((tabu.bst.swap.cost < bst.swap.cost) and (tabu.bst.swap.cost < bst.cost))* **then**
| Return *tabu.bst.swap.cost* and *tabu.bst.pi*;
end
else
| Return *bst.swap.cost* and *bst.pi*;
end

Algorithm 15can be used as a subroutine under the name Identify the Best Swap. The tabu search used in this thesis is specified as follows.

Algorithm 15: Tabu Swap Local Search Algorithm

Data: problem size n and A, B, C as $n \times n$ matrices, maximum iteration number *max.iter*
Result: *bst.cost* and *bst.sol*
iter $\leftarrow 0$;
bst.cost $\leftarrow \infty$;
bst.sol $\leftarrow \{1, 2, 3, ..., n\}$;
for *iter: 1 to max.iter* **do**
  $s \leftarrow$ Identify the Best Swap (A, B, C, π, iter, bst.cost);
  calculate and record the cost of the swap s, let it be *cur.cost*;
  calculate and record the solution of the swap s, let it be *cur.sol*;
  update Tabu List to record the best swap s;
  if *cur.cost* $<$ *bst.cost* **then**
    bst.cost $\leftarrow$ *cur.cost*;
    bst.sol $\leftarrow$ *cur.sol*;
  end
end
Return *bst.cost* and *bst.sol*.

On the basis of the original tabu search algorithm, Algorithm 16, we can develop a multistart tabu search to approach the MAP in this thesis. As the name suggests, this version of tabu search attempts to initiate the search procedure with the multiple starting solutions. The multiple starting solutions are randomly generated.

Algorithm 16: Multi-start Tabu Swap Local Search Algorithm

Data: problem size n and A, B, C as $n \times n$ matrices, maximum restart number $max.restart$

Result: $bst.cost$ and $bst.sol$

$restart.num \leftarrow 0$;

$bst.cost \leftarrow \infty$;

$bst.sol \leftarrow \{1, 2, 3, ..., n\}$;

if $restart.num < best.obj$ **then**

  $initial.sol \leftarrow rand()$;

  $cr.cost \leftarrow Tabu.Search(A, B, C, n)$

  if $cr.cost < bst.cost$ **then**

    $bst.cost \leftarrow cr.cost$;

    $bst.sol \leftarrow cr.arrow$;

  end

end

Return $bst.cost$ and $bst.sol$.

4.5 Experimental Results

This section is devoted to numerical experiments implementing the previously indicated local search algorithms.

Specifically, we test the following categories of instances.

1. The first group of instances we test is pseudo-random problems, in which all parameters are pseudo-random numbers. We test the problems of size n ranging from 10 to 55.

   (a) Randomly generated problems. The entries in parameter matrix A range from 0 to 100, B from 100 to 200, and C from 200 to 1000. All entries are pseudo-random integers generated in the same manner as in section 2.4.

   (b) Positively correlated random problems. The entries in parameter matrix A range from 0 to 100, B from 100 to 200, and C from 200 to 1000. All entries are pseudo-random integers generated in the same manner as in section 2.4. As the name suggests, the matrices A and B positively correlated. That is to say, each row in matrices A and B is sorted in ascending order.

   (c) Negatively correlated random problems. The entries in parameter matrix A range from 0 to 100, B from 100 to 200, and C from 200 to 1000. All entries are pseudo-random integers generated in the same manner as in section 2.4. As the name suggests, the matrices A and B negatively correlated. That is to say, each row in matrix A is sorted in ascending order and each row in matrix B is sorted in descending order.

2. The second group of instances we test is pseudo-random homogeneous problems. The only difference between this group of problems and the first group is that all the problems in this group do not have matrix C. We also test three different subgroups of pseudo-random homogeneous problems as for pseudo-random problems. We test the problems of size n ranging from 10 to 55.

3. The third group of instances we test is pseudo-random small C problems. The only difference between this group of problems and the first group is that all the problems in this group do have smaller matrix C. More specifically, all the entries in matrix C in this group of problems range from 0 to 200, as opposed to 200 to 1000 in group 1. We test the problems of size n ranging from 10 to 55.

4. The fourth group of instances is the instances from the Quadratic Assignment Problem Library (QAPLIB) [18]. We do not test all the instances from QAPLIB. We focus on testing the most difficult ones of them. As in the experimental protocol of their paper [13], Una Benlic and Jin-Kao Hao states, "Among the 135 instances, 101 instances (including all the real-life instance) can be considered as easy since BLS (and many other state-of-art QAP methods) can solve them to optimality in every singe trial within a very short computation time (often less than a second)." Considering that they tested on the QAP and we test on the MAP, and the MAP are computationally easier than the QAP, we test on the following four groups of QAPLIB problems: J. Skorin-Kapov, E.D. Taillard, U.W. Thonemann and A. Bolte, and M.R. Wilhelm and T.L. Ward.

5. The fifth group of instances are for the most powerful algorithms we have discussed. The instances of this group have larger size n ranging from 500 to 1000. We test them with the MILP7, which is the winner of all linearization methods, in comparison with tabu search. We are interested in comparing the results of the MILP7 and tabu search in terms of both solution quality and computation time. We would like to examine whether the trade-off between solution quality and computation time can be seen.

Following are experimental results. As in previous experiments, the results with computation time greater than 3600s are not shown in the table.

4.6 Experimental Analysis

To show that the effectivenesses of different approaches differ, we run Wilcoxon signed-rank test on the previously indicated methods. In short, Wilcoxon signed-rank test compare two samples and determine if they are significantly different. In this case, the small p-value of a Wilcoxon signed-rank test between the methods of two methods indicates a striking difference between the effectiveness of these two methods. The p-values of the methods are tabulated as follows. An obvious observation based on the p-values is that the best-performing non-tabu search method out of a variety of local search candidates usually resembles Tabu search in terms of solution quality.

According to Table 4.1, we can see that different non-tabu local search methods work best for MAPs with different structures. Specifically, $\alpha A+C$ is the best method for pseudo-random problems; various methods, but mainly ten random initial starts method work best for negatively correlated random problems; various methods work best for positively correlated random problems of different sizes.

Wilcoxon also indicates the similarity between results from different methods. For pseudo-random problems, the results of $\alpha A + C$ are significantly different from all non-tabu local search methods except for $\alpha A + C$ itself and tabu search with 1000 maximum iterations. Notably, $\alpha A + C$ performs similarly with tabu search with 100000 maximum iterations. For negatively correlated random problems, the best-performing methods are mainly ten random initial starts method. The best results are close to those of $\beta B+C$. For positively correlated random problems, tabu search with 100000 maximum iterations give the results close to the best results given by different methods for different problem sizes.

Table 4.1: Wilcoxon test results (p-value) between the best method and other local search

type	best	One Fixed Initial	Multi-starts(10RI)	A+C	B+C	A+B+C	$\alpha A + C$	$\beta B + C$	Tabu(1000)	Tabu(100000)
R	$\alpha A + C$	0.001953	0.003906	0.001953	0.001953	0.001953	—	0.0423	0.04232	0.2719
N	10RI	0.001953	—	0.001953	0.009152	0.009152	0.01427	0.08398	0.01427	0.01427
P	Various	0.001953	0.001953	0.02249	0.01427	0.01427	0.05906	0.001953	0.3223	0.2754

P-value less than 0.05 usually indicates different effectiveness between methods.
Best values indicate the best (with least objective function value) result among various local search methods apart from Tabu search.

Several noteworthy observations can be drawn from the above experimental results.

1. First, we conduct analysis for problems in the most normal form. We look at Table A.31 - A.36. Following are some of the findings.

   (a) Seeing from Table A.31, $\alpha A+C$ is the best-performing method for peusdo random problems. Out of the 10 instances we tested, $\alpha A + C$ found the best quality solutions in 9 of them. Furthermore, it takes only one move to find the final solutions in 7 of the tests. It may suggest the $\alpha A + C$ method takes advantage of some structural features of the MAP.

   (b) In contrast to finding 1, 10RI (multistart) method is the best-performing method for negatively correlated and positively correlated peusdo random problems. As Table A.33 - A.36 show, for all the 20 instances falling into these two categories, 10RI method maintains a steady performance of finding the best quality solutions in all of them. But notably, 10RI method does not use the least number of moves to reach the final solutions. It suggests the existence of the time-quality trade-off pervasive in heuristics.

   (c) Reading across Table A.31 - A.36, one fixed initial and multi-start methods (10RI) perform most steadily regardless of the structures of problems. Namely, one fixed initial and 10RI methods always give the same results in all these 30 tested instances, comprised of pseudo random problems, positively correlated pseudo random problems, and negatively correlated pseudo random problems.

2. Looking at Table A.37 - A.42, we can notice some differences between regular MAPs and homogeneous MAPs.

   (a) Instead of $\alpha A + C$, $A + C$ is the best-performing method for pseudo random homogeneous problems. Considering the structural differences between the MAP and the homogeneous MAP and the differential performances of the methods, we are led to conclude that in implementing heuristics, we need to employ methods according to the structural features of the problem.

   (b) Similar to the case of normal MAPs, multi-start methods (10RI) still outperform other competing local search algorithms in positively and negatively correlated pseudo random problems. This observation consolidates the stability of the multi-start methods in dealing with problems with differing structural characteristics. This finding is of practical significance that we can safely implement multi-start methods to approach problems of unknown structures and expect reasonable results.

3. Looking at Table A.37 - A.48, we can draw some points of interest about MAPs with small parameter matrix Cs.

(a) For pseudo random small C MAP problems, the most effective method is $\alpha A + C$, same with the that of normal pseudo random MAPs. What's more, the most effective approach in dealing with small C MAPs is 10RI method, also same with the situation of normal pseudo random MAPs. This similarity suggests that the size of the parameter C is not a decisive factor in terms of computational difficulty.

Chapter 5

Conclusion

In this thesis, we systematically studied the Multiplicative Assignment Problem (MAP).

First, we studied the formulation of the MAP and presented the MAP under the light of combinatorial optimization problem with product objective function (COPP).

We then approached the MAP by three groups of algorithms. The three groups of algorithms are linearization, Constrained Assignment Problem (CAP), and heuristics. The three groups of algorithms can be classified as exact and approximate algorithms in terms of the exactness of the final solution. Linearization and the CAP are exact algorithms, and heuristics is approximate. For the exact algorithms, we presented the theoretical foundations underpinning them. For heuristics, we presented the rationale for adopting them.

Extensive experiments were conduced to test the algorithms presented in this thesis. On the basis of the experimental results, remarks regarding the effectiveness and efficiency of the algorithms were given.

The overall conclusion is that one of the linearizations introduced in this thesis, MILP7, is the best algorithm to solve the MAP of small and medium size ($n < 500$), in terms of both effectiveness and efficiency. However, for large scale MAP ($n > 500$), heuristics, particularly Tabu Search, still remains very competitive in terms of efficiency.

Bibliography

[1] H. Aarts and K. Lenstra. *Local search in combinatorial optimization.* Princeton University Press, 1997.

[2] W. Adams and T. Johnson. Improved linear programming-based lower bounds for the quadratic assignment problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 16:43–75, 1994.

[3] V. Aggarwal. A lagrangean-relaxation method for the constrained assignment problem. *Computers & Operations Research*, 12(1):97–106, 1985.

[4] R. Ahuja. Minimum cost-reliability ratio path problem. *Computers & operations research*, 15(1):83–89, 1988.

[5] R. Ahuja. The balanced linear programming problem. *European Journal of Operational Research*, 101(1):29–38, 1997.

[6] R. Ahuja, J. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27(10):917–934, 2000.

[7] E. Balas and J. Mazzola. Quadratic 0-1 programming by a new linearization. In *Joint ORSA/TIMS National Meeting*, page 249, 1980.

[8] E. Balas and J. Mazzola. Nonlinear 0–1 programming: I. linearization techniques. *Mathematical Programming*, 30(1):1–21, 1984.

[9] E. Balas and J. Mazzola. Nonlinear 0–1 programming: Ii. dominance relations and algorithms. *Mathematical Programming*, 30(1):22–45, 1984.

[10] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[11] M. Bazaraa and H. Sherali. Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem. *Naval Research Logistics Quarterly*, 27(1):29–41, 1980.

[12] M. Bazaraa and H. Sherali. On the use of exact and heuristic cutting plane methods for the quadratic assignment problem. *Journal of the Operational Research Society*, 33(11):991–1003, 1982.

[13] U. Benlic and J. Hao. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9):4800–4815, 2013.

[14] D. Bertsekas, A. Nedi, A. Ozdaglar, et al. *Convex analysis and optimization.* Athena Scientific, 2003.

[15] E. Buffa, G. Armour, and T. Vollmann. Allocating facilities with craft. *Harvard Business Review*, 42(2):136–158, 1962.

[16] M. Burkard, R. Dell'Amico and S. Martello. *Assignment problems: revised reprint.* SIAM, 2012.

[17] R. Burkard, E. Çela, P. Pardalos, and L. Pitsoulis. The quadratic assignment problem. In *Handbook of combinatorial optimization, Vol. 3*, pages 241–237. Kluwer Academic Publishers, Boston, MA, 1998.

[18] R. Burkard, S. Karisch, and F. Rendl. Qaplib–a quadratic assignment problem library. *Journal of Global optimization*, 10(4):391–403, 1997.

[19] J. Carrell. *Fundamentals of linear algebra.* University of British Columbia, 2005.

[20] E. Cela. *The quadratic assignment problem: theory and algorithms.* Kluwer Academic Publishers, Dordrecht, 1998.

[21] E. Cela, V. Deineko, and G. Woeginger. Linearizable special cases of the qap. *Journal of Combinatorial Optimization*, 31(3):1–11, 2014.

[22] J. Chakrapani and J. Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*, 41(4):327–341, 1993.

[23] IBM ILOG CPLEX. V12. 1: User 's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[24] G. Dantzig. *Linear programming and extensions.* Princeton University Press, 2016.

[25] L. Davis. Genetic algorithms and simulated annealing. 1987.

[26] J. Dickey and J. Hopkins. Campus building arrangement using topaz. *Transportation Research*, 6(1):59–68, 1972.

[27] M. Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[28] M. Drigo, V. Maniezzo, and A. Colorni. The ant system: optimization by a colony of cooperation agents. *IEEE Transactions of Systems, Man, and Cybernetics*, (Part B):29–41, 1996.

[29] A. Elshafei. Hospital layout as a quadratic assignment problem. *Operational Research Quarterly*, pages 167–179, 1977.

[30] G. Erdoğan and B. Tansel. A note on a polynomial time solvable case of the quadratic assignment problem. *Discrete Optimization*, 3(4):382–384, 2006.

[31] C. Fleurent and J. Ferland. Genetic hybrids for the quadratic assignment problem. *Quadratic assignment and related problems*, 16:173–187, 1994.

[32] A. Frieze and J. Yadegar. On the quadratic assignment problem. *Discrete applied mathematics*, 5(1):89–98, 1983.

[33] L. Gambardella, E. Taillard, and M. Dorigo. Ant colonies for the qap. Technical report, Technical Report 4-97, IDSIA, Lugano, Switzerland, 1997.

[34] J. Gavett and N. Plyter. The optimal assignment of facilities to locations by branch and bound. *Operations Research*, 14(2):210–232, 1966.

[35] P. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the society for industrial and applied mathematics*, 10(2):305–313, 1962.

[36] F. Glover. Improved linear integer programming formulations of nonlinear integer problems. *Management Science*, 22(4):455–460, 1975.

[37] F. Glover. Tabu search part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[38] F. Glover. Tabu search part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[39] F. Glover and M. Laguna. *Tabu Search*. Springer, 2013.

[40] D. Goldberg and J. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[41] V. Goyal, L. Genc-Kaya, and R. Ravi. An fptas for minimizing the product of two non-negative linear cost functions. *Mathematical Programming*, 126(2):401–405, 2011.

[42] A. Gupta and J. Sharma. Tree search method for optimal core management of pressurised water reactors. *Computers & Operations Research*, 8(4):263–266, 1981.

[43] A. Gupte, S. Ahmed, M. Cheon, and S. Dey. Solving mixed integer bilinear problems using milp formulations. *SIAM Journal on Optimization*, 23(2):721–744, 2013.

[44] P. Hahn, M. Anjos, R. Burkard, S. Karisch, and F. Rendl. Qaplib-a quadratic assignment problem library. `http://www.seas.upenn.edu/qaplib`, 2006.

[45] J. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[46] IBM. Cplex optimization studio interfaces. `https://www-01.ibm.com/software/commerce/optimization/interfaces`. Accessed: 2016-08-07.

[47] S. Kabadi and A. P. Punnen. An o (n 4) algorithm for the qap linearization problem. *Mathematics of Operations Research*, 36(4):754–761, 2011.

[48] V. Kaibel. *Polyhedral combinatorics of the quadratic assignment problem.* na, 1997.

[49] L. Kaufman and F. Broeckx. An algorithm for the quadratic assignment problem using bender's decomposition. *European Journal of Operational Research*, 2(3):207–211, 1978.

[50] J. Kennington and F. Mohammadi. The singly constrained assignment problem: A lagrangian relaxation heuristic algorithm. *Computational Optimization and Applications*, 3(1):7–26, 1994.

[51] H. Konno and T. Kuno. Linear multiplicative programming. *Mathematical Programming*, 56(1):51–64, 1992.

[52] T. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society*, 25(1):53–76, 1957.

[53] A. Land. A problem of assignment with inter-related costs. *Journal of the Operational Research Society*, 14(2):185–199, 1963.

[54] J. Larusic and A. P. Punnen. The balanced traveling salesmanproblem. *Computers & Operations Research*, 38(5):868–875, 2011.

[55] E. Lawler. The quadratic assignment problem. *Management Science*, 9(4):586–599, 1963.

[56] P. Lieshout and A. Volgenant. A branch-and-bound algorithm for the singly constrained assignment problem. *European Journal of Operational Research*, 176(1):151–164, 2007.

[57] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *Knowledge and Data Engineering, IEEE Transactions on*, 11(5):769–778, 1999.

[58] E. Q. V. Martins. An algorithm to determine a path with minimal cost/capacity ratio. *Discrete Applied Mathematics*, 8(2):189–194, 1984.

[59] S. Mittal and A. Schulz. An fptas for optimizing a class of low-rank functions over a polytope. *Mathematical Programming*, 141(1-2):103–120, 2013.

[60] H. Müller-Merbach. *Optimale reihenfolgen*, volume 15. Springer-Verlag, 2013.

[61] C. Nugent, T. Vollmann, and J. Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, 16(1):150–173, 1968.

[62] M. Padberg and M. Rijal. *Location, scheduling, design and integer programming*, volume 3. Springer Science & Business Media, 2012.

[63] C. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[64] A. P. Punnen. On combined minmax-minsum optimization. *Computers & operations research*, 21(6):707–716, 1994.

[65] A. P. Punnen. Combinatorial optimization with multiplicative objective function. *International Journal of Operations and Quantitative Management*, 7(3):205–210, 2001.

[66] A. P. Punnen and Y. Aneja. A tabu search algorithm for the resource-constrained assignment problem. *Journal of the Operational Research Society*, 46(2):214–220, 1995.

[67] A. P. Punnen and S. K. Bhatt. Some ratio sharing models. *Asia-Pacific Journal of operational research*, 12(2):187–198, 1995.

[68] A. P. Punnen and S. Kabadi. A linear time algorithm for the koopmans–beckmann qap linearization and related problems. *Discrete Optimization*, 10(3):200–209, 2013.

[69] A. P. Punnen and K. P. K. Nair. Constrained balanced optimization problems. *Computers & Mathematics with Applications*, 37(9):157–163, 1999.

[70] M. Queyranne. Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems. *Operations Research Letters*, 4(5):231–234, 1986.

[71] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[72] H. Sherali and W. Adams. *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, volume 31. Springer Science & Business Media, 2013.

[73] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.

[74] É. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4):443–455, 1991.

[75] D. Tate and A. Smith. A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, 22(1):73–83, 1995.

[76] G. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (fptas)? *INFORMS Journal on Computing*, 12(1):57–74, 2000.

Appendix A

Tables

Table A.1: Relaxed Linearizations Results of Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
R1	10	16657	0.063	3441.94	0	293616	0.281	293616	0.468	3879.5	0.078
R2	15	17279.5	0.203	3780.02	0	335032	2.545	335032	6.006	4043.5	0.593
R3	20	5486.2	0.843	5340.41	0.031	405621	25.646	405621	64.754	5686.5	2.792
R4	25	6344.28	7.832	6348.22	0.078	548467	223.76	548467	652.162	6519	41.11
R5	30	7561.57	30.404	7363.08	0.172	559585	1128.37	559585	3063.73	7605.5	2826.22
R6	35	8495.43	672.474	8430.4	0.312	–	–	–	–	–	–
R7	40	–	–	9350.86	3.635	–	–	–	–	–	–
R8	45	–	–	10334.4	7.262	–	–	–	–	–	–
R9	50	–	–	11116.9	13.917	–	–	–	–	–	–
R10	55	–	–	12145.9	23.373	–	–	–	–	–	–

Table A.2: Relaxed Linearizations Results of Negatively Correlated Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
N1	10	43564	0.046	3444.01	0.002	675707	0.303	675707	0.483	3879.5	0.093
N2	15	45038	0.52	3780.22	0.009	1408180	2.663	1408180	8.838	4043.5	0.811
N3	20	6090	1.015	5339.55	0.024	2379440	28.373	2379440	100.122	5686.5	3.963
N4	25	6633	8.247	6348.1	0.067	3656940	222.672	3656940	661.389	6519	1022.37
N5	30	7723.47	31.679	7363.89	0.165	5224570	1170.85	559585	3063.73	–	–
N6	35	–	–	8430.05	0.312	–	–	–	–	–	–
N7	40	–	–	9350.96	3.748	–	–	–	–	–	–
N8	45	–	–	10334.3	7.484	–	–	–	–	–	–
N9	50	–	–	11116.8	13.874	–	–	–	–	–	–
N10	55	–	–	12145.6	24.376	–	–	–	–	–	–

Table A.3: Relaxed Linearizations Results of Positively Correlated Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
P1	10	43292	0.047	3444.87	0.001	647710	0.263	647710	0.482	3879.5	0.074
P2	15	44980.5	0.405	3780.08	0.013	1419460	2.425	1419460	8.279	4043.5	0.599
P3	20	6090	0.862	5339.68	0.025	2378980	27.236	2378980	103.034	5686.5	2.829
P4	25	6633	8.198	6348.17	0.064	3644840	201.503	3644840	725.087	6519	41.487
P5	30	7723.47	30.489	7363.95	0.159	5278980	1208.69	–	–	7605.5	3082.36
P6	35	8537.14	278.692	8430.03	0.312	–	–	–	–	–	–
P7	40	–	–	9350.87	0.712	–	–	–	–	–	–
P8	45	–	–	10334.3	7.516	–	–	–	–	–	–
P9	50	–	–	11116.9	13.774	–	–	–	–	–	–
P10	55	–	–	12145.6	23.92	–	–	–	–	–	–

Table A.4: Integer Solutions to Traditional Linearizations Results of Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
R1	10	293616	1572.37	293616	2.745	293616	0.234	293616	0.109	293616	46.74
R2	15	–	–	335032	248.686	335032	2.137	335032	0.92	–	–
R3	20	–	–	–	–	405621	12.012	405621	6.712	–	–
R4	25	–	–	–	–	548467	88.859	548467	28.128	–	–
R5	30	–	–	–	–	559585	477.359	559585	109.013	–	–
R6	35	–	–	–	–	549248	382.29	549248	240.524	–	–
R7	40	–	–	–	–	634078	1000.49	634078	549.307	–	–
R8	45	–	–	–	–	742206	3013.58	742206	1145.29	–	–
R9	50	–	–	–	–	–	–	847876	1883.61	–	–
R10	55	–	–	–	–	–	–	803139	2888.85	–	–

Table A.5: Integer Solutions to Traditional Linearizations Results of Negatively Correlated Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
N1	10	–	–	675707	59.015	675707	0.375	675707	0.468	–	–
N2	15	–	–	–	–	1408180	3.635	1408180	6.723	–	–
N3	20	–	–	–	–	2379440	35.616	2379440	72.072	–	–
N4	25	–	–	–	–	3656940	238.079	3656940	582.581	–	–
N5	30	–	–	–	–	5224570	1227.99	5224570	3369.36	–	–
N6	35	–	–	–	–	–	–	–	–	–	–
N7	40	–	–	–	–	–	–	–	–	–	–
N8	45	–	–	–	–	–	–	–	–	–	–
N9	50	–	–	–	–	–	–	–	–	–	–
N10	55	–	–	–	–	–	–	–	–	–	–

Table A.6: Integer Solutions to Traditional Linearizations of Positively Correlated Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
P1	10	–	–	647710	38.912	647710	0.359	647710	0.39	–	–
P2	15	–	–	–	–	1419460	3.519	1419460	6.351	–	–
P3	20	–	–	–	–	2378980	35.504	2378980	71.385	–	–
P4	25	–	–	–	–	3644840	224.194	3644840	547.496	–	–
P5	30	–	–	–	–	5278980	1180.48	5278970	3109.81	–	–
P6	35	–	–	–	–	–	–	–	–	–	–
P7	40	–	–	–	–	–	–	–	–	–	–
P8	45	–	–	–	–	–	–	–	–	–	–
P9	50	–	–	–	–	–	–	–	–	–	–
P10	55	–	–	–	–	–	–	–	–	–	–

Table A.7: Experimental Results of New Linearizations on Pseudo Random Problems

		MILP6		MILP7		MILP6 Integer		MILP7 Integer	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Integer Obj. Value	Time(s)	Integer Obj. Value	Time(s)
R1	10	251589	0	292552	0	293616	0.172	293616	0
R2	15	294711	0	335032	0	335032	0.031	335032	0.016
R3	20	353170	0.016	405621	0	405621	0.093	405621	0.015
R4	25	463894	0.062	543244	0	548467	0.421	548467	0.031
R5	30	435136	0.078	549237	0	559585	0.952	559585	0.266
R6	35	450316	0.078	546595	0	549248	0.717	549248	0.031
R7	40	529871	0.078	631020	0	634078	1.045	634078	0.031
R8	45	590955	0.094	742206	0	742206	8.991	742206	0.031
R9	50	667489	0.125	841481	0	847876	19.299	847876	0.234
R10	55	636610	0.156	797769	0.016	803139	59.246	803139	0.218

Table A.8: New Linearizations Experimental Results of Negatively Correlated Pseudo Random Problems

		MILP6		MILP7		MILP6 Integer		MILP7 Integer	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Integer Obj. Value	Time(s)	Integer Obj. Value	Time(s)
N1	10	633983	0	674861	0	675707	0.281	675707	0.031
N2	15	1364290	0.016	1407750	0.016	1408180	0.109	1408180	0.156
N3	20	2267150	0.015	2378890	0	2379440	7.724	2379440	0.031
N4	25	3483700	0.031	3655180	0	3656940	182.211	3656940	0.016
N5	30	5060900	0.078	5224270	0.016	5224570	472.668	5224570	0.031
N6	35	7123220	0.094	7410560	0.016	–	–	7411970	0.047
N7	40	9386540	0.093	9752550	0.015	–	–	9755480	0.032
N8	45	12090200	0.109	12507000	0	–	–	12509200	0.266
N9	50	15222500	0.125	15849200	0.015	–	–	15853700	0.437
N10	55	18507200	0.14	19169000	0.016	–	–	19172100	0.094

Table A.9: New Linearizations Experimental Results of Positively Correlated Pseudo Random Problems

		MILP6		MILP7		MILP6 Integer		MILP7 Integer	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Integer Obj. Value	Time(s)	Integer Obj. Value	Time(s)
P1	10	629620	0	647440	0	647710	0.203	647710	0.015
P2	15	1366830	0.015	1419130	0	1419460	0.14	1419460	0.15
P3	20	2267110	0.016	2377990	0	2378980	9.376	2378980	0.031
P4	25	3482760	0.032	3643850	0	3644840	50.606	3644840	0.015
P5	30	5066390	0.078	5277870	0.015	–	–	5278980	0.031
P6	35	7120120	0.094	7412310	0.016	–	–	7414230	0.046
P7	40	9387160	0.109	9754530	0	–	–	9758080	0.172
P8	45	12094800	0.094	12540000	0.016	–	–	12543000	0.109
P9	50	15212900	0.11	15774100	0	–	–	15778100	0.047
P10	55	18495400	0.125	19072200	0.016	–	–	19075300	0.312

Table A.10: Integer Solutions to Traditional Linearizations on Homogeneous Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
HR1	10	287448	947.187	287448	3.417	287448	0.219	287448	0.14	287448	63.22
HR2	15	–	–	325431	391.642	325431	1.81	325431	0.966	–	–
HR3	20	–	–	–	–	392620	9.548	392620	7.574	–	–
HR4	25	–	–	–	–	533181	96.549	533181	28.363	–	–
HR5	30	–	–	–	–	541722	519.951	541722	104.676	–	–
HR6	35	–	–	–	–	529686	343.998	529686	239.155	–	–
HR7	40	–	–	–	–	609336	992.838	609336	501.013	–	–
HR8	45	–	–	–	–	714228	2946.34	714228	1135.74	–	–
HR9	50	–	–	–	–	–	–	818944	1899.09	–	–
HR10	55	–	–	–	–	–	–	766176	2921.16	–	–

Table A.11: Integer Solutions to Traditional Linearizations Results of Negatively Correlated Homogeneous Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
HN1	10	–	–	668811	59.567	668811	0.359	668811	0.265	–	–
HN2	15	–	–	–	–	1401030	3.494	1401030	6.76	–	–
HN3	20	–	–	–	–	2366490	33.681	2366490	71.247	–	–
HN4	25	–	–	–	–	3642350	235.958	3642350	563.951	–	–
HN5	30	–	–	–	–	5204170	1167.91	5204170	3438.03	–	–
HN6	35	–	–	–	–	–	–	–	–	–	–
HN7	40	–	–	–	–	–	–	–	–	–	–
HN8	45	–	–	–	–	–	–	–	–	–	–
HN9	50	–	–	–	–	–	–	–	–	–	–
HN10	55	–	–	–	–	–	–	–	–	–	–

Table A.12: Integer Solutions to Traditional Linearizations Results of Positively Correlated Homogeneous Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
HP1	10	–	–	641516	45.731	641516	0.344	641516	0.25	641516	2870.5
HP2	15	–	–	–	–	1411890	3.573	1411890	6.786	–	–
HP3	20	–	–	–	–	2365500	37.066	2365500	75.147	–	–
HP4	25	–	–	–	–	3630090	237.748	3630090	572.593	–	–
HP5	30	–	–	–	–	5260990	1235.89	5260990	3365.41	–	–
HP6	35	–	–	–	–	–	–	–	–	–	–
HP7	40	–	–	–	–	–	–	–	–	–	–
HP8	45	–	–	–	–	–	–	–	–	–	–
HP9	50	–	–	–	–	–	–	–	–	–	–
HP10	55	–	–	–	–	–	–	–	–	–	–

Table A.13: New Linearizations Experimental Results of Homogeneous Pseudo Random Problems

		MILP6		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)
HR1	10	287448	0.265	287448	0.016
HR2	15	325431	0.078	325431	0.016
HR3	20	392620	0.109	392620	0.015
HR4	25	533181	0.421	533181	0.031
HR5	30	541722	0.951	541722	0.25
HR6	35	529686	0.998	529686	0.047
HR7	40	609336	1.076	609336	0.062
HR8	45	714228	9.766	714228	0.031
HR9	50	818944	23.622	818944	0.188
HR10	55	766176	22.588	766176	0.218

Table A.14: New Linearizations Experimental Results of Negatively Correlated Homogeneous Pseudo Random Problems

		MILP6		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)
HN1	10	668811	0.156	668811	0.016
HN2	15	1401030	0.156	1401030	0.016
HN3	20	2366490	9.313	2366490	0.015
HN4	25	3642350	247.695	3642350	0.031
HN5	30	5204170	464.433	5204170	0.25
HN6	35	–	–	7393290	0.047
HN7	40	–	–	9728830	0.156
HN8	45	–	–	12480400	0.078
HN9	50	–	–	15823800	0.343
HN10	55	–	–	19138300	0.218

Table A.15: New Linearizations Experimental Results of Positively Correlated Homogeneous Pseudo Random Problems

		MILP6		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)
HP1	10	641516	0.172	641516	0.016
HP2	15	1411890	0.156	1411890	0
HP3	20	2365500	7.558	2365500	0.015
HP4	25	3630090	56.259	3630090	0.031
HP5	30	–	–	5260990	0.031
HP6	35	–	–	7394530	0.032
HP7	40	–	–	9732460	0.25
HP8	45	–	–	12515400	0.047
HP9	50	–	–	15749400	0.063
HP10	55	–	–	19039500	0.281

Table A.16: Integer Solutions to Traditional Linearizations Results of Small C Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
SR1	10	288334	954.105	288334	3.417	288334	0.219	288334	0.125	288334	63.446
SR2	15	–	–	327067	391.642	327067	1.872	327067	0.921	–	–
SR3	20	–	–	–	–	394262	8.705	394262	0.991	–	–
SR4	25	–	–	–	–	535865	92.394	535865	6.991	–	–
SR5	30	–	–	–	–	544696	511.322	544696	91.402	–	–
SR6	35	–	–	–	–	533278	344.339	533278	294.085	–	–
SR7	40	–	–	–	–	612629	1009.6	612629	493.359	–	–
SR8	45	–	–	–	–	718662	3008.09	718662	1330.62	–	–
SR9	50	–	–	–	–	–	–	824355	1919.01	–	–
SR10	55	–	–	–	–	–	–	772015	3023.33	–	–

Table A.17: Integer Solutions to Traditional Linearizations Results of Negatively Correlated Small C Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
SN1	10	–	–	669772	53.572	669772	0.328	669772	0.281	–	–
SN2	15	–	–	1402490	53.572	1402490	3.525	1402490	6.63	–	–
SN3	20	–	–	–	–	2368480	33.741	2368480	71.36	–	–
SN4	25	–	–	–	–	3645030	238.552	3645030	565.237	–	–
SN5	30	–	–	–	–	5207260	1207.06	5207260	3456.02	–	–
SN6	35	–	–	–	–	–	–	–	–	–	–
SN7	40	–	–	–	–	–	–	–	–	–	–
SN8	45	–	–	–	–	–	–	–	–	–	–
SN9	50	–	–	–	–	–	–	–	–	–	–
SN10	55	–	–	–	–	–	–	–	–	–	–

Table A.18: Integer Solutions to Traditional Linearizations Results of Positively Correlated Small C Pseudo Random Problems

		Lawler		KB		FY		AJ		RLT	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
SP1	10	–	–	642524	46.47	642524	0.359	642524	0.265	642524	2826.21
SP2	15	–	–	–	–	1413360	3.401	1413360	6.505	–	–
SP3	20	–	–	–	–	2367440	34.647	2367440	73.648	–	–
SP4	25	–	–	–	–	3632610	226.764	3632610	543.925	–	–
SP5	30	–	–	–	–	5264010	1162.27	5264010	3240.91	–	–
SP6	35	–	–	–	–	–	–	–	–	–	–
SP7	40	–	–	–	–	–	–	–	–	–	–
SP8	45	–	–	–	–	–	–	–	–	–	–
SP9	50	–	–	–	–	–	–	–	–	–	–
SP10	55	–	–	–	–	–	–	–	–	–	–

Table A.19: New Linearizations Experimental Results of Small C Pseudo Random Problems

		MILP6		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)
SR1	10	288334	0.156	288334	0.016
SR2	15	327067	0.156	327067	0.016
SR3	20	394262	9.313	394262	0.016
SR4	25	535865	247.695	535865	0.016
SR5	30	544696	464.433	544696	0.187
SR6	35	–	–	533278	0.047
SR7	40	–	–	612629	0.062
SR8	45	–	–	718662	0.015
SR9	50	–	–	824355	0.297
SR10	55	–	–	772015	0.062

Table A.20: New Linearizations Experimental Results of Negatively Correlated Small C Pseudo Random Problems

		MILP6		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)
SN1	10	669772	0.156	669772	0
SN2	15	1402490	0.156	1402490	0.016
SN3	20	2368480	9.313	2368480	0.031
SN4	25	3645030	247.695	3645030	0.015
SN5	30	5207260	464.433	5207260	0.016
SN6	35	–	–	7397090	0.047
SN7	40	–	–	9732370	0.031
SN8	45	–	–	12484600	0.171
SN9	50	–	–	15829700	0.094
SN10	55	–	–	19144900	0.249

Table A.21: New Linearizations Experimental Results of Positively Correlated Small C Pseudo Random Problems

		MILP6		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)
SP1	10	642524	0.156	642524	0.125
SP2	15	1413360	0.156	1413360	0.016
SP3	20	2367440	9.313	2367440	0.031
SP4	25	3632610	247.695	3632610	0.016
SP5	30	5264010	464.433	5264010	0.031
SP6	35	–	–	7397590	0.032
SP7	40	–	–	9736320	0.187
SP8	45	–	–	12519600	0.078
SP9	50	–	–	15754200	0.063
SP10	55	–	–	19045100	0.265

Table A.22: CAP Algorithms Experimental Results on Randomly Generated Problems

		Iterated CAP		Iterated Relaxed CAP		Modified CAP		Modified Relaxed CAP	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
R1	10	293616	0.343	342081	0.016	287448	0.109	287448	0.031
R2	15	335032	1.139	325431	0.032	325431	0.016	325431	0.031
R3	20	405621	2.34	392620	0.031	392620	0.078	392620	0.032
R4	25	548467	3.354	544604	0.046	451328	0.156	533181	0.515
R5	30	559585	10.363	542581	0.047	541722	0.437	541722	0.842
R6	35	549248	13.682	529686	0.047	529686	0.546	529686	0.125
R7	40	634070	14.293	617034	0.047	609336	0.406	609336	1.03
R8	45	742206	29.933	714228	0.063	714228	0.842	714228	0.405
R9	50	848543	31.053	819060	0.062	818944	1.03	818944	0.78
R10	55	803139	59.62	771120	0.07	766176	1.139	766176	1.95

Table A.23: CAP Algorithms Experimental Results on Randomly Generated Negatively Correlated Problems

		Iterated CAP		Iterated Relaxed CAP		Modified CAP		Modified Relaxed CAP	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
N1	10	550592	0.078	668811	0.031	668811	0.109	668811	0.156
N2	15	1326586	0.343	1401057	0.031	1401034	0.246	1401030	0.188
N3	20	2162733	0.015	2366701	0.031	2151996	0.031	2366490	0.171
N4	25	3657100	4.883	3642373	0.047	3642354	0.718	3642350	0.249
N5	30	5224568	4.104	5213725	0.046	5204168	0.814	5204170	0.452
N6	35	7411971	8.33	7449126	0.062	7393287	1.357	7393290	0.718
N7	40	9546393	3.398	9780960	0.063	9728828	1.857	9728830	0.904
N8	45	12509195	10.975	12539442	0.078	12480435	4.336	12480400	0.937
N9	50	15853727	14.86	15846617	0.094	15823818	4.508	15823800	1.092
N10	55	19172323	19.968	19174357	0.125	19138328	3.775	19138300	1.436

Table A.24: CAP Algorithms Experimental Results on Randomly Generated Positively Correlated Problems

		Iterated CAP		Iterated Relaxed CAP		Modified CAP		Modified Relaxed CAP	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
P1	10	647710	0.141	641516	0.031	641516	0.078	641516	0.203
P2	15	1346674	0.047	1412726	0.031	1411893	0.203	1411890	0.109
P3	20	1910430	0.031	2365706	0.032	1900464	0.047	2365500	0.202
P4	25	3644844	4.379	3632389	0.047	3630088	0.531	3630090	0.249
P5	30	5278975	5.763	5261242	0.047	5261325	0.749	5260990	0.281
P6	35	7414227	7.719	7416915	0.047	7394530	1.139	7394530	0.561
P7	40	9758080	10.296	9777530	0.063	9732460	2.278	9732460	0.952
P8	45	12416182	7.111	12560694	0.078	12515412	2.761	12515400	1.076
P9	50	15778079	15.195	15779904	0.078	15749400	4.322	15749400	1.092
P10	55	19075502	17.456	19075873	0.125	19039514	3.775	19039600	1.529

Table A.25: CAP Algorithms Experimental Results on Randomly Generated Homogeneous Problems

		Iterated CAP		Iterated Relaxed CAP		Modified CAP		Modified Relaxed CAP	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
HR1	10	287448	0.468	342081	0.031	287448	0.234	287448	1.045
HR2	15	325431	1.172	325431	0.031	325431	0.093	325431	0.031
HR3	20	392620	2.358	392620	0.032	392620	0.078	392620	0.047
HR4	25	533181	3.202	544605	0.031	451328	0.141	533181	0.515
HR5	30	541722	10.811	542582	0.047	541722	0.437	541722	0.842
HR6	35	529686	13.313	529686	0.031	529686	0.437	529686	0.14
HR7	40	609336	14.235	617035	0.046	609336	0.435	609336	1.03
HR8	45	714228	30.008	714228	0.062	714228	0.714	714228	0.405
HR9	50	818944	30.489	819060	0.063	818944	1.038	818944	0.782
HR10	55	766176	61.283	771120	0.078	766176	1.141	766176	1.922

Table A.26: CAP Algorithms Experimental Results on Randomly Generated Negatively Correlated Homogeneous Problems

		Iterated CAP		Iterated Relaxed CAP		Modified CAP		Modified Relaxed CAP	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
HN1	10	545400	0.078	668811	0.031	668811	0.109	668811	0.156
HN2	15	1319200	0.374	1401060	0.031	1401034	0.296	1401030	0.188
HN3	20	2151996	0.015	2366700	0.047	2151996	0.031	2366490	0.171
HN4	25	3642354	4.88	3642370	0.047	3642354	0.749	3642350	0.249
HN5	30	5204168	4.056	5213720	0.047	5204168	0.844	5204170	0.452
HN6	35	7393287	7.85	7449130	0.047	7393287	1.386	7393290	0.702
HN7	40	9519976	3.154	9780960	0.063	9728828	1.934	9728830	0.906
HN8	45	12480435	10.811	12539400	0.062	12480435	4.04	12480400	0.936
HN9	50	15823818	15.14	15846600	0.078	15823818	4.399	15823800	1.076
HN10	55	19138328	21.094	19174400	0.14	19138328	3.807	19138300	1.44

Table A.27: CAP Algorithms Experimental Results on Randomly Generated Positively Correlated Homogeneous Problems

		Iterated CAP		Iterated Relaxed CAP		Modified CAP		Modified Relaxed CAP	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
HP1	10	641516	0.146	641516	0.146	668811	0.109	641516	0.219
HP2	15	1339888	0.047	1412730	0.047	1401034	0.047	1411890	0.109
HP3	20	1900464	0.046	2365710	0.046	2151996	0.046	236550	0.202
HP4	25	3630088	3.766	3632390	3.766	3642354	3.766	3630090	0.265
HP5	30	5260990	5.725	5261240	5.725	5204168	5.725	5260990	0.296
HP6	35	7394530	7.067	7416920	7.067	7393287	7.067	7394530	0.562
HP7	40	9732460	9.609	9777530	9.609	9728828	9.609	9732460	0.936
HP8	45	12392820	7.403	12560700	7.403	12480435	7.403	12515400	1.077
HP9	50	15749408	15.4	15779900	15.4	15823818	15.4	15749400	1.092
HP10	55	19039514	17.375	19075900	17.375	19138328	17.375	19039600	1.513

Table A.28: CAP Algorithms Experimental Results on Randomly Generated Small C Problems

		Iterated CAP		Iterated Relaxed CAP		Modified CAP		Modified Relaxed CAP	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
SR1	10	288334	0.499	342081	0.031	287448	0.188	287448	1.03
SR2	15	327067	1.234	325431	0.031	325431	0.094	325431	0.032
SR3	20	394262	2.309	392620	0.047	392620	0.078	392620	0.031
SR4	25	535865	3.229	544605	0.047	451328	0.156	533181	0.517
SR5	30	544696	10.36	542582	0.063	541722	0.483	541722	0.827
SR6	35	533278	13.65	529686	0.047	529686	0.468	529686	0.14
SR7	40	612629	13.841	617035	0.047	609336	0.423	609336	1.045
SR8	45	718662	29.614	714228	0.062	714228	0.765	714228	0.408
SR9	50	824515	30.334	819060	0.048	818944	0.951	818944	0.827
SR10	55	772015	59.794	771120	0.08	766176	1.108	766176	1.934

Table A.29: CAP Algorithms Experimental Results on Randomly Generated Negatively Correlated Small C Problems

		Iterated CAP		Iterated Relaxed CAP		Modified CAP		Modified Relaxed CAP	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
SN1	10	546163	0.078	668811	0.031	668811	0.109	668811	0.172
SN2	15	1320489	0.249	1401060	0.047	1401034	0.327	1401030	0.187
SN3	20	2153739	0.016	2366700	0.047	2151996	0.031	2366490	0.172
SN4	25	3645031	5.475	3642370	0.047	3642354	0.718	3642350	0.25
SN5	30	5207255	3.987	5213720	0.047	5204168	0.733	5204170	0.452
SN6	35	7397093	8.471	7449130	0.047	7393287	1.357	7393290	0.733
SN7	40	9523197	3.075	9780960	0.062	9728828	1.856	9728830	0.954
SN8	45	12484637	10.619	12539400	0.063	12480435	4.232	12480400	0.983
SN9	50	15829672	15.089	15846600	0.093	15823818	4.618	15823800	0.092
SN10	55	19144859	19.693	19174400	0.125	19138328	3.84	19138300	1.415

Table A.30: CAP Algorithms Experimental Results on Randomly Generated Positively Correlated Small C Problems

		Iterated CAP		Iterated Relaxed CAP		Modified CAP		Modified Relaxed CAP	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
SP1	10	642524	0.141	641516	0.031	641516	0.062	641516	0.219
SP2	15	1341024	0.047	1412730	0.031	1411893	0.234	1411890	0.109
SP3	20	1902067	0.046	2365710	0.032	1900464	0.047	2365500	0.202
SP4	25	3632611	3.838	3632390	0.047	3630088	0.577	3630090	0.249
SP5	30	5264008	5.776	5261240	0.047	5261325	0.733	5260990	0.297
SP6	35	7397588	7.073	7416920	0.047	7394530	1.281	7394530	0.592
SP7	40	9736322	9.556	9777530	0.062	9732460	2.246	9732460	1.045
SP8	45	12397163	7.261	12560700	0.078	12515412	2.778	12515400	1.17
SP9	50	15754168	15.02	15779900	0.093	15749408	4.336	15749400	1.076
SP10	55	19045077	16.674	19075900	0.14	19039514	3.822	19039600	1.529

Table A.31: Local Search Algorithms Experimental Results on Randomly Generated Problems (A)

		One Fixed Initial			Multi-starts(10RI)			A+C			B+C		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.
R1	10	312020	0	83	293616	1.461	83	330075	0.141	81	368101	0.219	77
R2	15	407354	0	72	385752	2.13	72	471856	0.171	67	572901	0.109	61
R3	20	592814	0	60	479161	4.552	56	560668	0.203	61	479621	0.202	66
R4	25	766827	0	52	749849	5.811	49	690927	0.234	55	778846	0.249	51
R5	30	916887	0	46	708564	5.357	44	908784	0.282	48	998810	0.297	45
R6	35	930878	0	45	837161	11.949	43	955766	0.328	46	1034600	0.592	43
R7	40	1205927	0	39	1034215	10.022	38	1259525	0.297	40	1353136	1.045	37
R8	45	1314719	0	36	1312691	16.538	31	1473292	0.343	36	1755674	1.17	32
R9	50	1838405	0	31	1580979	17.151	30	1542353	0.421	35	1864322	1.076	30
R10	55	1674918	0	31	1526631	22.502	30	2018849	0.484	31	2208373	1.529	27

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.32: Local Search Algorithms Experimental Results on Randomly Generated Problems (B)

		A+B+C			αA + C			βB + C			Best		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Best Obj. Value	Method	L.N.
R1	10	368101	0	77	350712	0.015	79	309039	0	84	309039	βB + C	84
R2	15	432500	0.015	70	335032	0	1	406266	0.016	72	335032	αA + C	1
R3	20	492834	0.016	66	405621	0.015	1	567368	0	61	405621	αA + C	1
R4	25	810242	0.016	51	563556	0	1	777013	0.015	52	563556	αA + C	1
R5	30	816772	0.031	50	559585	0.016	63	995853	0.016	45	559585	αA + C	63
R6	35	1254966	0.016	40	549248	0.015	1	843902	0.016	47	549248	αA + C	1
R7	40	1244558	0	40	641937	0.016	59	1033425	0.016	42	641937	αA + C	59
R8	45	1344473	0.016	37	742206	0.031	1	1810295	0.016	32	742206	αA + C	1
R9	50	1747908	0.031	34	848512	0.016	1	1582943	0.031	33	848512	αA + C	1
R10	55	1859502	0.016	32	810950	0.015	1	2154462	0.032	28	810950	αA + C	1

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.33: Local Search Algorithms Experimental Results on Randomly Generated Negatively Correlated Problems (A)

		One Fixed Initial			Multi-starts(10RI)			A+C			B+C		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.
N1	10	688892	0	56	675707	0	57	693404	0	56	679239	0.016	57
N2	15	1457297	0	39	1408915	0	39	1416293	0.016	39	1416293	0	39
N3	20	2411525	0	30	2379444	4.552	30	2412849	0.016	30	2412849	0	30
N4	25	3662905	0	25	3657613	5.811	25	3676528	0	25	3684055	0.016	25
N5	30	5267626	0	21	5245147	5.357	21	5264964	0.016	21	5234499	0.016	21
N6	35	7511250	0	17	7446833	11.949	17	7465399	0.016	17	7449528	0.015	17
N7	40	9827427	0	15	9776473	10.022	15	9811808	0.015	15	9805899	0.016	15
N8	45	12573188	0	13	12533434	16.538	14	12615456	0.031	13	12615456	0.015	13
N9	50	15967096	0	12	15894859	17.151	12	15918861	0.031	12	15918861	0.016	12
N10	55	19298339	0	11	19231265	22.502	11	19352583	0.031	11	19363335	0.016	11

L.N. represents the number of the loops (moves) (moves) of the local search component of the algorithm employs.

Table A.34: Local Search Algorithms Experimental Results on Randomly Generated Negatively Correlated Problems (B)

		A+B+C			αA + C			βB + C			Best		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Best Obj. Value	Method	L.N.
N1	10	693404	0	56	679239	0.015	1	693404	0	56	675707	10RI	57
N2	15	1416293	0.015	39	1415961	0	1	1423704	0.016	39	1408915	10RI	39
N3	20	2412849	0.016	30	2387324	0.015	31	2406927	0	30	2379444	10RI	30
N4	25	3676528	0.001	25	3664505	0.031	25	3664103	0.016	25	3657613	10RI	25
N5	30	5234499	0	21	5243901	0.015	21	5254121	0.016	21	5234499	B+C	21
N6	35	7496275	0.016	17	7422273	0.015	18	7451310	0.016	17	7422273	$\alpha A + C$	43
N7	40	9805899	0.016	15	9785430	0.015	15	9784682	0.016	15	9776473	10RI	15
N8	45	12615456	0.016	13	12568900	0.016	14	12570019	0.015	14	12533434	10RI	14
N9	50	15968276	0.015	12	15904835	0.032	12	15996018	0.015	14	15894859	10RI	12
N10	55	19352583	0.031	11	19204147	0.031	11	19368275	0.016	11	19204147	$\alpha A + C$	11

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.35: Local Search Algorithms Experimental Results on Randomly Generated Positively Correlated Problems (A)

		One Fixed Initial			Multi-starts(10RI)			A+C			B+C		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.
P1	10	647710	0	83	647710	—	83	647710	0.015	58	647710	0	58
P2	15	1459659	0	72	1419457	—	72	1419457	0	39	1419457	0.016	39
P3	20	2406053	0	60	2397984	—	56	2398475	0.016	30	2398475	0	30
P4	25	3671294	0	52	3647514	—	49	3715959	0.016	24	3692643	0.016	25
P5	30	5378898	0	46	5279519	—	44	5314930	0.016	21	5343883	0.016	20
P6	35	7445968	0	45	7441245	—	43	7472211	0.031	17	7483409	0.016	17
P7	40	9811411	0	39	9784692	—	38	9919094	0.032	15	9847182	0.031	15
P8	45	12668866	0	36	12596492	—	31	12606055	0.016	13	12606055	0.016	13
P9	50	15948291	0	31	15862198	—	30	15947250	0.015	12	15859588	0.016	12
P10	55	19177428	0	31	19145194	—	30	19129555	0.015	11	19238912	0.032	11

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.36: Local Search Algorithms Experimental Results on Randomly Generated Positively Correlated Problems (B)

		A+B+C			αA + C			βB + C			Best		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Best Obj. Value	Method	L.N.
P1	10	647710	0.016	58	647710	0	58	686858	0.015	57	647710	10RI	83
P2	15	1419457	0.015	39	1419457	0	39	1459566	0.016	39	1419457	10RI	72
P3	20	2398475	0.016	30	2383359	0.015	31	2434861	0.016	30	2383355	αA + C	56
P4	25	3719633	0.015	24	3649463	0	25	3649463	0.016	25	3647514	10RI	49
P5	30	5304582	0.015	21	5288394	0.016	21	5330998	0.015	21	5279519	10RI	44
P6	35	7483409	0.015	17	7448192	0.016	17	7474100	0.016	17	7441245	10RI	43
P7	40	9838892	0.015	15	9805774	0.016	15	9800800	0.016	15	9784692	10RI	38
P8	45	12610265	0.015	13	12585316	0.031	14	12614244	0.016	14	12585316	αA + C	31
P9	50	15891635	0.031	12	15855861	0.016	12	15902030	0.031	12	15855861	αA + C	30
P10	55	19241201	0.015	11	19139237	0.031	11	19188667	0.032	11	19129555	A + C	30

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.37: Local Search Algorithms Experimental Results on Randomly Generated Homogeneous Problems (A)

		One Fixed Initial			Multi-starts(10RI)			A+C			B+C		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.
HR1	10	312020	0	83	293616	—	83	345072	0.015	79	302480	0	84
HR2	15	407354	0	72	385752	—	72	325431	0.016	1	397056	0.016	72
HR3	20	592814	0	60	479161	—	56	392620	0	1	466440	0.016	67
HR4	25	766827	0	52	749849	—	49	548100	0	1	763552	0.016	52
HR5	30	916887	0	46	708564	—	44	541722	0.016	63	820170	0.016	49
HR6	35	930878	0	45	837161	—	43	529686	0.016	1	824428	0.015	47
HR7	40	1205927	0	39	1034215	—	38	617176	0.015	59	1009980	0.016	42
HR8	45	1314719	0	36	1312691	—	31	714228	0.046	1	1785420	0.016	32
HR9	50	1838405	0	31	1580979	—	30	819060	0.016	1	1721169	0.031	32
HR10	55	1674918	0	31	1526631	—	30	774810	0.031	1	2119424	0.032	28

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.38: Local Search Algorithms Experimental Results on Randomly Generated Homogeneous Problems (B)

		A+B+C			αA + C			βB + C			Best		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Best Obj. Value	Method	L.N.
HR1	10	324722	0	82	345072	0	79	302480	0	84	293616	10RI	83
HR2	15	424440	0	72	325431	0.015	1	397056	0	72	325431	A+C	1
HR3	20	515318	0.016	65	392620	0	1	466440	0.016	67	392620	A+C	1
HR4	25	700685	0.016	55	548100	0.015	1	763552	0	52	548100	A+C	1
HR5	30	909650	0	49	541722	0.015	63	820170	0.016	49	541722	A+C	63
HR6	35	898480	0.016	48	529686	0.015	1	824428	0.016	47	529686	A+C	1
HR7	40	1039104	0.016	45	617176	0.015	59	1009980	0.016	42	617176	A+C	59
HR8	45	1599864	0.016	37	714228	0.016	1	1785420	0.015	32	714228	A+C	1
HR9	50	1565728	0.016	37	819060	0.031	1	1721169	0.015	32	819060	A+C	1
HR10	55	2130432	0.015	32	774810	0.031	1	2119424	0.016	28	774810	A+C	1

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.39: Local Search Algorithms Experimental Results on Randomly Generated Homogeneous Negatively Correlated Problems (A)

		One Fixed Initial			Multi-starts(10RI)			A+C			B+C		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.
HN1	10	312020	0	83	293616	—	83	673672	0.015	1	673672	0	57
HN2	15	407354	0	72	385752	—	72	1408716	0.016	1	1416160	0	39
HN3	20	592814	0	60	479161	—	56	2375592	0.016	31	2396332	0	30
HN4	25	766827	0	52	749849	—	49	3648576	0.016	25	3649344	0.016	25
HN5	30	916887	0	46	708564	—	44	5223708	0.016	21	5228496	0.016	21
HN6	35	930878	0	45	837161	—	43	7403376	0.015	18	7485678	0.031	17
HN7	40	1205927	0	39	1034215	—	38	9778239	0.016	15	9806850	0.015	15
HN8	45	1314719	0	36	1312691	—	31	12540808	0.016	14	12605562	0.031	13
HN9	50	1838405	0	31	1580979	—	30	15874969	0.031	12	15903348	0.016	12
HN10	55	1674918	0	31	1526631	—	30	19180744	0.032	11	19232504	0.015	11

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.40: Local Search Algorithms Experimental Results on Randomly Generated Homogeneous Negatively Correlated Problems (B)

		A+B+C			αA + C			βB + C			Best		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Best Obj. Value	Method	L.N.
HN1	10	672360	0.016	57	673672	0	1	673672	0.015	57	293616	10RI	83
HN2	15	1412704	0.015	1	1408716	0	1	1416160	0.016	39	385752	10RI	72
HN3	20	2384438	0.016	30	2375592	0.015	31	2396332	0.016	30	479161	10RI	56
HN4	25	3650920	0	25	3648576	0.015	25	3649344	0	25	749849	10RI	49
HN5	30	5224600	0.015	21	5223708	0.016	21	5228496	0.015	21	708564	10RI	44
HN6	35	7427520	0.016	17	7403376	0.016	18	7485678	0.015	17	837161	10RI	43
HN7	40	9813755	0.016	15	9778239	0.015	15	9806850	0.032	15	1034215	10RI	38
HN8	45	12529740	0.016	14	12540808	0.015	14	12605562	0.016	13	1312691	10RI	31
HN9	50	15929204	0.031	12	15874969	0.015	12	15903348	0.032	12	1580979	10RI	30
HN10	55	19324614	0.031	11	19180744	0.016	11	19232504	0.015	11	1526631	10RI	30

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.41: Local Search Algorithms Experimental Results on Randomly Generated Homogeneous Positively Correlated Problems (A)

		One Fixed Initial			Multi-starts(10RI)			A+C			B+C		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.
HP1	10	312020	0	83	293616	—	83	641516	0	58	680930	0.016	57
HP2	15	407354	0	72	385752	—	72	1411893	0.016	39	1437480	0	39
HP3	20	592814	0	60	479161	—	56	2370240	0.016	31	2427126	0	30
HP4	25	766827	0	52	749849	—	49	3633696	0.015	25	3673722	0.016	25
HP5	30	916887	0	46	708564	—	44	5269398	0.016	21	5325818	0.016	20
HP6	35	930878	0	45	837161	—	43	7427560	0.016	18	7438442	0.015	17
HP7	40	1205927	0	39	1034215	—	38	9820802	0.031	15	9850303	0.016	15
HP8	45	1314719	0	36	1312691	—	31	12558186	0.016	14	12639489	0.015	13
HP9	50	1838405	0	31	1580979	—	30	15827110	0.015	12	15827110	0.031	12
HP10	55	1674918	0	31	1526631	—	30	19136874	0.031	11	19271558	0.031	11

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.42: Local Search Algorithms Experimental Results on Randomly Generated Homogeneous Positively Correlated Problems (B)

		A+B+C			αA + C			βB + C			Best		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Best Obj. Value	Method	L.N.
HP1	10	641516	0	1	641516	0.015	58	680930	0	57	293616	10RI	83
HP2	15	1431234	0.015	39	1411893	0	39	1437480	0.016	39	385752	10RI	72
HP3	20	2379492	0.016	31	2370240	0.015	31	2427126	0	30	479161	10RI	56
HP4	25	3701308	0.016	25	3633696	0.015	25	3673722	0	25	749849	10RI	49
HP5	30	5302351	0	21	5269398	0.015	21	5325818	0.016	20	708564	10RI	44
HP6	35	7438442	0.015	17	7427560	0.016	18	7438442	0.016	17	837161	10RI	43
HP7	40	9774512	0.015	15	9820802	0.016	15	9850303	0.015	15	1034215	10RI	38
HP8	45	12633621	0.016	14	12558186	0.031	14	12639489	0.016	13	1312691	10RI	31
HP9	50	15921290	0.016	12	15827110	0.031	12	15856536	0.016	12	1580979	10RI	30
HP10	55	19168416	0.016	11	19136874	0.031	11	19271558	0.031	11	1526631	10RI	30

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.43: Local Search Algorithms Experimental Results on Randomly Generated Small C Problems (A)

		One Fixed Initial			Multi-starts(10RI)			A+C			B+C		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.
SR1	10	312020	0	83	293616	—	83	288334	0.015	87	308055	0	83
SR2	15	407354	0	72	385752	—	72	520729	0	64	397304	0.016	73
SR3	20	592814	0	60	479161	—	56	466161	0.015	68	545655	0.016	61
SR4	25	766827	0	52	749849	—	49	891022	0.015	49	821300	0	50
SR5	30	916887	0	46	708564	—	44	690832	0.015	55	928254	0.016	46
SR6	35	930878	0	45	837161	—	43	1179982	0.016	42	1045990	0.016	43
SR7	40	1205927	0	39	1034215	—	38	1296990	0.016	40	1009623	0.015	41
SR8	45	1314719	0	36	1312691	—	31	1596079	0.031	36	1515957	0.016	35
SR9	50	1838405	0	31	1580979	—	30	1495245	0.016	37	2053880	0.031	30
SR10	55	1674918	0	31	1526631	—	30	1681672	0.016	35	1848754	0.031	30

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.44: Local Search Algorithms Experimental Results on Randomly Generated Small C Problems (B)

		A+B+C			αA + C			βB + C			Best		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Best Obj. Value	Method	L.N.
SR1	10	288334	0.016	87	303304	0	1	303304	0	84	288334	A+C	87
SR2	15	451263	0	69	327067	0.015	1	398700	0	72	327067	$\alpha A + C$	1
SR3	20	550168	0	62	394262	0.016	1	467940	0.015	67	394262	$\alpha A + C$	1
SR4	25	748406	0.016	54	550699	0.015	1	766463	0.015	52	550699	$\alpha A + C$	1
SR5	30	1120513	0.016	44	544696	0	63	954778	0.015	66	544696	$\alpha A + C$	63
SR6	35	1064691	0.015	44	533278	0.016	1	827797	0.015	47	533278	$\alpha A + C$	1
SR7	40	1233621	0.016	41	620416	0.016	59	1013851	0.015	42	620416	$\alpha A + C$	59
SR8	45	1377850	0.015	38	725759	0.016	1	1624126	0.016	33	725759	$\alpha A + C$	1
SR9	50	1870570	0.016	33	824355	0.015	1	1726656	0.032	32	824355	$\alpha A + C$	1
SR10	55	2137248	0.031	31	780204	0.016	1	2137318	0.031	28	780204	$\alpha A + C$	1

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.45: Local Search Algorithms Experimental Results on Randomly Generated Small C Negatively Correlated Problems (A)

		One Fixed Initial			Multi-starts(10RI)			A+C			B+C		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.
SN1	10	312020	0	83	293616	—	83	669772	0	57	687630	0.016	56
SN2	15	407354	0	72	385752	—	72	1409858	0	39	1402561	0	39
SN3	20	592814	0	60	479161	—	56	2396949	0.016	30	2408756	0	30
SN4	25	766827	0	52	749849	—	49	3647838	0.015	25	3649362	0.016	25
SN5	30	916887	0	46	708564	—	44	5249526	0.016	21	5256054	0.016	21
SN6	35	930878	0	45	837161	—	43	7492271	0.016	17	7442987	0.015	17
SN7	40	1205927	0	39	1034215	—	38	9872347	0.015	15	9836637	0.015	15
SN8	45	1314719	0	36	1312691	—	31	12537889	0.015	14	12566192	0.031	13
SN9	50	1838405	0	31	1580979	—	30	15949330	0.031	12	15928395	0.016	12
SN10	55	1674918	0	31	1526631	—	30	19316397	0.031	11	19247051	0.031	11

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.46: Local Search Algorithms Experimental Results on Randomly Generated Small C Negatively Correlated Problems (B)

		A+B+C			αA + C			βB + C			Best		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Best Obj. Value	Method	L.N.
SN1	10	669772	0	57	674764	0	1	687630	0.015	56	293616	10RI	83
SN2	15	1409858	0.015	39	1410057	0	1	1417707	0.016	39	385752	10RI	72
SN3	20	2416525	0.016	30	2377936	0.015	31	2398625	0	30	479161	10RI	56
SN4	25	3680327	0.016	25	3651238	0.015	25	3675355	0	25	749849	10RI	49
SN5	30	5218029	0.015	21	5226903	0.016	21	5252798	0	21	708564	10RI	44
SN6	35	7429162	0.015	17	7443539	0.016	17	7439767	0.015	17	837161	10RI	43
SN7	40	9770233	0.016	15	9769889	0.015	15	9803866	0.016	15	1034215	10RI	38
SN8	45	12616255	0.015	13	12546874	0.016	14	12592386	0.015	13	1312691	10RI	31
SN9	50	15988829	0.015	12	15898695	0.032	12	15890348	0.015	12	1580979	10RI	30
SN10	55	19377019	0.016	11	19219710	0.031	11	19286848	0.016	11	1526631	10RI	30

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.47: Local Search Algorithms Experimental Results on Randomly Generated Small C Positively Correlated Problems (A)

		One Fixed Initial			Multi-starts(10RI)			A+C			B+C		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.
SP1	10	312020	0	83	293616	—	83	642524	0	58	661327	0.016	57
SP2	15	407354	0	72	385752	—	72	1413358	0	39	1432886	0	39
SP3	20	592814	0	60	479161	—	56	2382917	0.016	30	2417136	0.016	30
SP4	25	766827	0	52	749849	—	49	3677444	0.016	25	3694913	0	25
SP5	30	916887	0	46	708564	—	44	5298076	0.016	21	5315297	0.016	20
SP6	35	930878	0	45	837161	—	43	7453081	0.015	17	7456969	0.016	17
SP7	40	1205927	0	39	1034215	—	38	9795186	0.016	15	9826469	0.016	15
SP8	45	1314719	0	36	1312691	—	31	12629628	0.031	13	12640198	0.016	13
SP9	50	1838405	0	31	1580979	—	30	15888357	0.016	12	15978450	0.031	12
SP10	55	1674918	0	31	1526631	—	30	1924563	0.031	11	19137238	0.016	11

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.48: Local Search Algorithms Experimental Results on Randomly Generated Small C Positively Correlated Problems (B)

		A+B+C			αA + C			βB + C			Best		
Test	n	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Obj. Value	Time(s)	L.N.	Best Obj. Value	Method	L.N.
SP1	10	642524	0	58	642524	0.015	58	681954	0	57	293616	10RI	83
SP2	15	1413358	0.015	39	1432886	0	39	1438898	0	39	385752	10RI	72
SP3	20	2438612	0	30	2372219	0.015	31	2429040	0.016	30	479161	10RI	56
SP4	25	3677240	0.016	25	3636432	0.015	25	3641196	0.016	25	749849	10RI	49
SP5	30	5344824	0.015	20	5272476	0.016	21	5375662	0.015	20	708564	10RI	44
SP6	35	7455064	0.015	17	7427213	0.016	18	7447051	0.016	17	837161	10RI	43
SP7	40	9848494	0.015	15	9813127	0.016	15	9794466	0.015	15	1034215	10RI	38
SP8	45	12579578	0.016	14	12554867	0.015	14	12614563	0.031	13	1312691	10RI	31
SP9	50	15888617	0.016	12	15817093	0.015	12	15906478	0.031	12	1580979	10RI	30
SP10	55	19226154	0.031	11	19126113	0.031	11	19234084	0.032	11	1526631	10RI	30

L.N. represents the number of the loops (moves) of the local search component of the algorithm employs.

Table A.49: Tabu Search Algorithms Experimental Results on Pseudo Random Problems

		Max Iter = 1000		Max Iter = 100000		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
R1	10	293616	0.11	293616	0.319	293616	0
R2	15	335032	0.212	335032	17.939	335032	0.016
R3	20	405621	0.348	405621	32.056	405621	0.015
R4	25	548467	0.524	548467	48.547	548467	0.031
R5	30	594166	0.746	559975	69.294	559585	0.266
R6	35	573702	1.003	549248	94.549	549248	0.031
R7	40	716125	1.313	653192	123.022	634078	0.031
R8	45	881552	1.661	785003	156.194	742206	0.031
R9	50	959428	2.062	884524	192.999	847876	0.234
R10	55	963328	2.505	830221	233.017	803139	0.218

For the two Tabu methods, the parameters tabu tenure is 10 and max restart number is 20.

Table A.50: Tabu Search Algorithms Experimental Results on Pseudo Random Problems of Large Sizes

		Max Iter = 1000		Max Iter = 100000		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
R11	100	2425164	7.942	2397652	79.609	1596151	3.0
R12	150	4596949	17.578	5070999	175.548	1883356	13.657
R13	200	8373862	31.918	8749658	316.441	1809620	37.177
R14	250	10474657	50.789	12833586	504.091	1712084	92.059
R15	300	16881247	72.855	17031035	725.958	1351850	188.328
R16	350	19145096	100.111	21581021	995.443	1380260	338
R17	400	25211743	131.86	26090778	1306.37	833746	581.14
R18	450	30882749	169.976	29783305	1668.66	982355	1014.727
R19	500	34222850	215.815	35182078	2102.1	735075	1466.883

For the two Tabu methods, the parameters tabu tenure is 10 and max restart number is 20.

Table A.51: Tabu Search Algorithms Experimental Results on Negatively Correlated Pseudo Random Problems

		Max Iter = 1000		Max Iter = 100000		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
N1	10	675707	0.109	675707	8.483	675707	0.031
N2	15	1408184	0.212	1408184	17.964	1408180	0.156
N3	20	2379444	0.348	2379444	31.371	2379440	0.031
N4	25	3656936	0.538	3656936	48.849	3656940	0.016
N5	30	5224568	0.759	5224568	70.053	5224570	0.031
N6	35	7411971	1.028	7411971	95.182	7411970	0.047
N7	40	9756934	1.361	9755479	125.724	9755480	0.032
N8	45	12511115	1.698	12509195	158.498	12509200	0.266
N9	50	15855336	2.087	15853703	195.665	15853700	0.437
N10	55	19172100	2.535	19172100	238.977	19172100	0.094

For the two Tabu methods, the parameters tabu tenure is 10 and max restart number is 20.

Table A.52: Tabu Search Algorithms Experimental Results on Negatively Correlated Pseudo Random Problems of Large Sizes

		Max Iter = 1000		Max Iter = 100000		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
N11	100	66604106	7.849	66587755	787.758	66576269	2.655
N12	150	152792829	17.637	152713346	1738.95	152550032	12.223
N13	200	274259561	31.955	274168354	3146.99	273832805	36.447
N14	250	432400040	50.961	—	—	431521336	86.947
N15	300	625862315	73.184	—	—	624477627	185.207
N16	350	856579992	100.897	—	—	854802643	348.002
N17	400	1121922081	132.88	—	—	1119664093	596.956
N18	450	1423272632	170.991	—	—	1420305572	935.773
N19	500	1764085456	219.103	—	—	1759958286	1459.871

For the two Tabu methods, the parameters tabu tenure is 10 and max restart number is 20.

Table A.53: Tabu Search Algorithms Experimental Results on Positively Correlated Pseudo Random Problems

		Max Iter = 1000		Max Iter = 100000		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
P1	10	675707	0.109	675707	8.58	647710	0.015
P2	15	1408184	0.212	1408184	18.058	1419460	0.15
P3	20	2379444	0.351	2379444	31.403	2378980	0.031
P4	25	3656936	0,531	3656936	49.598	3644840	0.015
P5	30	5224568	0.76	5224568	70.166	5278980	0.031
P6	35	7411971	1.024	7411971	95.178	7414230	0.046
P7	40	9756934	1.361	9755479	125.467	9758080	0.172
P8	45	12511115	1.699	12509195	157.994	12543000	0.109
P9	50	15855336	2.09	15853703	196.101	15778100	0.047
P10	55	19172100	2.502	19172100	239.891	19075300	0.312

For the two Tabu methods, the parameters tabu tenure is 10 and max restart number is 20.

Table A.54: Tabu Search Algorithms Experimental Results on Positively Correlated Pseudo Random Problems of Large Sizes

		Max Iter = 1000		Max Iter = 100000		MILP7	
Test	n	Obj. Value	Time(s)	Obj. Value	Time(s)	Obj. Value	Time(s)
P11	100	66675005	7.848	66587755	799.423	66606608	3.136
P12	150	152782541	17.442	152713346	1769.1	152580659	11.578
P13	200	274729435	31.603	274168354	3185.17	274309328	34.896
P14	250	432224187	50.647	—	—	431279238	86.834
P15	300	625308476	72.546	—	—	624119360	189.002
P16	350	856814745	99.605	—	—	854956724	348.559
P17	400	1121504322	130.969	—	—	1118996972	579.252
P18	450	1424067697	168.088	—	—	1420818322	974.577
P19	500	1763921389	218.186	—	—	1759916528	1455.806

For the two Tabu methods, the parameters tabu tenure is 10 and max restart number is 20.