# ALGORITHMS AND THEORETICAL TOPICS ON SELECTED COMBINATORIAL OPTIMIZATION PROBLEMS

by

Arman Kaveh
BS, Simon Fraser University, 2006

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the Department
of
Mathematics


SIMON FRASER UNIVERSITY
Fall 2010

# APPROVAL

**Name:** Arman Kaveh

**Degree:** Master of Science

**Title of Thesis:** Algorithms and Theoretical Topics on Selected Combinatorial Optimization Problems

**Examining Committee:** Dr. Zhaosong Lu
Chair

Dr. Abraham Punnen, Senior Supervisor

Dr. Tamon Stephen, Supervisor

Dr. Snezana Mitrovic-Minic, External Examiner

**Date Approved:** December 3, 2010

# Declaration of
# Partial Copyright Licence

# Abstract

We study the *Quadratic Assignment Problem* (QAP), *Three Dimensional Assignment Problem* (3AP) and *Quadratic Three Dimensional Assignment Problem* (Q3AP), which combines aspects of both QAP and 3AP. The three problems are known to be NP-hard. We propose new algorithms for obtaining near optimal solutions of QAP and 3AP and present computational results. Our algorithms obtain improved solutions in some benchmark instances of QAP and 3AP. We also discuss theoretical results on 3AP and Q3AP such as polynomially solvable special cases and approximation algorithms. A special case of 3AP is the constant 3AP where every feasible solution has the same cost. Necessary and sufficient conditions are presented for an instance of 3AP to have a constant solution and the result is extended to *Multidimensional Assignment Problems* (MAP).

# Acknowledgments

I would like to thank my senior supervisor, Dr. Abraham Punnen, for his guidance, support and patience throughout my graduate studies. He took me under his wings as an undergraduate student and introduced me to the field of Operations Research and Optimization. I have learned so much from him over the years in academics and life in general that will stay with me forever. I would like to thank Dr. Tamon Stephen for the invaluable lessons I have learned from him in optimization and the inspirations for many ideas used in this thesis. A sincere thank you to Dr. Zhaosong Lu for teaching me everything I know in continuous optimization. I am also indebted to Dr. Natalia Kouzniak and Dr. Randall Pyke for life-changing advice and never-ending support throughout the years. I would like to thank Dr. Snezana Mitrovic-Minic for the opportunity to work with her on different hands-on projects and providing me with tools of the trade for computational studies.

A great thank you is in order to my parents, brothers and sister for supporting me and giving me unconditional love. A heartfelt thank you to my fiancé, Neda, for always being there for me during my student years and giving me strength. Last but not least, I would like to thank my colleagues for helping me throughout my thesis while having their own work to attend to. So thank you Annie Zhang, Daniel Benvenuti, Bradley Woods, John LaRusic, Hua Zheng and Yong Zhang.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In this thesis we consider three combinatorial optimization problems: *Quadratic Assignment Problem* (QAP), *Three Dimensional Assignment Problem* (3AP) and *Quadratic Three Dimensional Assignment Problem* (Q3AP). Each problem is rich in theory and applications. We introduce the problems in this chapter and discuss the existing literature. Subsequent chapters include our contribution to these problems.

## 1.1 Introduction to Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) was first proposed by Koopmans and Beckmann in 1957 [81]. It can be formulated as follows. Suppose we are given two $n \times n$ matrices $A$ and $B$ where $A = [a_{ij}]$ is the distance between locations and $B = [b_{ij}]$ is the flow between facilities. The objective is to

$$\min_{\pi \in \pi_n} \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} a_{\pi(i)\pi(j)} \tag{1.1}$$

where $\pi_n$ denotes permutations on $n$. Research on QAP has led to interesting theory, applications and solution techniques. We will briefly describe some applications and solution techniques in this chapter. The reader is referred to some books and surveys in the literature as listed in [38, 99, 100].

### 1.1.1 Alternative Formulations

An Integer Programming formulation is stated as follows

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} b_{ij}a_{kl}x_{ik}x_{jl} \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i = 1,\ldots,n \\ & \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall\, j = 1,\ldots,n \\ & x_{ij} \in \{0,1\} \end{aligned} \tag{1.2}$$

The above formulation is due to Koopmans and Beckmann [81]. The variables can be interpreted as

$$x_{ij} = \begin{cases} 1 & \text{if facility } i \text{ is assigned to location } j \\ 0 & \text{otherwise} \end{cases}$$

The formulation proposed by Lawler [84] is more generic and is given by

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ijkl}x_{ij}x_{kl} \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i = 1,\ldots,n \\ & \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall\, j = 1,\ldots,n \\ & x_{ij} \in \{0,1\} \end{aligned} \tag{1.3}$$

This formulation can be expressed in terms of permutations

$$\min_{\pi\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i\pi(i)j\pi(j)} \tag{1.4}$$

Consider the following formulation of Assignment Problem (AP)

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i = 1,\ldots,n \\ & \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall\, j = 1,\ldots,n \\ & x_{ij} \geq 0 \qquad \forall\, i,j = 1,\ldots,n \end{aligned} \tag{1.5}$$

and the corresponding permutation formulation

$$\min_{\pi \in \pi_n} \sum_{i=1}^{n} c_{i\pi(i)} \tag{1.6}$$

The Lawler formulation (1.3) extends AP to QAP by making the objective function quadratic. The Koopmans and Beckmann formulation (1.2) assumes that the four-dimensional cost matrix $C$ in (1.3) is generated by a pair of $n \times n$ matrices $A$ and $B$ where $c_{ijkl} = b_{ik}a_{jl}$.

There is another slightly different formulation which introduces a linear term in the objective function

$$\min_{\pi \in \pi_n} \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} a_{\pi(i)\pi(j)} + \sum_{i=1}^{n} c_{i\pi(i)} \tag{1.7}$$

In the case that $c_{ij} = 0$ for all $1 \leq i, j \leq n$, we get the formulation (1.1). Our focus in this work is on formulations (1.1) and (1.2). Formulation (1.2) has a quadratic term in the objective function. It is possible to represent the problem in a linear form [54, 79].

A trace formulation was introduced by Edwards in [46, 47]. Given the matrices $A$ and $B$, a function $f_{A,B}$ can be defined on the set $\Pi_n$ of permutation matrices such that

$$f_{A,B}(X) = tr(AXBX^t)$$

where the superscript $^t$ denotes the transpose of a matrix. Then the trace formulation is

$$\min_{X \in \Pi_n} f_{A,B}(X) = tr(AXBX^t) \tag{1.8}$$

Trace formulation is used for deriving eigenvalue lower bounds as referenced in Section (1.4.3).

## 1.2 Applications of QAP

The QAP originated in 1957 from assigning a set of economic activities to a set of locations [81]. Therefore its first application was related to facility location problems which remains as one of its major applications to this date. Another dominant application of QAP is in wiring problems.

### 1.2.1 Facility Location Problem

Facility location or facility layout is the problem of assigning $n$ facilities to $n$ locations. The $n \times n$ matrices $A$ and $B$ represent the distance between locations and the flow between

facilities respectively. A complete assignment can be represented by a permutation $\pi \in \pi_n$ as follows. Consider assigning facility $i$ to the location $\pi(i)$ and facility $j$ to the location $\pi(j)$. Since each unit of flow between facilities $i$ and $j$ has to travel the distance between locations $\pi(i)$ and $\pi(j)$, the associated cost is $b_{ij}a_{\pi(i)\pi(j)}$. The total cost for such permutation $\pi$ is

$$\sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}a_{\pi(i)\pi(j)}$$

We are interested in finding the permutation which leads to the minimum cost, thus formulation (1.1). A comprehensive discussion of facility layout problem can be found in [51].

A related problem is the hospital layout problem [48]. It involves the location of various departments within a hospital so as to minimize the distance traveled by patients (per year) while moving from one department to another. The annual flow between each pair of departments is known and so is the distance between each pair of locations. The $n \times n$ matrices $A$ and $B$ represent the distances and flows respectively. The problem formulation is the same as (1.1) above.

### 1.2.2 Steinberg Wiring Problem

In this problem $n$ components have to be placed on a board with $n$ available slots where pairs of components are connected by wires. Let the $n \times n$ matrix $A$ represent the distance between pairs of slots on the board and the $n \times n$ matrix $B$ represent the number of wires between pairs of components. The objective is to place the components on the board in such a way to minimize the total length of wires used [112]. The problem formulation is that of QAP in (1.1).

Other applications of QAP include: typewriter keyboards [103], parallel and distributed computing [25], economic problems [74] and scheduling parallel production lines [59]. Some recent applications in bio-informatics can be found in [111, 120, 122, 123].

## 1.3 Computational Complexity of QAP

QAP is among the hardest combinatorial optimization problems. In 1976 Sahni and Gonzales showed that QAP is strongly NP-hard [109] which indicates that finding an efficient algorithm to solve the problem in polynomial time is very unlikely. They also showed that unless $P = NP$, it is not possible to find an $\epsilon$-approximation algorithm for any constant $\epsilon$ in

polynomial time. An $\epsilon$-approximation algorithm for a minimization problem finds a solution with cost at most $(1 + \epsilon)$ of the optimal cost. For more on the theory of NP-completeness please refer to [56].

Many NP-complete problems such as Traveling Salesman Problem (TSP), Graph Partitioning Problem and Maximum Clique Problem are special cases of QAP. In the case of TSP, the distance matrix of QAP corresponds to the distance matrix of TSP and the flow matrix corresponds to the adjacency matrix of a cycle of length $n$ in TSP.

Some special cases of QAP are solvable in polynomial time. The distance and flow matrices of such instances have a special structure which leads to polynomial time algorithms. For instance, if both matrices are weighted adjacency matrices of a tree, or one of the matrices is a weighted adjacency matrix of a double star then the problem can be solved in polynomial time [40]. Adolphson and Hu showed that if one matrix represents the weighted adjacency matrix of a tree while the other represents the distance matrix of a grid graph the problem is solvable in $O(n \log n)$ [40]. The reader is referred to [38] for more polynomially solvable cases of QAP.

A set of benchmark instances of QAP can be found in a unified collection called QAPLIB [33] accessible to the scientific community. It contains instances which have been solved to optimality and instances for which researchers are currently working on to obtain improved solutions. As a measure of how difficult QAP is consider the following. The time taken to find the optimal solution to the problem Kra30b with $n = 30$ available on QAPLIB would amount to the equivalent of 2.7 years on a single cpu HP-3000 workstation [11]. This shows that even a QAP problem of size $n = 30$ can prove to be computationally challenging. A study of the distribution of objective values of QAP can be found in [19].

## 1.4 Lower Bounds for QAP

A lower bound for a QAP instance is a value smaller than or equal to the optimal solution value of QAP. There is extensive literature on finding lower bounds for QAP from theoretical and computational perspectives. Despite the efforts, finding tight and computationally efficient lower bounds remains a challenge. The quality of lower bounds is crucial for many exact algorithms such as branch and bound. Exact methods perform implicit enumeration of search space by using lower bounds to avoid an exhaustive search. Also, lower bounds can be used to measure the quality of solutions obtained from heuristic algorithms. For the

QAP there are three main categories of lower bounds: combinatorial bounds, reformulation bounds and algebraic bounds.

### 1.4.1 Combinatorial Bounds

Combinatorial based bounds include Gilmore-Lawler bound (GL) [62, 84] and Christofides-Gerrard bound (CG) [41]. GL bound is one of the original bounds proposed for QAP. Given the matrices $A$ and $B$ consider the new $n \times n$ matrix $C$ defined as follows

$$c_{ij} = \min_{\pi \in \pi_n} \sum_{k=1}^{n} a_{i\pi(k)} b_{jk} \quad 1 \leq i, j \leq n \tag{1.9}$$

It takes $O(n^3)$ to calculate the matrix $C$. Given a permutation $\pi$ the following inequality holds

$$\sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} a_{\pi(i)\pi(j)} \geq \sum_{i=1}^{n} c_{\pi(i)i} \tag{1.10}$$

which implies the inequality

$$\min_{\pi \in \pi_n} \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} a_{\pi(i)\pi(j)} \geq \min_{\pi \in \pi_n} \sum_{i=1}^{n} c_{\pi(i)i} = GL(A, B) \tag{1.11}$$

After calculating matrix $C$, it takes $O(n^3)$ to calculate $GL(A, B)$ by solving a linear assignment problem. GL bounds are one of the simplest and cheapest bounds to compute but are not strong.

### 1.4.2 Reformulation Bounds

A reformulation of a QAP instance is another QAP instance with different matrices $A$ and $B$ such that any permutation generates the same objective value in both instances. The underlying idea behind reformulation bounds is to generate a sequence of reformulations $P_1, P_2, \ldots, P_k$ for a given problem $P_0$ and obtain bounds $B_1 \leq B_2 \leq \cdots \leq B_k$. One would aim to use the previous reformulation and bound to make the current calculation easier. Bounding techniques based on reformulation are addressed by Carraresi and Malucelli [37] and Assad and Xu [12]. Both authors use GL bounds in each iteration.

### 1.4.3 Algebraic Bounds

Algebraic bounds include eigenvalue related bounds [49, 69, 70, 107], variance reduction bounds [87] and semidefinite relaxation bounds [77]. Eigenvalue related bounds are the major type in algebraic bounds.

An introduction is in order for next section. Algorithms for solving NP-hard combinatorial optimization problems can be classified into three main categories: exact algorithms, heuristic algorithms and approximation algorithms. Exact algorithms strive to obtain the optimal solution and result in extensive computational time. As the problem size grows, the time required to obtain the optimal solutions grows exponentially. This is where heuristic algorithms defend their merit. Heuristic algorithms are able to find a near optimal solution in a much more reasonable time, especially for large instances. One drawback of heuristic algorithms is that they provide no guarantee of the solution quality. On the other hand, approximation algorithms provide a guarantee of solution quality, which makes them appealing from a theoretical perspective. However, approximation algorithms do not generally perform as well as heuristics in practice, which is why heuristic methods remain widespread in the industry and academia. In the case of the general QAP, as discussed in section 1.3, approximation algorithms cannot exist unless P=NP. Therefore any attempts to discover approximation algorithms have to be restricted to special cases of QAP.

## 1.5 Exact Algorithms for QAP

Methods for finding the exact optimal solution for QAP fall into three main categories: branch-and-bound, cutting plane and dynamic programming.

### 1.5.1 Branch and Bound

Many authors have proposed sequential and parallel branch-and-bound methods for QAP. These methods are among the most successful ones for QAP. They start with an empty initial solution and iteratively extend it to a full solution. The general idea is to avoid exhausting the entire search space by excluding certain assignments. Once we choose to branch on the possible values of a variable, we can use different methods to find bounds on these branches in order to determine which branches to avoid. Branch and bound methods can be characterized in three main categories with respect to branching rule: single assignment

methods [62, 84, 31, 47, 97], pair assignment methods [58, 83, 94] and relative positioning methods [91].

### 1.5.2 Cutting Plane

Cutting plane methods for QAP are described in [22, 21]. These methods involve solving the linear programming relaxation of the underlying integer program. In case the optimal solution to this linear program is an integer then it is the optimal solution to the integer program. Otherwise, the algorithm adds a constraint to the linear program which cuts away this non-integer solution but leaves all integer solutions in the feasible region. The integer linear programming formulation for these methods can be extremely large even for moderate size problems. As a result, computational experience with these methods is not satisfactory.

### 1.5.3 Dynamic Programming

Christofides and Benavent [39] used a dynamic programming approach to solve a special case of QAP. Dynamic programming solves a problem of size $n$ by starting from subproblems of size $1, 2, \ldots, n-1$. After solving subproblems of size $k$ it upgrades the solutions to size $k+1$. Problems may arise in dynamic programming if the solution to a subproblem or the upgrade procedure cannot be performed in polynomial time.

## 1.6 Heuristic Algorithms for QAP

Many researchers have studied heuristic methods for solving QAP instances as the problem remains NP-complete and even finding an $\epsilon$-approximate solution for any constant $\epsilon$ is NP-complete. Exact algorithms can only solve small instances with $n \leq 20$ whereas heuristic methods can obtain near optimal solutions for large problems (e.g. $n \geq 40$) in reasonable time. There are six main types of heuristics in the literature: construction methods, limited enumeration methods, improvement methods, metaheuristics, simulated annealing and genetic algorithms.

### 1.6.1 Construction Methods

Construction methods start with an empty partial permutation $p$ and expand $p$ into a full permutation with every step of the algorithm. Such methods are considered to be relatively

simple approaches with a poor solution quality. Gilmore introduced construction methods in 1962 [62]. A revised construction method was proposed by Burkard [28].

### 1.6.2 Limited Enumeration Methods

These methods are strongly related to exact methods such as branch-and-bound and cutting planes. The idea behind these algorithms is that a good suboptimal solution may be produced early in an enumerative search while finding an optimal solution takes much longer. Also an optimal solution may be found earlier in the search whereas the rest of the time is spent on proving the optimality of this solution. There are many ways to limit enumeration of the search space. One approach is to impose a time limit. Enumeration stops when the algorithm reaches a time limit or no improvement has been made in a predetermined time interval. These prespecified parameters can be problem specific. A second option is to decrease the requirement for optimality. For example, if no improvement has been made after a certain prespecified time interval, then the upper bound is decreased by a certain percentage resulting in deeper cuts in the enumeration tree. Although the optimal solution may be cut off, it differs from the obtained solution by the above percentage.

### 1.6.3 Improvement Methods

Most heuristics on QAP can be classified as improvement methods. One major subcategory is *Local Search*. Local search algorithms start with a feasible solution and iteratively try to find a better solution in the neighborhood of the current one. This process is repeated until no further improvement can be made. Local search methods rely heavily on the definition of neighborhood. A local optimum with respect to one neighborhood may not be optimum with respect to another neighborhood. The most common neighborhoods used for QAP are *pair exchanges* and *cyclic triple exchanges*. A combination of neighborhoods may be used in a local search algorithm. One can run local search several times with different initial solutions in order to get better solutions. Construction methods can be used to obtain the initial solution. However, Bruijs [27] proved that generally there is no strong argument for good quality initial solutions. Some local search based algorithms are discussed in [91, 93].

### 1.6.4 Metaheuristics

An example of metaheuristics is tabu search introduced by Glover [63, 64]. Tabu search tries to escape local optima in combinatorial search. It consists of neighborhood structure, a move, a tabu list and an aspiration criterion. A move in QAP is usually a pair-exchange that generates a new solution from the existing one. A tabu list is a list of forbidden moves. An aspiration criterion allows a tabu move to be executed if it is perceived to lead to a very good solution. A description of tabu search for QAP can be found in [20, 110, 114].

### 1.6.5 Simulated Annealing

Simulated annealing approaches try to overcome local optimality in hard combinatorial optimization problems by drawing analogies from statistical mechanics. The analogy led to a method called *simulated annealing* developed by Kirkpatrick, Gelatt and Vecchi [80]. They showed how the Metropolis algorithm [90] for simulating the behavior of a physical particle system can be used to apply techniques from statistical mechanics to optimization. They applied those ideas on Traveling Salesman Problem. Burkard and Rendl [35] applied simulated annealing to QAP and showed that simulated annealing is a general approach for combinatorial optimization problems which possess a neighborhood structure. Wilhelm and Ward further investigated simulated annealing for QAP [121].

### 1.6.6 Genetic Algorithms

Genetic algorithms are based on the ideas from natural selection. They maintain a population of solutions and apply evolutionary mechanisms to obtain a new population of solutions with better fitness values on average. The main point is to adapt these simple evolutionary mechanisms to combinatorial optimization problems. Genetic algorithms were first developed by Holland [75] and did not receive much enthusiasm until the advent of parallel computers.

A genetic algorithm starts with a set of initial feasible solutions called *initial population*. The algorithms selects a number of pairs of *parents* from the current population and produces a new feasible solution from each pair by *crossover rules*. A good description of genetic algorithms can be found in [44, 67]. Tate and Smith [116] developed a standard genetic algorithm for QAP and revealed some drawbacks of this approach despite the promising numerical results. Fleurent and Ferland [50] proposed hybrid approaches obtained by

combining genetic algorithms with other algorithms such as local search and tabu search. They improved the best known solution to some large scale test problems in QAPLIB [33]. Ahuja, Orlin and Tiwari [9] developed a *greedy genetic algorithm* and also obtained very good results on large scale problems from QAPLIB.

## 1.7 Introduction to 3-Dimensional Assignment Problem

The 3-Dimensional Assignment Problem was first introduced by Pierskalla in 1967 [101] as an extension of the Assignment Problem (AP) described in (1.5). A classical application of AP is the assignment of $n$ people to $n$ jobs, where $c_{ij}$ denotes the cost of assigning person $i$ to job $j$. The objective is to assign exactly one person to each job such that the total cost of assignments is minimized.

The 3-Dimensional Assignment Problem is an extension of AP and has two variations; *axial* (3AP) and *planar* (3PAP). The focus of this thesis is on the axial version 3AP.

### 1.7.1 Axial 3-Dimensional Assignment Problem

The 3AP is formulated as the following

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} & \\
\text{s.t.} \quad & \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 & \forall\, i = 1,\ldots,n \\
& \sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 & \forall\, j = 1,\ldots,n \\
& \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 & \forall\, k = 1,\ldots,n \\
& x_{ijk} \in \{0,1\} & \forall\, i,j,k = 1,\ldots,n
\end{aligned} \tag{1.12}$$

A real value $c_{ijk}$ is assigned to each triple $(i,j,k)$. In general, $c_{ijk}$ is arbitrary and cannot be represented by the sum of two edge costs (i.e. $c_{ijk} \neq c'_{ij} + c''_{jk}$). Otherwise, as discussed later in this chapter, the problem becomes fairly straightforward.

Alternatively, 3AP can be considered as an optimization problem on a complete tripartite graph $K_{n,n,n} = (I \cup J \cup K, (I \times J) \cup (I \times K) \cup (J \times K))$, where $I, J, K$ are disjoint sets of size $n$. The cost of choosing a triangle $(i,j,k) \in I \times J \times K$ is $c_{ijk}$. The objective is to find

a subset $A \subseteq I \times J \times K$ of $n$ disjoint triangles, such that every element of $I \cup J \cup K$ occurs in exactly one triangle of $A$ and the total cost $c(A)$ is minimized, where

$$c(A) = \sum_{(i,j,k) \in A} c_{ijk} \tag{1.13}$$

An alternative formulation is

$$\min_{p,q \in \pi_n} \sum_{i=1}^{n} c_{ip(i)q(i)} \tag{1.14}$$

where $\pi_n$ denotes permutations on $n$. Therefore this problem has $(n!)^2$ feasible solutions.

The 3AP as formulated in (1.12) is a special case of the Set Partitioning Problem (SPP), which is formulated as

$$\min \{cx \mid Ax = e, x \in \{0,1\}^q\} \tag{1.15}$$

where $A$ is a matrix of zeros and ones, and $e$ is a vector of ones. The reader is referred to [13, 57] for more details on SPP.

### 1.7.2 Planar 3-Dimensional Assignment Problem

The Planar 3-Dimensional Assignment Problem (3PAP) can be formulated as

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} & \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_{ijk} = 1 & \forall\, j,k = 1,\ldots,n \\
& \sum_{j=1}^{n} x_{ijk} = 1 & \forall\, i,k = 1,\ldots,n \\
& \sum_{k=1}^{n} x_{ijk} = 1 & \forall\, i,j = 1,\ldots,n \\
& x_{ijk} \in \{0,1\} & \forall\, i,j,k = 1,\ldots,n
\end{aligned} \tag{1.16}$$

In contrast to finding $n$ triples in 3AP, the 3PAP finds $n^2$ triples such that each pair of elements from $(I \times J) \cup (I \times K) \cup (J \times K)$ occurs in exactly one triple and the total cost is minimized. Every "flat" in the 3-dimensional array $x_{ijk}$ must contain a 2-dimensional assignment. Thus the number of feasible solutions of 3PAP corresponds to Latin squares of order $n$ [85] and hence increases very fast. The number of different Latin squares of order $n$ equals $n!(n-1)!T(n)$, where $T(n)$ is the number of reduced Latin squares. $T(n)$ grows very large. For instance, $T(9)$ was discovered to be 377,597,570,964,258,816 [16] and $T(9)$ was calculated as 5,363,937,773,277,371,298,119,673,540,771,840 [89].

### 1.7.3 Multidimensional Assignment Problem

The 3-Dimensional Assignment Problem can be further extended to higher dimensions resulting in Multidimensional Assignment Problem (MAP). MAP was considered for the first time by Pierskalla [102]. The axial MAP of dimension $d$ is formulated as follows

$$\begin{aligned}
\min \quad & \sum_{i_1=1}^{n} \cdots \sum_{i_d=1}^{n} c_{i_1 \ldots i_d} x_{i_1 \ldots i_d} & \\
\text{s.t.} \quad & \sum_{i_2=1}^{n} \cdots \sum_{i_d=1}^{n} x_{i_1 \ldots i_d} = 1 & \forall\, i_1 = 1, \ldots, n \\
& \sum_{i_1=1}^{n} \cdots \sum_{i_{k-1}=1}^{n} \sum_{i_{k+1}=1}^{n} \cdots \sum_{i_d=1}^{n} x_{i_1 \ldots i_d} = 1 & \forall\, k = 2, \ldots, d-1, \\
& & \forall\, i_k = 1, \ldots, n \\
& \sum_{i_1=1}^{n} \cdots \sum_{i_{d-1}=1}^{n} x_{i_1 \ldots i_d} = 1 & \forall\, i_d = 1, \ldots, n \\
& x_{i_1 \ldots i_d} \in \{0, 1\} & \forall\, i_1, \ldots, i_d = 1, \ldots, n
\end{aligned} \tag{1.17}$$

An equivalent formulation is an extension of (1.14). Let $\phi_1, \phi_2, \ldots, \phi_{d-1}$ be permutations on $n$. Then the problem is

$$\min_{\phi_1, \phi_2, \ldots, \phi_{d-1}} \sum_{i=1}^{n} c_{i \phi_1(i) \phi_2(i) \ldots \phi_{d-1}(i)} \tag{1.18}$$

## 1.8 Complexity of 3AP

3AP belongs to the class of NP-hard problems [56] and therefore an efficient algorithm to solve 3AP in polynomial time seems highly unlikely. 3AP is related to the 3-Dimensional Matching Problem (3DM), which is one of the original problems to be proven NP-hard by Karp [78]. The 3DM states that given a set $U \subseteq T \times T \times T$ where $T$ is a finite set, is there a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of $W$ agree in any coordinates. The reduction from 3DM to 3AP is as follows. Given an arbitrary instance of 3DM, create an instance of 3AP with

$$c_{ijk} = \begin{cases} 1 & \text{if } (i, j, k) \in U \\ 0 & \text{otherwise} \end{cases}$$

Then a minimum cost solution to 3AP corresponds to a "yes" answer for 3DM if and only if its cost is zero.

In addition to 3AP, the planar version 3PAP is NP-hard as well [52]. On the other hand, the AP is polynomially solvable in $O(n^3)$ [82].

### 1.8.1 Polynomially Solvable Special Cases

Although the general 3AP is NP-hard, some special cases of 3AP can be solved in polynomial time. A trivial case is where the costs $c_{ijk}$ can be represented as two sums such that $c_{ijk} = c'_{ij} + c''_{jk}$. Then the problem can be viewed as two separate Assignment Problems on matrices $C'$ and $C''$ which can be solved in polynomial time.

Furthermore, if the cost matrix satisfies a *Monge property* as defined below, then the problem is polynomially solvable by a lexicographical greedy algorithm [23].

**Definition 1.1.** *An $m \times n$ matrix $C$ is called a Monge matrix if*

$$c_{i_1 j_1} + c_{i_2 j_2} \leq c_{i_1 j_2} + c_{i_2 j_1} \quad \forall\, i_1 < i_2, j_1 < j_2$$

Aggarwal and Park [7, 8] extended this notion to higher dimensional matrices.

**Definition 1.2.** *For $d \geq 2$, an $n_1 \times n_2 \times \cdots \times n_d$ $d$-dimensional matrix $C = c_{i_1,i_2,\ldots,i_d}$ has the Monge property if for all entries $c_{i_1,i_2,\ldots,i_d}$ and $c_{j_1,j_2,\ldots,j_d}$ we have*

$$c_{s_1,s_2,\ldots,s_d} + c_{t_1,t_2,\ldots,t_d} \leq c_{i_1,i_2,\ldots,i_d} + c_{j_1,j_2,\ldots,j_d}$$

*where for $1 \leq k \leq d$, $s_k = \min\{i_k, j_k\}$ and $t_k = \max\{i_k, j_k\}$.*

Another special case which is polynomially solvable is discussed by Burkard et al. [36]. Consider three $n$-element sequences $a_i, b_i, d_i$ of nonnegative real numbers such that the cost matrix $C$ can be decomposed as $c_{ijk} = a_i b_j d_k$. They show that the problem $\max_{p,q \in \pi_n} \sum_{i=1}^{n} c_{ip(i)q(i)}$ is solvable in polynomial time whereas the minimization version $\min_{p,q \in \pi_n} \sum_{i=1}^{n} c_{ip(i)q(i)}$ remains NP-hard (Theorems (1.2) and (1.4) below). The maximization version is a special case of the minimization version as follows. The array defined as $c_{ijk} = -a_i b_j d_k$ fulfills the three-dimensional Monge property as first proposed by Aggarwal and Park [7, 8]. Therefore Theorems (1.2) and (1.3) can be seen as a special case of a more general result of Bein, Brucker, Park and Pathak [23]. These authors have shown that the lexicographical greedy algorithm solves the *Multidimensional Transportation Problem* if and only if the cost array possesses the Monge property. The *Multidimensional Transportation Problem* is an extension of the *Multidimensional Assignment Problem*. We

provide the formulation for *Three Dimensional Transportation Problem* (3TP) and leave the multidimensional case as a simple exercise for the reader.

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} \\
\text{s.t.} \quad & \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = a_i \qquad \forall\, i = 1,\ldots,n \\
& \sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = b_j \qquad \forall\, j = 1,\ldots,n \\
& \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = e_k \qquad \forall\, k = 1,\ldots,n \\
& x_{ijk} \geq 0 \\
& x_{ijk} \in \mathbb{Z}
\end{aligned} \tag{1.19}$$

where

$$\sum_{i=1}^{n} a_i = \sum_{j=1}^{n} b_j = \sum_{k=1}^{n} e_k$$

Note that 3TP can be turned into 3AP if $a_i = b_j = e_k = 1$ for all $i, j, k = 1, \ldots, n$.

Burkard et al. [36] discuss more special cases of the 3AP where $c_{ijk} = a_i b_j d_k$ and the sequences $a_i, b_i, d_i$ satisfy certain properties. Both the 2-dimensional versions of the minimization and maximization problems, i.e. $\min_{p\in\pi_n} \sum_{i=1}^{n} c_{ip(i)}$ and $\max_{p\in\pi_n} \sum_{i=1}^{n} c_{ip(i)}$ are trivial by the following proposition [73].

**Proposition 1.1.** *Let $a_i$ and $b_i$ be two $n$-element sequences of real numbers sorted in non-decreasing order and let $p$ be an arbitrary permutation. Then*

$$\sum_{i=1}^{n} a_i b_{n-i+1} \leq \sum_{i=1}^{n} a_i b_{p(i)} \leq \sum_{i=1}^{n} a_i b_i$$

The following theorems are due to Burkard et al. [36].

**Theorem 1.2.** *Given three sequences $a_i, b_i$ and $d_i$ consisting of $n$ nonnegative numbers sorted in non-decreasing order, we have*

$$\sum_{i=1}^{n} a_i b_i d_i = \max_{p,q\in\pi_n} \sum_{i=1}^{n} a_i b_{p(i)} d_{q(i)}$$

**Theorem 1.3.** *Let $p_i^{(j)}$ be $m$ $n$-element sequences with nonnegative elements sorted increasingly and let $\phi_j$ be $m$ arbitrary permutations on $n$. Then*

$$\sum_{i=1}^{n}\prod_{j=1}^{m} p_i^{(j)} = \max_{\phi_1,\ldots,\phi_m} \sum_{i=1}^{n}\prod_{j=1}^{m} p_{\phi_j(i)}^{(j)}$$

**Theorem 1.4.** *Let three $n$-element sequences $a_i, b_i$ and $d_i$ of nonnegative rational numbers and a bound $S$ be given.*

1. *Then it is NP-complete to decide whether there exist permutations $p$ and $q$ such that $\sum_{i=1}^{n} a_i b_{p(i)} d_{q(i)} \leq S$.*
2. *For each $k \geq 1$ it is NP-hard to approximate the optimum solution within a factor of $n^k$.*

**Theorem 1.5.** *Consider sequences of the form $A = (1,\ldots,1,x,\ldots,x), B = (1,\ldots,1,y,\ldots,y)$ and $D = (1,\ldots,1,z,\ldots,z)$ with $1 < x \leq y \leq z$. Then 3AP with $c_{ijk} = a_i b_j d_k$ can be solved in $O(n)$ time.*

**Theorem 1.6.** *Consider sequences of the form $A = (1,\ldots,1,x,\ldots,x), B = (1,\ldots,1,y,\ldots,y)$ and $D = (d_1, d_2, \ldots, d_n)$ with $1 < x \leq y$ and $1 = d_1 \leq d_2 \leq \cdots \leq d_n$. Then 3AP with $c_{ijk} = a_i b_j d_k$ can be solved in $O(n)$ time.*

**Theorem 1.7.** *If the sequences $A$ and $B$ contain together at most $k$ distinct values, where $k \geq 1$ is some fixed integer, then 3AP with $c_{ijk} = a_i b_j d_k$ can be solved in $O(n^{k^2+1} \log n)$.*

**Theorem 1.8.** *If the sequence $A$ contains $(n-k)$ times the value $x$, where $k \geq 1$ is some fixed integer, then 3AP with $c_{ijk} = a_i b_j d_k$ can be solved in $O(n^{2k+1} \log n)$.*

### 1.8.2 Approximation Algorithms

As part 2 Theorem (1.4) by Burkard et al. [36] indicates, it is NP-hard to approximate the general 3AP (with arbitrary cost matrix) to a constant factor. However, the first researchers to show this result are Crama and Spieksma [43], as stated in Theorem (1.9). On the other hand, Theorem (1.4) indicates a stronger result than that of Theorem (1.9) in the sense that the general 3AP cannot be approximated to any polynomial factor unless $P = NP$.

The proof by Crama and Spieksma follows by investigating the special case described next. Consider the formulation discussed in equation (1.13). Each edge $(u, v) \in (I \times$

$J) \cup (I \times K) \cup (J \times K)$ is assigned a nonnegative length $d_{uv}$ and the cost of a triangle $(i, j, k) \in (I \cup J \cup K)$ is defined by either $t_{ijk}$ or $s_{ijk}$, where

$$t_{ijk} = d_{ij} + d_{ik} + d_{jk} \tag{1.20}$$

$$s_{ijk} = \min\{d_{ij} + d_{ik}, d_{ij} + d_{jk}, d_{ik} + d_{jk}\} \tag{1.21}$$

In other words, $t_{ijk}$ is the total length of a triangle and $s_{ijk}$ is the sum of the lengths of its two shortest edges. The 3AP with $c_{ijk} = t_{ijk}$ or $c_{ijk} = s_{ijk}$ is referred to as problem $T$ or $S$ respectively.

**Theorem 1.9.** *(Crama and Spieksma [43]) Unless $P = NP$, there is no $\epsilon$-approximate polynomial algorithm for problems $T$ and $S$ for any $\epsilon \geq 0$.*

Since $T$ and $S$ are speical cases of 3AP, this theorem proves that the general 3AP with arbitrary cost cannot be approximated to any constant factor in polynomial time unless $P = NP$.

Unlike the general 3AP, Crama and Spieksma [43] proved that some special cases can be approximated within a constant factor as described next. Consider the special cases of $T$ and $S$ for which the following triangle inequality holds

$$d_{uv} \leq d_{uw} + d_{vw} \quad \forall\, u, v, w \in I \cup J \cup K \tag{1.22}$$

These problems are referred to as $T\triangle$ and $S\triangle$ respectively.

**Theorem 1.10.** *(Crama and Spieksma [43]) Problems $T\triangle$ and $S\triangle$ are NP-hard.*

They present a $\frac{1}{2}$-approximate algorithm for both $T\triangle$ and $S\triangle$, i.e. a heuristic which always returns a feasible solution whose cost is at most $\frac{3}{2}$ of the optimal cost. The algorithm proceeds in two phases.

*Phase 1.* Find an optimal solution $x^*$ of

$$\begin{aligned} \min \quad & \sum_{i \in I} \sum_{j \in J} d_{ij} x_{ij} & \\ \text{s.t.} \quad & \sum_{i \in I} x_{ij} = 1 & \forall\, j \in J \\ & \sum_{j \in J} x_{ij} = 1 & \forall\, i \in I \\ & x_{ij} \geq 0 & \forall\, i \in I, j \in J \end{aligned} \tag{1.23}$$

Let $M = \{(i,j) \mid x^*_{ij} = 1\}$.

*Phase 2.* Find an optimal solution $y^*$ of

$$\begin{aligned} \min \quad & \sum_{(i,j)\in M} \sum_{k\in K} c_{ijk} y_{ijk} & \\ \text{s.t.} \quad & \sum_{(i,j)\in M} y_{ijk} = 1 & \forall\, k \in K \\ & \sum_{k\in K} y_{ijk} = 1 & \forall\, (i,j) \in M \\ & y_{ijk} \in \{0,1\} & \forall\, (i,j) \in M, k \in K \end{aligned} \tag{1.24}$$

where $c_{ijk} = t_{ijk}$ or $c_{ijk} = s_{ijk}$ depending on whether the problem is an instance of $T\triangle$ or $S\triangle$. This algorithm is denoted by $H_{IJ}$ (since it minimizes the assignment from $I$ to $J$) and its cost is denoted by $cost(H_{IJ})$. One can obtain two similar algorithms $H_{IK}$ and $H_{JK}$ by modifying the problems (1.23) and (1.24) above. Notice that both problems (1.23) and (1.24) are instances of AP and hence can be solved in $O(n^3)$ [82]. It follows that $H_{IJ}$ can be solved in $O(n^3)$ as well. Let $OPT_\tau$ denote an optimal solution to a 3AP instance $\tau$.

**Theorem 1.11.** *(Crama and Spieksma [43]) $H_{IJ}$ is a $\frac{1}{2}$-approximate algorithm for problems $T\triangle$ and $S\triangle$. Moreover, there exist arbitrary large instances $\tau$ of $T\triangle$ and $S\triangle$ such that $cost(H_{IJ}) = \frac{3}{2} cost(OPT_\tau)$.*

One can easily modify the proof to obtain the same result for $H_{IK}$ and $H_{JK}$. Consider now the heuristic $H$ which applies all three heuristics $H_{IJ}, H_{IK}$ and $H_{JK}$ separateley to an instance of $T\triangle$ or $S\triangle$ and retains the best solution. Let $\gamma$ denote the solution obtained by $H$. Then $cost(\gamma) = \min\{cost(H_{IJ}), cost(H_{IK}), cost(H_{JK})\}$.

**Theorem 1.12.** *(Crama and Spieksma [43]) $H$ is a $\frac{1}{3}$-approximate algorithm for problems $T\triangle$ and $S\triangle$. Moreover, there exist arbitrary large instances $\tau$ of $T\triangle$ and $S\triangle$ such that $cost(\gamma) = \frac{4}{3} cost(OPT_\tau)$.*

## 1.9 Applications of 3AP

There are some practical applications of 3AP in the literature which we discuss next.

### 1.9.1 Assignment of Workers to Jobs to Machines

Consider the following example. There are $n$ workers, $n$ jobs and $n$ machines. For each triple $(i,j,k)$ the time required for worker $i$ to complete job $j$ on machine $k$ is $c_{ijk}$. Each

worker is assigned to exactly one job and exactly one machine. The objective is to minimize the total time taken by the workers to complete the jobs using the available machines. The formulation (1.12) above solves this problem [61]. Each $x_{ijk}$ is an indicator variable for worker $i$ doing job $j$ on machine $k$. The first set of constraints specifies that each worker is assigned to exactly one job-machine pair. The second set of constraints enforces that each job gets assigned to exactly one person and one machine. The third set of constraints requires each machine to be worked on by one person and for the purpose of one job.

### 1.9.2 Scheduling of Teaching Practices

Student teachers at colleges of education have to undertake teaching practices. Each student teacher is supervised by a tutor who monitors the student and provides feedback. There are a number of schools available for the student teacher to choose from. Arranging which school each student is assigned to and which tutor will supervise the student is complicated by the fact that students, tutors and schools have requirements and preferences.

Frieze and Yadegar [53] considered this problem and provided the following formulation. Suppose there are $m$ students, $n$ tutors and $p$ schools. Let $I = \{1, \ldots, m\}$, $J = \{1, \ldots, n\}$, $K = \{1, \ldots, p\}$. For each student $i$, tutor $j$ and school $k$ let $v_{ijk}$ be a *satisfaction value* associated with the assignment of student $i$ to school $k$ under the supervision of tutor $j$. Tutor $j$ is willing to supervise no more than $t_j$ students and school $k$ can have at most $s_k$ student teachers assigned to it. The problem formulation is

$$\begin{aligned}
\max \quad & \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} v_{ijk}x_{ijk} & \\
\text{s.t.} \quad & \sum_{j\in J}\sum_{k\in K} x_{ijk} = 1 & \forall\, i \in I \\
& \sum_{i\in I}\sum_{k\in K} x_{ijk} \le t_j & \forall\, j \in J \\
& \sum_{i\in I}\sum_{j\in J} x_{ijk} \le s_k & \forall\, k \in K \\
& x_{ijk} \in \{0,1\} & \forall\, i,j,k = 1,\ldots,n
\end{aligned} \tag{1.25}$$

The above formulation can be easily converted to the more convenient formulation (1.12) by means of adding duplicate nodes in $I$, $J$ and $K$ and turning the *max* into *min* by replacing $v_{ijk}$ with $M - v_{ijk}$ where

$$M = \max\{v_{ijk} \mid i \in I, j \in J, k \in K\}$$

### 1.9.3 Dynamic Facility Location

Consider the case where a company has to add $p$ new warehouses at $q$ potential sites over the next $r$ years. The company is not only faced with the problem of where to locate the warehouses but also when should the warehouses be built. Each $c_{ijk}$ is the total cost of building warehouse $i$ on site $j$ at time $k$. Many factors such as discounted construction costs, future transportation costs for shipping, operating costs, etc could be incorporated into each $c_{ijk}$. The company is interested in minimizing the total cost associated with this expansion [102].

Other notable applications include assembly of printed circuit boards [42], military troops assignment [101] and satellite coverage optimization [102].

### 1.9.4 Applications of 3PAP and MAP

An application of 3PAP involves trade-show scheduling [60]. At a trade show there are $r$ vendors, $s$ customers and $t$ time slots, where $r \geq s \geq t$. Vendors and customers are to meet one-on-one at different times. Each vendor can meet at most one customer per time slot and each customer can meet one vendor at a time. For each vendor, customer and time triple $(i, j, k)$ the cost $c_{ijk}$ is known, where lower cost indicates greater desirability. The problem formulation is as follows

$$\begin{aligned}
\min \quad & \sum_{i=1}^{r}\sum_{j=1}^{s}\sum_{k=1}^{t} c_{ijk}x_{ijk} & \\
\text{s.t.} \quad & \sum_{i=1}^{r} x_{ijk} = 1 & \forall\, j = 1,\ldots,s, \forall\, k = 1,\ldots,t \\
& \sum_{j=1}^{s} x_{ijk} \leq 1 & \forall\, i = 1,\ldots,r, \forall\, k = 1,\ldots,t \\
& \sum_{k=1}^{t} x_{ijk} \leq 1 & \forall\, i = 1,\ldots,r, \forall\, j = 1,\ldots,s \\
& x_{ijk} \in \{0,1\} & \forall\, i,j,k = 1,\ldots,n
\end{aligned} \tag{1.26}$$

The first set of constraints states that each customer must meet exactly one vendor in each time slot. The second constraints set allows each vendor to meet at most one customer per time slot. The third constraint set prevents customers and vendors from meeting each other more than once.

A dominant application of MAP is in the area of multi-sensor multi-target tracking

[104]. Suppose that we are given a sequence of observations made at times $t_1, t_2, \ldots, t_n$. Each observation contains objects with their respective positions. The goal is to match objects in each of the observations so that the matched objects in different observations have the maximum probability of being the same actual objects. By doing so we track the objects throughout the observations. An example is recognizing airplanes in a radar screen.

## 1.10 Exact Algorithms for 3AP

There are numerous exact algorithms in the literature for solving 3AP. These algorithms obtain the optimal solution and prove the optimality of the obtained solution. Balas and Saltzman [14] introduced a branch-and-bound method for 3AP. Branch-and-bound algorithms split the current problem into two subproblems by fixing one variable $x_{ijk}$ to 1 and 0 in order to decrease the subproblem size. The method of Balas and Saltzman exploits the polyhedral structure of the problem by fixing several variables at each branching step. Instead of the LP-relaxation, bounds are computed using a Lagrangian relaxation which incorporates facets of 3AP polytope. This relaxation is solved by a modified subgradient optimization method. Also, new and efficient primal heuristics are used to obtain successively improved approximate solutions.

Frieze and Yadegar [53] proposed another subgradient procedure for solving a Lagrangian relaxation approach to 3AP together with computational results. An example of a subgradient approach is discussed in [29, 32]. Consider taking two blocks of the constraints of formulation (1.12) into the objective function via Lagrangian multipliers. The Lagrangian relaxation is

$$L(\pi,\epsilon) = \min \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}(c_{ijk}+\pi_j+\epsilon_i)x_{ijk} - \sum_{j=1}^{n}\pi_j - \sum_{i=1}^{n}\epsilon_i \right\} \tag{1.27}$$

such that

$$\begin{aligned} &\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 \quad \forall\, k = 1,\ldots,n \\ &x_{ijk} \in \{0,1\} \qquad \forall\, i,j,k = 1,\ldots,n \\ &\pi \in \mathbb{R}^n \\ &\epsilon \in \mathbb{R}^n \end{aligned} \tag{1.28}$$

$L(\pi,\epsilon)$ is a concave function and a subgradient method can be used to find its maximum. Let $r$ be a counter of iterations. Algorithm (1.1) describes the procedure.

**Algorithm 1.1:** Maximizing $L(\pi, \epsilon)$

Start with $r \leftarrow 0, \pi^r \leftarrow 0, \epsilon^r \leftarrow 0$.

Use a greedy algorithm to minimize $L(\pi^r, \epsilon^r)$. Let $x^r_{ijk}$ be the corresponding optimal solution. Let

$v^r_{i_0} = |\{x^r_{i_0,j,k} \mid x_{i_0,j,k} = 1\}| - 1$ for $i_0 = 1, \ldots, n$ and

$w^r_{j_0} = |\{x^r_{i,j_0,k} \mid x_{i,j_0,k} = 1\}| - 1$ for $j_0 = 1, \ldots, n$.

**if** $v^r = w^r = (0, 0, \ldots, 0))$ **then**

the maximum is reached. Terminate.

**end if**

**if** a prespecified number of iterations is not yet reached **then**

$\pi^{r+1} \leftarrow \pi^r + \lambda_r w^r$

$\epsilon^{r+1} \leftarrow \epsilon^r + \lambda_r v^r$, where $\lambda_r$ is a suitable step length. Go to step 2.

**else**

terminate

**end if**

## 1.11 Heuristic Algorithms for 3AP

There are many heuristic algorithms in the literature for obtaining near optimal solutions for 3AP. Heuristic algorithms give no guarantee of the solution quality but in practice obtain good suboptimal and possibly optimal solutions. Pierskalla [101] was the first to propose a heuristic method for 3AP. In this section we discuss two of the more successful heuristics in terms of solution quality for benchmark instances; GRASP with path relinking [10] and Hybrid Genetic [76].

### 1.11.1 GRASP with Path Relinking

GRASP is a greedy randomized adaptive search procedure. It is a multistart metaheuristic for combinatorial optimization. GRASP includes a construction phase based on a greedy randomized algorithm and a local search phase. Path relinking is an intensification procedure which investigates trajectories that connect high quality solutions. The best overall solution is kept as the result. GRASP with path relinking was able to improve the solution quality of heuristics proposed by Balas and Saltzman [14], Burkard et al. [36], and Crama and Spieksma [43] on all instances proposed in those papers.

In the construction phase a feasible solution is iteratively constructed one element at a time. The choice of the next element to be added is determined by ordering all candidate elements according to a greedy function. This function measures the benefit of selecting each element. In order to make GRASP adaptive, the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the previous element. The random component stems from randomly choosing one of the best candidates in the list, which is not necessarily a top candidate.

In the local search phase the current solution is improved by searching its neighborhood. If an improvement is found the solution is updated and neighborhood search continutes. If no improvement is detected, then local search stops. Given a solution $s$, the definition of the neighborhood $N(s)$ is crucial for the performance of local search.

**Definition 1.3.** *For two permutations $p$ and $p'$ the Hamming distance between them is defined to be $d(p, p') = |\{i \mid p(i) \neq p'(i)\}|$.*

A solution $s$ contains two permutations $p$ and $q$. GRASP uses a 2-exchange neighborhood defined as

$$N_2(p, q) = \{p', q' \in \pi_n \mid d(p, p') + d(q, q') = 2\} \tag{1.29}$$

In other words, a 2-exchange swaps either 2 entries in $p$ or $q$. Hence the size of the neighborhood is $|N_2(p)| + |N_2(q)| = 2\binom{n}{2}$.

Path relinking was first introduced in the context of tabu search [66] as an approach to combine intensification and diversification in the search. Path relinking explores trajectories that connect high quality solutions. It achieves this by starting from an initial solution and generating a path in the neighborhood of this solution towards another solution called the guiding solution. This path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution. At each step all moves that incorporate attributes of the guiding solution are analyzed and the move that best improves the initial solution is chosen.

### 1.11.2 Hybrid Genetic

The GRASP algorithm discussed in the previous section obtained superior results to other existing heuristics. The hybrid genetic algorithm proposed by Huang and Lim [76] outperformed GRASP in every instance with regards to solution quality. This indicates that the

hybrid genetic algorithm is the most successful algorithm in the literature. In addition, the computation time was much shorter than that of GRASP.

Genetic algorithm (GA) is one of the most successful evolutionary algorithms and is based on an analogy with Darwin's evolution theory of natural selection and survival of the fittest [75]. Genetic algorithm has been applied to solve general combinatorial optimization problems [75]. The results can be further improved if problem specific knowledge is incorporated into GA. The hybridization between local search (LS) and GA reflects this idea.

The local search phase of the hybrid genetic algorithm takes advantage of formulation (1.14). Given two permutations $p$ and $q$ which map $I$ to $J$ and $I$ to $K$ respectively, let $r = q \circ p^{-1}$ be the mapping from $J$ to $K$. We can reduce the problem 3AP to AP in 3 ways:

- Case (1): fix the mapping from $I$ to $J$ (permutation $p$)
- Case (2): fix the mapping from $I$ to $K$ (permutation $q$)
- Case (3): fix the mapping from $J$ to $K$ (permutation $r$)

Consider case (1). Define an $n \times n$ matrix $D$ such that

$$d_{ij} = c_{p^{-1}(i)ij} \tag{1.30}$$

Then the problem reduces to the following AP with $D$ as the cost matrix

$$\min_{r \in \pi_n} \sum_{i=1}^{n} d_{ir(i)} \tag{1.31}$$

and the permutation $q$ has to be updated by $q = r \circ p$.

Consider case (2). Matrix $D$ is constructed by

$$d_{ij} = c_{ijq(i)} \tag{1.32}$$

Then the corresponding AP is

$$\min_{p \in \pi_n} \sum_{i=1}^{n} d_{ip(i)} \tag{1.33}$$

and $q$ is unchanged.

Consider case (3). Let

$$d_{ij} = c_{ijr(j)} \tag{1.34}$$

and the corresponding AP is the same as (1.33) and $q$ is updated by $q = r \circ p$. After reducing the 3AP to an AP, the hybrid genetic algorithm solves the AP to optimality by the Hungarian Method, which can be done in $O(n^3)$ [82]. Algorithm (1.2) describes the local search phase of hybrid genetic.

**Algorithm 1.2:** Local Search stage of Hybrid Genetic

```
Start with any initial solution to 3AP
finishFlag ← false
while not finishFlag do
  finishFlag ← true
  for cases 1 to 3 do
    Construct corresponding bipartite graphs according to Eqs. (1.30), (1.32) and
    (1.34). Solve the resulting AP by Hungarian Method.
    if objective value decreases then
      finishFlag ← false
    end if
  end for
end while
```

## 1.12 Introduction to Quadratic 3-Dimensional Assignment Problem

Pierskalla (1967) introduced the Quadratic 3-Dimensional Assignment Problem (Q3AP) in a technical memorandum. The work was never published in the open literature. Since then nothing on the subject appeared in the literature until [71] rediscovered Q3AP while working on a problem arising in data transmission system design.

Consider the Lawler [84] formulation of QAP as discussed in (1.3). This formulation extends AP to QAP by making the objective function quadratic. Consider the 3AP formulation mentioned in (1.12). Similar to extending AP to QAP, we can extend 3AP to Q3AP

by making the objective function in (1.12) quadratic as follows

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n}\sum_{m=1}^{n}\sum_{r=1}^{n} c_{ijklmr}x_{ijk}x_{lmr} & \\
\text{s.t.} \quad & \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 & \forall\, i = 1,\ldots,n \\
& \sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 & \forall\, j = 1,\ldots,n \\
& \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 & \forall\, k = 1,\ldots,n \\
& x_{ijk} \in \{0,1\} & \forall\, i,j,k = 1,\ldots,n
\end{aligned} \tag{1.35}$$

The constraints of Q3AP in equation (1.35) indicate that the quadratic expression in the objective function of Q3AP need not include any $x_{ijk}x_{lmr}$ terms for which $i = l$ or $j = m$ or $k = r$ unless all three equalities hold. If $i = l, j = m$ and $k = r$, then $x_{ijk}x_{lmr} = x_{ijk}$, otherwise $x_{ijk}x_{lmr} = 0$. Therefore, from a computational perspective, Q3AP can be more efficiently represented as

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} b_{ijk}x_{ijk} + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{\substack{l=1\\l\neq i}}^{n}\sum_{\substack{m=1\\m\neq j}}^{n}\sum_{\substack{r=1\\r\neq k}}^{n} c_{ijklmr}x_{ijk}x_{lmr} & \\
\text{s.t.} \quad & \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 & \forall\, i = 1,\ldots,n \\
& \sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 & \forall\, j = 1,\ldots,n \\
& \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 & \forall\, k = 1,\ldots,n \\
& x_{ijk} \in \{0,1\} & \forall\, i,j,k = 1,\ldots,n
\end{aligned} \tag{1.36}$$

where $b_{ijk} = c_{ijkijk}$. The permutation version of Q3AP is

$$\min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} \tag{1.37}$$

Viewd differently, Q3AP in (1.35) is the extension of the QAP formulation (1.3) to the three dimensional version.

## 1.13 Complexity of Q3AP

Since Q3AP is an extension of the NP-hard problems 3AP and QAP [56, 109], it is easy to verify that Q3AP is also NP-hard. Simply let $c_{ijklmr} = 0$ in (1.36) to reduce Q3AP to the special subproblem 3AP, or project the three dimensions to two dimensions in (1.35) to obtain QAP as a special subproblem. In Chapter 4 we discuss special cases of Q3AP as well as approximation algorithms and derive new theoretical results.

# Chapter 2

# Randomized Local Search for Quadratic Assignment Problem

In this chapter we propose a method called Randomized Local Search (RandLS) and apply it to Quadratic Assignment Problem (QAP). Although RandLS can be easily applied to other combinatorial optimization problems, the focus of this chapter is on QAP. We apply the algorithm to solve benchmark instances of QAP. Our algorithm produced an improved solution for the largest QAP benchmark problem arising from microarray studies in very reasonable running time. Also, we obtained very good experimental results on other benchmark instances. This chapter contains a discussion of our heuristic algorithm and is computational in nature.

## 2.1 Local Search and QAP

A mathematical optimization problem can be represented by an ordered pair $(F, f)$ where $F$ is the family of feasible solutions, $f : F \longrightarrow \mathbb{R}$ is the objective function which is used to compare solutions and $\mathbb{R}$ is the set of real numbers. Many optimization problems of practical interest are difficult to solve to optimality. Researchers and practitioners depend on good quality heuristic algorithms to compute near optimal solutions for such problems. Local search is perhaps the most well studied heuristic approach to handle hard optimization problems. Starting with a feasible solution, $x^0$, a local search algorithm explores its neighborhood, $N(x^0)$ for an improving solution. If no improving solution is found, $x^0$ is

a local optimum and the algorithm outputs this solution. If $N(x^0)$ contains an improving solution $x^1$, then $x^1$ takes the role of $x^0$ and the process is continued until a locally optimal solution is identified. In some variations of local search, $x^1$ is chosen as the best improving solution in $N(x^0)$. The major drawback of local search is that, for simple neighborhoods, the resulting local optimum could be far from a global optimum.

Research in overcoming this difficulty has resulted in various classes of meta-heuristics which include Tabu Search [63, 64, 66], Simulated Annealing [80, 118], Variable Neighborhood Search [72, 92], Tabu Thresholding [65] etc. We discuss a very simple strategy, called *Randomized Local Search*(RandLS) that uses ideas parallel to Simulated Annealing, Tabu Search and Tabu Thresholding. The simplicity of this algorithm and quality of the solutions obtained makes it a viable alternative to other local search based algorithms. We primarily consider two variants of randomized local search - A Simulated Annealing variant which we call RandLS-Sim and a Tabu Search variant. A preliminary version of RandLS-Sim was introduced by Punnen and Aneja [106] and reported limited experimental results on the Quadratic Assignment Problem (QAP). The Tabu Search version of RandLS is known as Tabu Thresholding and was introduced by Glover [65]. Successful application of Tabu Thresholding was reported by various authors on several classes of optimization problems [117, 119]. No experimental results using Tabu Thresholding is available for the QAP.

In this chapter we discuss the general Randomized Local Search paradigm which subsumes RandLS-Sim and Tabu Thresholding. Detailed experimental results are reported for the QAP using RandLS-Sim and Tabu Thresholding. Most interestingly, we obtained solutions that either match or are better than the best known solutions for the QAP benchmark instances arising out of microarray studies [3]. We also tested our algorithms on the QAPLIB [33] benchmark problems. These problems are thoroughly investigated by various authors [35, 50, 88] using a variety of heuristic algorithmic ideas. No heuristic algorithm uniformly outperformed other heuristics on these test problems. Some of the best known heuristic algorithms for QAP include Robust Tabu Search [115], Genetic Hybrid [50], GRASP [88] and Simulated Annealing [35]. Out of 135 QAPLIB instances, our algorithm obtained optimal or best known solutions for 111 problems. Among the RandLS algorithms we tested, Tabu Thresholding appears more robust and is a good alternative to compute quality solutions for QAP in very reasonable running time.

We next discuss the general RandLS algorithm and its specializations RandLS-Sim and Tabu Thresholding.

## 2.2 Randomized Local Search

Let $x^0$ be a feasible solution. An *eligible candidate list* (ECL) with respect to $x^0$ is a finite subset $E(x^0)$ of $N(x^0)$ such that $x \in N(x^0) \setminus E(x^0)$ implies $f(x) \geq f(y) \forall y \in E(x^0)$. That is $E(x^0)$ is the set of $k$ best solutions in $N(x^0)$ where $k = |E(x^0)|$. The number $k$ is referred to as *degree of freedom*. The RandLS algorithm starts with an initial value of the degree of freedom, an initial solution $x$ and constructs the candidate list $E(x)$. Then it selects a solution randomly from $E(x)$ and moves to this solution. Also, the algorithm maintains a "current best solution" which is updated, if possible, while exploring the neighborhood $N(x^0)$. The process is repeated until a prescribed *transition property* $P$ is satisfied. After this, the value of degree of freedom is reset and the process is continued until a prescribed termination criterion is satisfied. A formal description of RandLS is given by Algorithm (2.1) below.

**Algorithm 2.1:** RandLS

**Input**: $k$: initial degree of freedom; $P$: transition property; $x$: initial solution; $f(x)$: value of $x$

**Output**: $x_{\text{best}}$, the best solution found and its value $f(x_{\text{best}})$

```
x_best ← x;
f(x_best) ← f(x);
while termination criterion is not satisfied do
    repeat
        Construct the candidate list E(x);
        y ← best element of E(x);
        if f(y) < f(x_best) then
            x_best ← y; x ← y; f(x_best) ← f(y);
        else
            x ← a random element of E(x);
        end
    until P is satisfied;
    update k and (if necessary) P;
end
output x_best and f(x_best);
```

Let us now consider two important variations of Algorithm (2.1); RandLS-Sim and

Tabu Thresholding. In RandLS-Sim, we set the termination criterion as $k < 1$. That is, the algorithm terminates when the degree of freedom $k < 1$. The transition property $P$ is an iteration limit on the number of times the repeat loop is executed between two consecutive updates of $k$. After the transition property is satisfied, $k$ is updated as $k \leftarrow (k-1)$. RandLS-Sim is comparable to Simulated Annealing where the degree of freedom corresponds to the temperature parameter which is set high initially and brought down gradually.

In the Tabu Thresholding version, updating the transition property $P$ and $k$ is handled by Algorithm (2.2) below. Throughout the algorithm, the "repeat loop" refers to the one in Algorithm (2.1).

**Algorithm 2.2:** Update-Transition

**Input**: $k$: Degree of freedom; $iter(k)$: Iteration limit per degree of freedom

$k' \leftarrow k$;
$k \leftarrow 1$ in the repeat loop;
$P$ is satisfied when the current solution is optimal;
$k \leftarrow k'$ in the repeat loop;
$P$ is satisfied when the repeat loop is performed $iter(k)$ number of times;

In other words, the transition property $P$ is set initially as "local optimality of the current solution" with respect to the neighborhood $N$ and $k = 1$. Thus the repeat loop is essentially a standard local search. Once a local minimum is reached, increase $k$ to the degree of freedom, which is an input parameter usually between 10 and 20. Then $P$ is modified as an iteration limit of the repeat loop, which is the input parameter $iter(k)$ usually set between 10 and 30. After checking the termination criterion, $k$ is updated back to 1 and $P$ is updated back to local optimality and the sequence is alternated until termination criterion is satisfied which is an iteration limit. Thus Tabu Thresholding performs a sequence of local search operations while upon reaching a local optimum, random moves (controlled by the degree of freedom) are employed to escape from the local minimum while restricting to stay within a "not so bad" region. Tabu Thresholding is analogous to the Tabu Search but Tabu list has size 1 and includes the last move so that we avoid reversing the most recent move.

## 2.3 The QAP and Proposed Heuristics

The definition of neighbourhood is an integral part of local search. The next definition shines light on how we form neighbourhoods in our RandLS algorithm.

**Definition 2.1.** *For any permutation $\pi \in \pi_n$ the neighborhood $N(\pi)$ is defined as*

$$N(\pi) = \{\sigma \mid \sigma \in F \text{ and } \sigma \text{ can be obtained from } \pi \text{ by interchanging two locations (i.e. the location of two facilities)}\}.$$

We call $N(\pi)$ the two-exchange neighborhood of $\pi$. Thus $\sigma \in N(\pi)$ can be obtained by exchanging the locations of $r$ and $s$ for some $1 \leq r, s \leq n$. Let $\Delta(\pi, r, s) = f(\sigma) - f(\pi)$. Then $\Delta(\pi, r, s)$ is given by [115]

$$\begin{aligned}\Delta(\pi, r, s) =& b_{rr}(a_{\pi_s\pi_s} - a_{\pi_r\pi_r}) + b_{rs}(a_{\pi_s\pi_r} - a_{\pi_r\pi_s}) + b_{sr}(a_{\pi_r\pi_s} - a_{\pi_s\pi_r}) + \\ & b_{ss}(a_{\pi_r\pi_r} - a_{\pi_s\pi_s}) + \sum_{k=1, k\neq r,s}^{n} (b_{kr}(a_{\pi_k\pi_s} - a_{\pi_k\pi_r}) + \\ & b_{ks}(a_{\pi_k\pi_r} - a_{\pi_k\pi_s}) + b_{rk}(a_{\pi_s\pi_k} - a_{\pi_r\pi_k}) + b_{sk}(a_{\pi_r\pi_k} - a_{\pi_s\pi_k}))\end{aligned} \tag{2.1}$$

Thus given the matrices $A = (a_{ij})$ and $B = (b_{ij})$, $\Delta(\pi, r, s)$ can be obtained in $O(n)$ time. The change in the objective function due to a swap can be evaluated faster using information from a preceding exchange operation. Suppose $\psi$ is the assignment obtained by exchanging the locations $r$ and $s$ in $\pi$. Then the effect of swapping facilities $u$ and $v$ in $\psi$, where $(\{r, s\} \cap \{u, v\} = \emptyset)$, can be evaluated in constant time [115] using the equation

$$\begin{aligned}\Delta(\psi, u, v) =& \Delta(\pi, u, v) + (b_{ru} - b_{rv} + b_{sv} - b_{su})(a_{\pi_s\pi_u} - a_{\pi_s\pi_v} + a_{\pi_r\pi_v} - \\ & a_{\pi_r\pi_u}) + (b_{ur} - b_{vr} + b_{vs} - b_{us})(a_{\pi_u\pi_s} - a_{\pi_v\pi_s} + a_{\pi_v\pi_r} - a_{\pi_u\pi_r})\end{aligned} \tag{2.2}$$

Using equation (2.1) finding the neighborhood $N(\pi)$ for the first time takes $O(n^3)$ time. For subsequent iterations, the neighborhood can be evaluated in $O(n^2)$ time using equations (2.1) and (2.2).

### 2.3.1 RandLS-Sim for the QAP

Let us now discuss our implementation of the RandLS-Sim algorithm for the QAP. Let $initial(k)$ be the starting degree of freedom. In each iteration we select a random member from the $k$ best members of the 2-exchange neighborhood of the current permutation $\pi$,

where $k$ is the degree of freedom during the iteration. This is obtained by choosing the $j^{\text{th}}$ smallest element of the set $\{\Delta(\pi, r, s) \mid 1 \leq r, s, \leq n\}$ where $j$ is a random number between 1 and $k$. This can be achieved in $O(n^3)$ time for the first iteration and in $O(n^2)$ time in subsequent iterations using the formula (2.1) and (2.2) and the linear time algorithm [105] for computing the $j^{\text{th}}$ best element of a set. After selecting an element from $N(\pi)$ the algorithm moves to this solution and the current best solution is updated if applicable. Let $iter(k)$ be the number of iterations allowed at degree of freedom $k$ and when this limit is exceeded, $k$ is reduced by one. The algorithm terminates when a local minimum is reached after $k = 1$. A formal description is presented in Algorithm (2.3) below.

**Algorithm 2.3:** RandLS-Sim-QAP

**Input**: $k$: Initial degree of freedom; $iter(k)$: Iteration limit per degree of freedom;

**Output**: The best solution found and its value

$\pi \leftarrow$ A random permutation of $\{1, 2, \ldots, n\}$;
Compute $f(\pi)$;
solution $\leftarrow \pi$;
value $\leftarrow f(\pi)$;
**repeat**
  count $\leftarrow 0$;
  **repeat**
    $\Delta(\pi, p, q) \leftarrow \min\{\Delta(\pi, r, s) \mid 1 \leq r, s \leq n\}$;
    $\sigma \leftarrow$ The permutation obtained from $\pi$ by interchanging locations $p$ and $q$;
    **if** *value* $> f(\sigma)$ **then**
      solution $\leftarrow \sigma$; value $\leftarrow f(\sigma)$; $\pi \leftarrow \sigma$;
    **else**
      $j \leftarrow$ A random integer in $[1, k]$;
      $\Delta(\pi, u, v) \leftarrow j^{\text{th}}$ best element of $\{\Delta(\pi, r, s) \mid 1 \leq r, s \leq n\}$;
      $\pi \leftarrow$ The permutation obtained from $\pi$ by interchanging locations $u$ and $v$;
    **end**
    count $\leftarrow$ count+1;
  **until** $count = iter(k)$;
  k $\leftarrow$ k-1;
**until** $k = 1$;

During the experiments we observed that the stopping criteria of $k = 1$ causes the solution to cycle in the neighborhood. Therefore we decided to stop when $k = 4$. The performance of the basic algorithm may be enhanced by using multiple starts using different starting permutations. During these multiple starts, the parameters $k$ and $iter(k)$ may be changed as well to further diversify the search. When $k$ becomes very small, we noticed that the algorithm goes into cycling. Thus it is recommended that when $k$ reaches 4 go directly into the local search phase.

Another variation that we tested uses occasional *peek* into bad regions to diversify the search. This is implemented as follows. After completing $iter(k)$ iterations at the degree of freedom $k$, we reach into "bad" regions by setting $k = \binom{n}{2}$, the neighborhood size. Then we are able to make a random move outside the smallest $k$ neighbors. A number of *peek* moves are done before reverting $k$ to its previous value.

### 2.3.2 Tabu Thresholding Algorithm

As indicated earlier, this scheme was originally proposed by Glover [65] and it is a variation of our general RandLS paradigm. The algorithm starts with a random permutation and performs a local search (i.e. choosing $k = 1$). Upon reaching a local minimum, we increase the value of $k$ to a fixed number and $iter(k)$ iterations are performed at this level of $k$. During these iterations, if we encounter a solution better than the best solution so far, we always move to this solution and start local search again. Otherwise, after completing $iter(k)$ moves at degree of freedom $k$, we switch back to local search and the process is continued until a fixed number of iterations are performed and output the best solution obtained. A formal description is given in Algorithm (2.4). We found it useful to include a Tabu list of size 1 during the $iter(k)$ iterations to avoid going back to the preceding permutation.

## 2.4 Computational Results

Recently, de Carvalho Jr. and Rahmann [45] studied the layout problem of Oligonucleotide microarrays and formulated the microarray placement problem as QAP. The resulting test problems are available at [3] and their best known solutions are available at [1]. Specializing RandLS-Sim and Tabu Thresholding to QAP, we obtained efficient heuristics for QAP which resulted in improved solution to the largest instance in the Conflict Index class. Appendix 1 lists the improved solution.

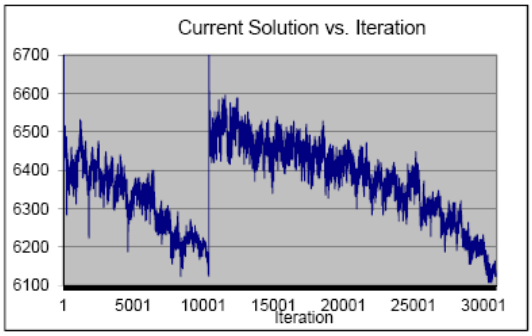


Figure 2.1: RandLS-Sim

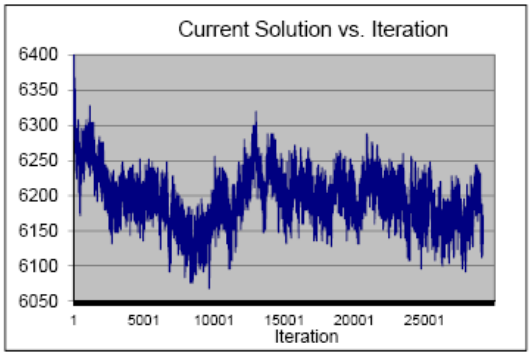


Figure 2.2: RandLS-Tabu

Algorithms (2.3) and (2.4) have been coded in C++ and tested on a Dell Precision workstation with 512 MB of memory and a 2 GHz Intel Xeon processor running Linux Mandrake operating system. The experiments were conducted in two stages. Stage 1 was designed to calibrate algorithm parameters to fix default values of the parameters. Using the default values identified in Stage 1, experiments were conducted using benchmark problems from the QAPLIB [33] and Microarray layout QAP problems [3].

After careful experimentation and fine-tuning we have concluded that we set the parameters $k = n$ initially and $iter(k) = 200$ for algorithm RandLS-Sim. The peek operation, although gave improved solutions in some isolated cases, was not found very effective and hence we did not use it in Stage 2 of our experiments. When $k$ reaches value 4, we proceed to local search directly to complete the algorithm. For the RandLS-Sim-QAP algorithm, Figure 1 gives the progression of current objective function against the number of iterations on a typical test problem and Figure 2 gives corresponding information for algorithm RandLS-Tabu-QAP for a particular instance. For algorithm RandLS-Tabu, we set default values as $k = 15$ during the random move phase and $iter(k) = 30$. The total number of iterations is set as $count - limit = 1,000,000$.

Using the default values indicated above we have tested QAP instances developed out of microarray layout problems [3]. The results are given in Tables (2.1) and (2.2) for the two different groups. The initial solution was chosen at random. After running our algorithms we compared our results with the best known solutions available on [1] as of March 2010. The "best known" column refers to those solutions. The run times in [1] range from less than 2 minutes per single run for the $n = 36$ instance to less than 9 hours for the $n = 144$

problems. RandLS-Sim and RandLS-Tabu are each run 5 times with the same default parameters as described above. The column "Obj Value" refers to the solution cost, which includes the average (rounded to the nearest integer) and the best cost over the 5 trials. Time is measured in seconds and has been rounded to the nearest integer. It includes the average time and the time taken to obtain the best solution (not necessarily the shortest time).

The bold entries in Tables (2.1) and (2.2) indicate the best known solutions. Underlined entries indicate that we matched the best known solutions. According to those tables, we did not outperform the best known solutions from [1], which may be deceiving. However, after fine-tuning the parameters for the specific instances, we did obtain better results. Appendix 1 lists the overall best solutions we obtained among all experiments (and the parameters used to obtain these solutions), which may not necessarily be reported in Tables (2.1) and (2.2). Out of the 14 test problems, our algorithm with fine-tuned parameter values obtained an improved solution for the Conflict Index instance with $n = 144$ and matched some of the best known solution to the remaining Conflict Index instances. As the table in Appendix 1 indicates, Tabu Thresholding worked better for most of the problems. The corresponding parameters used to obtain the table are reported as well.

For the microarray test problems, Table (2.3) summarizes the results obtained by using RandLS-Tabu solution as the initial solution for RandLS-Sim. The initial solution was chosen as the output generated by one of the five trials of RandLS-Tabu. More specifically, we used the trial which outperformed the remaining four. This experiment was run 3 times and the results are provided in Table (2.3). The column "Improved runs" shows how many trials out of 3 resulted in an improved solution compared to the initial solution. The column "init obj value" states the cost of initial solution. As Table (2.3) demonstrates, we obtained improved results for only 3 instances belonging to the Conflict Index group. Similarly, Tables (2.4) and (2.5) summarize the results obtained by using RandLS-Sim solution as the initial solution for RandLS-Tabu. RandLS-Tabu was able to improve the initial solution in all 14 instances. This seems to indicate that RandLS-Tabu can potentially reach better solutions in the search space than RandLS-Sim.

For the same class of microarray test problems, we performed 3 trials of RandLS-Sim and RandLS-Tabu using the solution reported in the website [3] as the initial solution. Each algorithm is run 3 times and the results are presented in Tables (2.6) and (2.7). Empty entries indicate that the algorithm did not improve the initial solution. Since the solutions

reported on [3] are good quality local minima, the purpose of this experiment was to assess the ability of our algorithm to get out of deep local minima to achieve better solutions.

In addition to microarray test problems, we tested RandLS-Sim and RandLS-Tabu against QAPLIB instances [33]. Out of the 135 test problems, 86 were solved to optimality as of March 2010. The remaining 49 had best known solutions reported by various algorithms. Out of the 135 test problems RandLS-Sim obtained optimal solutions for 77 problems and matched the best known solution value for 24 problems. Likewise, RandLS-Tabu produced optimal solutions for 81 problems and matched the best known solution for 23 problems. RandLS-Sim and RandLS-Tabu together matched 111 of 135 QAPLIB optimal or best known solutions. The best known solution to the non-optimal problems is due to many different algorithms. Of these algorithms, Robust Tabu Search [115], Genetic Hybrid [50], GRASP [88] and Simulated Annealing [35] appear more often as the first heuristic to reach the best known solution. Other algorithms may reach the same solution as well but QAPLIB only recognizes the first such algorithm. It may be noted that no known algorithms reportedly produced uniformly better solutions for all problems.

This raises the question of how our algorithms RandLS-Sim and RandLS-Tabu compare to the 4 algorithms mentioned above in the instances not proven to be optimal. Table (2.8) gives a comparison of these algorithms and ours in these "non-optimal" instances. A checkmark in the first 4 columns indicates which method first reported the best known solution to that problem as recognized by QAPLIB. A "Y" or "N" indicates whether the algorithm matches the best known solution. Blank entries indicate that the authors did not report a solution in their paper.

In addition to the QAPLIB and microarray benchmark instances, one can use QAP instance generators with known optimal solutions. Two such generators are developed by Palubeckis [96], Li and Pardalos [86].

In summary, our RandLS-Tabu algorithm obtained an improved solution for the largest instance of the Conflict Index class arising from microarray studies. Our solution and the time taken by the algorithm are reported in Appendix 1.

**Algorithm 2.4:** RandLS-Tabu-QAP

**Input**: $k$: Degree of freedom; $iter(k)$: Iteration limit per degree of freedom;
count-limit: Total number of iterations;
**Output**: The best solution found and its value

$\pi \leftarrow$ A random permutation of $\{1, 2, \ldots, n\}$;
Compute $f(\pi)$;
solution $\leftarrow \pi$;
value $\leftarrow f(\pi)$;
count $\leftarrow 0$;
**repeat**
  **while** $\pi$ *is not a local minimum* **do**
    $\Delta(\pi, p, q) \leftarrow \min\{\Delta(\pi, r, s) \mid 1 \leq r, s \leq n\}$;
    $\pi \leftarrow$ The permutation obtained from $\pi$ by interchanging locations $p$ and $q$;
    count $\leftarrow$ count+1;
  **end**
  **if** *value* $> f(\pi)$ **then** solution $\leftarrow \pi$; value $\leftarrow f(\pi)$;
  $i \leftarrow 1$;
  **repeat**
    $j \leftarrow$ A random integer in $[1, k]$;
    $\Delta(\pi, u, v) \leftarrow j^{\text{th}}$ best element of $\{\Delta(\pi, r, s) \mid 1 \leq r, s \leq n\}$;
    $\pi \leftarrow$ The permutation obtained from $\pi$ by interchanging locations $u$ and $v$;
    $i \leftarrow i + 1$;
    $\Delta(\pi, p, q) \leftarrow \min\{\Delta(\pi, r, s) \mid 1 \leq r, s \leq n\}$;
    $\sigma \leftarrow$ The permutation obtained from $\pi$ by interchanging locations $p$ and $q$;
    **if** *value* $> f(\sigma)$ **then**
      solution $\leftarrow \sigma$;
      $\pi \leftarrow \sigma$;
      value $\leftarrow f(\sigma)$;
      $i \leftarrow iter(k)$;
    **end**
    count $\leftarrow$ count+1;
  **until** $i = iter(k)$;
**until** *count* $\geq$ *count-limit*;
output solution and value;

Table 2.1: Border Length Instances

| | | RandLS-Sim | | | | RandLS-Tabu | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Obj Value | | Time | | Obj Value | | Time | |
| size | best known | avg | best | avg | best | avg | best | avg | best |
| 36 | **3296** | 3329 | 3320 | 26 | 26 | 3310 | 3304 | 226 | 226 |
| 49 | **4548** | 4610 | 4592 | 67 | 68 | 4582 | 4576 | 437 | 436 |
| 64 | **5988** | 6078 | 6056 | 156 | 156 | 6043 | 6024 | 746 | 745 |
| 81 | **7536** | 7661 | 7648 | 323 | 321 | 7605 | 7580 | 1210 | 1210 |
| 100 | **9272** | 9439 | 9428 | 617 | 613 | 9361 | 9336 | 1863 | 1865 |
| 121 | **11412** | 11654 | 11636 | 1098 | 1095 | 11535 | 11516 | 2757 | 2754 |
| 144 | **13472** | 13795 | 13760 | 1892 | 1881 | 13627 | 13616 | 3957 | 3951 |

Table 2.2: Conflict Index Instances

| | | RandLS-Sim | | | | RandLS-Tabu | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Obj Value | | Time | | Obj Value | | Time | |
| size | best known | avg | best | avg | best | avg | best | avg | best |
| 36 | **168,611,971** | 168,852,954 | 168,734,932 | 25 | 25 | 168,611,971 | 168,611,971 | 221 | 219 |
| 49 | **236,355,034** | 236,944,124 | 236,618,785 | 67 | 66 | 236,363,247 | 236,355,034 | 423 | 419 |
| 64 | **325,671,035** | 326,523,031 | 326,177,307 | 157 | 156 | 326,170,025 | 325,671,035 | 738 | 745 |
| 81 | **427,447,820** | 428,694,429 | 428,430,586 | 318 | 322 | 429,543,607 | 428,581,964 | 1203 | 1220 |
| 100 | **523,146,366** | 524,292,574 | 523,552,203 | 612 | 611 | 525,605,503 | 524,010,774 | 1848 | 1842 |
| 121 | **653,416,978** | 656,100,369 | 655,038,924 | 1095 | 1113 | 660,817,156 | 657,759,770 | 2751 | 2769 |
| 144 | **795,009,899** | 798,914,435 | 798,326,692 | 1873 | 1888 | 806,111,189 | 803,651,048 | 4029 | 3887 |

Table 2.3: RandLS-Sim Algorithm with initial solution as RandLS-Tabu output

| | | Obj Value | | Time | | Improved |
|---|---|---|---|---|---|---|
| size | init obj value | avg | best | avg | best | runs |
| 81 | 428,581,964 | 428,505,992 | 428,438,869 | 317 | 316 | 2 |
| 121 | 663,372,061 | 656,340,788 | 656,187,895 | 1091 | 1087 | 3 |
| 144 | 805,982,330 | 798,411,847 | 797,776,125 | 1876 | 1873 | 3 |

Table 2.4: RandLS-Tabu with initial solution as RandLS-Sim output - Border Length instances

| | | Obj Value | | Time | | Improved |
|---|---|---|---|---|---|---|
| size | init obj value | avg | best | avg | best | runs |
| 36 | 3340 | 3307 | 3304 | 229 | 228 | 3 |
| 49 | 4592 | 4584 | 4580 | 428 | 428 | 2 |
| 64 | 6084 | 6031 | 6020 | 759 | 763 | 3 |
| 81 | 7660 | 7611 | 7600 | 1234 | 1231 | 3 |
| 100 | 9432 | 9361 | 9340 | 1899 | 1896 | 3 |
| 121 | 11656 | 11551 | 11536 | 2816 | 2800 | 3 |
| 144 | 13800 | 13639 | 13620 | 4010 | 3986 | 3 |

Table 2.5: RandLS-Tabu with initial solution as RandLS-Sim output - Conflict Index instances

| | | Obj Value | | Time | | Improved |
|---|---|---|---|---|---|---|
| size | init obj value | avg | best | avg | best | runs |
| 36 | 168,889,122 | 168,611,971 | 168,611,971 | 228 | 225 | 3 |
| 49 | 237,198,653 | 236,365,581 | 236,355,034 | 427 | 426 | 3 |
| 64 | 326,252,430 | 326,039,320 | 326,039,320 | 751 | 747 | 3 |
| 81 | 428,430,586 | 428,147,875 | 428,129,252 | 1232 | 1225 | 3 |
| 100 | 524,742,033 | 524,175,251 | 524,035,169 | 1885 | 1881 | 3 |
| 121 | 655,038,924 | 654,464,810 | 654,416,694 | 2763 | 2762 | 3 |
| 144 | 798,326,692 | 797,383,203 | 797,187,445 | 3979 | 3986 | 3 |

Table 2.6: Border Length Instances - initial solution from website [3]

| | | RandLS-Sim | | | | | RandLS-Tabu | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Obj Value | | time | | Improved | Obj Value | | time | | Improved |
| size | best known | avg | best | avg | best | runs | avg | best | avg | best | runs |
| 64 | 6048 | | | | | 0 | 6035 | 6024 | 762 | 763 | 3 |
| 81 | 7644 | | | | | 0 | 7613 | 7612 | 1239 | 1235 | 3 |
| 100 | 9432 | | | | | 0 | 9367 | 9348 | 1894 | 1893 | 3 |
| 121 | 11640 | 11624 | 11624 | 1110 | 1110 | 1 | 11543 | 11524 | 2800 | 2807 | 3 |
| 144 | 13832 | 13769 | 13740 | 1893 | 1895 | 3 | 13619 | 13568 | 4011 | 4012 | 3 |

Table 2.7: Conflict Index Instances - initial solution from website [3]

| | | RandLS-Sim | | | | | RandLS-Tabu | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Obj Value | | time | | Improved | Obj Value | | time | | Improved |
| size | best known | avg | best | avg | best | runs | avg | best | avg | best | runs |
| 36 | 169,016,907 | 168,703,820 | 168,703,820 | 26 | 25 | 3 | 168,611,971 | 168,611,971 | 228 | 225 | 3 |
| 49 | 237,077,377 | 236,700,827 | 236,386,676 | 67 | 67 | 3 | 236,478,882 | 236,355,034 | 433 | 430 | 3 |
| 64 | 326,696,412 | 326,363,496 | 326,121,515 | 156 | 156 | 3 | 326,166,312 | 326,077,701 | 757 | 758 | 3 |
| 81 | 428,682,120 | 428,424,248 | 428,424,248 | 317 | 317 | 3 | 427,720,768 | 427,501,773 | 1233 | 1225 | 3 |
| 100 | 525,401,670 | 525,143,833 | 525,067,906 | 612 | 612 | 3 | 523,926,636 | 523,807,971 | 1881 | 1869 | 3 |
| 121 | 658,317,466 | 656,761,677 | 656,006,176 | 1094 | 1094 | 3 | 655,520,202 | 654,993,828 | 2794 | 2768 | 3 |
| 144 | 803,379,686 | 797,963,331 | 797,760,823 | 1882 | 1882 | 3 | 800,177,700 | 799,908,525 | 3996 | 4021 | 3 |

Table 2.8: Comparison of algorithms used for QAPLIB instances

| name | size | Ro-TS | GEN | GRASP | SIM | RandLS-Sim | RandLS-Tabu |
|---|---|---|---|---|---|---|---|
| bur26b | 26 | | | ✓ | | Y | Y |
| bur26c | 26 | | | ✓ | | Y | Y |
| bur26d | 26 | | | ✓ | | Y | Y |
| bur26e | 26 | | | ✓ | | Y | Y |
| bur26f | 26 | | | ✓ | | Y | Y |
| bur26g | 26 | | | ✓ | | Y | Y |
| bur26h | 26 | | | ✓ | | Y | Y |
| esc32a | 32 | ✓ | | Y | | Y | Y |
| esc32b | 32 | ✓ | | Y | | Y | Y |
| esc32c | 32 | | | Y | ✓ | Y | Y |
| esc32d | 32 | ✓ | | Y | | Y | Y |
| esc32h | 32 | ✓ | | Y | | Y | Y |
| esc64a | 64 | | | Y | ✓ | Y | Y |
| esc128 | 128 | | | ✓ | | Y | Y |
| sko42 | 42 | ✓ | | | N | Y | Y |
| sko49 | 49 | ✓ | | | N | N | Y |
| sko56 | 56 | ✓ | | | N | Y | Y |
| sko64 | 64 | ✓ | | | N | Y | Y |
| sko72 | 72 | ✓ | | | N | N | N |
| sko81 | 81 | | ✓ | | N | Y | N |
| sko90 | 90 | ✓ | | | N | N | N |
| sko100a | 100 | | ✓ | | | N | N |
| sko100b | 100 | | ✓ | | | N | N |
| sko100c | 100 | | ✓ | | | N | N |
| sko100d | 100 | | ✓ | | | N | N |
| sko100e | 100 | | ✓ | | | N | N |
| sko100f | 100 | | ✓ | | | N | N |
| tai25a | 25 | ✓ | | | | N | N |
| tai30a | 30 | ✓ | | | | N | Y |
| tai30b | 30 | ✓ | | | | Y | N |
| tai35a | 35 | ✓ | | | | N | Y |
| tai35b | 35 | ✓ | | | | Y | N |
| tai40b | 40 | ✓ | | | | Y | N |
| tai50a | 50 | | ✓ | | | N | N |
| tai50b | 50 | ✓ | | | | N | N |
| tai60a | 60 | ✓ | N | | | N | N |
| tai60b | 60 | ✓ | | | | N | N |
| tai64c | 64 | ✓ | | | | Y | Y |
| tai80a | 80 | ✓ | N | | | N | N |
| tai80b | 80 | ✓ | | | | N | N |
| tai100b | 100 | ✓ | | | | N | N |
| tai150b | 150 | | ✓ | | | N | N |
| tho40 | 40 | | | | ✓ | N | N |
| tho150 | 150 | | | | ✓ | N | N |
| wil50 | 50 | Y | | | ✓ | Y | Y |
| wil100 | 100 | N | ✓ | | N | Y | Y |

# Chapter 3

# Theory and Algorithms on Three Dimensional Assignment Problem

The general 3AP, as discussed in Chapter 1, is NP-hard to solve to optimality and approximate. Some polynomially solvable special cases were discussed in Chapter 1 as well as approximation algorithms for special cases. The theoretical contents of this chapter include new special cases and approximation algorithms for special cases. The most notable special case is the *constant 3AP* where any feasible solution is optimal. We provide necessary and sufficient conditions for an instance to be a constant 3AP and extend the result to MAP. We then present heuristic algorithms for the general 3AP in order to obtain near optimal solutions in reasonable time and present computational results. Two of our three algorithms outperform the most successful heuristics in the literature in terms of solution quality.

## 3.1 Polynomially Solvable Special Cases

In this section we introduce new special cases of 3AP which can be solved in polynomial time.

### 3.1.1 Constant 3AP

Let $G = K_n$ be the complete graph on $n$ nodes. Associate a cost $c_{ij}$ for each edge $(i, j)$ of $G$. A Hamiltonian cycle (tour) $H$ of $G$ is a cycle that visits all the nodes in $G$ exactly once. The cost of a tour $H$ is given by $c(H) = \sum_{(i,j)\in H} c_{ij}$. The Traveling Salesman Problem

(TSP) is the problem of finding the tour with minimum length.

Let $\pi_n^k$ denote the set of permutations on $n$ constituting the product of $k$ independent cycles. Each Hamiltonian tour $H$ is an element of $\pi_n^1$. The TSP can be formulated as

$$\min_{p \in \pi_n^1} \sum_{i=1}^{n} c_{ip(i)} \tag{3.1}$$

Essentially TSP is a constrained version of AP where we only consider a specific subset of $\pi_n$, namely the cycles of length $n$.

**Definition 3.1.** *The Traveling Salesman Problem (TSP) is called constant if every tour $\pi \in \pi_n^1$ has the same cost.*

**Theorem 3.1.** *(Gabovich [55]) The TSP is constant if and only if there exist real numbers $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ such that the cost matrix satisfies $c_{ij} = a_i + b_j$ for all $i, j = 1, 2, \ldots, n$.*

It is easy to verify the claim when $c_{ij} = a_i + b_j$. In this case every solution has a cost equal to $\sum_{i=1}^{n}(a_i + b_i)$. Gabovich proved the converse that if every solution has the same cost, then it must be possible to break up the costs $c_{ij}$ as a sum.

**Corollary 3.2.** *The AP is constant if and only if there exist real numbers $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ such that the cost matrix satisfies $c_{ij} = a_i + b_j$ for all $i, j = 1, 2, \ldots, n$.*

*Proof.* Since all permutations $\pi \in \pi_n$ have the same cost in AP, so do all permutations $\pi \in \pi_n^1$. Considering only the permutations $\pi \in \pi_n^1$ reduces the AP to an instance of TSP. Apply Theorem (3.1). □

We next present an alternative proof of Corollary (3.2) which will be extended to higher dimensions later in this chapter. The following definition will prove useful for our discussion.

**Definition 3.2.** *Given two vectors $(i_1, j_1)$ and $(i_2, j_2)$ in AP, a 2-exchange on these two vectors results in the vectors $(i_2, j_1)$ and $(i_1, j_2)$.*

*Proof.* If $c_{ij} = a_i + b_j$ for all $i, j = 1, 2, \ldots, n$, then any feasible solution has a cost equal to $\sum_{i=1}^{n}(a_i + b_i)$. For the converse, given an instance $AP$ and the cost matrix $C$, define a new problem $AP'$ and a matrix $C'$ such that

$$c'_{ij} = c_{ij} - c_{in} - c_{nj} \tag{3.2}$$

Note that the above transformation does not change the optimal solution of problems $AP$ and $AP'$ in the sense that any optimal solution to $AP$ is optimal for $AP'$ and vice-versa, although the transformation may change the optimal cost.

If at least one index is equal to $n$, then equation (3.2) yields the following results

$$\begin{aligned} c'_{in} &= c_{in} - c_{in} - c_{nn} &&= -c_{nn} \\ c'_{nj} &= c_{nj} - c_{nn} - c_{nj} &&= -c_{nn} \\ c'_{nn} &= c_{nn} - c_{nn} - c_{nn} &&= -c_{nn} \end{aligned} \tag{3.3}$$

Hence if $i, j$ or both are equal to $n$, then $c'_{ij} = -c_{nn}$. We next prove that if $i, j \neq n$, then $c'_{ij} = -c_{nn}$ as well.

Consider any edge $(i, j)$ where $i, j \neq n$ (the case where $i$ and $j$ are equal is allowed). Let $S$ be a solution of $AP'$ where the edges $(i, j), (n, n) \in S$. By Definition (3.2), a 2-exchange of the two vectors $(i, j)$ and $(n, n)$ produces the vectors $(n, j)$ and $(i, n)$. Since any solution to $AP'$ is optimal and this 2-exchange does not affect other vectors in $S$, we have

$$\begin{aligned} c'_{ij} + c'_{nn} &= c'_{nj} + c'_{in} \\ c'_{ij} - c_{nn} &= -c_{nn} - c_{nn} \qquad \text{by (3.3)} \\ c'_{ij} &= -c_{nn} \end{aligned}$$

Therefore, combining this result with equation (3.3) shows that

$$c'_{ij} = -c_{nn} \quad \text{for all } i, j = 1, \ldots, n \tag{3.4}$$

We next construct the numbers $a_i, b_i$ for $i = 1, \ldots, n$. Let $a_n = 0$ and $b_n = c_{nn}$. Compute the real numbers $a_1, \ldots, a_{n-1}$ by letting

$$\begin{aligned} c_{1n} &= a_1 + b_n \\ c_{2n} &= a_2 + b_n \\ &\vdots \\ c_{n-1,n} &= a_{n-1} + b_n \end{aligned} \tag{3.5}$$

Compute the real numbers $b_1, \ldots, b_{n-1}$ by letting

$$\begin{aligned} c_{n1} &= b_1 \\ c_{n2} &= b_2 \\ &\vdots \\ c_{n,n-1} &= b_{n-1} \end{aligned} \tag{3.6}$$

We next prove that the seqeunces $A$ and $B$ obtained above satisfy $c_{ij} = a_i + b_j$ for all $i, j = 1, \ldots, n$. Consider

$$\begin{array}{rcll} c'_{ij} & = & c_{ij} - c_{in} - c_{nj} & \text{by (3.2)} \\ -c_{nn} & = & c_{ij} - (a_i + b_n) - b_j & \text{by (3.4), (3.5) and (3.6)} \\ -b_n & = & c_{ij} - a_i - b_n - b_j & \\ c_{ij} & = & a_i + b_j & \text{for all } i, j = 1, \ldots, n \end{array}$$

□

We now extend Corollary (3.2) to 3AP. We first define the notion of 2-exchange in 3AP.

**Definition 3.3.** *Given two vectors $(i_1, j_1, k_1)$ and $(i_2, j_2, k_2)$ in 3AP, a 2-exchange can be obtained in three different ways:*

1. *A 2-exchange on the first coordinates leads to the vectors $(i_2, j_1, k_1)$ and $(i_1, j_2, k_2)$.*
2. *A 2-exchange on the second coordinates leads to the vectors $(i_1, j_2, k_1)$ and $(i_2, j_1, k_2)$.*
3. *A 2-exchange on the third coordinates leads to the vectors $(i_1, j_1, k_2)$ and $(i_2, j_2, k_1)$.*

**Theorem 3.3.** *The 3AP with $n \geq 3$ (i.e. $|K| \geq 3$) is constant if and only if there exist real numbers $a_1, a_2, \ldots, a_n$, $b_1, b_2, \ldots, b_n$ and $d_1, d_2, \ldots, d_n$ such that the cost matrix satisfies $c_{ijk} = a_i + b_j + d_k$ for all $i, j, k = 1, 2, \ldots, n$.*

*Proof.* If $c_{ijk} = a_i + b_j + d_k$ for all $i, j, k = 1, 2, \ldots, n$, then any feasible solution has a cost equal to $\sum_{i=1}^{n}(a_i + b_i + d_i)$. Hence any feasible solution is optimal. For the converse, given an instance $3AP$ and the cost matrix $C$, define a new problem $3AP'$ and a matrix $C'$ such that

$$c'_{ijk} = c_{ijk} - c_{inn} - c_{njn} - c_{nnk} \tag{3.7}$$

Note that the above transformation does not change the optimal solution of problems $3AP$ and $3AP'$ in the sense that any optimal solution to $3AP$ is optimal for $3AP'$ and vice-versa, although the transformation may change the optimal cost.

If at least two indices are equal to $n$, then equation (3.7) yields the following results

$$\begin{array}{rcllll} c'_{inn} & = & c_{inn} - c_{inn} - c_{nnn} - c_{nnn} & = & -2c_{nnn} \\ c'_{njn} & = & c_{njn} - c_{nnn} - c_{njn} - c_{nnn} & = & -2c_{nnn} \\ c'_{nnk} & = & c_{nnk} - c_{nnn} - c_{nnn} - c_{nnk} & = & -2c_{nnn} \\ c'_{nnn} & = & c_{nnn} - c_{nnn} - c_{nnn} - c_{nnn} & = & -2c_{nnn} \end{array} \tag{3.8}$$

Hence if at least two of $i, j, k$ are equal to $n$, then $c'_{ijk} = -2c_{nnn}$.

Consider the case where no index is equal to $n$. Let $S_1$ be a solution of $3AP'$ where $(i, j, k), (n, n, n) \in S_1$ and $i, j, k \neq n$. By Definition (3.3), a 2-exchange of these two vectors on first coordinates produces the vectors $(n, j, k)$, and $(i, n, n)$. Since any solution to $3AP'$ is optimal and this 2-exchange does not affect other vectors in $S_1$, we have

$$\begin{aligned} c'_{ijk} + c'_{nnn} &= c'_{njk} + c'_{inn} \\ c'_{ijk} - 2c_{nnn} &= c'_{njk} - 2c_{nnn} \qquad \text{by (3.8)} \\ c'_{ijk} &= c'_{njk} \end{aligned}$$

Similarly, if given vectors $(i, j, k)$ and $(n, n, n)$ we perform a 2-exchange on second or third coordinates instead, then the result is

$$c_{ijk} = \begin{cases} c'_{ink} & \text{if 2-exchange on second coordinates} \\ c'_{ijn} & \text{if 2-exchange on third coordinates} \end{cases}$$

Therefore

$$c'_{ijk} = c'_{njk} = c'_{ink} = c'_{ijn} \quad \text{for all } i, j, k \neq n \tag{3.9}$$

Consider the case where exactly one index is equal to $n$. Let $S_2$ be a solution of $3AP'$ such that the vectors $(n, j, k), (i, n, v) \in S_2$, where $i, j, k, v \neq n$ and $v \neq k$. Such a $v \in K$ exists as $|K| \geq 3$ by the theorem assumption. Performing a 2-exchange on index $j$ produces the vectors $(n, n, k)$ and $(i, j, v)$. Since any solution to $3AP'$ is optimal and this 2-exchange does not affect other vectors in $S_2$, we have

$$c'_{inv} + c'_{njk} = c'_{nnk} + c'_{ijv} \tag{3.10}$$

Substituting $c'_{inv} = c'_{ijv}$ from equation (3.9) into (3.10) gives

$$c'_{njk} = c'_{nnk} = -2c_{nnn} \quad \text{by (3.8)} \tag{3.11}$$

Combining equations (3.8), (3.9) and (3.11) proves that

$$c'_{ijk} = -2c_{nnn} \quad \text{for all } i, j, k = 1, \ldots, n \tag{3.12}$$

Hence all entries in $C'$ are equal to $-2c_{nnn}$. We next construct the numbers $a_i, b_i, d_i$ for $i = 1, \ldots, n$. Let $a_n = b_n = 0$ and $d_n = c_{nnn}$. Compute the real numbers $a_1, \ldots, a_{n-1}$ by

letting

$$\begin{aligned} c_{1nn} &= a_1 + d_n \\ c_{2nn} &= a_2 + d_n \\ &\vdots \\ c_{n-1,n,n} &= a_{n-1} + d_n \end{aligned} \tag{3.13}$$

Compute the real numbers $b_1, \ldots, b_{n-1}$ by letting

$$\begin{aligned} c_{n1n} &= b_1 + d_n \\ c_{n2n} &= b_2 + d_n \\ &\vdots \\ c_{n,n-1,n} &= b_{n-1} + d_n \end{aligned} \tag{3.14}$$

Compute the real numbers $d_1, \ldots, d_{n-1}$ by letting

$$\begin{aligned} c_{nn1} &= d_1 \\ c_{nn2} &= d_2 \\ &\vdots \\ c_{n,n,n-1} &= d_{n-1} \end{aligned} \tag{3.15}$$

We next prove that the sequences $A, B$ and $D$ obtained above satisfy $c_{ijk} = a_i + b_j + d_k$ for all $i, j, k = 1, \ldots, n$. Consider

$$\begin{aligned} c'_{ijk} &= c_{ijk} - c_{inn} - c_{njn} - c_{nnk} && \text{by (3.7)} \\ -2c_{nnn} &= c_{ijk} - (a_i + d_n) - (b_j + d_n) - d_k && \text{by (3.12) to (3.15)} \\ -2d_n &= c_{ijk} - a_i - b_j - d_k - 2d_n \\ c_{ijk} &= a_i + b_j + d_k \end{aligned}$$

□

We remark that the condition $n \geq 3$ in Theorem (3.3) is necessary as we can provide a counter-example for $n = 2$. Consider the 3AP with $n = 2$ and cost matrix defined as

$$c_{ijk} = \begin{cases} 0 & \text{if 2 or 3 indices are equal to 2} \\ 1 & \text{if 0 or 1 indices are equal to 2} \end{cases}$$

All possible feasible solutions of the above 3AP can be listed as

$$(1,1,1) \quad \text{and} \quad (2,2,2) \tag{3.16}$$

$$(1,1,2) \quad \text{and} \quad (2,2,1) \tag{3.17}$$

$$(1,2,2) \quad \text{and} \quad (2,1,1) \tag{3.18}$$

$$(1,2,1) \quad \text{and} \quad (2,1,2) \tag{3.19}$$

It is an easy exercise to show that any feasible solution has cost equal to 1. If there exist $a_1, a_2, b_1, b_2, d_1, d_2$ such that $c_{ijk} = a_i + b_j + d_k$, then

$$c_{111} = a_1 + b_1 + d_1 \tag{3.20}$$

$$c_{112} = a_1 + b_1 + d_2 \tag{3.21}$$

$$c_{121} = a_1 + b_2 + d_1 \tag{3.22}$$

$$c_{211} = a_2 + b_1 + d_1 \tag{3.23}$$

Since $c_{111} = c_{112} = c_{121} = c_{211} = 1$, comparing equations (3.20) and (3.21) shows that $d_1 = d_2$. Similarly, comparing equations (3.20) and (3.22) leads to $b_1 = b_2$. Finally, equations (3.20) and (3.23) yield $a_1 = a_2$. Since $c_{222} = a_2 + b_2 + d_2$, we get $c_{222} = a_1 + b_1 + d_1 = c_{111}$, which is impossible since $c_{222} = 0$ but $c_{111} = 1$.

### 3.1.2 Constant MAP

In this section we generalize Theorem (3.3) to any MAP with $d, n \geq 3$, where $d$ is the dimension and $n$ is the number of elements in each independent set.

**Theorem 3.4.** *The MAP with $d \geq 3$ and $n \geq 3$ is constant if and only if there exist real sequences $a^{(1)}, a^{(2)}, \ldots, a^{(d)}$ such that the cost matrix satisfies $c_{i_1,i_2,\ldots,i_d} = a^{(1)}_{i_1} + a^{(2)}_{i_2} + \cdots + a^{(d)}_{i_d}$ for all $i_1, i_2, \ldots, i_d = 1, 2, \ldots, n$.*

*Proof.* If $c_{i_1,i_2,\ldots,i_d} = a^{(1)}_{i_1} + a^{(2)}_{i_2} + \cdots + a^{(d)}_{i_d}$ for all $i_1, i_2, \ldots, i_d = 1, 2, \ldots, n$, then any feasible solution has a cost equal to $\sum_{i=1}^{n}(a^{(1)}_i + a^{(2)}_i + \cdots + a^{(d)}_i)$. For the converse, given an instance $MAP$ and the cost matrix $C$, define a new problem $MAP'$ and a matrix $C'$ such that

$$c'_{i_1,i_2,\ldots,i_d} = c_{i_1,i_2,\ldots,i_d} - c_{i_1,n,\ldots,n} - c_{n,i_2,n,\ldots,n} - \cdots - c_{n,\ldots,n,i_d} \tag{3.24}$$

Note that the above transformation does not change the optimal solution of problems $MAP$ and $MAP'$ in the sense that any optimal solution to $MAP$ is optimal for $MAP'$ and vice-versa, although the transformation may change the optimal cost.

If at least $d-1$ indices are equal to $n$, then equation (3.24) yields the following results

$$\begin{array}{rcll} c'_{i_1,n,\ldots,n} &=& c_{i_1,n,\ldots,n} - c_{i_1,n,\ldots,n} - \underbrace{c_{n,\ldots,n} - \cdots - c_{n,\ldots,n}}_{d-1 \text{ times}} &= -(d-1)c_{n,\ldots,n} \\ c'_{n,i_2,n,\ldots,n} &=& c_{n,i_2,n,\ldots,n} - c_{n,\ldots,n} - c_{n,i_2,n,\ldots,n} - \cdots - c_{n,\ldots,n} &= -(d-1)c_{n,\ldots,n} \\ &\vdots & & \\ c'_{n,\ldots,n,i_d} &=& c_{n,\ldots,n,i_d} - c_{n,\ldots,n} - \cdots - c_{n,\ldots,n} - c_{n,\ldots,n,i_d} &= -(d-1)c_{n,\ldots,n} \\ c'_{n,\ldots,n} &=& c_{n,\ldots,n} - c_{n,\ldots,n} - \cdots - c_{n,\ldots,n} &= -(d-1)c_{n,\ldots,n} \end{array} \tag{3.25}$$

Hence if at least $d-1$ indices are equal to $n$, then $c'_{i_1,i_2,\ldots,i_d} = -(d-1)c_{n,\ldots,n}$.

Consider the case where no index is equal to $n$. Let $S_1$ be a solution of $MAP'$ where the vectors $(i_1, i_2, \ldots, i_d), (n, \ldots, n) \in S_1$ and $i_1, i_2, \ldots, i_d \neq n$. A 2-exchange of these two vectors on first coordinates produces the vectors $(n, i_2, \ldots, i_d)$ and $(i_1, n, \ldots, n)$. Since any solution to $MAP'$ is optimal and this 2-exchange does not affect other vectors in $S_1$, we have

$$\begin{array}{rcll} c'_{i_1,i_2,\ldots,i_d} + c'_{n,\ldots,n} &=& c'_{n,i_2,\ldots,i_d} + c'_{i_1,n,\ldots,n} & \\ c'_{i_1,i_2,\ldots,i_d} - (d-1)c_{n,\ldots,n} &=& c'_{n,i_2,\ldots,i_d} - (d-1)c_{n,\ldots,n} & \text{by (3.25)} \\ c'_{i_1,i_2,\ldots,i_d} &=& c'_{n,i_2,\ldots,i_d} & \end{array}$$

Similarly, by performing different 2-exchanges on $(i_1, i_2, \ldots, i_d)$ and $(n, \ldots, n)$ we obtain

$$c'_{i_1,i_2,\ldots,i_d} = c'_{n,i_2,\ldots,i_d} = c'_{i_1,n,i_3,\ldots,i_d} = \cdots = c'_{i_1,i_2,\ldots i_{d-1},n} \quad \text{for all } i_1, \ldots, i_d \neq n \tag{3.26}$$

Suppose the $r$-th coordinate of an element $c'_{i_1,i_2,\ldots,i_r,\ldots,i_d}$ is not equal to $n$ (i.e. $i_r \neq n$). We next prove that we can replace the $r$-th coordinate by $n$ so that

$$c'_{i_1,i_2,\ldots,i_{r-1},i_r,i_{r+1},\ldots,i_d} = c'_{i_1,i_2,\ldots,i_{r-1},n,i_{r+1},\ldots,i_d} \tag{3.27}$$

Let $I_1, I_2, \ldots, I_d$ denote the independent sets in $MAP$. Choose numbers $j_1, j_2, \ldots, j_d$ such that

$$\begin{array}{rcl} j_r &=& n \\ j_s &\in& I_s - \{i_s, n\} \quad \text{for } s = 1, \ldots, d \text{ and } s \neq r \end{array}$$

Each $j_s$ as defined above exists since $|I_s| \geq 3$. Let $S_2$ be a solution to $MAP'$ where

$$(i_1, i_2, \ldots, i_d), (j_1, j_2, \ldots, j_d) \in S$$

A 2-exchange of these two vectors on $r$-th coordinates produces the vectors

$$\begin{array}{l} (i_1, i_2, \ldots, i_{r-1}, n, i_{r+1}, \ldots, i_d) \quad \text{and} \\ (j_1, j_2, \ldots, j_{r-1}, i_r, j_{r+1}, \ldots, j_d) \end{array}$$

Since any solution to $MAP'$ is optimal and this 2-exchange does not affect other vectors in $S_2$, we have

$$\begin{aligned} & c'_{i_1,i_2,\ldots,i_{r-1},i_r,i_{r+1},\ldots,i_d} + c'_{j_1,j_2,\ldots,j_{r-1},n,j_{r+1},\ldots,j_d} \\ = \; & c'_{i_1,i_2,\ldots,i_{r-1},n,i_{r+1},\ldots,i_d} + c'_{j_1,j_2,\ldots,j_{r-1},i_r,j_{r+1},\ldots,j_d} \end{aligned} \tag{3.28}$$

Equation (3.26) indicates that in equation (3.28) we have

$$c'_{j_1,j_2,\ldots,j_{r-1},i_r,j_{r+1},\ldots,j_d} = c'_{j_1,j_2,\ldots,j_{r-1},n,j_{r+1},\ldots,j_d} \tag{3.29}$$

Therefore, substituting equation (3.29) in (3.28) leads to

$$c'_{i_1,i_2,\ldots,i_{r-1},i_r,i_{r+1},\ldots,i_d} = c'_{i_1,i_2,\ldots,i_{r-1},n,i_{r+1},\ldots,i_d}$$

which asserts the claim (3.27).

Equation (3.27) states that one can start with any element of $C'$ and iteratively replace the coordinates not equal to $n$ with $n$ in at most $d$ steps. Since the subsequent values are equal, we have

$$c'_{i_1,i_2,\ldots,i_d} = c'_{n,n,\ldots,n} = -(d-1)c_{n,\ldots,n} \quad \text{for all } i_1,\ldots,i_d = 1,\ldots,n \tag{3.30}$$

We next construct the sequences $a^{(1)}, a^{(2)}, \ldots, a^{(d)}$. Let $a_n^{(1)} = a_n^{(2)} = \cdots = a_n^{(d-1)} = 0$ and $a_n^{(d)} = c_{n,n,\ldots,n}$. For all $r = 1,\ldots,d-1$ and all $i_r = 1,\ldots,n$, compute the $r$-th sequence $a^{(r)}$ by letting

$$c_{n,\ldots,n,i_r,n,\ldots,n} = a_{i_r}^{(r)} + a_n^{(d)} \tag{3.31}$$

Compute $a^{(d)}$ by letting $a_{i_d}^{(d)} = c_{n,n,\ldots,n,i_d}$ for all $i_d = 1,\ldots,n$. It remains to show that $c_{i_1,i_2,\ldots,i_d} = a_{i_1}^{(1)} + a_{i_2}^{(2)} + \cdots + a_{i_d}^{(d)}$ for all $i_1, i_2, \ldots, i_d = 1, 2, \ldots, n$. Consider

$$\begin{aligned} c'_{i_1,i_2,\ldots,i_d} &= c_{i_1,i_2,\ldots,i_d} - c_{i_1,n,\ldots,n} - c_{n,i_2,n,\ldots,n} - \cdots - c_{n,n,\ldots,i_{d-1},n} - c_{n,\ldots,n,i_d} \\ -(d-1)c_{n,\ldots,n} &= c_{i_1,i_2,\ldots,i_d} - (a_{i_1}^{(1)} + a_n^{(d)}) - (a_{i_2}^{(2)} + a_n^{(d)}) - \cdots - (a_{i_{d-1}}^{(d-1)} + a_n^{(d)}) - a_{i_d}^{(d)} \\ -(d-1)c_{n,\ldots,n} &= c_{i_1,i_2,\ldots,i_d} - (a_{i_1}^{(1)} + a_{i_2}^{(2)} + \cdots + a_{i_d}^{(d)}) - (d-1)c_{n,\ldots,n} \\ c_{i_1,i_2,\ldots,i_d} &= a_{i_1}^{(1)} + a_{i_2}^{(2)} + \cdots + a_{i_d}^{(d)} \end{aligned}$$

□

### 3.1.3 Other Special Cases

Consider formulation (1.14) of 3AP. We may also reformulate this problem as

$$\min_{p,r\in\pi_n} \sum_{i=1}^{n} c_{i,p(i),r(p(i))} \tag{3.32}$$

The relationship between formulations (1.14) and (3.32) is that $q = r \circ p$. Note that $p$ is a permutation that maps $I$ to $J$, $r$ maps $J$ to $K$ and $q$ maps $I$ to $K$. The following theorem addresses the case where $c_{ijk} = a_i + b_j + d_{jk}$ for all $i, j, k = 1, \ldots, n$ and $a_i, b_j, d_{jk}$ are arbitrary numbers.

**Theorem 3.5.** *The 3AP cost matrix with $n \geq 3$ can be decomposed as $c_{ijk} = a_i + b_j + d_{jk}$ if and only if any specific permutation $r$ leads to solutions with the same cost regardless of $p$ (i.e. the cost of a solution depends only on the permutation $r$).*

*Proof.* Suppose that $c_{ijk} = a_i + b_j + d_{jk}$ for every $i, j, k = 1, \ldots, n$. Then for a fixed $r^* \in \pi_n$ and any $p \in \pi_n$ we have

$$\begin{aligned}
\sum_{i=1}^{n} c_{i,p(i),r^*(p(i))} &= \sum_{i=1}^{n} \left(a_i + b_{p(i)} + d_{p(i),r^*(p(i))}\right) && (3.33)\\
&= \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_{p(i)} + \sum_{i=1}^{n} d_{p(i),r^*(p(i))} && (3.34)\\
&= \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i + \sum_{i=1}^{n} d_{p(i),r^*(p(i))} && (3.35)\\
&= \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i + \sum_{i=1}^{n} d_{i,r^*(i)} && (3.36)
\end{aligned}$$

Equation (3.35) follows from (3.34) since $p$ is a permutation and omitting $p$ merely changes the order of terms in the summation. Similarly, equation (3.36) follows from (3.35). Note that given $r^*$, equation (3.36) is a constant regardless of $p$. This also shows that if $r^*$ is not fixed, then the optimal solution to 3AP reduces to solving the assignment problem in equation (3.36).

Conversely, suppose that any specific $r$ leads to solutions with the same cost regardless of $p$. Fix a permutation $r^*$ and reduce 3AP to an instance of AP with the cost matrix $e_{ij} = c_{ijr^*(j)}$ as per equation (1.34). Then the AP instance has the property that any permutation yields the same cost. Therefore, by Corollary (3.2), there exist real numbers

$a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ such that $e_{ij} = a_i + b_j$ for all $i, j = 1, \ldots, n$. Thus

$$c_{ijr^*(j)} = a_i + b_j \quad \text{for all } i, j = 1, \ldots, n \tag{3.37}$$

We next construct the elements $d_{jk}$. Equation (3.37) indicates that we may let $d_{jr^*(j)} = 0$ for all $j = 1, \ldots, n$. We obtain the remaining elements $d_{jk}$ by letting

$$c_{njk} = a_n + b_j + d_{jk} \quad \text{for all } j, k = 1, \ldots, n \tag{3.38}$$

Note that in equation (3.38) if $k = r^*(j)$, then $d_{jr^*(j)} = 0$ as expected. It remains to show that $c_{ijk} = a_i + b_j + d_{jk}$ for any $i \neq n$ and any $j, k = 1, 2, \ldots, n$. Choose an arbitrary vector $(i, j, k)$ such that $i \neq n$. We may assume that $k \neq r^*(j)$; otherwise equation (3.37) applies. Let $k = r^*(u)$ for some $u \in J - \{j\}$. Choose an arbitrary $j' \in J - \{j, u\}$. Such a $j'$ exists since $n \geq 3$ by assumption. Note that $k \neq r^*(j), r^*(j')$.

Let $S_1$ be a solution of 3AP where $(i, j, k), (n, j', r^*(j')) \in S_1$ and the permutation $r$ in $S_1$ for the remaining elements in $J$ is arbitrary. A 2-exchange of these two vectors on first coordinates produces the vectors $(n, j, k)$ and $(i, j', r^*(j'))$. Since the cost of a solution only depends on the permutation $r$ and this 2-exchange does not affect other vectors in $S_1$, we have

$$\begin{aligned} c_{ijk} + c_{nj'r^*(j')} &= c_{njk} + c_{ij'r^*(j')} \\ c_{ijk} + (a_n + b_{j'}) &= (a_n + b_j + d_{jk}) + (a_i + b_{j'}) \qquad \text{by (3.37) and (3.38)} \\ c_{ijk} &= a_i + b_j + d_{jk} \end{aligned}$$

□

We remark that the condition $n \geq 3$ in Theorem (3.5) is necessary as we can provide a counter-example for $n = 2$. Suppose that an instance of 3AP has the property that for a given permutation $r$, the cost of a solution is constant regardless of $p$. Choose arbitrary real numbers $a_1, b_1, d_{11}$ such that $c_{111} = a_1 + b_1 + d_{11}$. Let $b_2$ be an arbitrary number. Then compute $a_2, d_{22}, d_{12}$ and $d_{21}$ by following the equations below in the order specified.

$$\begin{aligned} c_{211} &= a_2 + b_1 + d_{11} \\ c_{122} &= a_1 + b_2 + d_{22} \\ c_{212} &= a_2 + b_1 + d_{12} \\ c_{221} &= a_2 + b_2 + d_{21} \end{aligned}$$

The four possible feasible solutions are listed in (3.16) to (3.19). Since equations (3.16) and (3.18) share the same $r$, they have the same solution cost. Therefore

$$\begin{aligned} c_{111} + c_{222} &= c_{122} + c_{211} \\ a_1 + b_1 + d_{11} + c_{222} &= a_1 + b_2 + d_{22} + a_2 + b_1 + d_{11} \\ c_{222} &= a_2 + b_2 + d_{22} \end{aligned}$$

This shows that $c_{222}$ must satisfy $c_{222} = a_2 + b_2 + d_{22}$ as expected. Consider the cost of solutions (3.17) and (3.19). Since they share the same $r$ we must have

$$\begin{aligned} c_{112} + c_{221} &= c_{121} + c_{212} \\ c_{112} + a_2 + b_2 + d_{21} &= c_{121} + a_2 + b_1 + d_{12} \\ c_{112} &= c_{121} + b_1 - b_2 + d_{12} - d_{21} \end{aligned} \tag{3.39}$$

Let $c_{121}$ be an arbitrary number such that $c_{121} \neq a_1 + b_2 + d_{21}$. Then as long as $c_{112}$ satisfies equation (3.39) the 3AP instance has the same cost for a given $r$ regardless of $p$.

The following two theorems can be proven by a similar argument as Theorem (3.5).

**Theorem 3.6.** *The 3AP cost matrix with $n \geq 3$ can be decomposed as $c_{ijk} = a_{ij} + b_j + d_k$ if and only if any specific permutation $p$ leads to solutions with the same cost regardless of $r$.*

**Theorem 3.7.** *The 3AP cost matrix with $n \geq 3$ can be decomposed as $c_{ijk} = a_i + b_j + d_{ik}$ if and only if any specific permutation $q$ leads to solutions with the same cost regardless of $p$.*

## 3.2 Approximation Algorithms

Consider the problems $T$ and $S$ in Section (1.8.2) where the cost matrix is defined by $t_{ijk}$ and $s_{ijk}$ respectively, where

$$\begin{aligned} t_{ijk} &= d_{ij} + d_{ik} + d_{jk} \\ s_{ijk} &= \min\{d_{ij} + d_{ik}, d_{ij} + d_{jk}, d_{ik} + d_{jk}\} \end{aligned}$$

If the distances satisfy the triangle inequality such that

$$d_{uv} \leq d_{uw} + d_{vw} \quad \forall\, u, v, w \in I \cup J \cup K \tag{3.40}$$

then the problems are referred to as $T\triangle$ and $S\triangle$. The triangle inequality (3.40) can be generalized to the *parameterized triangle inequality* by the following.

**Definition 3.4.** *For a parameter $\delta \geq \frac{1}{2}$, the parameterized triangle inequality is characterized by*

$$d_{uv} \leq \delta \left( d_{uw} + d_{vw} \right) \quad \forall\, u, v, w \in I \cup J \cup K \tag{3.41}$$

Note that $\delta = \frac{1}{2}$ forces $d_{uv}$ to be a constant for all $u, v \in I \cup J \cup K$ and $\delta < \frac{1}{2}$ is infeasible. If $\delta > 1$, then the problem is said to satisfy a *relaxed triangle inequality.*

We refer to problems $T\triangle$ and $S\triangle$ that satisfy the parameterized triangle inequality in (3.41) as $T\triangle_\delta$ and $S\triangle_\delta$ respectively and provide approximation algorithms whose ratio is a function of $\delta$.

The proposed algorithm exploits the fact that $t_{ijk} = d_{ij} + d_{jk} + d_{ik}$, where $i \in I, j \in J, k \in K$. The 3AP formulation (1.14) motivates us to let permutations $p, q$ and $r$ correspond to mappings from $I$ to $J$, $I$ to $K$ and $J$ to $K$ respectively as also described in Section (1.11.2). Note that $q = r \circ p$ and $r = q \circ p^{-1}$. Consider the following three separate problems.

$$\min_{p \in \pi_n} \sum_{i \in I} d_{ip(i)} \tag{3.42}$$

$$\min_{q \in \pi_n} \sum_{i \in I} d_{iq(i)} \tag{3.43}$$

$$\min_{r \in \pi_n} \sum_{j \in J} d_{jr(j)} \tag{3.44}$$

Consider the following algorithm.

**Algorithm 3.1:** Approximate $T\triangle_\delta$ and $S\triangle_\delta$

Let $A_1$ be the solution comprised of optimal $p$ and $r$ in (3.42) and (3.44).
Let $A_2$ be the solution comprised of optimal $p$ and $q$ in (3.42) and (3.43).
Let $A_3$ be the solution comprised of optimal $q$ and $r$ in (3.43) and (3.44).
Let *Approx* be the best solution among $A_1, A_2, A_3$.
Return *Approx* as the approximate solution.

**Theorem 3.8.** *The solution Approx from Algorithm (3.1) satisfies*

$$cost(Approx) \leq \frac{2}{3}(1 + \delta)\, cost(OPT)$$

*for any instance in $T\triangle_\delta$.*

*Proof.* Let $F$ denote an optimal solution of a given $T\triangle_\delta$ instance. By equation (1.20) we have

$$\begin{aligned}
cost(A_1) &= \sum_{(i,j,k)\in A_1} (d_{ij} + d_{jk} + d_{ik}) && (3.45)\\
&= \sum_{(i,j,k)\in A_1} d_{ij} + \sum_{(i,j,k)\in A_1} d_{jk} + \sum_{(i,j,k)\in A_1} d_{ik} && (3.46)\\
&\leq \sum_{(i,j,k)\in F} d_{ij} + \sum_{(i,j,k)\in F} d_{jk} + \sum_{(i,j,k)\in A_1} \delta\,(d_{ij} + d_{jk}) && (3.47)\\
&= \sum_{(i,j,k)\in F} d_{ij} + \sum_{(i,j,k)\in F} d_{jk} + \delta \sum_{(i,j,k)\in A_1} d_{ij} + \delta \sum_{(i,j,k)\in A_1} d_{jk} && (3.48)\\
&\leq \sum_{(i,j,k)\in F} d_{ij} + \sum_{(i,j,k)\in F} d_{jk} + \delta \sum_{(i,j,k)\in F} d_{ij} + \delta \sum_{(i,j,k)\in F} d_{jk} && (3.49)\\
&= (1+\delta) \sum_{(i,j,k)\in F} (d_{ij} + d_{jk}) && (3.50)
\end{aligned}$$

Equation (3.47) follows from (3.46) since $A_1$ contains the optimal $p$ and $r$. The third summand in (3.47) is due to (3.41). Similarly equation (3.49) follows from (3.48) since $A_1$ contains the optimal $p$ and $r$. Therefore, since $cost(Approx) \leq cost(A_1)$, we have

$$cost(Approx) \leq (1+\delta) \sum_{(i,j,k)\in F} (d_{ij} + d_{jk}) \qquad (3.51)$$

Similarly we have

$$\begin{aligned}
cost(A_2) &= \sum_{(i,j,k)\in A_2} (d_{ij} + d_{jk} + d_{ik})\\
&= \sum_{(i,j,k)\in A_2} d_{ij} + \sum_{(i,j,k)\in A_2} d_{jk} + \sum_{(i,j,k)\in A_2} d_{ik}\\
&\leq \sum_{(i,j,k)\in F} d_{ij} + \sum_{(i,j,k)\in A_2} \delta\,(d_{ij} + d_{ik}) + \sum_{(i,j,k)\in F} d_{ik}\\
&= \sum_{(i,j,k)\in F} d_{ij} + \delta \sum_{(i,j,k)\in A_2} d_{ij} + \delta \sum_{(i,j,k)\in A_2} d_{ik} + \sum_{(i,j,k)\in F} d_{ik}\\
&\leq \sum_{(i,j,k)\in F} d_{ij} + \delta \sum_{(i,j,k)\in F} d_{ij} + \delta \sum_{(i,j,k)\in F} d_{ik} + \sum_{(i,j,k)\in F} d_{ik}\\
&= (1+\delta) \sum_{(i,j,k)\in F} (d_{ij} + d_{ik})
\end{aligned}$$

and hence

$$cost(Approx) \leq (1+\delta) \sum_{(i,j,k)\in F} (d_{ij} + d_{ik}) \qquad (3.52)$$

A similar argument for $cost(A_3)$ shows that

$$cost(Approx) \leq (1+\delta) \sum_{(i,j,k)\in F} (d_{ik} + d_{jk}) \tag{3.53}$$

Combining inequalities (3.51), (3.52) and (3.53) shows that

$$\begin{aligned} 3\ cost(Approx) &\leq 2(1+\delta) \sum_{(i,j,k)\in F} (d_{ij} + d_{jk} + d_{ik}) \\ cost(Approx) &\leq \frac{2}{3}(1+\delta)\ cost(OPT) \end{aligned}$$

□

Note that if $\delta = 1$, then $cost(Approx) \leq \frac{4}{3} cost(OPT)$ as obtained by algorithm $H$ in Theorem (1.12). We can also prove that $cost(A_1) \leq (\delta + \frac{1}{2}) cost(OPT)$ by the same ideas in the proof of Theorem (3.8). However, we omit this since Theorem (3.8) is a stronger result. Solutions $A_2$ and $A_3$ exhibit the same approximation ratio as well and if $\delta = 1$, then $cost(A_1) \leq \frac{3}{2} cost(OPT)$ as obtained by Algorithm $H_{IJ}$ in Theorem (1.11).

**Theorem 3.9.** *The solution Approx from Algorithm (3.1) satisfies*

$$cost(Approx) \leq \frac{2}{3}(1+\delta)\ cost(OPT)$$

*for any instance in* $S\triangle_\delta$.

*Proof.* Let $F$ denote an optimal solution of a given $S\triangle_\delta$ instance. By equation (1.21) we have

$$\begin{aligned} cost(A_1) &= \sum_{(i,j,k)\in A_1} \min\{d_{ij}+d_{jk}, d_{ij}+d_{ik}, d_{jk}+d_{ik}\} \\ &\leq \sum_{(i,j,k)\in A_1} (d_{ij}+d_{jk}) && \text{by minimality of } s_{ijk} \\ &\leq \sum_{(i,j,k)\in F} (d_{ij}+d_{jk}) && \text{by optimality of } p, r \end{aligned}$$

Since $cost(Approx) \leq cost(A_1)$, we have

$$cost(Approx) \leq \sum_{(i,j,k)\in F} (d_{ij} + d_{jk}) \tag{3.54}$$

Similarly by considering $A_2$ and $A_3$ we can show that

$$cost(Approx) \leq \sum_{(i,j,k)\in F} (d_{ij} + d_{ik}) \tag{3.55}$$

$$cost(Approx) \leq \sum_{(i,j,k)\in F} (d_{jk} + d_{ik}) \tag{3.56}$$

Combining (3.54), (3.55) and (3.56) results in

$$cost(Approx) \leq \frac{2}{3} \sum_{(i,j,k)\in F} (d_{ij} + d_{jk} + d_{ik}) \tag{3.57}$$

Suppose that $s_{ijk} = d_{ij} + d_{jk}$ for some $(i, j, k) \in F$. Then

$$d_{ij} + d_{jk} + d_{ik} \leq d_{ij} + d_{jk} + \delta\,(d_{ij} + d_{jk}) = (1 + \delta)\, s_{ijk}$$

By a similar reason we can conclude that regardless of how $s_{ijk}$ is obtained we have

$$d_{ij} + d_{jk} + d_{ik} \leq (1 + \delta)\, s_{ijk} \tag{3.58}$$

Substituting (3.58) in (3.57) proves that

$$cost(Approx) \leq \frac{2}{3}(1 + \delta) \sum_{(i,j,k)\in F} s_{ijk}$$

and thus $cost(Approx) \leq \frac{2}{3}(1 + \delta)\ cost(OPT)$. □

We next discuss our proposed algorithm that builds on ideas from Algorithm (1.2).

## 3.3 Proposed Algorithms

As 3AP is NP-hard, numerous heuristic algorithms have been proposed in the literature. We propose three heuristic algorithms and compare them with two most successful heuristics; GRASP and Hybrid Genetic [10, 76]. Computational studies show that 2 of our 3 algorithms, namely Lagrangian Relaxation and LP Peeling, outperform GRASP and Hybrid Genetic in terms of solution quality in a reasonable time.

### 3.3.1 Fix Mapping

This method is derived from the generic iterative local search. In local search, starting with a feasible solution $x^0$, the algorithm explores the neighborhood of $x^0$, denoted $N(x^0)$, for an improving solution. If no improving solution is found, $x^0$ is a local optimum and the algorithm outputs this solution. If $N(x^0)$ contains an improving solution $x^1$, then $x^1$ takes the role of $x^0$ and the process is continued until a locally optimal solution is identified. In iterative local search we incorporate multiple re-starts. After finding a locally optimal solution, the algorithm moves to a new starting solution $x^0$ and local search is repeated until some termination criterion is satisfied.

In the case of 3AP, we can alternate fixing the three mappings of $I$ to $J$, $I$ to $K$ and $J$ to $K$ as described in Section (1.11.2). As a result, we will reduce the current 3AP feasible solution to an AP feasible solution using the same cost matrix as in equations (1.30), (1.32) and (1.34). Our approach, however, differs from that of hybrid genetic algorithm where the authors solve the resulting AP to optimality. In our approach, after reducing the 3AP solution to an AP solution, we merely strive to obtain a better AP solution rather than optimizing it. Obtaining an improved solution was implemented using the primal method by Balinski and Gomory [15]. Our algorithm will take longer to terminate, due to small improvements in each step, but experimental studies indicate that the solution quality will be better. A full description of the algorithm follows.

In line (1), we solve the LP relaxation using CPLEX. CPLEX is a software for solving mathematical programming problems including linear programming, integer programming and mixed integer programming. If $x_{ijk} = 1$ for some $i, j, k$, then we include the $(i, j, k)$ vector in $x^0$. Note that in this chapter we use the term "vector $(i, j, k)$" to denote $x_{ijk} = 1$. If we proceed in this way, $x^0$ may not be a complete solution. In order to make $x^0$ feasible we add the missing vectors by a greedy algorithm which is based on the following observation. If $x^0$ does not contain $n$ vectors, then there must be some $i \in I, j \in J, k \in K$ such that $i, j, k$ are not in any vectors of $x^0$. Consider all such $i \in I, j \in J, k \in K$ and the corresponding cost $c_{ijk}$. The next algorithm describes the procedure.

In line (4) the value of MAX is an input and is set to $MAX = 10$ in our experiments. Line (9) needs more explanation. Suppose we are given two vectors $(i_1, j_1, k_1)$ and $(i_2, j_2, k_2)$. A *2-exchange* on these two vectors can be done in three ways; switching $i$'s, $j$'s or $k$'s. Switching $i_1$ and $i_2$ generates the vectors $(i_2, j_1, k_1)$ and $(i_1, j_2, k_2)$. Switching $j_1$ and $j_2$

**Algorithm 3.2:** Fix Mapping

1: Get initial $x^0$ by LP relaxation
2: $x \leftarrow x^0$
3: $iteration \leftarrow 0$
4: **while** $iteration <$ MAX **do**
5: **repeat**
6: **repeat**
7: Alternate fixing the 3 mappings using equations (1.30), (1.32) and (1.34) and update $x$
8: **until** $x$ does not improve
9: Perform *3-exchange* on $x$ to escape local min
10: **until** $x$ does not improve
11: Save $x$ in elite solutions $E$
12: Diversify from $x$
13: $iteration \leftarrow iteration + 1$
14: **end while**
15: Run CPLEX on $E$ to get $x_{\text{best}}$

**Algorithm 3.3:** Greedy Initial

Let an incomplete solution $x^0$ be given.
**for** $i = 1$ to $n$ in $I$ **do**
**if** $i$ is not a vector in $x^0$ **then**
Find the minimum $c_{ijk}$ such that $j, k$ are not in vectors of $x^0$.
Add the vector $(i, j, k)$ to $x^0$.
**end if**
**end for**

yields the vectors $(i_1, j_2, k_1)$ and $(i_2, j_1, k_2)$. Switching $k_1$ and $k_2$ results in vectors $(i_1, j_1, k_2)$ and $(i_2, j_2, k_1)$.

**Proposition 3.10.** *If fixing the three mappings of $I$ to $J$, $I$ to $K$ and $J$ to $K$ does not improve the current solution (i.e. line (9) of Algorithm (3.2)), then none of the 2-exchanges of $i$'s, $j$'s or $k$'s above can improve the current solution.*

*Proof.* Switching $i$'s cannot improve the solution since after fixing the mapping of $J$ to $K$ the current AP solution is already optimal. Similarly, switching $j$'s or $k$'s is redundant since fixing the mapping of $I$ to $K$, or $I$ to $J$ results in an optimal AP solution. □

A 3-exchange involves a rearrangement of three vectors $(i_1, j_1, k_1), (i_2, j_2, k_2)$ and $(i_3, j_3, k_3)$.

A simple counting argument shows that there are $(3 \times 2)^2 = 36$ different arrangements for these three vectors. Some (not all) 3-exchanges can help the current solution escape local minimum in line (9) of Algorithm (3.2). After considering all 36 arrangements it is a trivial exercise to show that only 20 cases can improve the current solution since fixing the three mappings already considers 16 of those 3-exchanges. Given $x$, we consider all $\binom{n}{3}$ possible 3-exchanges and from the $20\binom{n}{3}$ resulting cases we perform the 3-exchange with the most improvement in the objective value of $x$.

If no improving 3-exchange can be found, we proceed to line (11) and add the current solution $x$ to the list $E$. The list $E$ contains all solutions from previous iterations. Since we set $MAX = 10$, we have $|E| \leq 10$. Line (15) solves a small subproblem of the original 3AP using the list $E$. The subproblem is constructed as follows. Instead of having all $n^3$ cost entries $c_{ijk}$, the subproblem only consists of $c_{ijk}$ for which $x_{ijk} = 1$ in some $x \in E$. Therefore, we only consider up to $10n$ cost entries and solve the corresponding integer program to optimality using CPLEX. The list $E$ simulates the "gene pool" in genetic algorithms and by solving the integer program we simulate all the gene pool operations. In other words, we combine all elite solutions in the gene pool to obtain the best one, $x_{\text{best}}$, which is the program output.

In line (12) we diversify from the current solution in an attempt to move away from the local minimum. We move to another region in the search space in a systematic way similar to Algorithm (2.3), i.e. the RandLS-Sim algorithm for QAP in Section 2.3.1 . The backbone of our diversification method is finding the 2-exchange neighborhood of the current solution $x$ and moving to a random member from the $k$ best members of the neighborhood, where $k$ is a prespecified parameter. Given two vectors $(i_1, j_1, k_1)$ and $(i_2, j_2, k_2)$ in $x$, we have three options for a 2-exchange; swapping $i$'s, $j$'s or $k$'s. Define

$$\begin{aligned} N_2(x) = &\{\text{2-exchanges of } x \text{ obtained by swapping two elements in} \\ &I, J \text{ or } K\} \end{aligned} \tag{3.59}$$

The following algorithm describes the diversification procedure. In line (2), $iter(k)$ is a prespecified parameter as per the RandLS-Sim method in Section 2.3.1. In line (5) the objective value of $x$ is denoted by $f(x)$ and $x_{\text{incumbent}}$ denotes the incumbent solution, which is the best solution encountered so far by Algorithm (3.2). If in the process of diversification the algorithm finds a better solution $y$ than $x_{\text{incumbent}}$, then we should save $y$ and exit diversification so that Algorithm (3.2) explores the newly found solution $y$. We

observed through computational experiments that appropriate values for $k$ and $iter(k)$ are 15 and 20 respectively.

**Algorithm 3.4:** 3AP-Diversification

```
1: i ← 1
2: while i < iter(k) do
3:    Find N_2(x) as described in Eqn. (3.59)
4:    y ← the best neighbor in N_2(x)
5:    if f(x_incumbent) > f(y) then
6:       x_incumbent ← y
7:       x ← y
8:       i ← iter(k)
9:    else
10:      j ← a random integer in [1, k]
11:      x ← j^th best neighbor in N_2(x)
12:      i ← i + 1
13:   end if
14: end while
```

### 3.3.2 Lagrangian Relaxation

In this section we propose a heuristic method that is based on Lagrangian relaxation. The algorithm reformulates the 3AP as a *Minimum Cost Flow* problem (MCF) with additional constraints and moves the constraints to the objective function using weights. The process of moving constraints to the objective function is referred to as Lagrangian relaxation. After solving the resulting MCF problem, we recover the corresponding 3AP solution, $x$. However, $x$ may not be a feasible 3AP solution as some constraints in the objective function of MCF may have been violated. We continually modify the weights in an attempt to satisfy more constraints and make $x$ feasible. As discussed next, if a solution to MCF is feasible for 3AP, then it is optimal for 3AP. In this situation, our heuristic method is able to prove the optimality of $x$ for 3AP, which is a potential feature of the algorithm. Heuristic methods are not generally capable of proving the optimality of a solution which gives our Lagrangian relaxation algrithm an added benefit.

On the other hand, finding the appropriate weights for constraints in order to force feasibility of the solution is not trivial. In fact, it is NP-hard. Otherwise one could formulate a Lagrangian relaxation of the problem and find the right weights to make the solution feasible

in polynomial time. This is where subgradient optimization techniques prove useful. A subgradient optimization algorithm addresses the problem of modifying the constraint weights in Lagrangian relaxation by generating a sequence of weights that eventually converge to a solution $\lambda$. Using $\lambda$ as constraint weights makes the solution feasible. There are, however, some practical issues associated with subgradient optimization such as convergence rate. At some point one has to stop the algorithm and convert the obtained solution to a feasible solution, which may prove to be difficult and not lead to an optimal solution. Therefore, we propose an alternative for subgradient optimization that tries to achieve the same goal by a different approach.

The Minimum Cost Flow problem is described as follows. Let $G = (N, A)$ be a directed network defined by a set $N$ of $n$ nodes and a set $A$ of $m$ directed arcs. Each arc $(i, j) \in A$ has three values associated with it: $c_{ij}, u_{ij}$ and $l_{ij}$. The value $c_{ij}$ denotes the cost per unit flow on the arc $(i, j)$. The values $u_{ij}$ and $l_{ij}$ denote the upper and lower capacities on each arc $(i, j)$ and represent the maximum and minimum amount that can flow on the arc. We associate with each node $i \in N$ an integer number $b(i)$ representing its supply or demand. If $b(i) > 0$, node $i$ is a supply node and if $b(i) < 0$, node $i$ is a demand node. A node with $b(i) = 0$ is a transshipment node. The decision variables $x_{ij}$ are arc flows and represent the amount of flow on arc $(i, j)$. The MCF is formulated as

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i) \quad \forall\, i \in N \\ & l_{ij} \le x_{ij} \le u_{ij} \qquad\qquad\qquad\qquad \forall\, (i,j) \in A \end{aligned} \tag{3.60}$$

where $\sum_{i=1}^{n} b(i) = 0$.

The transformation of 3AP to MCF is as follows. Given formulation (1.12), create the node set $N$ in MCF with $N = R \cup S \cup T$ such that

$$\begin{aligned} R &= \{r_1, r_2, \ldots, r_n\} \\ S &= \{s_1, s_2, \ldots, s_{n^2}\} \\ T &= \{r_1, r_2, \ldots, r_n\}. \end{aligned} \tag{3.61}$$

Create the arc set $A$ where

$$\begin{aligned} A &= R \times S \\ &\cup \{(s_j, t_1) : j = 1, \ldots, n\} \\ &\cup \{(s_j, t_2) : j = n+1, \ldots, 2n\} \\ &\vdots \\ &\cup \{(s_j, t_n) : j = (n-1)n+1, \ldots, n^2\} \end{aligned} \tag{3.62}$$

In other words, the edges between $R$ and $S$ form a complete bipartite graph with all arcs oriented from $R$ to $S$. For every $n$ nodes in $S$ there is only one adjacent node in $T$ with arcs oriented from $S$ to $T$. As a result, there are $n^3$ arcs from $R$ to $S$ and $n^2$ arcs from $S$ to $T$.

All lower capacities in this network are 0 and upper capacities are 1. The node supplies are

$$b(i) = \begin{cases} 1 & \text{if } i \in R \\ 0 & \text{if } i \in S \\ -1 & \text{if } i \in T \end{cases} \tag{3.63}$$

Before describing what the arc costs represent, we will demonstrate the purpose of the above network. The sets $R$ and $T$ in MCF correspond to the sets $I$ and $K$ in 3AP respectively. The supply of $+1$ for nodes in $R$ and $-1$ for nodes in $T$ enforces that every element in $R$ and $T$ appears in exactly one solution of 3AP. The set $J$ in 3AP corresponds to the set $S$ in MCF as follows

- node $j_1$ corresponds to nodes $s_1, s_{n+1}, s_{2n+1}, \ldots, s_{(n-1)n+1}$
- node $j_2$ corresponds to nodes $s_2, s_{n+2}, s_{2n+2}, \ldots, s_{(n-1)n+2}$
  $\vdots$
- node $j_n$ corresponds to nodes $s_n, s_{2n}, s_{3n}, \ldots, s_{n^2}$

We will refer to this as "expanding the set $J$". After solving the MCF problem, a solution to 3AP can be recovered by the following procedure, as also described by Algorithm (3.6). If $x_{r_i,s_j} = 1$ for some $r_i \in R$ and $s_j \in S$, since $s_j$ is a transshipment node with supply of zero, we must have $x_{s_j,t_k} = 1$ for some $t_k \in T$ (where $k = \lceil \frac{j}{n} \rceil$ to be exact). These two arc flows correspond to the vector $(i, j \bmod n, k)$ in 3AP if $j \bmod n \neq 0$, and $(i, n, k)$ if $j \bmod n = 0$. Hence every element of $I$ and $K$ appears in exactly one vector. The drawback is that there

may be elements of $J$ which appear in more than one vector. For example, for node $j_1$ in 3AP, more than one of $\{s_1, s_{n+1}, s_{2n+1}, \ldots, s_{(n-1)n+1}\}$ may be present in an MCF solution. We introduce additional constraints in MCF to avoid this problem. Consider

$$\begin{aligned} x_{s_1,t_1} + x_{s_{n+1},t_2} + x_{s_{2n+1},t_3} + \cdots + x_{s_{(n-1)n+1},t_n} &= 1 \\ x_{s_2,t_1} + x_{s_{n+2},t_2} + x_{s_{2n+2},t_3} + \cdots + x_{s_{(n-1)n+2},t_n} &= 1 \\ \vdots & \\ x_{s_n,t_1} + x_{s_{2n},t_2} + x_{s_{3n},t_3} + \cdots + x_{s_{n^2},t_n} &= 1 \end{aligned} \tag{3.64}$$

Each line in the equations above ensures that the corresponding $j \in J$ in 3AP is unique. We can summarize the constraints (3.64) in one line as follows

$$\sum_{k=1}^{n} x_{s_{(k-1)n+j},t_k} = 1 \qquad \forall\, j = 1, \ldots, n \tag{3.65}$$

The arc costs in the MCF formulation are denoted by matrix $D$ to avoid confusion with $C$ in 3AP. The relationship between them is

$$c_{ijk} = d_{r_i,s_{(k-1)n+j}} \tag{3.66}$$

and

$$d_{r_i,s_j} = \begin{cases} c_{i,j \bmod n,\lceil \frac{j}{n} \rceil} & \text{if } j \bmod n \neq 0 \\ c_{i,n,\lceil \frac{j}{n} \rceil} & \text{otherwise} \end{cases} \tag{3.67}$$

Also, $d_{s,t} = 0$ for every $s \in S, t \in T$ where $(s,t) \in A$. Therefore, 3AP can be formulated as MCF with equation (3.65) added as a constraint set to produce

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n^2} d_{r_i,s_j} x_{r_i,s_j} + \sum_{j=1}^{n} d_{s_j,t_1} x_{s_j,t_1} + \sum_{j=n+1}^{2n} d_{s_j,t_2} x_{s_j,t_2} \\ & + \cdots + \sum_{j=(n-1)n+1}^{n^2} d_{s_j,t_n} x_{s_j,t_n} \\ \text{s.t.} \quad & \sum_{j=1}^{n^2} x_{r_i,s_j} = 1 && \forall\, i = 1, \ldots, n \\ & x_{s_j,t_{\lceil \frac{j}{n} \rceil}} - \sum_{i=1}^{n} x_{r_i,s_j} = 1 && \forall\, j = 1, \ldots, n^2 \\ & \sum_{k=1}^{n} x_{s_{(k-1)n+j},t_k} = 1 && \forall\, j = 1, \ldots, n \\ & 0 \leq x_{ij} \leq 1 && \forall\, (i,j) \in A \end{aligned} \tag{3.68}$$

Our next task is to move the additional constraints (3.65) to the objective function using Lagrangian multipliers (i.e. weights) $m_1, m_2, \ldots, m_n$ as constants. Consider the objective function

$$\begin{aligned}\min \sum_{i=1}^{n}\sum_{j=1}^{n^2} d_{r_i,s_j}x_{r_i,s_j} + \sum_{j=1}^{n} d_{s_j,t_1}x_{s_j,t_1} + \sum_{j=n+1}^{2n} d_{s_j,t_2}x_{s_j,t_2} \\ + \cdots + \sum_{j=(n-1)n+1}^{n^2} d_{s_j,t_n}x_{s_j,t_n} + m_1\left(\sum_{k=1}^{n} x_{s_{(k-1)n+1},t_k} - 1\right) \\ + m_2\left(\sum_{k=1}^{n} x_{s_{(k-1)n+2},t_k} - 1\right) + \cdots + m_n\left(\sum_{k=1}^{n} x_{s_{(k-1)n+n},t_k} - 1\right)\end{aligned} \tag{3.69}$$

After multiplying the weights $m_1, m_2, \ldots, m_n$ through, the objective function will have the term $M = -m_1 - m_2 - \cdots - m_n$ at the end. Since $M$ is a constant, it can be removed from the objective function without affecting the solution $x$. The updated objective function is

$$\begin{aligned}\min \sum_{i=1}^{n}\sum_{j=1}^{n^2} d_{r_i,s_j}x_{r_i,s_j} + \sum_{j=1}^{n} d_{s_j,t_1}x_{s_j,t_1} + \sum_{j=n+1}^{2n} d_{s_j,t_2}x_{s_j,t_2} \\ + \cdots + \sum_{j=(n-1)n+1}^{n^2} d_{s_j,t_n}x_{s_j,t_n} + m_1\sum_{k=1}^{n} x_{s_{(k-1)n+1},t_k} \\ + m_2\sum_{k=1}^{n} x_{s_{(k-1)n+2},t_k} + \cdots + m_n\sum_{k=1}^{n} x_{s_{(k-1)n+n},t_k}\end{aligned} \tag{3.70}$$

Also, we can rearrange the constraints in the objective function so that the weights appear as part of the cost in the summation. Therefore, given the weights $m_1, m_2, \ldots, m_n$, we are finally able to formulate the MCF with Lagrangian relaxation.

$$
\begin{aligned}
\min \sum_{i=1}^{n}\sum_{j=1}^{n^2} d_{r_i,s_j}x_{r_i,s_j} + \sum_{j=1}^{n}(d_{s_j,t_1}+m_j)x_{s_j,t_1} + \sum_{j=n+1}^{2n}(d_{s_j,t_2}+m_{j-n})x_{s_j,t_2} \\
+\cdots+ \sum_{j=(n-1)n+1}^{n^2}(d_{s_j,t_n}+m_{j-(n-1)n})x_{s_j,t_n} \\
\text{s.t.} \sum_{j=1}^{n^2} x_{r_i,s_j} = 1 \qquad \forall\, i=1,\ldots,n \\
x_{s_j,t_{\lceil \frac{j}{n} \rceil}} - \sum_{i=1}^{n} x_{r_i,s_j} = 1 \qquad \forall\, j=1,\ldots,n^2 \\
0 \le x_{ij} \le 1 \qquad \forall\,(i,j)\in A
\end{aligned}
\tag{3.71}
$$

The MCF formulation above was based on "expanding the set $J$". We can derive a very similar network by expanding the set $I$ or $K$. As a result, there are three different ways of obtaining the MCF network. The only difference is that the cost matrix $D$ in equations (3.66) and (3.67) has to be set up accordingly, whereas equations (3.61), (3.62), (3.63), (3.65) and (3.68) apply to all three cases.

In order to expand the set $I$, let $R, S$ and $T$ correspond to $J, I$ and $K$ respectively. Define

$$c_{ijk} = d_{r_j,s_{(k-1)n+i}} \tag{3.72}$$

and

$$d_{r_i,s_j} = \begin{cases} c_{j \bmod n,i,\lceil \frac{j}{n} \rceil} & \text{if } j \bmod n \neq 0 \\ c_{n,i,\lceil \frac{j}{n} \rceil} & \text{otherwise} \end{cases} \tag{3.73}$$

Also, $d_{s,t} = 0$ for every $s \in S, t \in T$ where $(s,t) \in A$. After solving the MCF problem, the 3AP solution can be recovered as follows. Given $x_{r_i,s_j} = x_{s_j,t_k} = 1$ for some $r_i \in R, s_j \in S$ and $t_k \in T$ (where $k = \lceil \frac{j}{n} \rceil$), add the vector $(j \bmod n, i, k)$ in 3AP if $j \bmod n \neq 0$, and $(n,i,k)$ if $j \bmod n = 0$. This recovery procedure is described in Algorithm (3.6).

Similarly, in order to expand the set $K$, let $R, S$ and $T$ correspond to $I, K$ and $J$ respectively. Then

$$c_{ijk} = d_{r_i,s_{(j-1)n+k}} \tag{3.74}$$

and

$$d_{r_i,s_j} = \begin{cases} c_{i,\lceil \frac{j}{n} \rceil, j \bmod n} & \text{if } j \bmod n \neq 0 \\ c_{i,\lceil \frac{j}{n} \rceil, n} & \text{otherwise} \end{cases} \tag{3.75}$$

Also, $d_{s,t} = 0$ for every $s \in S, t \in T$ where $(s,t) \in A$. After solving the MCF problem, the 3AP solution can be recovered by Algorithm (3.6) as follows. Given $x_{r_i,s_j} = x_{s_j,t_k} = 1$ for some $r_i \in R, s_j \in S$ and $t_k \in T$ (where $k = \lceil \frac{j}{n} \rceil$), add the vector $(i, k, j \bmod n)$ in 3AP if $j \bmod n \neq 0$, and $(i, k, n)$ if $j \bmod n = 0$.

Expanding the sets $I, J$ and $K$ is not symmetric with respect to the MCF optimal solution. In other words, expanding $I$ or $J$ may lead to different objective values in the MCF formulation (3.71).

**Algorithm 3.5:** Lagrangian Relaxation

```
1:  for i = 1, ..., n do
2:     Initialize m_i ← 1000
3:  end for
4:  for i = 1, ..., MAX do
5:     Solve the MCF problem (3.71)
6:     Recover the 3AP solution, x, by Algorithm (3.6)
7:     if x is feasible for 3AP then
8:        x is optimal for 3AP
9:        Report optimality of x and exit
10:    end if
11:    for all n nodes j ∈ J do
12:       picked_j ← number of times nodes j appears in x
13:       if picked_j > 1 then
14:          weight_j ← weight_j + 2(picked_j − 1)
15:       else if picked_j = 0 and weight_j > 0 then
16:          weight_j ← weight_j − 1
17:       end if
18:    end for
19:    Convert x to a feasible solution by the greedy algorithm (3.7)
20:    Add x to elite solutions E
21: end for
22: Run CPLEX on E to get x_best
```

Algorithm (3.5) describes the Lagrangian relaxation using the MCF formulation. In line (2) we initialize the weights $m_i$ to 1000 so that we can decrease the weights in line (16) while keeping them non-negative. Line (4) uses the input parameter MAX to determine

how many Lagrangian relaxation problems we solve. The value of MAX depends on the problem size $n$, where $n = |I| = |J| = |K|$. Setting $MAX = 10n$ gave the best results in our experiments. The MCF solver used for solving the MCF problem was developed by Loebel [2], available for academic use free of charge. It uses a network simplex algorithm.

**Algorithm 3.6:** Recovering 3AP Solution from Lagrangian Relaxation

```
Let y denote the MCF solution
Initialize the 3AP solution x ← 0⃗
for i = 1, ..., n do
  Find the nodes r_i ∈ R, s_j ∈ S such that y_{r_i,s_j} = 1
  k ← ⌈j/n⌉
  if the set I was expanded then
    if j mod n ≠ 0 then
      x_{j mod n,i,k} ← 1
    else
      x_{n,i,k} ← 1
    end if
  else if the set J was expanded then
    if j mod n ≠ 0 then
      x_{i,j mod n,k} ← 1
    else
      x_{i,n,k} ← 1
    end if
  else
    if j mod n ≠ 0 then
      x_{i,k,j mod n} ← 1
    else
      x_{i,k,n} ← 1
    end if
  end if
end for
```

The following proposition proves why an optimal solution to (3.71) which is feasible for 3AP is also optimal for 3AP.

**Proposition 3.11.** *Consider the optimization problem*

$$\begin{aligned} \min \quad & L(x) = c^T x \\ s.t. \quad & Ax = b \\ & x \in D \end{aligned} \tag{3.76}$$

*where $D$ is some domain, $b$ is a vector and $x^1$ is an optimal solution. Consider the Lagrangian relaxation*

$$\begin{aligned} \min \quad & L(x,m) = c^T x + m^T(Ax - b) \\ s.t. \quad & x \in D \end{aligned} \tag{3.77}$$

*where $m$ is a given vector and $x^2$ is an optimal solution. Then*

$$L(x^2, m) \leq L(x^1)$$

*Furthermore, if $x^2$ is feasible for (3.76) then it is also optimal for (3.76).*

*Proof.* Any feasible solution to (3.76) is also feasible for (3.77) but (3.77) may consider more feasible solutions since the constraint $Ax = b$ is removed. Thus $L(x^2, m) \leq L(x^1)$. If, in addition, $x^2$ is feasible for (3.76), then $L(x^2, m) = L(x^2) = L(x^1)$ and $x^2$ is also optimal for (3.76). □

**Algorithm 3.7:** Greedy Conversion from Lagrangian to Feasible

Initialize $x' \leftarrow \vec{0}$
**for** all $j$ with $picked_j = 1$ **do**
  Find the vector $(i, j, k)$ with $x_{ijk} = 1$
  $x'_{ijk} \leftarrow 1$
  Remove $i$ from $I$, $j$ from $J$ and $k$ from $K$
**end for**
**for** all $j$ with $picked_j > 1$ **do**
  $L \leftarrow \{(i, j, k) \mid x_{ijk} = 1, i \in I, j \in J, k \in K\}$
  $(i, j, k) \leftarrow$ the vector in $L$ with minimum $c_{ijk}$
  $x'_{ijk} \leftarrow 1$
  Remove $i$ from $I$, $j$ from $J$ and $k$ from $K$
**end for**
**for** all $j$ with $picked_j = 0$ **do**
  $(i, j, k) \leftarrow$ the vector with minimum $c_{ijk}$ where $i \in I, j \in J, k \in K$
  $x'_{ijk} \leftarrow 1$
  Remove $i$ from $I$, $j$ from $J$ and $k$ from $K$
**end for**
$x \leftarrow x'$

The for loop in lines (11) to (18) modifies the weights $m_i$ in order to encourage $x$ to be feasible in the next MCF solution. If $picked_j = 1$ for some node $j \in J$, then the corresponding constraint in the objective function is satisfied and if $picked_j \neq 1$, then the

corresponding constraint is violated. In the case that $picked_j > 1$, increasing $m_j$ discourages multiple occurences of node $j$ by driving up the cost in the objective function. In line (13) node $j$ appears $(picked_j - 1)$ times more than allowed. It seems reasonable to take the value $(picked_j - 1)$ into consideration when increasing $m_j$. In fact our experiments indicate that an increase of $2(picked_j - 1)$ leads to better results. On the other hand, if $picked_j = 0$, decreasing $m_j$ (provided $m_j$ stays non-negative) encourages going through node $j$. Experiments showed that a decrease of 1 is more suitable.

Similar to Algorithm (3.2), line (20) adds $x$ to the list of elite solutions $E$ in order for line (22) to solve the 3AP subproblem by an exact integer solver such as CPLEX. Since the number of triangles in $E$ is at most $n \times MAX$ compared to $n^3$ in the original 3AP, finding an exact solution is very quick.

Modifying the weights of Lagrangian relaxation leads to different solutions where each solution may exhibit signs of being in an optimal solution. We predict that the method of generating elite solutions $E$ using Lagrangian relaxation and using an exact integer solver on $E$ can be successfully applied to other combinatorial optimization problems. We believe that this approach can be a powerful tool in obtaining good quality solutions to many NP-hard optimization problems. The choice of how to generate elite solutions $E$ may be problem dependent.

### 3.3.3 Linear Programming Peeling

This simple and efficient algorithm is based on solving the linear programming relaxation of 3AP and continually reducing the problem size by "peeling" (i.e. removing) the nodes $i \in I, j \in J, k \in K$ for which $x_{ijk} = 1$.

Consider the integer programming (IP) formulation (1.12). The associated linear programming (LP) relaxation is

$$\begin{array}{lll} \min & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} & \\ \text{s.t.} & \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 & \forall\, i = 1,\ldots,n \\ & \sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 & \forall\, j = 1,\ldots,n \\ & \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 & \forall\, k = 1,\ldots,n \\ & x_{ijk} \in [0,1] & \forall\, i,j,k = 1,\ldots,n \end{array} \tag{3.78}$$

If $IP^*$ and $LPR^*$ are the optimal solutions to the IP and LP relaxation respectively and $f$ is the objective function, we have $f(LPR^*) \leq f(IP^*)$ since $IP^*$ is a feasible solution to the LP relaxation as well.

A mixed integer program (MIP) is a combination of IP and LP where some variables only take on integer values and the remaining variables assume real values. An example of MIP for 3AP is where $x_{ijk} \in \{0,1\}$ for some variables and $x_{ijk} \in [0,1]$ for the remaining variables.

The LP Peeling algorithm deals with an MIP. We may assume that the original LP relaxation is an MIP with all real variables and no integer variables. Let

$$\begin{array}{lcl} S_{=1} & = & \{x_{ijk} \mid x_{ijk} = 1 \text{ in the MIP solution}\} \\ S_{<1} & = & \{x_{ijk} \mid x_{ijk} < 1 \text{ in the MIP solution}\} \end{array} \tag{3.79}$$

The algorithm forces some variables to be integer (i.e. either 0 or 1 due to the constraints) and some variables to be zero in the next MIP. Note that after peeling the current MIP, the remaining solution in $S_{<1}$ remains feasible for the new MIP and potentially optimal. In this case, $S_{=1} = \emptyset$ in the new MIP. In order to avoid this, we modify some variable types in MIP addition to removing some nodes. Define

$$\begin{array}{lcl} F_{int} & = & \{x_{ijk} \mid \text{the algorithm forces to be integer}\} \\ F_0 & = & \{x_{ijk} \mid \text{the algorithm forces to be zero}\} \end{array} \tag{3.80}$$

The size of the MIP throughout the algorithm is denoted by |MIP| and we define it to be equal to $|I|$ in the current MIP. Note that the peeling process removes equal number of

nodes from the sets $I, J$ and $K$ and hence $|I| = |J| = |K|$ in the MIP . Algorithm (3.8) below describes the LP Peeling algorithm.

**Algorithm 3.8:** LP Peeling

1: $F_{int} \leftarrow \emptyset$
2: $F_0 \leftarrow \emptyset$
3: MIP $\leftarrow$ LP relaxation of original 3AP
4: **while** |MIP| > inputSize **do**
5: Solve MIP using CPLEX
6: **if** $S_{=1} \neq \emptyset$ **then**
7: Remove all variables in $S_{=1}$ from MIP
8: $F_0 \leftarrow \emptyset$
9: **else**
10: $F_0 \leftarrow F_0 \cup F_{int}$
11: **end if**
12: $F_{int} \leftarrow$ 5 largest variables in $S_{<1}$
13: Apply $F_{int}, F_0$ to MIP
14: **end while**
15: Solve the remaining IP using CPLEX

In line (4) of the algorithm we set $inputSize = 4$. This is due to the size of benchmark instances used for testing the algorithm. These instances start with $n = 4, 6, 8, 10, \ldots$ and setting the value of inputSize any higher reduces the algorithm to solving those instances as an exact IP. For practical applications one could solve instances of size up to $n = 10$ by CPLEX at almost the same time as $n = 4$ in less than a second. Removing the variables in $S_{=1}$ in MIP requires removing the corresponding nodes in $I, J$ and $K$ from MIP and therefore reduces |MIP| by at least one. Line (13) forces the variables in $F_{int}$ to take on integer values and the variables in $F_0$ to be equal to zero in the new MIP.

## 3.4 Computational Results

In this section we present the results of algorithms (3.2), (3.5) and (3.8). These are the Fix Mapping, Lagrangian Relaxation and LP Peeling algorithms respectively. The experiments were implemented using C++ and conducted on the same machine as described in Section 2.4. There are three types of benchmark instances available in the literature: Balas and Saltzman instances, Burkard, Rudolf and Woeginger instances and instances due to Crama and Spieksma. They are available for download from the OR Library [4] under Quadratic

Assignment Problem. The OR library is a collection of operations research related benchmark instances.

Tables 1 to 5 below summarize the results obtained by the three algorithms Fix Mapping, Lagrangian Relaxation and LP Peeling. The columns named "Obj" refer to the objective value of the solution and the "Time" columns report the time in seconds taken by the algorithm. The bold entries indicate the best solution value among the five algorithms. We also ran CPLEX to solve the integer program to optimality. Interestingly, every solution reported by Lagrangian Relaxation turned out to be optimal. The entry "CPLEX Time" reports the time taken by CPLEX to solve the integer problem to optimality.

### 3.4.1 Balas and Saltzman Dataset

This dataset is generated by Balas and Saltzman [14]. It includes 60 test instances with the problem size $n = 4, 6, 8, \ldots, 22, 24, 26$. For each $n$, five instances are randomly generated with the integer cost coefficients $c_{ijk}$ uniformly distributed in the interval [0,100].

Table 3.1: Balas and Saltzman Dataset

| | GRASP | | Hybrid Gen. | | Fix Mapping | | Lagrangian | | LP Peel. | | CPLEX |
|---|---|---|---|---|---|---|---|---|---|---|---|
| n | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Time |
| 4 | - | - | **42.2** | 0.00 | **42.2** | 0.01 | **42.2** | 0.01 | **42.2** | 0.01 | 0.014 |
| 6 | - | - | **40.2** | 0.01 | **40.2** | 0.02 | **40.2** | 0.02 | 41.2 | 0.01 | 0.016 |
| 8 | - | - | **23.8** | 0.03 | 25.4 | 0.03 | **23.8** | 0.04 | 24.6 | 0.02 | 0.034 |
| 10 | - | - | **19** | 0.37 | 21.8 | 0.03 | **19** | 0.07 | **19** | 0.04 | 0.056 |
| 12 | **15.6** | 74.79 | **15.6** | 0.87 | 23.2 | 0.10 | **15.6** | 0.13 | 16.6 | 0.10 | 0.142 |
| 14 | **10** | 106.55 | **10** | 1.73 | 15.2 | 0.22 | **10** | 0.21 | 12 | 0.22 | 0.224 |
| 16 | 10.2 | 143.89 | **10** | 1.89 | 14 | 0.39 | **10** | 0.38 | 10.4 | 0.37 | 0.412 |
| 18 | 7.4 | 190.88 | 7.2 | 2.95 | 9.4 | 1.11 | **6.4** | 0.76 | 6.6 | 0.94 | 0.74 |
| 20 | 6.4 | 246.70 | 5.2 | 4.01 | 5.6 | 1.89 | **4.8** | 1.34 | 5 | 3.24 | 1.432 |
| 22 | 7.8 | 309.64 | 5.6 | 4.54 | 13 | 1.47 | **4** | 2.13 | **4** | 4.11 | 2.71 |
| 24 | 7.4 | 382.45 | 3.2 | 5.66 | 5.8 | 2.13 | **1.8** | 4.68 | 2.2 | 6.10 | 8.042 |
| 26 | 8.4 | 465.20 | 3.6 | 10.78 | 2.6 | 7.46 | **1** | 8.75 | 1.6 | 11.15 | 25.894 |

### 3.4.2 Burkard, Rudolf and Woeginger Dataset

Burkard et al. [36] described this dataset with decomposable cost coefficients, which means that $c_{ijk} = a_i b_j d_k$ as described in Section 1.8.1. For each problem size $n = 4, 6, 8, \ldots, 16$ they generated 100 instances.

Table 3.2: Burkard, Rudolf and Woeginger Dataset

| | GRASP | | Hybrid Gen. | | Fix Mapping | | Lagrangian | | LP Peel. | | CPLEX |
|---|---|---|---|---|---|---|---|---|---|---|---|
| n | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Time |
| 4 | - | - | **443.6** | 0.00 | **443.6** | 0.01 | **443.6** | 0.01 | **443.6** | 0.01 | 0.0061 |
| 6 | - | - | 633.72 | 0.01 | **633.71** | 0.01 | **633.71** | 0.01 | **633.71** | 0.01 | 0.014 |
| 8 | - | - | **819.16** | 0.03 | **819.16** | 0.02 | **819.16** | 0.03 | **819.16** | 0.01 | 0.0318 |
| 10 | - | - | **959.41** | 0.07 | 959.42 | 0.03 | **959.41** | 0.04 | **959.41** | 0.03 | 0.0555 |
| 12 | **1186.81** | 68.30 | **1186.81** | 0.13 | 1186.84 | 0.05 | **1186.81** | 0.06 | 1186.85 | 0.06 | 0.0992 |
| 14 | **1467.74** | 98.10 | **1467.74** | 0.23 | 1468.05 | 0.10 | **1467.74** | 0.11 | 1467.75 | 0.13 | 0.1715 |
| 16 | **1475.13** | 139.30 | **1475.13** | 0.40 | 1475.43 | 0.18 | **1475.13** | 0.19 | **1475.13** | 0.32 | 0.3031 |

### 3.4.3 Crama and Spieksma Dataset

Crama and Spieksma generated this dataset by restricting coefficients to be $c_{ijk} = d_{ij} + d_{ik} + d_{jk}$ [43] as discussed in Section 1.8.2. There are three types of instances in this dataset. For each type, three instances of size $n = 33$ and three instances of size $n = 66$ are generated.

Table 3.3: Crama and Spieksma Dataset, Type 1

| | GRASP | | Hybrid Gen. | | Fix Mapping | | Lagrangian | | LP Peel. | | CPLEX |
|---|---|---|---|---|---|---|---|---|---|---|---|
| n | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Time |
| 33 | **1608** | 660.50 | **1608** | 0.03 | **1608** | 0.71 | **1608** | 2.17 | **1608** | 0.50 | 1.31 |
| 33 | **1401** | 680.50 | **1401** | 0.11 | **1401** | 0.65 | **1401** | 2.79 | **1401** | 0.69 | 2.67 |
| 33 | **1604** | 676.10 | **1604** | 0.11 | **1604** | 0.68 | **1604** | 2.25 | **1604** | 0.51 | 1.29 |
| 66 | 2664 | 15470.10 | **2662** | 0.55 | **2662** | 8.14 | **2662** | 7.34 | **2662** | 7.08 | 16.95 |
| 66 | **2449** | 15010.90 | **2449** | 0.27 | **2449** | 9.26 | **2449** | 7.83 | **2449** | 7.72 | 17.5 |
| 66 | 2759 | 15084.60 | **2758** | 0.58 | 2760 | 10.06 | **2758** | 11.21 | **2758** | 10.01 | 31.06 |

Table 3.4: Crama and Spieksma Dataset, Type 2

| | GRASP | | Hybrid Gen. | | Fix Mapping | | Lagrangian | | LP Peel. | | CPLEX |
|---|---|---|---|---|---|---|---|---|---|---|---|
| n | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Time |
| 33 | **4797** | 766.06 | **4797** | 0.11 | **4797** | 1.33 | **4797** | 2.56 | **4797** | 1.23 | 2.02 |
| 33 | 5068 | 772.84 | **5067** | 0.26 | **5067** | 1.23 | **5067** | 2.89 | **5067** | 1.14 | 2.76 |
| 33 | **4287** | 762.19 | **4287** | 0.26 | **4287** | 1.27 | **4287** | 2.97 | **4287** | 1.14 | 2.81 |
| 66 | 9694 | 14629.10 | **9684** | 4.86 | 9685 | 68.23 | **9684** | 31.22 | **9684** | 69.72 | 138.16 |
| 66 | 8954 | 14922.90 | **8944** | 3.35 | 8948 | 65.77 | **8944** | 30.19 | 8945 | 117.98 | 131.06 |
| 66 | 9751 | 14391.70 | **9745** | 3.09 | 9746 | 62.98 | **9745** | 28.06 | **9745** | 70.52 | 93.87 |

In addition to these benchmark instances, Grundel and Pardalos devised an algorithm to generate Multidimensional Assignment Problems of controlled size with a known optimum solution [68]. They made the code publicly available on their website [5].

Table 3.5: Crama and Spieksma Dataset, Type 3

| | GRASP | | Hybrid Gen. | | Fix Mapping | | Lagrangian | | LP Peel. | | CPLEX |
|---|---|---|---|---|---|---|---|---|---|---|---|
| n | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Obj | Time | Time |
| 33 | **133** | 490.79 | **133** | 0.01 | **133** | 0.82 | **133** | 2.53 | **133** | 1.03 | 2.41 |
| 33 | **131** | 471.21 | **131** | 0.03 | **131** | 0.78 | **131** | 2.69 | **131** | 1.28 | 2.47 |
| 33 | **131** | 451.72 | **131** | 0.02 | **131** | 0.69 | **131** | 2.57 | **131** | 0.71 | 2.53 |
| 66 | **286** | 5322.97 | **286** | 0.15 | **286** | 13.21 | **286** | 10.91 | **286** | 13.15 | 28.19 |
| 66 | **286** | 5126.86 | **286** | 0.16 | **286** | 10.65 | **286** | 8.35 | **286** | 9.84 | 19.15 |
| 66 | **282** | 5059.06 | **282** | 0.23 | **282** | 8.91 | **282** | 9.52 | **282** | 8.59 | 24.05 |

## 3.5 Three Dimensional Traveling Salesman Problem

In this section we introduce a new problem, Three Dimensional TSP (3TSP), as an extension of TSP and provide an application for it.

The TSP is a special case of AP where the permutation is a cycle of length $n$. Consider the 3AP formulation (1.14). If the permuation $p$ is a tour consisting of one cycle, then we refer to the problem as 3TSP. Alternatively, we can obtain 3TSP from 3AP if the permutation $q$ is a cycle of length $n$. An application of 3TSP is the following.

### 3.5.1 An Application of 3TSP

An application of TSP can be stated as follows. A traveling salesman has some dictionaries that he needs to sell in different cities on the tour. He packs the dictionaries before the trip and sells them as he goes through the cites. Imagine a "Marco Polo" version where the traveling salesman buys products from cities on the trip and sells them in the next city. In other words, the salesman has only one luggage which he fills in city $i$ and sells in city $j$ as he travels from $i$ to $j$. Each item has a purchase price in city $i$ and a selling price in city $j$. The traveling salesman is interested in making the most profit as he goes through the cities.

For this purpose, let each $c_{ijk}$ in 3AP be obtained by

$$c_{ijk} = d_{ij} + e_{jk} + f_{ik} \tag{3.81}$$

where $d_{ij}$ is the distance between cities $i$ and $j$, $e_{jk}$ is the cost of selling item $k$ in city $j$ (or the negative of the selling price in city $j$), and $f_{ik}$ is the cost of purchasing item $k$ in city $i$. Then 3TSP can be formulated as

$$\min_{p\in\pi_n^1, q\in\pi_n} \sum_{i=1}^{n} c_{ip(i)q(i)} \tag{3.82}$$

where $\pi_n$ denotes permutations on $n$ and $\pi_n^1$ denotes cycles on $n$.

3TSP is NP-complete since as a special case, it considers matrices $e$ and $f$ to be the same as distance matrix $d$ [43]. Alternatively, if the matrices $e$ and $f$ are identically zero, then 3TSP cotains TSP as a special case.

In this chapter we discussed the necessary and sufficient conditions for an instance of 3AP to be constant. The result was extended to Multidimensional Assignment Problems. Other polynomially solvable special cases were discussed as well as an approximation algorithm for a special case. Also, three heuristic algorithms were introduced and computational experiments were conducted. Out of the three proposed algorithms, Lagrangian Relaxation outperforms the competing heuristics in the literature.

# Chapter 4

# Topics on Quadratic Three Dimensional Assignment Problem

The Quadratic Three Dimensional Assignment Problem can be considered an extension of QAP or 3AP as discussed in Section (1.12). One may obtain Q3AP as the quadratic version of 3AP, or the three-dimensional version of QAP. This natural extension lends itself to studying the Q3AP through QAP and 3AP. This chapter focuses on theoretical aspects of Q3AP such as polynomially solvable special cases and approximation algorithms for special cases. In addition to extending results in QAP and 3AP, we present a new polynomially solvable case that does not rely on QAP nor 3AP.

## 4.1 Polynomially Solvable Special Cases

We propose new theorems to extend the special cases of 3AP and QAP to Q3AP.

**Theorem 4.1.** *If the cost coefficients of Q3AP are decomposable as* $c_{ijklmr} = d_{ijk}d_{lmr}$ *where* $d_{ijk}$ *is the cost coefficient in 3AP, then Q3AP and 3AP have the same optimal solution and*

$$\min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} = \left(\min_{p,q\in\pi_n} \sum_{i=1}^{n} d_{ip(i)q(i)}\right)^2 \tag{4.1}$$

*Proof.* Let $p^*, q^*$ be the permutations that minimize $\sum_{i=1}^{n} d_{ip(i)q(i)}$. Then

$$\min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} \tag{4.2}$$

$$= \min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ip(i)q(i)}d_{jp(j)q(j)} \tag{4.3}$$

$$= \min_{p,q\in\pi_n} \left(\sum_{i=1}^{n} d_{ip(i)q(i)} \sum_{j=1}^{n} d_{jp(j)q(j)}\right) \tag{4.4}$$

$$= \left(\min_{p,q\in\pi_n} \sum_{i=1}^{n} d_{ip(i)q(i)}\right)\left(\min_{p,q\in\pi_n} \sum_{j=1}^{n} d_{jp(j)q(j)}\right) \tag{4.5}$$

$$= \left(\min_{p,q\in\pi_n} \sum_{i=1}^{n} d_{ip(i)q(i)}\right)^2 \tag{4.6}$$

$$= \left(\sum_{i=1}^{n} d_{ip^*(i)q^*(i)}\right)^2 \tag{4.7}$$

Equation (4.5) follows from equation (4.4) since both summands in (4.4) are identical. In other words, the same permutations $p^*$ and $q^*$ in 3AP minimize both sums in (4.4). Note that $p^*$ and $q^*$ are optimal for Q3AP as well. □

**Theorem 4.2.** *If the cost coefficients of Q3AP are decomposable as $c_{ijklmr} = d_{ijk} + d_{lmr}$ where $d_{ijk}$ is the cost coefficient in 3AP, then Q3AP and 3AP have the same optimal solution and*

$$\min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} = 2n \min_{p,q\in\pi_n} \sum_{i=1}^{n} d_{ip(i)q(i)} \tag{4.8}$$

*Proof.* Let $p^*, q^*$ be the permutations that minimize $\sum_{i=1}^{n} d_{ip(i)q(i)}$. Then

$$\begin{aligned}
& \min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} \\
= & \min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} \left(d_{ip(i)q(i)} + d_{jp(j)q(j)}\right) \\
= & \min_{p,q\in\pi_n} \left( \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ip(i)q(i)} + \sum_{i=1}^{n}\sum_{j=1}^{n} d_{jp(j)q(j)} \right) \\
= & \min_{p,q\in\pi_n} \left( n\sum_{i=1}^{n} d_{ip(i)q(i)} + n\sum_{j=1}^{n} d_{jp(j)q(j)} \right) \\
= & 2n \min_{p,q\in\pi_n} \sum_{i=1}^{n} d_{ip(i)q(i)} \\
= & 2n \sum_{i=1}^{n} d_{ip^*(i)q^*(i)}
\end{aligned}$$

The permutations $p^*$ and $q^*$ in 3AP are also feasible for Q3AP and therefore minimize Q3AP. □

The following corollary describes how to extend polynomially solvable special cases of 3AP to polynomially solvable cases of Q3AP. Therefore, we can extend the special cases in Section (1.8.1) to Q3AP, as well as new special cases of 3AP to be discovered in the future by researchers.

**Corollary 4.3.** *Consider an instance of 3AP with entries $d_{ijk}$ and construct an instance of Q3AP where $c_{ijklmr} = d_{ijk}d_{lmr}$ or $c_{ijklmr} = d_{ijk} + d_{lmr}$. If an algorithm $H$ solves 3AP to optimality in polynomial time, then $H$ also solves Q3AP to optimality in polynomial time.*

*Proof.* The result follows from Theorems (4.1) and (4.2) as well as the value of the solution in Q3AP. □

We next present a polynomially solvable class of Q3AP that is obtained independently rather than an extension of QAP or 3AP. Suppose that each entry $c_{ijklmr}$ can be characterized as

$$c_{ijklmr} = a^{(1)}_{ij} + a^{(2)}_{jk} + a^{(3)}_{kl} + a^{(4)}_{lm} + a^{(5)}_{mr} + a^{(6)}_{ri} \tag{4.9}$$

where $A^{(1)}, \ldots, A^{(6)}$ are two-dimensional matrices with arbitrary real entries.

**Theorem 4.4.** *If the coefficients of Q3AP can be characterized as equation (4.9), then the optimal solution of Q3AP is obtained in polynomial time by solving the following problems*

$$\min_{p\in\pi_n} \sum_{i=1}^{n} b_{ip(i)} \tag{4.10}$$

*and*

$$\min_{r\in\pi_n} \sum_{i=1}^{n} d_{ir(i)} \tag{4.11}$$

*where* $b_{ij} = a^{(1)}_{ij} + a^{(4)}_{ij}$ *and* $d_{ij} = a^{(2)}_{ij} + a^{(5)}_{ij}$. *The optimal solution is comprised of* $p$ *as above and* $q = r \circ p$.

*Proof.*

$$\min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} \tag{4.12}$$

$$= \min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} \left( a^{(1)}_{ip(i)} + a^{(2)}_{p(i)q(i)} + a^{(3)}_{q(i)j} + a^{(4)}_{jp(j)} + a^{(5)}_{p(j)q(j)} + a^{(6)}_{q(j)i} \right) \tag{4.13}$$

$$= \min_{p,q\in\pi_n} \left( \sum_{i=1}^{n}\sum_{j=1}^{n} a^{(1)}_{ip(i)} + \sum_{i=1}^{n}\sum_{j=1}^{n} a^{(4)}_{jp(j)} \right.$$
$$\left. + \sum_{i=1}^{n}\sum_{j=1}^{n} a^{(2)}_{p(i)q(i)} + \sum_{i=1}^{n}\sum_{j=1}^{n} a^{(5)}_{p(j)q(j)} + \sum_{i=1}^{n}\sum_{j=1}^{n} a^{(3)}_{q(i)j} + \sum_{i=1}^{n}\sum_{j=1}^{n} a^{(6)}_{q(j)i} \right) \tag{4.14}$$

$$= \min_{p,q\in\pi_n} \left( n\sum_{i=1}^{n} a^{(1)}_{ip(i)} + n\sum_{j=1}^{n} a^{(4)}_{jp(j)} \right.$$
$$\left. + n\sum_{i=1}^{n} a^{(2)}_{p(i)q(i)} + n\sum_{j=1}^{n} a^{(5)}_{p(j)q(j)} + \sum_{i=1}^{n}\sum_{j=1}^{n} a^{(3)}_{q(i)j} + \sum_{i=1}^{n}\sum_{j=1}^{n} a^{(6)}_{q(i)j} \right) \tag{4.15}$$

The terms $\sum_{i=1}^{n}\sum_{j=1}^{n} a^{(3)}_{q(i)j}$ and $\sum_{i=1}^{n}\sum_{j=1}^{n} a^{(6)}_{q(i)j}$ in equation (4.15) are constant regardless of choice of $q$. They are simply the sum of all entries in $a^{(3)}$ and $a^{(6)}$. Hence the optimal solution of (4.12) is the same as the optimal solution of

$$\min_{p,q\in\pi_n} \left( n\sum_{i=1}^{n} a^{(1)}_{ip(i)} + n\sum_{j=1}^{n} a^{(4)}_{jp(j)} + n\sum_{i=1}^{n} a^{(2)}_{p(i)q(i)} + n\sum_{j=1}^{n} a^{(5)}_{p(j)q(j)} \right) \tag{4.16}$$

$$= n \min_{p,q\in\pi_n} \left( \sum_{i=1}^{n} \left( a^{(1)}_{ip(i)} + a^{(4)}_{ip(i)} \right) + \sum_{i=1}^{n} \left( a^{(2)}_{p(i)q(i)} + a^{(5)}_{p(i)q(i)} \right) \right) \tag{4.17}$$

Define matrices $B$ and $D$ such that $b_{ij} = a_{ij}^{(1)} + a_{ij}^{(4)}$ and $d_{ij} = a_{ij}^{(2)} + a_{ij}^{(5)}$. Then solving (4.17) is equivalent to solving the following

$$\min_{p,q\in\pi_n} \left( \sum_{i=1}^{n} b_{ip(i)} + \sum_{i=1}^{n} d_{p(i)q(i)} \right) \tag{4.18}$$

$$= \min_{p,r\in\pi_n} \left( \sum_{i=1}^{n} b_{ip(i)} + \sum_{i=1}^{n} d_{ir(i)} \right) \tag{4.19}$$

$$= \left( \min_{p\in\pi_n} \sum_{i=1}^{n} b_{ip(i)} \right) + \left( \min_{r\in\pi_n} \sum_{i=1}^{n} d_{ir(i)} \right) \tag{4.20}$$

Equation (4.19) follows from equation (4.18) since $p$ is a permutation and ignoring $p$ in the second summand has the effect of changing the order of terms in the summation. This change requires the permutation $q$ to be replaced by $r$. The reason is that $q$ in equation (4.18) maps $I$ to $K$ whereas the second sum in equation (4.19) indicates a mapping from $J$ to $K$. We keep track of this change by noting that $q = r \circ p$. Equation (4.20) indicates that in order to find the optimal solution of (4.12), we can solve the two separate instances of AP in equation (4.20). Since AP can be solved in $O(n^3)$ [82], problem (4.12) can also be solved in $O(n^3)$. □

We next construct Q3AP instances from QAP instances in an attempt to extend the results in QAP to Q3AP.

**Theorem 4.5.** *If $c_{ijklmr} = b_{il}a_{jm} + b_{il}d_{kr}$, then*

$$\min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} = \left( \min_{p\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}a_{p(i)p(j)} \right) + \left( \min_{q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}d_{q(i)q(j)} \right) \tag{4.21}$$

*Proof.*

$$\begin{aligned}
& \min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} \\
= & \min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} \left( b_{ij}a_{p(i)p(j)} + b_{ij}d_{q(i)q(j)} \right) \\
= & \min_{p,q\in\pi_n} \left( \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}a_{p(i)p(j)} + \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}d_{q(i)q(j)} \right) \\
= & \left( \min_{p\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}a_{p(i)p(j)} \right) + \left( \min_{q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}d_{q(i)q(j)} \right)
\end{aligned}$$

□

**Theorem 4.6.** *If* $c_{ijklmr} = b_{il}a_{jm} + a_{jm}d_{kr}$, *then*

$$\min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} = \left(\min_{p\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}a_{p(i)p(j)}\right) + \left(\min_{r\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}d_{r(i)r(j)}\right) \tag{4.22}$$

*where* $q = r \circ p$.

*Proof.*

$$\min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} \tag{4.23}$$

$$= \min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} \left(b_{ij}a_{p(i)p(j)} + a_{p(i)p(j)}d_{q(i)q(j)}\right) \tag{4.24}$$

$$= \min_{p,q\in\pi_n} \left(\sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}a_{p(i)p(j)} + \sum_{i=1}^{n}\sum_{j=1}^{n} a_{p(i)p(j)}d_{q(i)q(j)}\right) \tag{4.25}$$

$$= \min_{p,r\in\pi_n} \left(\sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}a_{p(i)p(j)} + \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}d_{r(i)r(j)}\right) \tag{4.26}$$

$$= \left(\min_{p\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}a_{p(i)p(j)}\right) + \left(\min_{r\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}d_{r(i)r(j)}\right) \tag{4.27}$$

Equation (4.26) follows from equation (4.25) by the same reason mentioned in the proof of Theorem (4.4) and noting that $q = r \circ p$. □

Theorems (4.5) and (4.6) describe two scenarios where Q3AP can be reduced to solving two instances of QAP.

**Corollary 4.7.** *Suppose an instance of Q3AP is of the form stated in Theorem (4.5) or (4.6), and in addition, each resulting QAP instance is polynomially solvable. Then the original Q3AP instance is polynomially solvable with solution provided by Theorem (4.5) or (4.6) respectively.*

## 4.2 Approximation Algorithms

In this section we propose a theorem that extends any approximation algorithm for 3AP to an approximation algorithm for Q3AP.

Let $OPT_{3AP}$ and $OPT_{Q3AP}$ denote the optimal solutions of 3AP and Q3AP respectively. Any feasible solution $S$ to 3AP is also feasible for Q3AP. Let $cost(S_{3AP})$ and $cost(S_{Q3AP})$ denote the costs of $S$ in 3AP and Q3AP respectively.

**Theorem 4.8.** *Consider an instance of 3AP with entries $d_{ijk}$ and construct an instance of Q3AP such that $c_{ijklmr} = d_{ijk}d_{lmr}$. Suppose $H$ is an approximation algorithm to 3AP which returns a solution $S$ with $cost(S_{3AP}) \leq \alpha\ cost(OPT_{3AP})$. Then $H$ is also an approximation algorithm for Q3AP where $cost(S_{Q3AP}) \leq \alpha^2 cost(OPT_{Q3AP})$.*

*Proof.* Let $OPT_{3AP}$ be comprised of permutations $p^*$ and $q^*$. Let $S$, the solution of $H$, be comprised of permutations $p'$ and $q'$. Then since $H$ is an approximation algorithm for 3AP, we have

$$cost(S_{3AP}) = \sum_{i=1}^{n} d_{ip'(i)q'(i)} \leq \alpha \sum_{i=1}^{n} d_{ip^*(i)q^*(i)} = cost(OPT_{3AP}) \tag{4.28}$$

and hence we have

$$\begin{aligned}
cost(S_{Q3AP}) &= \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip'(i)q'(i)jp'(j)q'(j)} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ip'(i)q'(i)}d_{jp'(j)q'(j)} \\
&= \sum_{i=1}^{n} d_{ip'(i)q'(i)} \sum_{j=1}^{n} d_{jp'(j)q'(j)} \\
&\leq \left(\alpha \sum_{i=1}^{n} d_{ip^*(i)q^*(i)}\right)^2 && \text{by equation (4.28)} \\
&= \alpha^2 \min_{p,q\in\pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip(i)q(i)jp(j)q(j)} && \text{by Theorem (4.1)} \\
&= \alpha^2 cost(OPT_{Q3AP})
\end{aligned}$$

Therefore, $cost(S_{Q3AP}) \leq \alpha^2 cost(OPT_{Q3AP})$. □

In order to make Theorem (4.8) more general, we do not require $\alpha$ to be a constant. It may be a function of problem input such as $\log(n)$. This theorem allows us to extend the approximation algorithms to 3AP in Section (1.8.2) to Q3AP, as well as new approximation algorithms for 3AP to be discovered in the future by researchers.

**Corollary 4.9.** *If the cost coefficients of Q3AP are decomposable as $c_{ijklmr} = d_{ijk}d_{lmr}$ where $d_{ijk}$ is the cost coefficient in 3AP of type T△ or S△ (as defined in Section 1.8.2),*

*then the algorithm $H$ in Theorem (1.12) returns a solution within $\frac{16}{9}$ of the optimal Q3AP solution.*

*Proof.* Algorithm $H$ in Theorem (1.12) returns a solution within $\frac{1}{3}$ of the optimal 3AP solution. The result follows by Theorem (4.8). □

In this chapter we presented special cases of the Q3AP as extension of special cases of QAP and 3AP. Also, a new special case was discussed without relying on the QAP or 3AP problems. Approximation algorithms were discussed as well for special cases of Q3AP.

# Chapter 5

# Conclusion

Chapter 1 introduced three combinatorial optimization problems: Quadratic Assignment Problem (QAP), Three Dimensional Assignment Problem (3AP) and Quadratic Three Dimensional Assignment Problem (Q3AP). These problems are NP-hard.

In Chapter 2 we discussed a general heuristic paradigm called Randomized Local Search. Extensive experimental results are reported using two variations of the algorithm on the Quadratic Assignment Problem; RandLS-Sim and RandLS-Tabu, with benchmark QAPLIB problems and microarray instances as the test bed. Out of the 14 microarray instances, our algorithm obtained an improved solution for the largest instance and matched 3 others in the Conflict Index class. RandLS-Sim and RandLS-Tabu together matched 111 of 135 QAPLIB instances with optimal or best known solutions.

We expect that RandLS works well for other combinatorial optimization problems as well. A preliminary implementation of the method for Quadratic Minimum Spanning Tree problem is reported in [95].

In Chapter 3 we presented new polynomially solvable cases of the Three Dimensional Assignment Problem. The most notable such special case is the constant 3AP where every feasible solution is optimal. We provided full characterization of the constant 3AP: a 3AP instance is constant if and only if the cost matrix can be decomposed as the sum of three one-dimensional arrays. The result was extended to Multidimensional Assignment Problems. Other special cases involve cost matrices that are decomposable as the sum of two one-dimensional arrays and a two-dimensional array.

We also provided approximation algorithms for 3AP instances where $t_{ijk} = d_{ij} + d_{ik} + d_{jk}$ or $s_{ijk} = \min\{d_{ij} + d_{ik}, d_{ij} + d_{jk}, d_{ik} + d_{jk}\}$. These problems are referred to as $T$ and $S$

respectively. If the distances satisfy the parameterized triangle inequality with parameter $\delta$, then the problems are referred to as $T\triangle_\delta$ and $S\triangle_\delta$. Our approximation algorithms return solutions whose cost is at most $\frac{2}{3}(1+\delta)$ times the optimal cost.

Chapter 3 also introduced three heuristic algorithms for the general 3AP: Fix Mapping, Lagrangian Relaxation and LP Peeling. The Lagrangian Relaxation and LP Peeling algorithms outperform the two most successful heuristics in the literature in terms of solution quality. Experimental results were presented using benchmark instances. We expect that the methods used in our Lagrangian Relaxation algorithm can be successfully applied to other combinatorial optimization problems. The chapter concluded with a new problem called Three Dimensional Traveling Salesman Problem (3TSP) as a three-dimensional version of the well-known Traveling Salesman Problem (TSP). We are hoping that 3TSP gets the attention of researchers as a potential application is illustrated.

In Chapter 4 we presented polynomially solvable special cases of the Quadratic Three Dimensional Assignment Problem, as well as approximation algorithms for special cases of Q3AP. Since there are only a few papers written on the Q3AP, we anticipate more work in the future on this problem.

## 5.1 Appendix 1

The improved solution obtained for one of the Conflict Index Instances from the QAP micorarray dataset is given below.

n: 144
cost: 794811636
assignment: 19 101 16 47 107 129 48 59 61 137 35 108 85 40 134 57 36 63 69 53 18 139 87 120 37 130 15 76 78 43 75 111 58 128 86 33 10 82 39 13 117 45 14 62 143 44 38 4 132 56 141 95 74 136 124 66 77 68 126 60 71 105 6 79 80 52 27 123 98 94 93 125 24 67 99 81 70 131 12 8 3 23 144 133 55 73 109 91 31 122 34 65 112 41 21 119 72 22 142 7 50 121 42 114 138 100 30 11 1 102 89 32 26 103 64 46 25 9 116 5 118 140 28 97 135 54 90 49 2 96 20 88 110 127 115 51 106 17 92 113 83 84 104 29
algorithm: RandLS-Tabu
k: 30
iter(k): 30
total number of iterations: 2000000

We present below the best solutions we have obtained over all experiments for the microarray QAP test problems. These results are due to fine-tuning parameters for specific instances, whereas Tables (2.1) and (2.2) use default parameter values as reported earlier. The bold entries indicate the best known solutions so far and the underlined entries match the best known solutions that we obtained.

In addition, the run times in [1] range from less than 2 minutes per single run for the $n = 36$ instance to less than 9 hours for the $n = 144$ problems. As the following 2 tables indicate, our solution to $n = 144$ take far less time.

Table 5.1: Border Length Instances

| size | best known | our best solution | time | algorithm used |
|---|---|---|---|---|
| 36 | **3296** | 3304 | 28 | RandLS-Sim |
| | | | 226 | RandLS-Tabu |
| 49 | **4548** | 4572 | 648 | RandLS-Tabu |
| 64 | **5988** | 6020 | 763 | RandLS-Tabu |
| 81 | **7536** | 7580 | 1210 | RandLS-Tabu |
| 100 | **9272** | 9336 | 1865 | RandLS-Tabu |
| 121 | **11412** | 11504 | 2755 | RandLS-Tabu |
| 144 | **13472** | 13588 | 1977 | RandLS-Tabu |

Table 5.2: Conflict Index Instances

| size | best known | our best solution | time | algorithm used |
|---|---|---|---|---|
| 36 | **168,611,971** | 168,611,971 | 219 | RandLS-Tabu |
| 49 | **236,355,034** | 236,355,034 | 419 | RandLS-Tabu |
| 64 | **325,671,035** | 325,671,035 | 98 | RandLS-Sim |
| | | | 745 | RandLS-Tabu |
| 81 | **427,447,820** | 427,501,773 | 1197 | RandLS-Tabu |
| 100 | **523,146,366** | 523,552,203 | 611 | RandLS-Sim |
| 121 | **653,416,978** | 654,416,694 | 2780 | RandLS-Tabu |
| 144 | 795,009,899 | **794,811,636** | 8042 | RandLS-Tabu |

The following sections describe the specifics of the algorithms used in order to obtain the two tables above.

### 5.1.1 Border Length instances

n: 36

cost: 3304

assignment: 4 9 20 28 26 30 29 7 34 19 1 25 21 3 8 6 31 32 2 33 15 5 13 12 10 18 24 22 35 16 23 14 27 11 36 17

algorithm: RLS-Sim

initial k: n

initial iter(k): 100

When k = 4 , jump to local search

Multiple start parameters:

number of repeats: 4
number of peeks at local minimum: 10
increments of iter(k): 100
increments of k: 0.5 * initial k

**Alternative solution**
assignment: 23 10 24 22 2 29 14 18 33 15 35 13 21 8 3 5 6 31 12 16 20 34 19 32 36 17 9 7 1 25 28 26 30 4 27 11
algorithm: RLS-Tabu
k: 15
iter(k): 30
Total number of iterations: 1000000

n: 49
cost: 4572
assignment: 39 12 37 2 30 28 32 6 22 27 19 48 45 40 47 18 31 34 36 43 42 23 13 35 4 15 11 8 16 46 33 17 24 25 10 26 5 41 7 21 29 14 1 44 9 20 49 3 38
algorithm: RLS-Tabu
k: 15
iter(k): 30
total number of iterations: 1500000

n: 64
cost: 6020
assignment: 9 58 32 50 18 26 13 53 61 33 60 48 10 43 54 47 38 16 22 59 2 36 62 28 35 1 15 24 20 37 29 34 8 25 51 6 42 45 63 12 4 3 46 31 17 41 23 19 21 56 14 40 5 39 52 30 57 49 55 7 11 27 44 64
algorithm: RLS-Tabu
k: 15
iter(k): 30
total number of iterations: 1000000

n: 81

cost: 7580
assignment: 46 59 67 61 24 48 25 16 53 34 79 9 19 8 81 44 39 15 51 71 76 23 80 3 33 43 36 29 57 55 7 11 18 54 1 45 14 6 52 30 2 26 49 10 74 58 63 32 72 64 4 56 70 38 62 40 5 41 35 27 42 28 37 17 47 66 77 60 31 21 78 65 69 50 68 20 12 22 73 75 13
algorithm: RLS-Tabu
k: 15
iter(k): 30
total number of iterations: 1000000

n: 100
cost: 9336
assignment: 23 13 54 5 26 92 56 68 44 4 35 69 61 65 80 87 45 98 14 74 9 22 73 20 75 84 93 94 79 53 6 72 58 59 55 38 66 17 19 63 21 8 34 100 25 99 12 50 52 36 91 10 78 33 16 67 28 83 2 43 90 70 86 15 41 88 27 42 32 47 89 60 71 95 1 31 97 57 29 96 7 49 3 30 51 39 64 81 40 82 11 48 77 18 62 46 37 85 24 76
algorithm: RLS-Tabu
k: 15
iter(k): 30
total number of iterations: 1000000

n: 121
cost: 11504
assignment: 48 32 55 72 104 17 105 11 53 98 113 37 99 62 117 80 14 69 43 2 36 46 85 71 115 111 84 116 15 114 3 57 51 10 42 91 73 23 82 64 100 52 41 4 58 31 119 66 19 49 87 45 76 8 107 103 5 108 1 97 16 65 33 28 60 67 24 118 102 63 86 56 20 21 121 59 95 77 26 6 47 112 70 92 89 27 120 68 50 109 13 61 81 78 12 30 35 40 74 39 22 18 29 38 94 83 54 44 110 93 9 90 34 75 25 7 79 101 96 106 88
algorithm: RLS-Tabu
k: 15
iter(k): 30
total number of iterations: 1000000

n: 144

cost: 13588
assignment: 32 51 17 87 120 76 140 89 11 6 8 23 64 85 35 128 4 3 12 68 94 24 123 113 46 9 58 111 38 126 143 93 25 138 114 49 110 96 60 125 86 104 14 131 10 37 54 75 130 40 44 77 27 29 97 7 13 117 90 52 103 134 15 81 133 84 115 91 95 74 136 124 26 50 67 21 144 83 122 31 141 69 45 98 108 135 73 55 101 2 34 121 127 36 43 47 99 92 142 28 105 39 42 82 5 59 63 129 66 118 41 109 106 33 112 56 20 61 19 48 72 1 100 53 102 88 107 132 116 78 16 70 22 119 62 18 30 137 139 71 57 79 65 80
algorithm: RLS-Tabu
k: 15
iter(k): 30
total number of iterations: 500000

### 5.1.2 Conflict Index Instances

n: 36
cost: 168611971
assignment: 29 18 22 24 5 19 2 21 33 15 6 27 28 26 1 35 31 32 36 17 3 13 12 10 30 7 4 9 25 23 8 16 34 20 11 14
algorithm: RLS-Tabu
k: 15
iter(k): 30 total number of iterations: 1000000

n: 49
cost: 236355034
assignment: 1 3 38 2 40 32 28 24 49 5 7 9 20 39 4 21 45 41 19 23 12 6 34 33 46 8 27 22 30 29 26 16 47 18 31 37 25 11 42 44 35 13 48 17 36 43 15 14 10
algorithm: RLS-Tabu
k: 15
iter(k): 30 total number of iterations: 1000000

n: 64
cost: 325671035
assignment: 13 9 14 29 37 12 28 30 53 58 33 34 45 62 36 47 21 32 18 11 63 19 2 59 56 60 50

27 55 23 61 22 16 48 42 39 52 54 3 15 38 64 44 5 49 57 25 51 24 10 20 41 31 35 1 7 6 26 43 4 8 17 46 40
Algorithm: RLS-Sim
initial k: n
initial iter(k): 200
when k = 4 , jump to local search.
Multiple start parameters:
number of repeats: 3
number of peeks at local minimum: 15
increments of iter(k): 50
increments of k: 0.5 * initial k

**Alternative solution**
assignment: 6 24 38 16 56 21 53 13 26 10 64 48 60 32 58 9 43 20 44 42 50 18 33 14 4 41 5 39 27 11 34 29 8 31 49 52 55 63 45 37 17 35 57 54 23 19 62 12 46 1 25 3 61 2 36 28 40 7 51 15 22 59 47 30
algorithm: RLS-Tabu
k: 15
iter(k): 30
total number of iterations: 1000000

n: 81
cost: 427501773
assignment: 74 23 25 55 26 66 68 50 16 38 24 48 7 18 77 43 39 37 65 8 80 11 3 33 10 44 62 22 19 9 81 76 54 1 15 53 40 47 79 2 28 52 57 49 70 67 73 59 30 75 42 4 56 6 21 61 34 46 13 27 72 32 71 31 69 17 64 35 41 45 63 5 12 60 20 14 29 51 36 78 58
algorithm: RLS-Tabu
k: 30
iter(k): 30
total number of iterations: 1000000

n: 100
cost: 523552203

assignment: 94 3 14 69 85 80 16 55 23 1 6 71 86 61 22 65 78 9 35 81 10 70 20 73 58 34 5 21 2 54 11 95 13 72 8 59 56 100 45 60 24 7 47 96 15 92 83 28 26 90 36 63 52 17 33 88 19 50 37 89 43 31 97 57 87 53 68 44 40 98 82 32 29 42 27 79 93 4 12 46 76 51 30 18 77 74 66 38 99 39 67 62 41 49 48 75 91 84 64 25
algorithm: RLS-Sim
initial k: n
initial iter(k): 200
when k = 4 , jump to local search.
Multiple start parameters:
number of repeats: 4
number of peeks at local minimum: 15
increments of iter(k): 50
increments of k: 0.5 * initial k

n: 121
cost: 654416694
assignment: 54 101 28 35 76 41 98 58 66 113 57 39 77 33 36 85 43 10 103 38 25 52 92 30 65 2 50 109 45 15 79 7 97 46 81 5 9 53 90 34 83 69 112 1 20 12 8 104 51 26 74 110 17 86 63 70 94 96 72 108 3 22 18 105 56 14 32 78 48 106 44 114 31 13 6 84 23 80 117 62 87 100 93 42 11 102 116 19 118 99 115 37 29 88 67 121 59 120 55 47 71 91 75 49 16 60 27 89 40 24 21 73 111 64 82 119 107 4 95 61 68
algorithm: RLS-Tabu
k: 15
iter(k): 30
total number of iterations: 1000000

n: 144
cost: 794811636
assignment: 19 101 16 47 107 129 48 59 61 137 35 108 85 40 134 57 36 63 69 53 18 139 87 120 37 130 15 76 78 43 75 111 58 128 86 33 10 82 39 13 117 45 14 62 143 44 38 4 132 56 141 95 74 136 124 66 77 68 126 60 71 105 6 79 80 52 27 123 98 94 93 125 24 67 99 81 70 131 12 8 3 23 144 133 55 73 109 91 31 122 34 65 112 41 21 119 72 22 142 7 50 121 42 114 138 100 30 11 1 102 89 32 26 103 64 46 25 9 116 5 118 140 28 97 135 54 90 49 2 96 20 88

110 127 115 51 106 17 92 113 83 84 104 29
algorithm: RLS-Tabu
k: 30
iter(k): 30
total number of iterations: 2000000

# Bibliography


[1] Results for de Carvalho et al. Problems: http://cbeweb-1.fullerton.edu/isds/zdrezner/Carvalho.htm.

[2] A. Loebel, (2004). MCF Version 1.3 - A network simplex implementation: www.zib.de.

[3] Microarray Library: http://gi.cebitec.uni-bielefeld.de/comet/chiplayout/qap.

[4] OR Library website: http://people.brunel.ac.uk/∼mastjjb/jeb/info.html.

[5] Test problem generator for Multidimensional Assignment Problem: www.math.ufl.edu/∼coap.

[6] A. Aggarwal and J.K. Park. Notes on searching in multidimensional monotone arrays. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 497–512, 1988.

[7] A. Aggarwal and J.K. Park. Parallel searching in multidimensional monotone arrays. Research Report RC 14826, IBM T.J. Watson Research Center, Yorktown Heights, NY, 1989. Portions of this paper appeared in [6].

[8] A. Aggarwal and J.K. Park. Sequential searching in multidimensional monotone arrays. Research Report RC 15128, IBM T.J. Watson Research Center, Yorktown Heights, NY, 1989. Portions of this paper appeared in [6].

[9] R.K. Ahuja, J.B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research*, 27:917–934, 2000.

[10] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. Grasp with path relinking for three-index assignment. *INFORMS Journal on Computing*, 17:224–247, 2005.

[11] K. Anstreicher, N. Brixius, J.P. Goux, and J. Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, 91:563–588, 2002.

[12] A.A. Assad and W. Xu. On lower bounds for a class of quadratic 0,1 programs. *Operations Research Letters*, 4:175–180, 1985.

[13] E. Balas and M.W. Padberg. Set partitioning: a survey. *SIAM Review*, 18:710–760, 1976.

[14] E. Balas and M.J. Saltzman. An algorithm for the three-index assignment problem. *Operations Research*, 39:150–161, 1991.

[15] M.L. Balinski and R.E. Gomory. A primal method for the assignment and transportation problems. *Management Science*, 10:578–593, 1964.

[16] S.E. Bammel and J. Rothstein. The number of $9 \times 9$ latin squares. *Discrete Mathematics*, 11:93–95, 1975.

[17] H.J. Bandelt, Y. Crama, and F.C.R. Spieksma. Approximation algorithms for multidimensional assignment problems with decomposable costs. *Discrete Applied Mathematics*, 49:25–50, 1994.

[18] H.J. Bandelt, A. Maas, and F.C.R. Spieksma. Local search heuristics for multi-index assignment problems with decomposable costs. *Journal of the Operational Research Society*, 55:694–704, 2004.

[19] A. Barvinok and T. Stephen. The distribution of values in the quadratic assignment problem. *Mathematics of Operations Research*, 28(1):64–91, 2003.

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

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

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

[23] W.W. Bein, P. Brucker, J.K. Park, and P.K. Pathak. A monge property for the d-dimensional transportation problem. *Discrete Applied Mathematics*, 58:97–109, 1995.

[24] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[25] S.H. Bokhari. *Assignment Problems in Parallel and Distributed Computing*. Kluwer Academic Publishers, 1987.

[26] N.W. Brixius and K.M. Anstreicher. The steinberg wiring problem. Technical report, Microsoft Corporation, Department of Management Scineces, University of Iowa, 2001.

[27] P.A. Bruijs. On the quality of heuristic solutions to a 19 x 19 quadratic assignment problem. *European Journal of Operational Research*, 17:21–30, 1984.

[28] R.E. Burkard. Locations with spatial interactions: the quadratic assignment problem. In P.B. Mirchandani and R.L. Francis, editors, *Discrete Location Theory*, pages 387–437. John Wiley and Sons, 1991.

[29] R.E. Burkard. Selected topics on assignment problems. *Discrete Applied Mathematics*, 123:257–302, 2002.

[30] R.E Burkard and T. Bonniger. A heuristic for quadratic boolean programs with applications to quadratic assignment problems. *European Journal of Operational Research*, 13:374–386, 1983.

[31] R.E. Burkard and U. Derigs. *Assignment and matching problems: solutions methods with Fortran programs*, volume 184 of *Lectures Notes in Economics and Mathematical Systems*. Springer, 1980.

[32] R.E. Burkard and K. Fröhlich. Some remarks on 3-dimensional assignment problems. *Methods of Operations Research*, 36:31–36, 1980.

[33] R.E. Burkard, S. Karisch, and F. Rendl. QAPLIB - a quadratic assignment problem library. *European Journal of Operational Research*, 55:115–119, 1991. http://www.opt.math.tu-graz.ac.at/qaplib.

[34] R.E. Burkard, B. Klinz, and R. Rudolf. Perspectives of monge properties in optimization. *Discrete Applied Mathematics*, 70:95–161, 1996.

[35] R.E. Burkard and F. Rendl. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, 17(2):169–174, 1984.

[36] R.E. Burkard, R. Rudolf, and G.J Woeginger. Three dimensional axial assignment problems with decomposable cost coefficients. *Discrete Applied Mathematics*, 65:123–139, 1996.

[37] P. Carraresi and F. Malucelli. A new lower bound for the quadratic assignment problem. *Operations Research*, 40(1):S22–S27, 1992.

[38] E. Çela. *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer Academic Publishers, 1998.

[39] N. Christofides and E. Benavent. An exact algorithm for the quadratic assignment problem on a tree. *Operation Research*, 37(5):760–768, 1989.

[40] N. Christofides and M. Gerrard. Special cases of the quadratic assignment problem. Technical report, Management Science Research Report 391, Carnegie Mellon University, 1976.

[41] N. Christofides and M. Gerrard. A graph theoretic analysis of bounds for the quadratic assignment problem. In *Studies on Graphs and Discrete Programming*, pages 61–68. North-Holland, 1981.

[42] Y. Crama, A.W.J. Kolen, A.G. Oerlemans, and F.C.R. Spieksma. Throughput rate optimization in the automated assembly of printed circuit boards. *Annals of Operations Research*, 26:455–480, 1990.

[43] Y. Crama and F.C.R. Spieksma. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research*, 60:273–279, 1992.

[44] L. Davis. *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers, 1987.

[45] S.A. de Carvalho Jr. and S. Rahmann. Microarray layout as quadratic assignment problem. In *Proceedings of the German Conference on Bioinformatics P-83*, pages 11–20, 2006. Lecture Notes in Informatics.

[46] C.S. Edwards. The derivation of a greedy approximator for the koopmans-beckmann quadratic assignment problem. In *Proceedings of the 77-th Combinatorial Programming Conference (CP77)*, pages 55–86, 1977.

[47] C.S. Edwards. A branch and bound algorithm for the koopmans-beckmann quadratic assignment problem. *Mathematical Programming Study*, 13:35–52, 1980.

[48] A.N. Elshafei. Hospital layout as a quadratic assignment problem. *Operational Research Quarterly*, 28(1):167–179, 1977.

[49] G. Finke, R.E. Burkard, and F. Rendl. Quadratic assignment problems. *Annals of Discrete Mathematics*, 31:61–82, 1987.

[50] C. Fleurent and J.A. Ferland. Genetic hybrids for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 173 – 187. American Mathematical Society, 1994.

[51] R.L. Francis and J.A. White. *Facility layout and location: An analytical approach*. Prentice-Hall, 1974.

[52] A.M. Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13:161–164, 1983.

[53] A.M. Frieze and J. Yadegar. An algorithm for solving 3-dimensional assignment problems with application to scheduling a teaching practice. *Journal of the Operational Research Society*, 32:989–995, 1981.

[54] A.M. Frieze and J. Yadegar. On the quadratic assignment problem. *Discrete Applied Mathematics*, 5:89–98, 1983.

[55] E.Y. Gabovich. Constant discrete programming problems on substitution sets. *Translated from Kibernetika*, 5:128–134, 1976. in Russian.

[56] M.R. Garey and D.S. Johnson. *Computers and intractibility - A guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[57] R.S. Garfinkel and G.L. Nemhauser. The set-partitioning problem: set covering with equality constraints. *Operations Research*, 17:848–856, 1969.

[58] J.W. Gavett and N.V. Plyter. The optimal assignment of facilities to locations by branch-and-bound. *Operations Research*, 14:210–232, 1966.

[59] A.M. Geoffrion and G.W. Graves. Scheduling parallel production lines with changeover costs: practical applications of a quadratic assignment/lp approach. *Operations Research*, 24:595–610, 1976.

[60] K.C. Gilbert and R.B. Hofstra. An algorithm for a class of three-dimensional assignment problems arising in scheduling operations. *Institute of Industrial Engineers Transactions*, 19:29–33, 1987.

[61] K.C. Gilbert and R.B. Hofstra. Multidimensional assignment problems. *Decision Sciences*, 19:306–321, 1988.

[62] P.C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal on Applied Mathematics*, 10:305–313, 1962.

[63] F. Glover. Tabu search–part I. *ORSA Journal on Computing*, 1(3):190–206, 1989. "orsa" is called informs today.

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

[65] F. Glover. Tabu thresholding: Improved search by nonmonotonic trajectories. *ORSA Journal on Computing*, 7(4):426–442, 1995.

[66] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[67] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[68] D.A. Grundel and P.M. Pardalos. Test problem generator for the multidimensional assignment problem. *Computational Optimization and Applications*, 30:133–146, 2005.

[69] S.W. Hadley, F. Rendl, and H. Wolkowicz. Bounds for the quadratic assignment problem using continuous optimization techniques. In *Integer Programming and Combinatorial Optimization*, pages 237–248. University of Waterloo Press, 1990.

[70] S.W. Hadley, F. Rendl, and H. Wolkowicz. A new lower bound via projection for the quadratic assignment problem. *Mathematics of Operations Research*, 17:727–739, 1992.

[71] P.M. Hahn, B.J. Kim, T. Stützle, S. Kanthak, W.L. Hightower, H. Samra, Z. Ding, and M. Guignard. The quadratic three-dimensional assignment problem: exact and approximate solution methods. *European Journal of Operational Research*, 184(2):416–428, 2008.

[72] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[73] G.H. Hardy, J.E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, 1967.

[74] D.R. Heffley. The quadratic assignment problem: A note. *Econometrica*, 40:1155–1163, 1972.

[75] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[76] Gaofeng Huang and Andrew Lim. A hybrid genetic algorithm for the three-index assignment problem. *European Journal of Operational Research*, 172:249–257, 2006.

[77] S.E. Karisch. *Nonlinear approaches for quadratic assignment and graph partition problems*. PhD thesis, Graz University of Technology, 1995.

[78] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. R.E. Miller and J.W. Thatcher, editors, Plenum Press, 1972.

[79] L. Kaufman and F. Broeckx. An algorithm for the quadratic assignment problem using bender's decomposition. *European Journal of Operational Research*, 2:204–211, 1978.

[80] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[81] T.C. Koopmans and M.J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

[82] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[83] A.M. Land. A problem of assignment with interrelated costs. *Operational Research Quarterly*, 14:185–198, 1963.

[84] E.L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.

[85] C.F. Laywine and G.L. Mullen. *Discrete Mathematics Using Latin Squares*. John Wiley & Sons, New York, 1998.

[86] Y. Li and P.M. Pardalos. Generating quadratic assignment test problems with known optimal permutations. *Computational Optimization and Applications*, 1:163–184, 1992.

[87] Y. Li, P.M. Pardalos, K.G. Ramakrishnan, and M.G.C. Resende. Lower bounds for the quadratic assignment problem. *Operations Research*, 50:387–410, 1994.

[88] Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

[89] B.D. McKay and I.M. Wanless. On the number of latin squares. *Annals of Combinatorics*, 9:335–344, 2005.

[90] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[91] P. B. Mirchandani and T. Obata. Algorithms for a class of quadratic assignment problems. 1979. Presented at the Joint ORSA/TIMS National Meeting, 1979, New Orleans.

[92] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[93] K.A. Murthy, P. Pardalos, and Y. Li. A local search algorithm for the quadratic assignment problem. *Informatica*, 3:524–538, 1992.

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

[95] T. Öncan and Punnen A.P. The quadratic minimum spanning tree problem: A lower bounding procedure and an efficient search algorithm. *Computers and Operations Research*, 37(10):1762–1773, 2010.

[96] G.S. Palubeckis. Generation of quadratic assignment test problems with known optimal solutions. *U.S.S.R. Comput. Maths. Math. Phys.*, 28:97–98, 1988. (in Russian).

[97] P. M. Pardalos and J. V. Crouse. A parallel algorithm for the quadratic assignment problem. In *Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, pages 351–360. ACM Press, 1989.

[98] P. M. Pardalos, K. A. Murthy, and T. P. Harrison. A computational comparison of local search heuristics for solving quadratic assignment problems. *Informatica*, 4:172–187, 1993.

[99] P.M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–42. American Mathematical Society, 1994.

[100] P.M. Pardalos and H. Wolkowicz. *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1994.

[101] W.P. Pierskalla. The tri-substitution method for the three-dimensional assignment problem. *Journal of the Canadian Operational Research Society*, 5:71–81, 1967.

[102] W.P. Pierskalla. The multidimensional assignment problem. *Operations Research*, 16:422–431, 1968.

[103] M.A. Pollatschek, N. Gershoni, and Y.T. Radday. Optimization of the typewriter keyboard by simulation. *Angewandte Informatik*, 17:438–439, 1976.

[104] A.P. Poore, N. Rijavec, M.E. Liggins, and V.C. Vannicola. Data association problems posed as multidimensional assignment problems: problem formulation. In O.E. Drummond, editor, *Signal and Data Processing of Small Targets*, volume 1954, pages 552–563. SPIE, 1993.

[105] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[106] A.P. Punnen and Y.P. Aneja. Randomized local search algorithms. In *TIMS/ORSA Chicago*, 1993.

[107] F. Rendl and H. Wolkowicz. Applications of parametric programming and eigenvalue maximization to the quadratic assignment problem. *Mathematical Programming*, 53:63–78, 1992.

[108] A.J. Robertson. A set of greedy randomized adaptive local search procedure (grasp) implementations for the multidimensional assignment problem. *Computational Optimization and Applications*, 19:145–164, 2001.

[109] S. Sahni and T. Gonzales. P-complete approximation problems. *Journal of the Association for Computing Machinery*, 23:555–565, 1976.

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

[111] M. Solimanpur, P. Vrat, and R. Shankar. Ant colony optimization algorithm to the inter-cell layout problem in cellular manufacturing. *European Journal of Operational Research*, 157:592–606, 2004.

[112] L. Steinberg. The backboard wiring problem: A placement algorithm. *SIAM Review*, 3:37 – 50, 1961.

[113] L.W. Swanson. N-dimensional scheduling. Unpublished manuscript, Northwestern University, 1976.

[114] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

[115] E.D. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3:87–105, 1995.

[116] D.E. Tate and A.E. Smith. A genetic approach to the quadratic assignment problem. *Computers and Operations Research*, 22:73–83, 1995.

[117] V. Valls, R. Martí, and P. Lino. A tabu thresholding algorithm for arc crossing minimization in bipartite graphs. *Annals of Operations Research*, 63(2):233–251, 1996.

[118] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, 1987.

[119] D. Vigo and V. Maniezzo. A genetic/tabu thresholding hybrid algorithm for the process allocation problem. *Journal of Heuristics*, 3(2):91–110, 1997.

[120] S.J. Wang and B.R. Sarker. Locating cells with bottleneck machines in cellular manufacturing systems. *International Journal of Production Research*, 40:403–424, 2002.

[121] M.R. Wilhelm and T.L. Ward. Solving quadratic assignment problems by simulated annealing. *IEEE Transactions*, 19(1):107–119, 1987.

[122] H. Youssef, S.M. Sait, and H. Ali. Fuzzy simulated evolution algorithm for vlsi cell placement. *Computers and Industrial Engineering*, 44:227–247, 2003.

[123] J. Yu and B.R. Sarker. Directional decomposition heuristic for a linear machine-cell location problem. *European Journal of Operational Research*, 149:142–184, 2003.