Heuristic Solution Approaches to the Solid Assignment Problem

Dhia A. Kadhem

A Thesis Submitted for the Degree of

Doctor of Philosophy

Department of Mathematical Sciences
University of Essex

December, 2017

Dedicated to

My affectionate parents, my wife Mrs Raghda Mehdi, my sons, my daughter and all my grandchildren.

Acknowledgements

I would like to extend tremendous thanks to my supervisors, Professor Abdellah Salhi and Dr. Xinan Yang for providing me with support, generous guidance and valuable insight. I am very grateful for the opportunity to study a PhD at Essex University conducting researching work in the field of Operations Research. Their vast knowledge in the subject of solving the Solid Assignment Problems and the related subjects such as the Genetic Algorithms and the Monge Assignment Problem has been especially helpful. I am also very thankful for their continual encouragement, training, workshops and conferences which made finding answers, advice and comments easier.

I would like to give special thanks to my dissertation committee and everybody within the Department of Mathematical Sciences, for creating such a friendly atmosphere which made it so enjoyable to study. I would like to acknowledge the post graduate students secretary Mrs. Shauna Meyers for her reliable and fantastic commitment to her role. I really appreciate Dr. Mohamed Mehbali from London South Bank University for his assistance while reading my thesis and express his unquestionable feedback. A special thanks goes to my colleagues Ms. Rewayda Abo Alsabeh, Dr. Omar Karakchi and Dr. Ghazwan Alsoufi for their supportive friendship.

Finally I am most grateful to my family, especially to my parents who have shown

me the road to further education and the importance of the continual betterment of ones intellect before they passed away.

I am ever so thankful to my wife Mrs. Raghda Mehdi who during my long studies provided encouragement and support in every possible way. I would like to recognise the time and efforts of my sons Haval, Dlair, Himen, Allan and my daughter Aven for their supports.

Abstract

The 3-dimensional assignment problem, also known as the Solid Assignment Problem (SAP), is a challenging problem in combinatorial optimisation. While the ordinary or 2-dimensional assignment problem is in the **P**-class, SAP which is an extension of it, is **NP**-hard. SAP is the problem of allocating n jobs to n machines in n factories such that exactly one job is allocated to one machine in one factory. The objective is to minimise the total cost of getting these n jobs done. The problem is commonly solved using exact methods of integer programming such as Branch-and-Bound (B&B). As it is intractable, only approximate solutions are found in reasonable time for large instances. Here, we suggest a number of approximate solution approaches, one of them the Diagonals Method (DM), relies on the Kuhn-Tucker Munkres algorithm, also known as the Hungarian Assignment Method. The approach was discussed, hybridised, presented and compared with other heuristic approaches such as the Average Method, the Addition Method, the Multiplication Method and the Genetic Algorithm. Moreover, a special case of SAP which involves Monge-type matrices is also considered. We have shown that in this case DM finds the exact solution efficiently.

We sought to provide illustrations of the models and approaches presented whenever appropriate. Extensive experimental results are included and discussed. The thesis ends with a conclusions and some suggestions for further work on the same and related topics.

Declaration

I hereby declare that, except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Contents

Acknowledgements **iii**

Abstract **v**

Declaration **vi**

Acronyms **xvi**

1 Methods of Optimisation **2**

1.1 Introduction 2

1.2 The Hungarian Method 4

1.3 Complexity and NP-Completeness 7

1.4 Solution Approaches to Optimisation Problems 11

1.4.1 Exact Methods 11

1.4.2 Approximation Algorithms 12

1.4.3 Meta-heuristics 12

1.4.4 Review of Meta-heuristic Methods 13

1.4.5 Simulated Annealing 14

1.4.6 Genetic Algorithm 14

1.4.7 Discrete Particle Swarm Optimisation 16
1.5 Linear, Non-Linear and Integer Programming Problem 17
1.6 Thesis Aims and Objectives 21
1.7 Thesis Organisation 24
1.8 Summary 25

2 Literature Review and Mathematical Background 26

2.1 The Two Dimensional Assignment Problem 26
2.2 The Three Dimensional Assignment Problem 29
2.3 Types of the Solid Assignment Problems 33
2.3.1 The 3-Index Assignment Problem 33
2.3.2 The Planar 3-Assignment Problem 34
2.4 The Characteristics of the Assignment Problems 35
2.5 Related Mathematical Methods 36
2.5.1 Branch-and-Bound 37
2.5.2 The Primal-Dual Implicit Enumeration Method 39
2.5.3 Special Cases 39
2.6 Summary 40

3 The Diagonals Method 41

3.1 Introduction 41
3.2 The Target of the Heuristic Diagonals Method 42
3.3 The Structure of the Diagonals Method 46
3.4 Numerical Examples 47

3.5 Case Study: Hybridisation of the Diagonals Method 55

3.6 The Diagonals Method: Tie Cases 64

3.7 Summary 71

4 Further Heuristic Approaches to SAP 72

4.1 Introduction 72

4.2 The Average Cost Method 73

4.3 Case Study 78

4.4 The Addition Method 85

4.5 Tie Cases 89

4.5.1 Tie problem: Case 1 90

4.5.2 Tie problem: Case 2 90

4.5.3 Tie problem: Case 3 91

4.5.4 Tie problem: Case 4 91

4.5.5 Tie problem: Case 5 92

4.6 Case Study: Different Functions for the Addition Method 93

4.7 The Multiplication Method 98

4.8 Summary 101

5 The Genetic Algorithm 102

5.1 Introduction 102

5.2 The Genetic Algorithm for SAP 107

5.3 Implementing the Fitness Function 109

5.4 Random Permuted Numbers Method to Solve SAP 110

5.5 Random Permuted Numbers and the Hungarian Method 113
5.6 The Genetic Algorithm to Solve SAP 115
5.7 Summary 117

6 SAP with Monge Matrices 119

6.1 Introduction 119
6.2 Related Research Works 122
6.3 Monge Minimisation Algorithm 123
6.4 Monge Array 125
6.5 Monge Sequence Special Case for SAP 127
6.6 Special Case of the d-Dimensional Assignment Problem 129
6.7 Observations: 131
6.8 Summary 134

7 Conclusion 136

7.1 Conclusion 136

List of Figures

1.1 Venn diagram explains the complexity relationship 10

3.1 DM: Average Cost Comparisons 62

3.2 DM: Average Time Comparisons 62

3.3 DM: Average Total Cost Comparisons 63

3.4 DM: Average Total Time Comparisons 63

4.1 ACM: Cost Comparisons 80

4.2 ACM: Time Comparisons 80

4.3 ACM: Average Total Cost Comparisons 81

4.4 ACM: Average Total Time Comparisons 81

List of Tables

1.1 Assignment costs for four workers to do four jobs 6

1.2 Subtracting the minimum number of each row 6

1.3 Subtracting the minimum number of each column 6

1.4 The boxed positions of the workers to do the jobs 7

1.5 The optimum allocation costs 7

3.1 SAP, two factories costs matrix 43

3.2 The Construction table of Factories, Machines, Jobs and costs 46

3.3 Example 3.4.2 Costs Matrix. 51

3.4 Case Study: Large Sizes DM 56

3.5 DM: Case Study Hybridisation 57

3.6 Five instances average comparison $(4 \times 4 \times 4)$ 58

3.7 Five instances average comparison $(6 \times 6 \times 6)$ 58

3.8 Five instances average comparison $(8 \times 8 \times 8)$ 58

3.9 Five instances average comparison $(10 \times 10 \times 10)$ 59

3.10 Five instances average comparison $(12 \times 12 \times 12)$ 59

3.11 Five instances average comparison $(14 \times 14 \times 14)$ 59

3.12 Five instances average comparison ($16 \times 16 \times 16$) 59

3.13 Five instances average comparison ($18 \times 18 \times 18$) 60

3.14 Five instances average comparison ($20 \times 20 \times 20$) 60

3.15 Five instances average comparison ($22 \times 22 \times 22$) 60

3.16 Five instances average comparison ($24 \times 24 \times 24$) 60

3.17 Five instances average comparison ($26 \times 26 \times 26$) 61

4.1 Case Study: ACM Hybridisation 79

4.2 Average Method: Five instances average comparison ($4 \times 4 \times 4$) 82

4.3 Average Method: Five instances average comparison ($6 \times 6 \times 6$) 82

4.4 Average Method: Five instances average comparison ($8 \times 8 \times 8$) 82

4.5 Average Method: Five instances average comparison ($10 \times 10 \times 10$) 83

4.6 Average Method: Five instances average comparison ($12 \times 12 \times 12$) 83

4.7 Average Method: Five instances average comparison ($14 \times 14 \times 14$) 83

4.8 Average Method: Five instances average comparison ($16 \times 16 \times 16$) 83

4.9 Average Method: Five instances average comparison ($18 \times 18 \times 18$) 84

4.10 Average Method: Five instances average comparison ($20 \times 20 \times 20$) 84

4.11 Average Method: Five instances average comparison ($22 \times 22 \times 22$) 84

4.12 Average Method: Five instances average comparison ($24 \times 24 \times 24$) 84

4.13 Average Method: Five instances average comparison ($26 \times 26 \times 26$) 85

4.14 AM: The $\mathbf{A_1}$ and $\mathbf{A_2}$ total cost comparisons 94

4.15 AM: The $\mathbf{A_3}$ and $\mathbf{A_4}$ total cost comparisons. 95

4.16 AM: The Poisson $\mathbf{B_1}$ comparisons, $\lambda = 3$ for 20 instances. 96

4.17 AM: The Poisson $\mathbf{B_2}$ comparisons, $\lambda = 5$ for 20 instances. 97

5.1 GA: SAP costs matrix. 107
5.2 GA: Symbolised Costs. 108
5.3 GA:Chromosome Allocations. 108
5.4 GA: Chromosomes Matrix. 108
5.5 GA: Job Allocations. 109
5.6 GA: SAP Size 4 Problem 111
5.7 GA: Machine Allocations. 111
5.8 GA: SAP Allocation Costs Matrix. 112
5.9 GA: The SAP Allocation. 112
5.10 GA: Circled Allocation Costs 112
5.11 The SAP allocation. 114
5.12 GA: Population numbers, Hungarian method and fitness value 115
5.13 GA: The Final Allocations. 116

List of Algorithms

1 The Hungarian Method 5

2 Simulated Annealing, [1] 15

3 Genetic Algorithm, [2] 16

4 **Discrete Particle Swarm Optimisation, [3,4]** 18

5 Branch-and-Bound 38

6 The Diagonal Method 47

7 The Average Cost Method 75

8 The Addition Method 86

9 The Multiplication Method 98

10 Randomly Permute Number and the Hungarian Method 113

Acronyms

3d-AP: 3 Dimensional Assignment Problem.

ACM: The Average Cost Method.

AD: The Addition Method.

B&B: Branch-and-Bound.

BFS: Basic feasible Solution.

DM: The Diagonal Method.

EGAP: Elastic Generalised Assignment Problem.

GA: Genetic Algorithm.

GAP: Generalised Assignment Problem.

ILP: Integer Linear Programming.

LP: Linear Programming.

MM: The Multiplication Method.

MSM: Monge Sequence Matrix.

P3AP: Planar 3-Assignment Problem.

P3IAP: Planar 3-Index Assignment Problem.

PD: Primal Dual Algorithm.

RIPP: Relaxed Integer Programming Problem.

SA:The Length of Shortest Triangle Sides.

SA: Simulated Annealing.

SAP: The Solid Assignment Problem.

SI: Swarm Intelligence.

TA: The Cost of the Sum of the Triangle Length.

Chapter 1

Methods of Optimisation

1.1 Introduction

Optimisation is a mathematical process. It strives to minimise the cost of production or to maximise the efficiency of production, possibly subject to constraints. Such problems have typically many different solutions; we aim to find the optimum or near optimum solutions. Studying the constraints and special cases using the optimisation process will guide us to know if there is an optimum solution, an unbounded one or there is no solution i.e. the problem is infeasible. Proper procedures and appropriate methodology are necessary to find the optimum solution or show that it is unbounded or does not exist.

An optimisation algorithm is the process or the procedure to find an optimum or a satisfactory solution. If we have a problem with an initial solution in the first stage, it is possible to improve it in subsequent stages using an improvement procedure. We repeat the procedure until reaching a satisfactory or optimum solution.

The methods used to find the optimal solution can be classified into many categories

based on the nature of the optimisation problem. Optimisation is categorised according to the linearity and/or the non-linearity of the objective function and the constraints. For example optimisation is called quadratic if the objective function is quadratic and the constraints are linear.

Optimisation is the operation of maximising or minimising an objective function possibly subject to constraints. The mathematical representation of such a problem is

$$\begin{aligned} &\textit{maximize/minimize} \quad f(\mathbf{x}), \quad \mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbf{R}^n. \\ &\textit{Subject to}: \quad \phi_j(\mathbf{x}) = 0, \quad (j = 1, 2, ..., M), \\ &\qquad\qquad\qquad \lambda_k(\mathbf{x}) \leq 0, \quad (k = 1, 2, ..., N), \end{aligned} \tag{1.1}$$

where $f(\mathbf{x})$, $\phi_j(\mathbf{x})$ and $\lambda_k(\mathbf{x})$ are functions of $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbf{R}^n$. In (1.1), vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ are the decision variables. $f(\mathbf{x})$ is called the objective function and, $\phi_j(\mathbf{x}) = 0$ and $\lambda_k(\mathbf{x}) \leq 0$ are called the equality constraint and the inequality constraint respectively. The feasible region/search space consists all $\mathbf{x}$ points that satisfy all the constraints, [5,6].

Decision variables can be continuous, discrete or mixed i.e. both real and integer points. In continuous problems the decision variables take real values, and in the discrete ones, they take integer values. If the problem does not have any constraints, it is called an unconstrained optimisation problem. Otherwise, it is constrained, [5,6].

Definition 1.1.1 Let a real valued function f be defined over a feasible set $S \subset \mathbf{R}^n$. The solution $\mathbf{x}^*$ is said to be in the neighbourhood of all solutions if there exists an $\epsilon > 0$ such that $|\mathbf{x} - \mathbf{x}^*| < \epsilon$.

For continuous optimisation problems, ϵ is usually a small positive number larger than 0. For combinatorial problems, for instance, it can be defined as the number of changes in the permutation, [6].

Definition 1.1.2 Let a real valued function f be defined over a set $S \subset \mathbf{R}^n$. A solution $\mathbf{x}^*$ is said to be a global maximum if $f(\mathbf{x}^*) \geq f(\mathbf{x}), \forall\ \mathbf{x} \in S \subset \mathbf{R}^n$. If $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall\ \mathbf{x} \in S \subset \mathbf{R}^n$, the solution $\mathbf{x}^*$ is said to be a global minimum, [7].

Definition 1.1.3 Let a real valued function f be defined over a set $S \subset \mathbf{R}^n$. A solution $\mathbf{x}^*$ is said to be a local maximum if $f(\mathbf{x}^*) \geq f(\mathbf{x}), \forall\ \mathbf{x} \in N(\mathbf{x}^*, \epsilon)$. If $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall\ \mathbf{x} \in N(\mathbf{x}^*, \epsilon)$, then $\mathbf{x}^*$ is said to be a local minimum, where $N(\mathbf{x}^*, \epsilon)$ denotes the ϵ-neighbourhood of $\mathbf{x}^*$, [7].

The difference between AP and SAP is that, AP deals with $n \times n$ dimensions while SAP deals with $n \times n \times n$ dimensions or indices, the complexity of solving AP in linear polynomial time is $O(n^3)$ as it will be explained in the next section. The Three dimensional assignment problem is NP-hard problem and can be solved only for limited size problem not more than $n = 26$. We applied different approaches to solve SAP with large sizes problems in less time. For more details on the thee assignment problem, see Burkard, [8].

1.2 The Hungarian Method

The Hungarian method is an algorithm to solve two dimensional assignment problems (AP). To solve the problem is to assign n number of sources to n destinations allowing only one source to be assigned to one destination at the same time. The objective is to calculate the minimum allocation cost. There are $n!$ ways to assign n resources to n tasks,

that means as n becomes large the problem needs too many trials.

The assignment and the transportation problems both can be solved by using the simplex linear programming technique. The assignment is a special case of the transportation problem where the supplies and demands are equal to one. Both problems can be solved by using different algorithms, [9]. The following Theorem 1.2.1 is useful to apply the Hungarian method [10].

Theorem 1.2.1 If a number added or subtracted from all of the entries of any row or column of a cost matrix, then the optimal assignment for the resulting cost matrix is also an optimal assignment for the original matrix.

Algorithm 1 applies Theorem 1.2.1 for a given $n \times n$ cost matrix to find an optimal assignment.

Algorithm 1: The Hungarian Method

1: Subtract the row minimum from each row.
2: Subtract the column minimum from each column.
3: Cover all zeros with minimum number of lines.
4: Optimality Condition

- If the number of lines is equal to n. An optimal assignment of zeros is possible. Match the zeros to the original matrix and obtain the optimal solution. Stop.
- If the number of lines is less than n. The optimal assignment is not possible. Go to 5.

5: Select the minimum uncovered number, subtract it from each uncovered row, then add it to each covered column.
6: Go to 3.

The following Example 1.2.1 explains how to apply the two assignment Hungarian method.

Example 1.2.1 A company has four workers available to do work on four separate jobs. Only one worker can work on any one job. The cost is given in Table 1.1. The objective is to minimise the total cost of the assignment.

	Jobs			
Workers	J_1	J_2	J_3	J_4
W_1	10	04	05	09
W_2	02	06	07	08
W_3	03	05	06	02
W_4	04	03	08	07

Table 1.1: Assignment costs for four workers to do four jobs

Apply Algorithm 1, we have the subtraction of minimum number of each row table and the subtraction of minimum number of each column table.

	Jobs			
Workers	J_1	J_2	J_3	J_4
W_1	6	0	1	5
W_2	0	4	5	6
W_3	1	3	4	0
W_4	1	0	5	4

Table 1.2: Subtracting the minimum number of each row

	Jobs			
Workers	J_1	J_2	J_3	J_4
W_1	6	0	0	5
W_2	0	4	4	6
W_3	1	3	3	0
W_4	1	0	4	4

Table 1.3: Subtracting the minimum number of each column

Table 1.5 shows the final optimal costs, the total cost = 12. The first worker will do the

	Jobs			
Workers	J_1	J_2	J_3	J_4
W_1	6	0	[0]	5
W_2	[0]	4	4	6
W_3	1	3	3	[0]
W_4	1	[0]	4	4

Table 1.4: The boxed positions of the workers to do the jobs

	Jobs			
Workers	J_1	J_2	J_3	J_4
W_1	10	4	[5]	9
W_2	[2]	6	7	8
W_3	3	5	6	[2]
W_4	4	[3]	8	7

Table 1.5: The optimum allocation costs

third job, the second worker will do the first job, the third worker will do the fourth job and finally the fourth worker will do the second job. For mor examples and details [11].

1.3 Complexity and NP-Completeness

The time complexity of an algorithm is the number of operations it requires in the worst-case to solve a problem of a given size. $Big-O$ notation is used to represent the dominant aspect of the required computation. If the argument of the $O-notation$ is a polynomial function **P**, the algorithm is said to be "efficient". Otherwise, it is "inefficient" and the problem is intractable, [12].

A problem is referred to as tractable if a polynomial-time algorithm can solve it. Otherwise, it is said to be intractable; in other words, no polynomial-time algorithm can solve the problem. However, in practice, some exponential time algorithms work well on instances that have small input length, [12].

A problem is in the non-deterministic polynomial (**NP**) class, if the answer to its decision problem form can be verified in polynomial time. There is another class of problems called **NP**-hard. They do not necessarily have to be in **NP** and they do not have to be decision problems. A problem is said to be in **NP**-hard class if it can be solved in polynomial time and also another problem in **NP**-complete class can be reduced to it in polynomial time. A problem is said to be in **NP**-complete class, if it is in both **NP**-hard and **NP**-class and all other problems in **NP**-class can be reduced to it in polynomial time, [12, 13].

The SAP is known as one of the interesting and challenging problems in combinatorial optimisation. SAP models find applications in optimal allocation, minimal idling time of a rolling mill, optimal location of production plants in regions, optimum number of satellites in different directions and orbits for maximisation of the scanned regions. As described in the facility location context, non-polynomial problem deals with assigning n type-1 entities and simultaneously n type-2 locations to n destinations. The problem is to assign all entities to different destinations with the goal of minimising the total cost.

The complexity of the problems are classified based on their difficulties to solve. The problem is **P**-class if it is solved in polynomial time. The computation is bounded by the power of the problem's size and the number of steps to solve it. All decision problems are classified as **P**-class if we can solve them by a deterministic Turing machine in polynomial time. Examples of **P**-class problems are shortest path problem, assignment and transshipment, transportation problems, allocation and scheduling problems.

All decision problems are classified as **NP**-class if the answer to the instances solution is **YES** and have efficiently verifiable proof. The solutions are efficient in terms of the values

of the decision variables when it reaches the optimum and using a reliable algorithm with less calculations and in short time. An efficient and acceptable proof of the solution is also required. Hence **NP**-class is a decision problem that can be easy to verify but difficult to find an efficient solution and test them in polynomial time.

Mertens and Stephan, [14] showed that a problem is said to be **NP**-hard if an algorithm for solving it can be translated into one to solve any other **NP** problem. It is much easier to show that a problem is **NP** than to show that it is **NP**-hard. A problem which is both NP and NP-hard is called an **NP**-complete problem.

A problem is **NP**-hard if the algorithm that solve it can be used as a solution approach to any **NP**-class problem. A problem **P** is said to be **NP**-hard if all variables of **NP** polynomially reduce to **P**. A decision problem is **NP**-complete (*NPC*) when it is both in **NP** and **NP**-hard. **NP**-complete is the hardest problem of **NP** problems and it can only be solved by non-deterministic polynomial time. The following Venn diagram 1.1 shows that when $\mathbf{P} \subseteq \{\mathbf{NP} \cap \mathbf{NP}\text{-hard}\}$, **P** is **NP**-complete.

Decision making for mangers in the industry is important, problems where the answer required is either **YES** or **NO** are called decision problems. The focus is on the relations that polynomially bounded, the relation $\mathbf{R} \subseteq \{0,1\} \times \{0,1\}$ is polynomially bounded if there exist a polynomial p such that $(x, y) \in \mathbf{R}$ it holds that $|y| \leq p(|x|)$.

The polynomial class problem **P** can be defined as, let $(x, y) \in \mathbf{R}$ or state that no such y exists. Hence the class **P** related to the class of search problems that can be solvable in polynomial time or we can say that there exists a polynomial-time algorithm that given x find y such that $(x, y) \in \mathbf{R}$ or state that no such y exists Goldreich, [15]. Woeginger, [16] explained in his survey the exact algorithms for **NP**-hard problems. The exact solution can

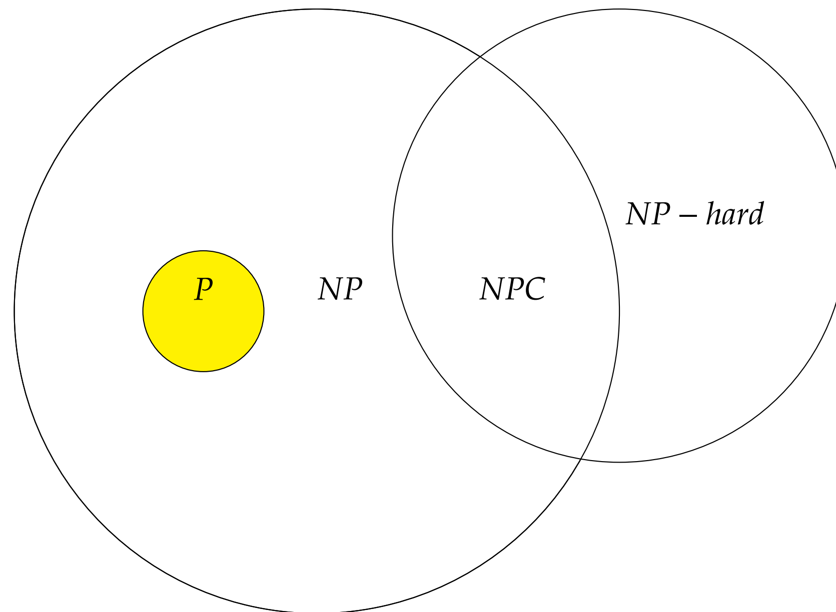

Figure 1.1: Venn diagram explains the complexity relationship

be reached in polynomial time **P** taking in consideration the input data size which relies on the usage of the computer memory storage or the execution time of the algorithm.

The second type of decision problems is called **NP**. Sometimes a problem cannot be solved because of its complexity or the solution cannot be accepted or rejected because the final outcome is unknown and there is no method to determine if the solution is correct or incorrect. SAP is **NP**-hard problem, the difficulty of solving it increases as the dimensional size or indices n of the problem increases, Karp, [17].

The minimisation problem of the special cases of geometric versions of the 3-dimensional assignment problem under general norms was also proven to be **NP**-hard, while the maximisation problem was solvable in a polynomial time, Custic et al. [18]. The third type of decision problems is called **NP**-complete, a polynomially bounded relation is defined as **NP**-complete if it is in **NP**-hard and every set in **NP**-hard is reducible to it. In the same way as testing the **NP** problems it is required at least one verifying proof to be accepted

when the answer is **YES**.

1.4 Solution Approaches to Optimisation Problems

In this section, exact methods, approximation algorithms and meta-heuristic methods will be reviewed.

1.4.1 Exact Methods

Exact methods are used to find the optimum solution for a given combinatorial problem. The drawback of these methods is that, as the size of the instance increases, the total computation time increases excessively. Nevertheless, instances of small size can be solved efficiently by these methods. Some of them will be referred to in the following methods. Branch-and-Bound (B&B) and Dynamic Programming (DP) [19, 20] are two of the classical methods that give exact optimum solutions by partially searching the feasible solution set. In B&B, a branch is a subset of solutions of the partitioned problem, and the bound is the lower bound computed that helps to find the optimum. The B&B algorithmic framework has been used successfully to find exact solutions for a wide array of optimization problems. [6, 21]. B&B uses a tree search strategy to implicitly enumerate all possible solutions to a given problem, applying pruning rules to eliminate regions of the search space that cannot lead to a better solution. There are three algorithmic components in B&B that can be specified. These components are the search strategy, the branching strategy, and the pruning rules. As DP solves sub problems, it keeps the solutions as a future reference, i.e if it finds the same sub problem uses the result that was

stored. In order to find the optimum value it starts from the bottom sub problem and goes to the main problem, which also guarantees that the sub problems are solved, [6,22].

The cutting-plane method is another rigorous approach for combinatorial problems. In this method, the feasible set is renewed by adding linear inequalities at each iteration. These inequalities are referred as cuts. This method is inefficient, and has low convergence rate, [23,24].

Although the cutting-plane method is said to be inefficient itself, it was combined with B&B; the Branch and Cut Algorithm is the result, [23,24].

1.4.2 Approximation Algorithms

Approximation algorithms for combinatorial problems do not necessarily provide an optimal solution. However, they approximate the optimum solution to within a guaranteed error value α, [25,26].

Greedy and local search algorithms are two standard approximation algorithms. In the greedy algorithm each step guarantees that the solution provided is locally optimal. The local search algorithm, on the other hand, starts with an initial solution and iteratively improves it by making changes to find a better local optimum solution, [27].

1.4.3 Meta-heuristics

Most optimisation problems, including non-trivial 3D assignment problems, are *NP*-hard. Thus, exact algorithms are inefficient and costly, especially when the problem size is large. Instead of finding the optimum solution, meta-heuristics generally find good approximations to it in acceptable computational times. For this, they are widely used

in the literature in the last decade, or so, [28]. They are, in general, a combination of the random search and the local search, [26]. Some well-known meta-heuristic methods and the ones developed recently will be reviewed in the following.

Definition 1.4.1 Heuristic

The term heuristic is used in mathematics for algorithms to solve a problem which is hard to find an exact solution. Select the best solution among all the outcomes of the heuristic algorithm used.

The presumed best solution, if it is found will need to be tested and approved in order to be accepted. The heuristic algorithms usually find a solution near to the optimal, easy to implement and fast in time. Rafael and Reinelt [29] explained different heuristic methods.

Definition 1.4.2 Meta-heuristic

It is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic algorithms [30].

1.4.4 Review of Meta-heuristic Methods

As said earlier, combinatorial or discrete optimisation problems are often computationally demanding. They more often than not belong to the so-called **NP**-hard class of problems, [12], [31]. As such, it is not reasonable to expect exact solutions to perform well when solving large and practical instances. Thus, approximation methods, heuristics and meta-heuristics, are almost the norm when it comes to solving them, [6, 32, 33].

1.4.5 Simulated Annealing

Simulated Annealing (SA) was introduced by Kirkpatrick et al., [1]. It is inspired by the annealing process of metals which involves slowing down heated metal. It is important to choose the appropriate initial temperature and the cooling down rate to avoid imperfections. Here, the temperature is denoted by t, k_B is the Boltzmann's constant, [28], and the current candidate solution c_{old} is replaced by c_{new}, if the newly generated solution is better. But, if c_{new} is worse than the current solution, then it may be accepted with probability P given by the below formula. This process is implemented for global optimisation. The cooling schedule has nothing to do with the constraints of the problem, for instance. It is there to regulate the convergence of the process to a cooled state which then points to local optima. If the cooling is too fast then convergence will not be to good local optima. If it is slow then it has more chances of finding the global optimum. The strength of SA is in its ability to get out of local optima, i.e. unstuck!

$$P(t, c_{new}, c_{old}) = e^{\frac{-(value(c_{new}) - value(c_{old}))}{k_B\ t}},\ t \geq 0, \tag{1.2}$$

to replace the current solution c_{old}. This is what allows SA to escape from the local optima. The pseudo-code of SA is given as in Algorithm 2 .

1.4.6 Genetic Algorithm

The Genetic Algorithm (GA) was developed by Holland, [2]. It is based on the idea of natural selection. The algorithm works with three operators; crossover, mutation and reproduction. Basics of GA are discussed below.

Algorithm 2: Simulated Annealing, [1]

$t \leftarrow$ initially a high temperature;
$c_{old} \leftarrow$ some initial guess;
$c_{best} \leftarrow c_{old}$;
Repeat
  $c_{new} \leftarrow$ update (c_{old});
  If value(c_{new})< value(c_{old}) or $rand[0,1] < P$ **Then**
    $c_{old} \leftarrow c_{new}$;
    Reduce the temperature t;
  End If
  If value(c_{old})¡value(c_{best}) **Then**
    $c_{best} \leftarrow c_{old}$;
  End If
Until Best solution is found, or termination criterion is reached, or $t \leq 0$;
Return Best solution c_{best} as the candidate optimum solution.

Initial Population A predetermined number of individuals is randomly generated to form an initial population. The basic GA starts with this population.

Fitness Function This measure is essential for the implementation of GA. It allows to randomise individual solutions in the population. It is often the objective function of the optimisation problem.

Selection of Parents The main idea of selection is choosing individuals from the population to be parents to new individuals. The latter are expected to be better than the parents. There are different selection methods such as the Roulette Wheel and Tournament Selection, [34].

Genetic Operators: There are three such operators.

Crossover Operator The crossover operator selects a random point which shows a position on the individual. Then, parts of two selected individuals are exchanged to generate two new individual. This procedure is called a single-point crossover. Another type is called two-point crossover. In this variant, two random positions are selected and parts of

parents are exchanged, [34].

Mutation Operator A predetermined number of individuals are mutated. This is done by changing/flipping some of the entries of an individual. This operator helps exploration in GA.

Reproduction Operator This copies good individuals into the new population as they are.

Stopping Criteria The algorithm stops when the number of generations reaches a predetermined maximum number of generations. Another commonly used stopping criterion is the maximum number of generations without improvement in the current best, [2, 26].

The pseudo-code of the algorithm is as in Algorithm 3.

Algorithm 3: Genetic Algorithm, [2]

```
f ← Objective function
Generate an initial random population of individuals (Parents),
    Repeat
        Select the number of individuals based on the rate,
        Generate new offspring using crossover (with probability p_c)
        or mutation (with probability p_m),
        Evaluate the fitness of the children,
        Update the population,
        Update the generation counter,
    Until (The stopping criteria is met)
Return Current best solution as candidate optimum.
```

1.4.7 Discrete Particle Swarm Optimisation

Discrete Particle Swarm Optimisation (DPSO) was introduced by Kennedy and Eberhart, [35]. It is based on flocking birds, fish schooling and any animals when moving as a group. Each particle in a swarm represents a solution. Each particle moves in a multidimensional search space for exploration and exploitation. In Discrete PSO or DPSO, and binary DPSO, in particular, [4], each particle is considered as a position in an N-dimensional space and

each entry of a particle position can take value 1 or 0 which mean "included" and "not included", respectively. Each particle also has a velocity vector attached to it, [3]. The velocity vector is updated at each iteration using *two* pieces of information. One is the current best, *pbest*, that a particle achieved and the other is the best kept in the memory from the beginning of the algorithm, *nbest*. The equations below are used to update the velocity and position vectors [3]:

$$v_i(t+1) = v_i(t) + \rho_1 C_1(pbest_i - X_i(t)) + \rho_2 C_2(nbest_i - X_i(t)). \tag{1.3}$$

where v_i denotes the velocity of the ith particle, and t denotes time. ρ_1 and ρ_2 are random values between $[0, 1]$ and C_1 and C_2 are learning factors.

$$X_i(t+1) = \begin{cases} 1 & \text{if } sig(v_i(t+1)) > r_i \\ 0 & \text{otherwise} \end{cases} \tag{1.4}$$

where $sig(v_i(t+1))$ is the sigmoid function,

$$sig(v_i(t+1)) = \frac{1}{1 + \exp(-v_i(t+1))}. \tag{1.5}$$

The pseudo-code of DPSO is given as Algorithm 4:

1.5 Linear, Non-Linear and Integer Programming Problem

A general optimisation problem is to select n decision variables $x_1, ..., x_n$ from a given feasible region in such a way to optimise (minimise or maximise) a given objective function,

Algorithm 4: Discrete Particle Swarm Optimisation, [3,4]

Initialize with a randomly generated $N - dimensional$ swarm with P particles
Repeat
  For all swarm i
    If $f(X_i) > f(pbest_i)$ **Then** $pbest_i = X_i$; **End If**
    If $f(pbest_i) > f(nbest_i)$ **Then** $nbest_i = pbest_i$; **End If**
  End For
  For all swarm i
    Update the velocity and the position vectors;
  End For
Until The stopping criteria is reached.
Return Best solution as candidate optimum

$$\text{Max or Min} \quad Z = f(x_1, x_2, ..., x_n), \tag{1.6}$$

The optimisation problem can be with constraints or without constraints. The general mathematical formulation for the linear programming problem is,

$$\text{Min or Max} \quad Z = c_1x_1 + c_2x_2 + ... + c_nx_n. \tag{1.7}$$

$$\text{Subject to:} \quad a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \ (\leq=\geq)\ b_1 \tag{1.8}$$

$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \ (\leq=\geq)\ b_2 \tag{1.9}$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \ (\leq=\geq)\ b_m \tag{1.10}$$

$$x_1, x_2, ..., x_n \geq 0, \tag{1.11}$$

where $x_1, x_2, ..., x_n$ are the decision variables.

c_j, $(j = 1, 2, ..., n)$ is the cost or the profit to the jth variable.

b_i, $(i = 1, 2, ..., m)$ is the requirement of the ith constraint.

$x_1, x_2, ..., x_n \geq 0$ is the set of non-negative restriction. The linear programming problem can be written in canonical form as follows;

$$\text{Max} \qquad \mathbf{Z} = \mathbf{c}^{\mathrm{T}}\boldsymbol{x} \tag{1.12}$$

$$\text{Subject to:} \quad \mathbf{A}\boldsymbol{x} \leq \mathbf{b} \tag{1.13}$$

$$\boldsymbol{x} \geq 0, \tag{1.14}$$

where $\boldsymbol{x}$ is an n-dimensional column vector of decision variables, c^T is an n-dimensional row cost or profit vector and $\boldsymbol{b}$ is an m-dimensional column requirement vector, $\mathbf{A}$ is an $m \times n$ matrix of coefficients and T is the transpose sign. The strict condition is that each component of x either zero or non-negative value.

The non-linear programming problem is similar to the linear programming. Both are formulated of objective function, constraint equations and variables bound. The difference is that the non-linear programming problems are included in at least one non-linear function either in the objective or in one or in all the constraint equations.

The Integer Programming Problem (IPP) is a branch of mathematical programming and used for many discrete problems. In real life, the variables of many problems are not always continuous. They are restricted in discrete problem to be integers. An integer programming is linear if some or all the constraint variables are integers and both the objective function and the constraints are linear. IPP generally are much more complicated than linear programming problems, Bosch, [36].

Definition 1.5.1 Uni-modular Matrix

A square integer matrix **A** is called uni-modular if $|\det \mathbf{A}| = 1$.

Definition 1.5.2 Totally Uni-modular Matrix

An integer matrix $\mathbf{A} \in \mathbf{R}^{\mathbf{m}\times\mathbf{n}}$ is totally uni-modular if every square non-singular sub-matrix of **A** is uni-modular or the determinant of each square sub matrix of **A** is $0, -1$, or $+1$.

Theorem 1.5.1 If an integer matrix $\mathbf{A} \in \mathbf{R}^{\mathbf{m}\times\mathbf{n}}$ is totally uni-modular, then every vertex solution of $\mathbf{Ax} \geq \boldsymbol{b}$ is integral.

Theorem 1.5.2 If an integer matrix $\mathbf{A} \in \mathbf{R}^{\mathbf{m}\times\mathbf{n}}$ is totally uni-modular, then both the primary and dual are integer programming problems.

From the definitions 1.5.1 and 1.5.2 and the theorems 1.5.1 and 1.5.2 it is clear that there is a relation between the totally uni-modular and the solution of the integer programming problem. If we can prove that the problem is totally uni-modular then it has a solution. For more information about uni-modular matrices, [37]. Integer programming problems can be classified into three types as follows;

1. Pure Integer Programming.
2. Mixed Integer Programming.
3. Binary or $\{0, 1\}$ Integer Programming.

The general mathematical formulation for Pure and mixed integer programming can

be written as follows,

$$\text{Max} \quad Z = \sum_{j=1}^{n} c_j x_j \tag{1.15}$$

$$\text{Subject to:} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad (i = 1, 2, ..., m) \tag{1.16}$$

$$x_j \ge 0 \quad (j = 1, 2, ..., n) \tag{1.17}$$

$$x_j \text{ integer for some or all } (j = 1, 2, ..., n). \tag{1.18}$$

The general mathematical formulation for the $\{0, 1\}$ integer programming is written as follows;

$$\text{Max} \quad Z = \sum_{j=1}^{n} c_j x_j \tag{1.19}$$

$$\text{Subject to:} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad (i = 1, 2, ..., m) \tag{1.20}$$

$$x_j \in \{0, 1\} \text{ for all } (j = 1, 2, ..., n). \tag{1.21}$$

For more details and examples about integer programming see, [9,38].

1.6 Thesis Aims and Objectives

The aims and objectives of this thesis are to investigate SAP in terms of formulation, solution approaches and implementation for large scale instances. They also include efficiency, reliability and robustness of solutions. Note that solving SAP approximately is often the only reasonable expectation since large instance as intractable. In fact, exact solutions can only expected when dealing with low dimensional instances of SAP, i.e. $n \le$

26. Furthermore, the solution to these instances are still costly in terms of computational time.

There are various methods to find an optimum or near-optimum solution for different types of SAP problems based on a number of ideas and philosophies. Broadly speaking, they can be exact or approximate, heuristic, meta-heuristic and ad hoc. The issue is to choose the right approach for a given instance. And, as said earlier, because of the **NP**-Complete nature of SAP, exact approaches are really of limited use. That is why, in this thesis, the emphasis is on approximate, heuristic and hybrid methods. Some of these already exist and we use them for comparison purposes and or in hybrid meta-heuristic. Others, however, are new and introduced for the first time here.

One of the methods introduced here, namely the Diagonals Method (DM), as well as some other heuristic approaches are applied by first converting the problem from three dimensional into the more tractable well know 2-dimensional version. This then allows us to apply and take advantage of the well understood Hungarian method. The Hungarian method is known to solve the problem in polynomial time with complexity $O(n^3)$.

The different heuristic approaches used to solve SAP in this thesis are Average Cost Method (ACM), the Addition Method (AM), the Multiplication Method (MM) and the Genetic Algorithm (GA). The aim is to understand why the methods work at all, how reliable they are, how robust, accurate and efficient compared to other approaches. The heuristic approaches DM, AM and MM are fast; for example, the total average execution time for the DM is $0.0462s$ while it is $436.7797s$ for the same problem using Branch-and-Bound; please see chapter 3, Table 3.5, for explanation. An important aspect of our work is that we can solve instances of the problem with much larger size going up to $n = 1000$;

a feasible solution is guaranteed by all these methods. Of course, their quality may well be the issue in some cases.

Although the basic feasible solution is not always close to the optimum, there is still a benefit from the speed and efficiency of the algorithm to solve the problem and get approximate solution for larger size problem.

By using different methodology, it helps to understand the complexity of the problem and why it is difficult to solve (**NP**-hard or **NP**-complete).Using different methodology help identify and compare the advantage and disadvantage of each method

Tie case may happen in any methodology. A tie case means that there are several different optimum solutions in stages. The results of the tie case will be misleading because there is no specific guide on selection which can effect the final optimum solution. The tie cases happen because of the nature of the numbers (integers, real , normal distribution or extreme). This phenomena was studied and explained with examples to show the effectiveness of tie when there are many choices to select the same cost of allocation and how it will divert the optimum solution.

Two methods have been applied using the Genetic Algorithm (GA). Both methods have two stages to obtain the fitness function. The first method generates a random permuted number in both stages. The second method generates a random permuted number function in the first stage but in the second stage the Hungarian Method has been applied. Crossover, Mutation, Elitism and Roulette wheel selection were used to generate the population using chromosomes and genes. Examples and have been given and comparisons to show the difference between the two methods.

Finally we discussed SAP with Monge Matrices. Two theorems and a lemma with

proof are established related to Monge sequence. We explained the problem by using examples and algorithms. Some useful ideas and observations also have been discussed.

1.7 Thesis Organisation

This thesis is organised as follows. In Chapter 1, we have explained some useful mathematical information about optimisation and combinatorial problems such as the integer programming problem. We also discussed the exact and approximation methods.

In Chapter 2, we have reviewed the literature and the related topics to our study. We have explained the two dimensional assignment and its algorithm. We have discussed both types of the three dimensional assignment problems, the first type is SAP and the second type is the planer problem.

Chapters 3, 4, 5 and 6 contribute the novel work of the thesis. In Chapter 3, We initiate the idea of the Diagonals Method (DM). It is a new approach to solve SAP and to find an optimum or near optimum solution.

Chapter 4, provides three further methods. The Average Cost Method (ACM) dealt with the SAP as in the DM but instead of summing the diagonals of the factories, the method is based on the relation between the amount of cost of every individual allocated worker and the average cost of the non-allocated workers. The Addition Method (AM) and the Multiplication Method (MM) are two methods discussed in Chapter 4

The Genetic Algorithm is discussed in Chapter 5. We have shown two methods to solve SAP. Permute function is used to randomly generate the selection of factories in the fitness function.

Chapter 6, provides Monge Sequence Method and explain it in detail. We have explained some aspects of the theoretical part of the problem and consider an algorithm constructed to solve SAP.

Finally, Chapter 7 presents the conclusions of this research, and avenues of future work that opens up new research and further exploration.

1.8 Summary

In this chapter we have briefly described the background to optimisation and reviewed the most common approaches to its solution. These can be exact, approximate of heuristic/meta-heuristic. The two dimensional assignment problem was explained with the Hungarian algorithm and a simple example is given and solved in section 1.2. The aims and objective of the thesis were discussed in Section 1.5. We have also presented the organisation of the thesis.

Chapter 2

Literature Review and Mathematical Background

2.1 The Two Dimensional Assignment Problem

In Chapter 1 section 1.2 we discussed briefly the two dimensional assignment problem and the Hungarian algorithm. The aim of this section is to describe the two dimensional assignment (AP) or the linear sum assignment problem (LSAP) and review the literature on the problem and the main approaches to solve it.

Definition 2.1.1 The Assignment Problem

Suppose there are n tasks, the matrix $\mathbf{C} = \{\mathbf{c_{ij}}\}$ is the allocation costs for all tasks i to agents j where (i and $j = 1, 2, ..., n$). The requirements are to assign all n tasks to all n agents such that only one task in each row is assigned to only one agent in each column. The objective function is to minimise the allocation costs, [39].

The mathematical formulation of the assignment problem can be expressed as follows. Let there are n tasks to be assigned to n agents such that one task is assigned to only one agent, let i and $j = 1, 2, ..., n$, represent tasks and agents receptively. Let $C = \{c_{ij}\}$ be the cost of performing i^{th} task by j^{th} agent. Let $X = \{x_{ij}\}$ be the number of the ith task assigned to the jth agent.

$$x_{ij} = \begin{cases} 1 & \text{if task } i \text{ is assigned to agent } j \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Total cost = $c_{11}x_{11} + c_{12}x_{12} + ... + c_{nn}x_{nn}$,

The mathematical formulation for the AP can be written as follows,

$$\text{Min Z} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}x_{ij}, \tag{2.2}$$

$$\text{Subject to:} \quad \sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \in I. \tag{2.3}$$

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

$$x_{ij} \in \{0, 1\}, \tag{2.5}$$

where I and J are two disjoint sets with $|I| = |J| = n$.

Definition 2.1.2 The Linear Sum Assignment Problem

Burkard, [40] defined LSAP through a graph theory model as follows; consider a bipartite graph $\mathbf{G} = (\mathbf{U}, \mathbf{V}; \mathbf{E})$ having a vertex of $\mathbf{U}$ for each row, $\mathbf{V}$ for each column and a cost $\mathbf{C} = \{\mathbf{c_{ij}}\}$ associated with $\mathbf{E_{i,j}}$ where (i and $j = 1, 2, ..., n$). The problem is then to determine a minimum cost perfect matching in $\mathbf{G}$ (weighted bipartite problem: find a subset of edges such that each vertex belongs to exactly one edge and the sum of the costs

of these edges is a minimum).

The two dimensional assignment problem in general can be solved by linear programming method. The transportation problem is a special case of linear programming and the assignment problem is a special case of the transportation problem where the supplies and demands are equal to one. Both the assignment and transportation problem can be solved by the simplex method, [9].

Kuhn, [41], Tucker, [39] and Munkres, [42] studied and developed an algorithm known as the Hungarian method to solve the two dimensional assignment problem, the objective was to do all n allocations at minimum cost.

The generalised assignment problem (GAP) is concerned with assigning n jobs to n machines such that each job is assigned to exactly one machine, while the total resource capacity is not exceeded. The objective function of the GAP is to minimise the total allocation cost. Fisher, [43] showed this problem to be **NP**-hard.

Nauss, [44] studied the elastic generalised assignment problem (EGAP). The version of the GAP will be affected by the cost and it will be violated if additional cost is added. Also he studied another version which allowed assessing the cost if the job was not completed in time because of not using the resources in an optimum way as required.

Martello and Toth, [45] considered the minimum-maximum version of the generalised assignment problem. Krumke and Thielen, [46] considered a variant of the generalised assignment problem; they studied in detail the complexity of different versions of the problem.

Alni, [47] presented two exact algorithms for the general assignment problem (GAP) using the Hungarian method which require $O(n^3)$ time and $O(n)$ space. She also presented

an $O(n^3)$ algorithm for solving a special case of the GAP.

2.2 The Three Dimensional Assignment Problem

The study of AP and GAP extended to the three dimensional assignment problem or SAP. The difference between AP and SAP is that AP deals with $n \times n$ dimensional problems while SAP deals with $n \times n \times n$ dimensional problems. There are two types of the three dimensional assignment problems. The first type is called the axial three dimensional assignment problem and the second is called the planar three dimensional assignment problem.

From a combinatorial point of view, one of the most important properties of SAP or multidimensional assignment problems is that they generally fall in the category of **NP**-hard problems, Garey and Johnson, [48] while AP can be solved in a polynomial time, Kuhn, [41].

The applications of SAP are common in the real world, as in scheduling workers to jobs in factories, vehicle routing, supply chain management and for many other problems.

Many methods and algorithms have been designed in the last decades such as B&B, heuristics and meta-heuristics. Pierskalla, [49] proposed the method of integer programming to solve SAP using B&B. Balas, [50] described B&B algorithm for solving the axial index assignment problem. Instead of using linear programming relaxation, their procedure was to find good lower bounds in their Lagrangian relaxation algorithm which incorporate the facet of the SAP polytopes. They used a modified sub-gradient optimisation to solve the Lagrangian relaxation. The primal dual heuristic was applied to obtain

an improved approximate solution.

Crama and Spieksma, [51] considered a distance defined on a set of points used two heuristic methods to solve the three-dimensional assignment problem and to find a minimum-weight collection of N triangles covering each point exactly once. They considered the special cases of SAP where a distance (verifying the triangle inequalities) is defined on the set of points. They presumed that (TA) is the cost of the sum of the lengths of triangle sides and (SA) is the length of its shortest sides. They proved that (TA) and (SA) were both **NP**-hard. They represented heuristics which always find a feasible solution. The algorithms were different from B&B and the heuristic reduction method. Computational experiments indicated that the performance of these heuristics was excellent on randomly generated instances of (TA) and (SA).

Gwan and Qi, [52] discussed the inequalities for the three dimensional assignment polytopes and they identified two new classes of facets in the work of Balas and Saltzman, [53] and Balas and Qi, [54]. The polytopes P of order n of SAP is mathematically defined as follows.

Consider three disjoint n-sets and the collection of all weighted triplets with one element in each n-set. The three-index assignment (or three-dimensional matching) problem asks for a minimum-weight set of triplets that partitions the union of the three n-sets. It

can be stated as $\{0, 1\}$ programming problem.

$$Min \quad \Sigma\{c_{ijk}x_{ijk} : i \in I,\ j \in J \text{ and } k \in K\} \tag{2.6}$$

$$\text{Subject to:} \quad \Sigma\{x_{ijk} : j \in J,\ k \in K,\ \forall i \in I\} \tag{2.7}$$

$$\Sigma\{x_{ijk} : i \in I,\ k \in K,\ \forall j \in J\} \tag{2.8}$$

$$\Sigma\{x_{ijk} : i \in I,\ j \in J,\ \forall k \in K\} \tag{2.9}$$

$$x_{ijk} \in \{0, 1\}, \forall\ i, j \text{ and } k, \tag{2.10}$$

where I, J and K are three disjoint sets with $|I| = |J| = |K| = n$. let $\mathbf{A}$ be the coefficient matrix for the constraints 2.7 - 2.10. Then $R = I \cup J \cup K$ is the row index set of $\mathbf{A}$. Let S be the column index set of $\mathbf{A}$. Let G_A be the intersection graph of $\mathbf{A}$. Then S is the node set of G_A. Let $\quad \mathrm{P} = \{x \in \mathbf{R}^{n^3} : \mathbf{A}x = \mathbf{e},\ \mathbf{x} \geq \mathbf{0}\}$. where $\quad e = (1, 1, ..., 1)^T \in \mathbf{R}^{n^3}$, then $P_I = \{x \in conv\{0, 1\}^{n^3} : x \in P\}$ is the three index polytopes of order n. Balas and Saltzman suggested two new facets called bull facet and comb facet [53].

Burkard, [55] studied SAP with decomposable cost coefficients. In their paper they investigated a special case of SAP in which they can decompose the cost coefficients d_{ijk} into the product of three values a_i, b_j and c_k. The maximization problem was proved to be solved in polynomial time by sorting the cost coefficients sequences a_i, b_j and c_k in increasing order. They used Hardy, [56] proposition about the n elements permutations sequences and other lemmas and theorems to prove that minimisation problem of SAP remains **NP**-hard. They had considered some special cases which were solved in polynomial time.

Cao, [57] proposed an efficient implementation of the Munkres algorithm for the assignment problem using MATLAB. We have used this code as a function to find the

optimum allocation for $n \times n$ dimensional factories.

Hahn, [58] studied the exact solution of emerging quadratic assignment problem (EQAP) and suggested a taxonomy that provided a framework to help using and extending the problem.

Anuradha and Pandian, [49] proposed a reduction method which is not based on the Hungarian assignment for finding an optimal or near optimal solution to SAP. We compared the result of the example they had used with our DM algorithm.

Pandian and Kavitha, [59] studied the sensitivity analysis of the fuzzy solid assignment problem; they explained different methods of solving type II sensitivity analysis. It was demonstrated the parametric method provides a better type II sensitivity range than the labelling algorithm.

Easterfield, [60] presented an algorithm for AP. It was a non-polynomial $O(2^n n^2)$ time approach, based on iterated application of a particular class of admissible transformations.

Kuhn, [41] and Tucker, [39] presented the famous Hungarian method, the Primal-Dual (PD) algorithm was applied and it was the first polynomial-time method to solve AP. The original formulation solves the problem in $O(n^4)$ time. Munkres, [42] used a different and improved algorithm to solve the AP.

Edmonds and Karp, [61] showed that shortest path computations on the reduced costs produce an $O(n^3)$ time algorithm for the AP.

The first $O(n^3)$ algorithm for AP had been introduces by Dinic and Kronrod, [62]. But the best time complexity for a Hungarian algorithm is $O(n^3)$ proposed by Lawler in 1976 [10].

2.3 Types of the Solid Assignment Problems

In this section we introduce two types of SAP, the first is the axial 3-index assignment problem and the second is called the planar 3-index assignment problem. The SAP is famous and well known problem in combinatorial optimisation field of study and has been investigated thoroughly in the literature, [63]. SAP was introduced by Pierskalla, [64] used B&B algorithm to solve the problem. Pierskalla explained that the extended dimension could be space or time [49].

2.3.1 The 3-Index Assignment Problem

The first type of SAP or the 3-index assignment problem can be defined as follows.

Definition 2.3.1 The First Type of the SAP

Given three disjoint n-sets I, J, K and a weight function $\omega : I \times J \times K \to \mathbf{R}^+$, it asks for a collection of triples $M \subseteq I \times J \times K$ such that each element of each set appears is exactly in one triple, and the function ω is minimized.

The first type of SAP mathematical formulation is as follows.

$$\text{Minimise Z} = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \omega_{ijk}x_{ijk}. \tag{2.11}$$

$$\text{Subject to:} \quad \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1, \quad \forall i \in I, \tag{2.12}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1, \quad \forall j \in J, \tag{2.13}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1, \quad \forall k \in K, \tag{2.14}$$

$$x_{ijk} \in \{0, 1\}, \ \forall\, i \in I, \ j \in J \ \text{ and } k \in K. \tag{2.15}$$

Let $\mathbf{A^n}$ denote the $\{0, 1\}$ matrix corresponding to the constraints 2.12 - 2.15, which has n^3 columns and $3n$ rows. Notice that hereafter n denotes the cardinality of each set being "assigned", hence the number of variables is n^3. For a survey on the 3-index assignment problem and more details, see Spieksma, [63]. Karp, [17] showed that the SAP is **NP**-hard. Spieksma, [65] defined the geometric three dimensional assignment problems as follows.

Definition 2.3.2 The Geometric 3-Dimensional Assignment Problem

Given three sets $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$, each set with $n \times n$ elements, and a cost function $\mathbf{c} : \mathbf{A} \times \mathbf{B} \times \mathbf{C} \rightarrow \mathbf{R}^{+}$. The goal is to partition $\mathbf{A} \cup \mathbf{B} \cup \mathbf{C}$ into n three triples $t_i = (a_{j(i)},\ b_{r(i)},\ c_{l(i)})$ such that to minimise $\sum_{i=1}^{n} c(t_i)$.

2.3.2 The Planar 3-Assignment Problem

The second type is the Planar 3-Assignment Problem (P3AP) or the Planar 3-Index Assignment Problem (P3IAP). The objective of the P3AP is to find the minimum assignment cost among the elements of three distinct sets demanding that every pair of elements, each representing a different set, appears in the same solution exactly once, in other words the P3AP is defined as follows.

Definition 2.3.3 The Planar 3-Assignment Problem

Given three n-sets $\mathbf{A_1}$, $\mathbf{A_2}$ and $\mathbf{A_3}$. For each triple in $\mathbf{A_1} \times \mathbf{A_2} \times \mathbf{A_3}$ there is a real weight number ω_{ijk} known for all $i \in I$, $j \in J$ and $k \in K$. The problem is to find n^2 triples such that each pair of elements from $(\mathbf{A_1} \times \mathbf{A_2}) \cup (\mathbf{A_1} \times \mathbf{A_3}) \cup (\mathbf{A_2} \times \mathbf{A_3})$ is in exactly one triple. Hence the required output in the P3AP is to find n^2 triples containing each pair of indices exactly once. The optimum solution is either to maximise or to minimise the sum of the weights

ω_{ijk}.

The planar formulation of P3AP as an integer linear program is as follows. Given real weights ω_{ijk} find a vector $x \in \mathbf{R}^{n^3}$ which satisfies

$$\text{Minimise} \quad Z = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \omega_{ijk} \, x_{ijk}. \tag{2.16}$$

$$\text{Subject to:} \quad \sum_{i=1}^{n} x_{ijk} = 1, \quad \forall \, j, \, k \in I \tag{2.17}$$

$$\sum_{j=1}^{n} x_{ijk} = 1, \quad \forall \, i, \, k \in J \tag{2.18}$$

$$\sum_{k=1}^{n} x_{ijk} = 1, \quad \forall \, i, \, j \in K \tag{2.19}$$

$$x_{ijk} \in \{0, 1\}, \; \forall \, i \in I, \; j \in J \; \text{ and } k \in K \tag{2.20}$$

Frieze, [66] proved that P3AP is **NP**-complete even when $\omega_{ijk} \in \{0, 1\}$.

2.4 The Characteristics of the Assignment Problems

Until now, the assignment problems have been dealt with independently. A lot of attempts were given to developing a unified approach, and different types of researches were done in the last decades. The individual problems of the assignment can be divided into four by the following characteristics.

1. Dimensionality:

   The dimension of the objective function; as it is known, the assignment problems can be two, three or multidimensional problems.

2. Degree:

   The objective function varies from linear, quadratic, cubic, bi-quadratic and higher degrees.

3. Mapping:

   In some problems, the allocations are one-to-one such as two dimensional assignment problems while they are many-to-one in the generalised assignment problem.

4. Linearisation:

   Linearisation and relaxation options which depend on the level of reformulation required.

2.5 Related Mathematical Methods

There are many methods to find the exact or approximate solution of SAP. We will explain some of these methods and the differences between them. We will also discuss the characteristic and the complexity of each one. The methods are divided into three parts as follows.

1- Branch-and-Bound (B&B).

2- Primal-Dual Implicit Enumeration.

3- Special Cases.

2.5.1 Branch-and-Bound

The Branch-and-Bound (B&B) technique is a common algorithm to solve the integer programming problems. The basic concept is to branch or divide the problem into sub-problems and conquer (fathom) the unrequired branch in each stage. By partitioning the feasible solutions, it is dividing or branching the problem into two sets. Bounding the good or the best solution in the subset, conquered (fathomed) the worst or the unrequired set. Continue this process until the optimum solution is reached. The classical B&B is able to solve both discrete and continuous problems, it was first introduced by Land and Doig, [67].

The algorithm application varies from problem to another depending on the specificity of each problem. B&B uses the basic tree enumeration method. It first solves a linear programming problem by relaxing the integral conditions; if the resultant solution is an integer, then the problem is solved; otherwise a tree search proceeds.

The disadvantage of the B&B technique, is that is much slower than other approximate methods and it often leads to exponential time but if it is used in a proper way and applied carefully it can run fast and reach an optimum solution in a reasonable time.

Poole, [68] defined the B&B search as to maintain the lowest-cost for a problem. The objective and the cost can be reached through a certain path of a tree.

Suppose that Z_{LP} and Z_{RIPP} are the objective functions of the linear programming and the relaxed integer programming problem respectively, then the concepts of bound on the optimal integer value are:

- For maximisation problem, the optimum relaxed objective value function is an upper

bound on the integer value.

$Z_{LP} \geq Z_{RIPP}$, an upper bound.

- For minimisation problem, the optimal relaxed objective value function is a lower bound on the optimal integer value.

  $Z_{LP} \leq Z_{RIPP}$, a lower bound.

- The optimal solution to a *LP* relaxation of an RIPP gives us a bound on the optimal RIPP functions value.

Suppose this cost is bound. If the search encounters a branch or a path p such that cost $(p) + h(p) \geq$ bound, path p can be pruned. If a non-pruned path to a goal is found, it must be better than the previous best path. This new solution is remembered and the bound is set to the cost of this new solution. Then the search continues until you reach the best. The following B&B Algorithm 5 explains how to solve an integer programming relaxation.

Algorithm 5: Branch-and-Bound

1: Relax the integer programming problem (RIPP).
2: Divide a problem into subproblems.
3: **IF** RIPP has no feasible solution, then there is no solution. **END IF**
4: **IF** RIPP has an integer feasible solution,
compare the feasible solution with the best solution known (the incumbent).
5: **IF** RIPP has a solution that is worse than the incumbent,
change the incumbent to be the best solution.
6: **IF** RIPP has an optimal solution but not all integer are better than the incumbent,
divide this subproblem further.
7: Go to 2.

2.5.2 The Primal-Dual Implicit Enumeration Method

Balas, [69] used the method of implicit enumeration which is often used to solve integer programming problems. It is based on the fact that each variable must be equal to 0 or 1 to simplify both the branching and bounding components.

The tree technique used in implicit enumeration is similar to B&B and the branching variable must be either 0 or 1. Suppose a $\{0, 1\}$ problem has n variables then it has exactly 2^n possible solutions and this makes it difficult to examine all the possibilities especially if n is large. Using the tree search will help to examine some of the feasible solutions and exclude the infeasible or unrequired solutions.

2.5.3 Special Cases

Crama and Spieksma, [51] considered that the SAP where the cost coefficients fulfil some special triangle inequalities. This makes the problem easier to approximate but it still remains **NP**-hard.

Burkard, [55] considered that SAP case with decomposable cost coefficients $\omega_{ijk} = a_i b_j c_k$. So given three n-elements sequences a_i, b_i and c_i of non-negative numbers, there are two permutations ϕ and ψ such that $\sum_{i=1}^{n} a_i b_{\phi(i)} c_{\psi(i)}$ reaches its minimum (maximum)

respectively. The formulation of the SAP will be as follows.

$$\text{Min (Max)} \qquad Z = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} a_i b_j c_k x_{ijk}. \tag{2.21}$$

$$\text{Subject to:} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1,\ \forall k = 1, ..., n, \tag{2.22}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1, \forall j = 1, ..., n, \tag{2.23}$$

$$\sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1, \forall i = 1, ..., n, \tag{2.24}$$

$$x_{ijk} \in \{0, 1\}. \tag{2.25}$$

The change of the decomposable cost coefficient at the objective function $\omega_{ijk} = a_i b_j c_k$ will affect the computational complexity.

2.6 Summary

In this chapter we went through the literature and the mathematical background related to the linear assignment and the three dimensional assignment problems. Many methods and algorithms have been designed in the last decades such as B&B method, heuristics and meta-heuristics.

We have discussed the general linear assignment problem and two types of the SAP, the first SAP is the 3-index assignment problem and the second is the planar 3-dimensional assignment problem. We also listed four characteristics of the SAP, the related mathematical methods and mentioned some special cases to solve the SAP.

Chapter 3

The Diagonals Method

3.1 Introduction

The Diagonals Method (DM) is a new heuristic for solving SAP. The aim is to find an optimal or near optimal solution for the three dimensional assignment problems. In the last several decades there were many attempts of solving this type of assignment problem.

The SAP can be solved exactly for small size instances but it becomes more difficult when the size of the problem is larger. Balas and Saltzman, [50] used the Lagrange relaxation method to find the exact solution for the first type of SAP, the axial three dimensional problem up to size $n = 26$. There were many attempts to solve the problems in special cases, [55]. Pierskalla used the B&B algorithm tree technique to solve the problem. The SAP was mentioned by Pierskalla in 1967, [64].

The DM is a fast algorithm for larger problems. Like all heuristics, we guarantee a feasible solution, but we sacrifice optimality for solution efficiency. We have tested the problem for n size up to 1000 and we obtain feasible solutions in few seconds. Obtain-

ing good solution, fast is important compared to obtaining exact ones only for limited dimensions in lengthy procedures requiring long time to process.

The DM is based on counting the maximum summation of both the diagonal and the anti-diagonal cost of each factory. Then we have rearranged the factories in descending or ascending order according to their maximum diagonal summation costs, that means the factory with the maximum summation of its diagonal or anti-diagonal cost will be allocated first. After rearranging the factories in descending or ascending order we allocate them individually one by one using the Hungarian method. The DM and how to apply the Hungarian algorithm will be explained with examples in more details later in this chapter.

We have used MATLAB software to generate and run our code, the Munkres function also had been used which implemented by Cao, [57] to assign the required allocations.

3.2 The Target of the Heuristic Diagonals Method

The target of applying DM is to reach an optimum (minimum cost) or near optimum solution to SAP and determine the best allocation. Although the three-dimensional SAP is **NP**-Hard, the DM was successfully applied to allocate and schedule a problem with size up to $n = 500$ with a reasonable elapsed time. The characteristic property of DM algorithm is to re-arrange the initial sources (Factories) according to their maximum diagonal and anti-diagonal costs in descending or ascending order. The other important property for the DM is converting the problem from three dimensions into two and solving it by applying the Hungarian method. As it is known that the Hungarian method solves the

problem in polynomial time with complexity $O(n^3)$, [47].

Solving SAP with size $n = 2$ is easy; simply count all the four possible allocations and select the minimum cost allocation. It is always easy to do that and have an exact solution, because we are simply selecting the minimum from all the four ($2^2 = 4$) possible combinations. To explain how to reach an optimum solution let us assume that we have two factories. In each factory there are two machines, our target is to assign two workers such that each worker does only one job on only one machine in only one factory and vice versa. The given costs matrix $C = \{c_{ijk}\}$ is to allocate the job J_i to the machine M_j in the factory F_k where (i, j and $k = 1, 2$). The cost $C = \{c_{ijk}\}$ for each factory is explained in Table 3.1. The optimal solution can be reached by selecting the minimum sum of the possible

Factories	F_1		F_2	
Machines	M_1	M_2	M_1	M_2
Jobs J_1	c_{111}	c_{121}	c_{112}	c_{122}
J_2	c_{211}	c_{221}	c_{212}	c_{222}

Table 3.1: SAP, two factories costs matrix

four combinations that satisfy the constraints of SAP, while the other combinations are either similar to the selected combination or do not satisfy the constraints. In the following table we have explained how to select the minimum sums of the allocated costs.

$$Z = \text{Min}\,\{c_{111} + c_{222},\ c_{121} + c_{212},\ c_{112} + c_{221},\ c_{122} + c_{211}\} \tag{3.1}$$

As we have mentioned earlier, SAP was solved by Pierskalla in 1967 using B&B method. Problems with sizes up to $n = 26$ have been solved with exact solution. The reason for this bound of the size n is the large combination and the requirement of large memories for the input and output data during the search procedure of the B&B. We can solve SAP

for small sizes n number problems but it will be too difficult to solve when n becomes large and it will be hard to find an exact solution when the size of the problem increases. For example to solve the SAP with a size $n = 26$, the number of combinations of the solutions of the problem is equal to 2^{26}. The problem can be solved and reached an exact solution for $n = 26$ using the B&B method. It is clear that the number of combination is $2^{26} = 67,108,864$. Imagine the number of combinations when the size of the problem increased to $n = 27$, the number of combinations is $2^{27} = 134,217,728$. It is almost twice the problem's size $n = 26$.

Hence the objective of our work is to solve SAP and due to the way B&B do the search, our investigation aim is to develop algorithms that are based on a different search strategy to be able to solve bigger size of SAPs.

We have used the outcomes of the B&B method to help us test all the methods that we have established and compare their results. Mart'i, [29] suggested the error percentage deviation

$$\text{Percentage Deviation} = \mid (C_{opt} - C_A)/C_{opt} \mid * 100, \qquad (3.2)$$

where C_A is the approximate value of the solution delivered by the heuristic method and C_{opt} is the optimum value of the given example, as a measure of robustness.

Definition 3.2.1 Upper and Lower Diagonal Lines

Let $\mathbf{C} = \{\mathbf{c_{ij}}\}$ be a square matrix such that

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{pmatrix},$$

where ($i,$ and $j = 1, 2, 3, \cdots, n$). The summation of the diagonal elements in **C** is an upper diagonal line (UDL) or simply an anti-diagonal line such that.

$$UDL = \sum_{\substack{i=1 \\ j=(n-i+1)}}^{n} c_{ij} = c_{1n} + c_{2(n-1)} + \cdots + c_{n1}. \tag{3.3}$$

The summation of the diagonal elements in C is a lower diagonal line (LDL) or simply a diagonal line such that

$$LDL = \sum_{i=1}^{n} c_{ii} = c_{11} + c_{22} + \cdots + c_{nn}. \tag{3.4}$$

Example 3.2.1 This example is explaining both UDL and LDL.

Let $\mathbf{C} = \{\mathbf{c_{ij}}\}$ be the costs of a square matrix with dimension 3×3 such that

$$\mathbf{C} = \begin{pmatrix} 3 & 6 & 5 \\ 6 & 1 & 7 \\ 8 & 2 & 9 \end{pmatrix}.$$

UDL = 5 + 1 + 8 = 14 and LDL = 3 + 1 + 9 = 13.

3.3 The Structure of the Diagonals Method

The Diagonals Method (DM) is a new heuristic approach to solve the SAP. The benefit of the DM is that it is easy to construct. It is also fast. Therefore, we can use it for large dimension problems. The basic feasible solution can be used as a useful and good initial starting solution for the B&B method for instance.

SAP can be constructed of n factories and there are n machines in each factory. We have n workers and each is responsible to do one job on one machine in one factory. For each job to be done by the workers there is a cost c_{ijk} to be consider for each worker doing the job in the factory. The SAP can be constructed as it is shown in Table 3.2. The problem consists of n factories and there are n machines in each factory. We have n workers and each worker is responsible for only one job on only one machine in only one factory. For each job to be done by the workers there is a cost c_{ijk} to be consider for each worker doing the job in the factory.

Factories	F_1	F_2	$\cdots$	F_n
Machines	M_1 M_2 $\cdots$ M_n	M_1 M_2 $\cdots$ M_n	$\cdots$	M_1 M_2 $\cdots$ M_n
J_1	c_{111} c_{121} $\cdots$ c_{1n1}	c_{112} c_{122} $\cdots$ c_{1n2}	$\cdots$	c_{11n} c_{12n} $\cdots$ c_{1nn}
Jobs J_2	c_{211} c_{221} $\cdots$ c_{2n1}	c_{212} c_{222} $\cdots$ c_{2n2}	$\cdots$	c_{21n} c_{22n} $\cdots$ c_{2nn}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
J_n	c_{n11} c_{n21} $\cdots$ c_{nn1}	c_{n12} c_{n22} $\cdots$ c_{nn2}	$\cdots$	c_{n1n} c_{n2n} $\cdots$ c_{nnn}

Table 3.2: The Construction table of Factories, Machines, Jobs and costs

The following Algorithm 6 explains the steps of the Diagonal Method.

Algorithm 6: The Diagonal Method

1: Initialise the problem, set $n \geq 2$. to be the number of jobs, machines and factories.
2: If SAP is not balanced add zero costs to the rows or columns as appropriate to balance it.
3: Let i, $j$and $k = 1, 2, ..., n$ be the indices number of the jobs, machines and factories respectively.
4: Let the matrix $\mathbf{C} = \{\mathbf{c_{ijk}}\}$ be the allocation costs, where i, j and $k = 1, 2, ..., n$.
5: Select the maximum sum of the upper (UDL) or the lower (LDL) diagonal line for each factory individually.
6: Re-arrange the factories in descending or ascending order according to their maximum diagonal sum.
7: Apply the Hungarian method to the first factory and assign the worker with the minimum cost c_{ijk} to do the job J_i on the machine M_j in the factory F_k.
8: Remove F_1 completely, delete J_i and all M_j from all the remaining factories.
9: If two factories remain go to 10 otherwise go to 7.
10: Calculate the allocation cost of the last two factories as follows;

$$\begin{pmatrix} c_{111} & c_{121} & & c_{112} & c_{122} \\ c_{211} & c_{221} & & c_{212} & c_{222} \end{pmatrix}.$$

Select the minimum cost from the four combinations set,
$\{c_{111} + c_{222}, c_{121} + c_{212}, c_{112} + c_{221}, c_{112} + c_{211}\}$.
11: Sum all the allocation costs from 7 and 10. Stop.

3.4 Numerical Examples

In this section, numerical examples are presented to explain the methodology of the proposed DM and to discuss the different observations that arise in these examples.

Example 3.4.1 In this example we will apply the DM to solve SAP by applying the Hungarian method, the Definition 3.2.1 and the Algorithm 6. Let $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ be three square

matrices, each of size $n = 3$ as follows.

$$\mathbf{C}_1 = \begin{pmatrix} 4 & 2 & 3 \\ 6 & 4 & 3 \\ 2 & 1 & 4 \end{pmatrix}, \quad \mathbf{C}_2 = \begin{pmatrix} 5 & 2 & 1 \\ 3 & 1 & 6 \\ 6 & 2 & 4 \end{pmatrix}, \quad \mathbf{C}_3 = \begin{pmatrix} 4 & 3 & 1 \\ 2 & 5 & 2 \\ 3 & 1 & 4 \end{pmatrix}.$$

Put the matrices in descending order according to their maximum UDL or LDL as it was explained in Definition 3.2.1 and calculate the total optimal cost in each matrix using the Hungarian method. Then we will find the assignment allocations for each matrix taking in consideration SAP constraints.

From $\mathbf{C}_1$ the maximum value of the sum of the upper and lower diagonal lines are as follows; UDL = 3 + 4 + 2 = 9 and LDL = 4 + 4 + 4 = 12, the Max{UDL, LDL} = 12. Applying the Hungarian method, the row vector index is [2 3 1] or it can be represented as $\mathbf{C(row, column)}$; $\mathbf{C_1(1, 2) = 2}$, $\mathbf{C_1(2, 3) = 3}$ and $\mathbf{C_1(3, 1) = 2}$. The total assignment cost is the sum of all allocation costs, 2 + 3 + 2 = 7.

Repeating the same procedure on $\mathbf{C}_2$ we will have; UDL = 1 + 1 + 6 = 8 and LDL = 5 + 1 + 4 = 10, the Max {UDL, LDL} = 10. Applying the Hungarian method, the row vector allocation's index is [3 1 2] or $\mathbf{C_2(1, 3) = 1}$, $\mathbf{C_2(2, 1) = 3}$ and $\mathbf{C_2(3, 2) = 2}$. The total assignment cost is the sum of all allocation costs, 1 + 3 + 2 = 6.

Repeating the same procedure on $\mathbf{C}_3$, we have; UDL = 1 + 5 + 3 = 9 and LDL = 4 + 5 + 4 = 13. The Max{UDL, LDL} = 13. Applying the Hungarian method, we have, the row vector allocation's index is [3 1 2] or $\mathbf{C_3(1, 3) = 1}$, $\mathbf{C_3(2, 1) = 2}$ and $\mathbf{C_3(3, 2) = 1}$. The total assignment cost is the sum of all allocation costs, 1 + 2 + 1 = 4.

Hence we have the set $\{12, 10, 13\}$ of maximum values of UDL and LDL for $\mathbf{C}_1, \mathbf{C}_2$ and $\mathbf{C}_3$

respectively. Rearrange them in descending order we have $\mathbf{C_3}$, $\mathbf{C_1}$ and $\mathbf{C_2}$. Assign the optimum allocation's values to the first matrix $\mathbf{C_3}$ using the Hungarian assignment method to do the allocation as follows;

$$\mathbf{C_3} = \begin{pmatrix} 4 & 3 & 1 \\ 2 & 5 & 2 \\ 3 & 1 & 4 \end{pmatrix}, \ \mathbf{C_1} = \begin{pmatrix} 4 & 2 & 3 \\ 6 & 4 & 3 \\ 2 & 1 & 4 \end{pmatrix}, \ \mathbf{C_2} = \begin{pmatrix} 5 & 2 & 1 \\ 3 & 1 & 6 \\ 6 & 2 & 4 \end{pmatrix}.$$

Apply the Hungarian assignment method. The allocation values for the first matrix $\mathbf{C_3}$ is

$$\mathbf{C_3} = \begin{pmatrix} - & - & 1 \\ 2 & - & - \\ - & 1 & - \end{pmatrix}.$$

Select the first minimum number $\mathbf{C_3(1,3)} = \mathbf{1}$, remove matrix $\mathbf{C_3}$ then delete the first row and the third column from the remaining matrices $\mathbf{C_1}$ and $\mathbf{C_2}$. The deletions we made will prevent the other workers from doing the job allocated to the first worker subject to the constraints. The remaining matrices are as follows;

$$\mathbf{C_1} = \begin{pmatrix} 6 & 4 \\ 2 & 1 \end{pmatrix}, \ \mathbf{C_2} = \begin{pmatrix} 3 & 1 \\ 6 & 2 \end{pmatrix},$$

or their concatenated 2×4 matrix

$$\mathbf{C_1 C_2} = \begin{pmatrix} 6 & 4 & & 3 & 1 \\ & & & & \\ 2 & 1 & & 6 & 2 \end{pmatrix}.$$

As mentioned in Algorithm 6, only two matrices remain; apply step 10 and select the minimum cost from the four choices set of the matrix $\mathbf{C_1}$ and $\mathbf{C_2}$ as follows; $\{c_{111} + c_{222}, c_{121} + c_{212}, c_{112} + c_{211}, c_{122} + c_{211}\}$ and $c_{122} + c_{211} = 1 + 2 = 3$,
is the minimum cost allocated in the last two remaining matrices. Let us assume that the matrices $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ be the cost values for three factories named F_1, F_2 and F_3 respectively. Let the jobs J_1, J_2 and J_3 be the jobs to be assigned to the machines M_1, M_2 and M_3 respectively in each factory.

To obtain the minimum allocation in each factory, the cost we have obtained from $\mathbf{C_3}$ is 1. It is obvious that the first worker is allocated to do J_1 on M_3 in F_3, the second worker is allocated to do J_2 on M_2 in F_2 and the third worker is allocated to do J_3 on M_1 in F_1 or we can write the optimal allocation in all the factories are as follows.

Factory 1 : Job 3 : Machine 1 = 02.

Factory 2 : Job 2 : Machine 2 = 01.

Factory 3 : Job 1 : Machine 3 = 01.

The total minimum cost = 2 + 1 + 1 = 4.

Example 3.4.2 Anuradha and Pandian, [70] solved this example by a reduction method. We will solve it using DM. Suppose that there are three factories denoted by F_1, F_2 and F_3, three machines denoted by M_1, M_2 and M_3, and three jobs denoted by J_1, J_2 and J_3. It is known that c_{ijk} is the cost of the assigning Job J_i to be performed by Machine M_j in

the Factory F_k. Besides, three machines, three factories and three jobs can be associated with only one of the others, that is, only one job on only one machine in only one factory. The assignment costs c_{ijk} are given in Table 3.3. We use the diagonals method to compare the solution with the result obtained by D. Anuradha and P. Pandian. It is only a basic feasible solution not an optimum.

Factories		F_1			F_2			F_3	
Machines	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3
J_1	10	08	12	09	10	27	15	10	13
Jobs J_2	08	06	07	09	06	12	07	11	12
J_3	09	07	06	10	07	12	08	06	08

Table 3.3: Example 3.4.2 Costs Matrix.

Let the matrices $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ be the cost matrices for the three combinations of jobs (J_1, J_2, J_3), machines (M_1, M_2, M_3) and factories (F_1, F_2, F_3) respectively, then calculate the upper and lower diagonal lines for each factory. Let $\mathbf{C_1}$ be the cost matrix for factory $\mathbf{F_1}$ such that,

$$\mathbf{C_1} = \begin{pmatrix} 10 & 08 & 12 \\ 08 & 06 & 07 \\ 09 & 07 & 06 \end{pmatrix}.$$

LDL = 22 and UDL = 27. Let the maximum diagonal value be $d_1 = Max\{22, 27\} = 27$. The Hungarian allocation cost = 10 + 6 + 6 = 22. Let $\mathbf{C_2}$ be the cost matrix for factory F_2 such that

$$\mathbf{C_2} = \begin{pmatrix} 09 & 10 & 27 \\ 09 & 06 & 12 \\ 10 & 07 & 12 \end{pmatrix}.$$

LDL = 27 and UDL = 43. Let the maximum diagonal value be $d_2 = Max\{27, 43\} = 43$.

The Hungarian allocation cost = 09 + 06 + 12 = 27.

Let $\mathbf{C_3}$ be the cost matrix for factory F_3 such that

$$\mathbf{C_3} = \begin{pmatrix} 15 & 10 & 13 \\ 07 & 11 & 12 \\ 08 & 06 & 08 \end{pmatrix}.$$

LDL = 34 and UDL = 32. Let the maximum diagonal value be $d_3 = Max\{34, 32\} = 34$.

The Hungarian allocation cost = 10 + 7 + 8 = 25.

Rearranging the maximum diagonal values d_1, d_2 and d_3 for the three factories F_1, F_2 and F_3 in descending order, we get $\mathbf{C_2}$, $\mathbf{C_3}$ and $\mathbf{C_1}$.

$$\mathbf{C_2} = \begin{pmatrix} 09 & 10 & 27 \\ 09 & 06 & 12 \\ 10 & 07 & 12 \end{pmatrix}, \ \mathbf{C_3} = \begin{pmatrix} 15 & 10 & 13 \\ 07 & 11 & 12 \\ 08 & 06 & 08 \end{pmatrix}, \ \mathbf{C_1} = \begin{pmatrix} 10 & 08 & 12 \\ 08 & 06 & 07 \\ 09 & 07 & 06 \end{pmatrix}.$$

Starting from the first descending order factory cost $\mathbf{C_2}$. The minimum assignment cost of factory $\mathbf{C_2}$ is 06 and it is allocated to J_2 in M_2.

$$\mathbf{C_2} = \begin{pmatrix} 09 & - & - \\ - & 06 & - \\ - & - & 12 \end{pmatrix}.$$

Delete the second row and the second column from both $\mathbf{C_3}$ and $\mathbf{C_1}$ respectively and follow

the steps of Algorithm 6 we have,

$$\mathbf{C_3} = \begin{pmatrix} 15 & 13 \\ 08 & 08 \end{pmatrix}, \ \mathbf{C_1} = \begin{pmatrix} 10 & 12 \\ 09 & 06 \end{pmatrix},$$

or $\mathbf{C_3C_1}$ the concatenated 2×4 matrix,

$$\mathbf{C_3C_1} = \begin{pmatrix} 15 & 13 & & 10 & 12 \\ 08 & 08 & & 09 & 06 \end{pmatrix}.$$

Since only two factories are left, as mentioned in step 10 of Algorithm 6, we select the minimum cost from the four choices set of the two factories F_3F_1 costs matrix $\mathbf{C_3C_1}$ as follows;

$\{15 + 06, 13 + 09, 10 + 08, 12 + 08\}$. Hence, $10 + 08$ is the minimum cost allocated in the last two remaining factories, where 10 is the minimum cost in F_1 and 08 is the minimum cost in F_3. The allocation in all the factories are as follows.

Factory 1 : Job 1 : Machine 1 = 10.

Factory 2 : Job 2 : Machine 2 = 06.

Factory 3 : Job 3 : Machine 3 = 08.

Total minimum cost = 24.

Let solve the problem by arranging the factories in ascending order $\mathbf{C_1}, \mathbf{C_3}$ and $\mathbf{C_2}$.

$$\mathbf{C_1} = \begin{pmatrix} 10 & 08 & 12 \\ 08 & 06 & 07 \\ 09 & 07 & 06 \end{pmatrix}, \ \mathbf{C_3} = \begin{pmatrix} 15 & 10 & 13 \\ 07 & 11 & 12 \\ 08 & 06 & 08 \end{pmatrix}, \ \mathbf{C_2} = \begin{pmatrix} 09 & 10 & 27 \\ 09 & 06 & 12 \\ 10 & 07 & 12 \end{pmatrix}.$$

Starting from the first ascending order factory cost $\mathbf{C_1}$. The minimum assignment cost of factory one $\mathbf{C_1}$ is 06 and it is allocated to J_2 in M_2. There is a tie as the minimum value 6 appears in two allocations but we will select only the first minimum value of the row vector of the Hungarian assignment allocation [10 06 06]

$$\mathbf{CF_1} = \begin{pmatrix} 10 & - & - \\ - & 06 & - \\ - & - & 06 \end{pmatrix}.$$

Delete the second row and column from both $\mathbf{C_3}$ and $\mathbf{C_2}$ respectively and follow the steps of the diagonal algorithm we will have,

$$\mathbf{C_3} = \begin{pmatrix} 15 & 13 \\ 08 & 08 \end{pmatrix}, \ \mathbf{C_2} = \begin{pmatrix} 09 & 27 \\ 10 & 12 \end{pmatrix}.$$

or $\mathbf{C_3C_2}$ the concatenated 2×4 matrix

$$\mathbf{C_3C_2} = \begin{pmatrix} 15 & 13 & & 09 & 27 \\ 08 & 08 & & 10 & 12 \end{pmatrix}.$$

Since only two factories are left, as mentioned in step 10 of Algorithm 6, select the minimum cost from the four choices set of the two factories F_3F_2 as follows;

$$\{15 + 12, 13 + 10, 09 + 08, 27 + 08\}.$$

Hence, 09 + 08 is the minimum cost allocated in the last two remaining factories, where

09 is the minimum cost in F_2 and 08 is the minimum cost in F_3. The minimum allocation in all the machines are as follows.

Factory 1 : Job 2 : Machine 2 = 06.

Factory 2 : Job 1 : Machine 1 = 09.

Factory 3 : Job 3 : Machine 3 = 08.

Total minimum cost = 23.

It is clear that the total minimum cost 23 is the same as the result that is found by Anuradha and Pandian. Both previous results of 24 and 23 are not optimal because the following allocation is better and an optimal solution.

Factory 1 : Job 2 : Machine 3 = 07.

Factory 2 : Job 1 : Machine 1 = 09.

Factory 3 : Job 3 : Machine 2 = 06.

Total optimal cost = 22.

3.5 Case Study: Hybridisation of the Diagonals Method

To judge the results of our basic feasible solution in all the methods that we have established, we will use the B&B method to hybridise the solution and reach a better improved solution. We have notice that applying the DM with larger sizes dimension is fast and run in seconds. for example the test results we have obtained from applying DM for problems with different size n as it is explained in the Table 3.4.

The DM was tested for problems with size up to $n = 500$ and the number of the variables was equivalent to $500^3 = 125000000$ on a normal memory capacity laptop computer. This

n	Cost	Time in Seconds	No. of Variables	No. of Combinations
100	82	1.768860	1,000,000	1.267651 e^{30}
150	67	6.572607	3,375,000	1.427248 e^{45}
200	95	17.442538	8,000,000	1.606938 e^{60}
250	73	39.178091	15,625,000	1.809251 e^{75}
300	71	75.413410	27,000,000	2.037036 e^{90}
350	111	135.038553	42,875,000	2.293499 e^{105}
400	146	221.470414	64,000,000	2.58225 e^{120}
450	105	350.552412	91,125,000	2.907355 e^{135}
500	62	516.558064	125,000,000	3.273391 e^{150}

Table 3.4: Case Study: Large Sizes DM

size of problem is no way to be solved by using the B&B method because of the large size of the problem and the huge number of combinations is equivalent to $2^{500} = 3.273391\ e^{150}$. The variable are randomly selected from normal distribution generated MATLAB function equal to $fix(100 * rand(n, n, n))$ of two integer digits.

In this case study the exact cost and execution time in seconds are represented by (B&B Cost) and (B&B Time) respectively for SAP. We also represent the heuristic cost and time of the diagonal method as (DM Cost) and (DM Time). To hybridise we have considered the diagonal method cost (DM Cost) to generate an initial feasible solution (X0) in the hybridised B&B such that: (X0 = DM_X0 Cost) and the DM hybridisation time is (DM_X0 Time). In other words we have used the hybridisation technique by applying the outcome cost of the DM and using it as an initial basic feasible solution in B&B method. We have discussed B&B hybridisation and compared the obtained result as it is shown in Table 3.5. The results are reordered in 12 tables. The data is randomly selected for the problems of size 4 and dimension $4 \times 4 \times 4$ up to size 26 and dimension $26 \times 26 \times 26$; each time we have increased the size of the problem by 2. We have compared the total average cost and time of each problem solved by B&B, DM and the hybridised DM_X0. Graph 3.1 to Graph 3.4

give further explanations.

n	B&B Cost	B&B Time	DM Cost	DM Time	DM_X0 Cost	DM_X0 Time
4	49.42	0.2530	94.28	0.0164	49.42	0.0490
6	36.64	0.3294	90.76	0.0128	36.64	0.1893
8	24.42	0.9433	99.00	0.0229	24.42	0.7347
10	20.79	2.5689	112.38	0.0266	20.79	2.2989
12	21.28	7.4436	95.31	0.0315	21.28	7.2635
14	18.05	17.5816	111.58	0.0361	18.05	17.2285
16	19.43	31.2333	105.40	0.0409	19.43	30.6277
18	17.91	126.8548	114.23	0.0347	17.91	121.5693
20	12.74	51.6672	97.32	0.0614	12.74	50.1720
22	13.28	445.9688	151.39	0.0608	13.28	429.8912
24	10.84	1599.6080	125.39	0.1231	10.84	1410.3500
26	11.36	2956.9040	100.29	0.0878	11.36	2894.0040
Average	**21.35**	**436.7797**	**108.11**	**0.0462**	**21.35**	**413.6982**

Table 3.5: DM: Case Study Hybridisation

In the first column of Table 3.5, n is the size of the problem or the dimension $n \times n \times n$. We have applied five instances SAP problem randomly selected and calculate the average time in seconds and cost. The second and third columns are the average cost and time for the instances where B&B method was used. The fourth and fifth columns are representing the average cost and time for the same five instances of the DM and the last two columns represent the hybridisation of the DM method. The outcomes of the DM was used as an initial basic feasible solution in B&B then we calculated the hybridised average cost and time for the same five instances.

The Figure 3.1 shows the average cost comparison for five instances with randomly selected costs and different sizes starting from size 4 to 26. The 12 tables related to this graph are listed in this section. The comparison was between B&B, DM and the hybridisation DM_X0. It is clear from the graph that the cost of DM is large compared to the cost obtained by B&B but it becomes the same cost when we applied the hybridisation technique. The Figure 3.2 shows the average time comparison for the same 12 tables.

The completion time of the DM is considerably shorter than B&B, while the hybridisation DM_X0 is shorter than B&B. From Table 3.5; the Figures 3.3 and 3.4 show the average of the total cost and the time comparisons for B&B, DM and the hybridisation DM_X0. The DM cost is larger than B&B, but when we apply the outcomes of DM as an initial basic feasible solution as a warm start of B&B or hybridisation of it we have obtained the same cost with less time as it is shown in Figure 3.4.

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	4	56.828	0.64069	125.62	0.040929	56.828	0.038089
2	4	65.668	0.42096	84.953	0.031383	65.668	0.061518
3	4	26.889	0.03039	110.61	0.0052278	26.889	0.027384
4	4	30.097	0.09662	30.097	0.0022864	30.097	0.049261
5	4	67.602	0.07655	120.14	0.0023353	67.602	0.068543
Average	**4**	**49.4168**	**0.25304**	**94.284**	**0.0164323**	**49.4168**	**0.048959**

Table 3.6: Five instances average comparison ($4 \times 4 \times 4$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	6	34.466	0.83146	119.96	0.044511	34.466	0.12887
2	6	28.204	0.07844	104.96	0.0077563	28.204	0.073163
3	6	29.859	0.18503	94.099	0.003989	29.859	0.20151
4	6	46.509	0.17658	83.794	0.003614	46.509	0.17691
5	6	44.175	0.37538	50.997	0.0039847	44.175	0.36587
Average	**6**	**36.6426**	**0.32938**	**90.762**	**0.012771**	**36.6426**	**0.1892646**

Table 3.7: Five instances average comparison ($6 \times 6 \times 6$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	8	31.13	2.4214	59.779	0.044532	31.13	1.7393
2	8	25.389	0.93663	67.5	0.036858	25.389	0.5444
3	8	21.068	0.77812	26.84	0.011915	21.068	0.8235
4	8	27.613	0.17273	236.45	0.01398	27.613	0.17624
5	8	16.888	0.40768	104.42	0.0071936	16.888	0.39004
Average	**8**	**24.4176**	**0.94331**	**98.9978**	**0.02289572**	**24.4176**	**0.734696**

Table 3.8: Five instances average comparison ($8 \times 8 \times 8$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	10	16.288	3.7124	66.25	0.057581	16.288	2.8588
2	10	21.63	1.9913	108.3	0.044436	21.63	1.6032
3	10	26.222	0.30373	160.43	0.011376	26.222	0.28832
4	10	13.79	0.84317	161.7	0.011521	13.79	0.85477
5	10	26.037	5.9938	65.199	0.0080316	26.037	5.8893
Average	**10**	**20.7934**	**2.56888**	**112.376**	**0.02658912**	**20.7934**	**2.298878**

Table 3.9: Five instances average comparison ($10 \times 10 \times 10$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	12	25.738	16.678	135.76	0.062476	25.738	16.334
2	12	19.98	3.5679	78.345	0.044485	19.98	3.1509
3	12	21.573	5.0089	59.821	0.019917	21.573	5.1153
4	12	20.421	8.5269	71.134	0.015202	20.421	8.4408
5	12	18.712	3.4361	131.47	0.015376	18.712	3.2764
Average	**12**	**21.2848**	**7.44356**	**95.306**	**0.0314912**	**21.2848**	**7.26348**

Table 3.10: Five instances average comparison ($12 \times 12 \times 12$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	14	17.567	5.9759	179.19	0.064642	17.567	5.1514
2	14	19.523	25.491	118.73	0.053921	19.523	24.912
3	14	18.924	11.887	76.499	0.023935	18.924	12.111
4	14	14.854	10.97	49.303	0.020406	14.854	10.179
5	14	19.37	33.584	134.16	0.017498	19.37	33.789
Average	**14**	**18.0476**	**17.5816**	**111.576**	**0.0360804**	**18.0476**	**17.22848**

Table 3.11: Five instances average comparison ($14 \times 14 \times 14$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	16	20.511	4.3335	76.219	0.064598	20.511	3.5903
2	16	17.918	63.964	100.42	0.058687	17.918	65.099
3	16	20.96	62.44	91.407	0.031104	20.96	59.328
4	16	19.763	16.059	140.54	0.022227	19.763	16.244
5	16	17.992	9.37	118.41	0.028062	17.992	8.8774
Average	**16**	**19.4288**	**31.2333**	**105.399**	**0.0409356**	**19.4288**	**30.62774**

Table 3.12: Five instances average comparison ($16 \times 16 \times 16$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	18	16.854	197.59	128.32	0.038865	16.854	196.19
2	18	20.62	288.23	73.877	0.029698	20.62	269.21
3	18	14.533	83.776	73.432	0.037877	14.533	78.338
4	18	18.199	5.048	187.4	0.032592	18.199	5.0294
5	18	19.35	59.63	108.14	0.034578	19.35	59.079
Average	**18**	**17.9112**	**126.855**	**114.234**	**0.034722**	**17.9112**	**121.56928**

Table 3.13: Five instances average comparison ($18 \times 18 \times 18$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	20	9.1919	11.551	73.636	0.068706	9.1919	10.625
2	20	16.595	25.212	132.28	0.085912	16.595	22.89
3	20	10.166	60.299	96.4	0.030842	10.166	59.936
4	20	15.142	62.851	86.479	0.034181	15.142	59.666
5	20	12.629	98.423	97.822	0.087455	12.629	97.743
Average	**20**	**12.74478**	**51.6672**	**97.3234**	**0.0614192**	**12.74478**	**50.172**

Table 3.14: Five instances average comparison ($20 \times 20 \times 20$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	22	11.018	29.944	175.32	0.098285	11.018	26.656
2	22	12.465	341.18	186.62	0.053231	12.465	323.47
3	22	16.259	864.33	151.9	0.051155	16.259	856.45
4	22	15.08	872.33	109.73	0.057362	15.08	824.76
5	22	11.558	122.06	133.37	0.044006	11.558	118.12
Average	**22**	**13.276**	**445.969**	**151.388**	**0.0608078**	**13.276**	**429.8912**

Table 3.15: Five instances average comparison ($22 \times 22 \times 22$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	24	9.4895	609.24	66.812	0.049442	9.4895	624.92
2	24	11.422	1544.9	189.72	0.16449	11.422	1468.6
3	24	11.871	4155.1	152.04	0.10504	11.871	3299.3
4	24	10.427	684.2	146.12	0.055228	10.427	661.22
5	24	10.973	1004.6	72.257	0.24106	10.973	997.71
Average	**24**	**10.8365**	**1599.61**	**125.39**	**0.123052**	**10.8365**	**1410.35**

Table 3.16: Five instances average comparison ($24 \times 24 \times 24$)

	n	B&B cost	B&B time	DM cost	DM time	DM_X0 cost	DM_X0 time
1	26	9.9917	609.51	74.298	0.12552	9.9917	556.08
2	26	11.43	990.91	100.9	0.069959	11.43	937.94
3	26	11.667	3488.2	73.559	0.06185	11.667	3402.4
4	26	12.51	5396.4	112.69	0.076044	12.66	5594.2
5	26	11.205	4299.5	140.02	0.10544	11.205	3979.4
Average	**26**	**11.36074**	**2956.9**	**100.293**	**0.0877626**	**11.39074**	**2894.004**

Table 3.17: Five instances average comparison ($26 \times 26 \times 26$)

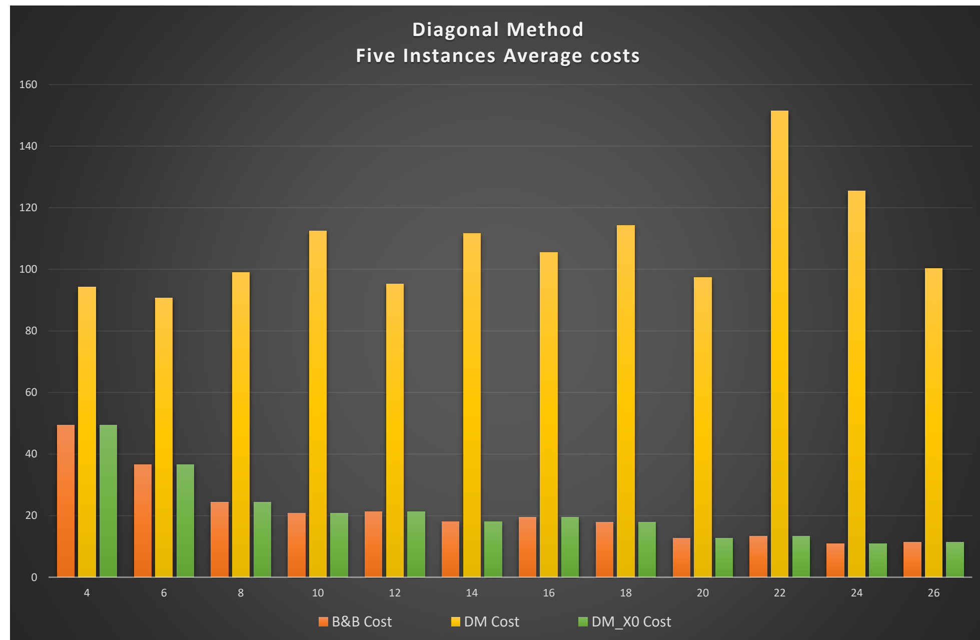

Figure 3.1: DM: Average Cost Comparisons

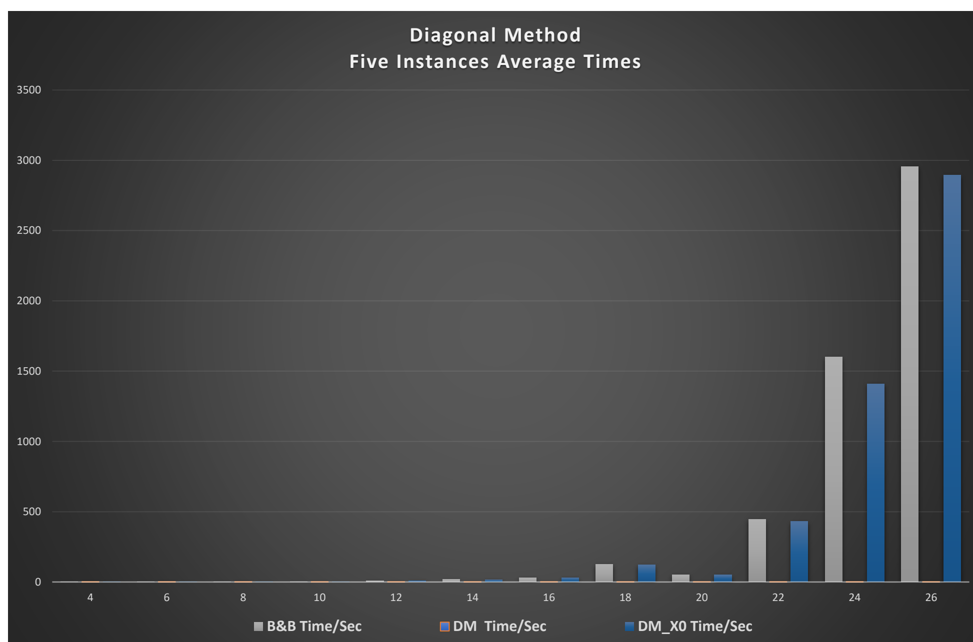

Figure 3.2: DM: Average Time Comparisons

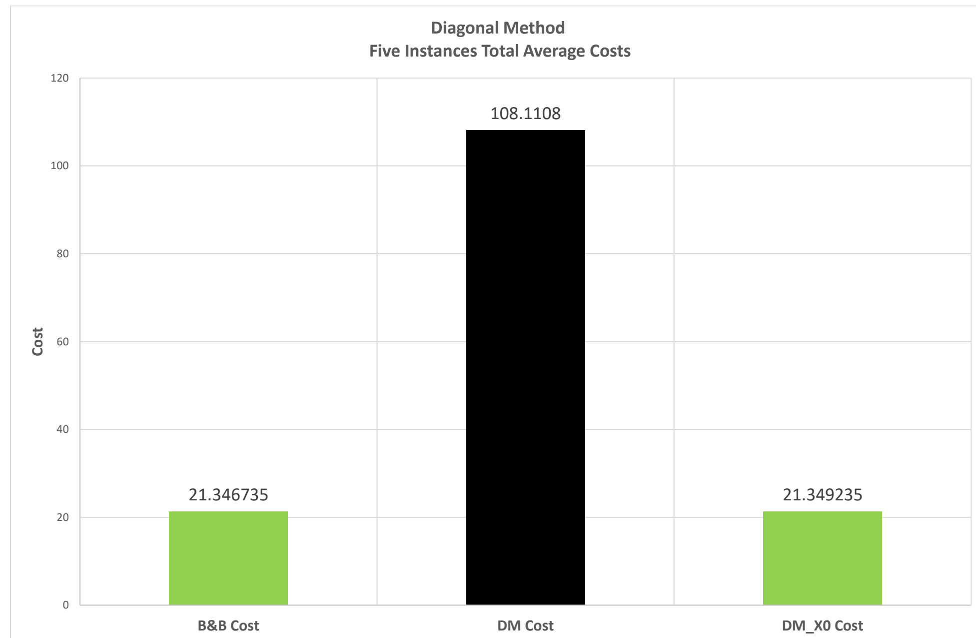

Figure 3.3: DM: Average Total Cost Comparisons

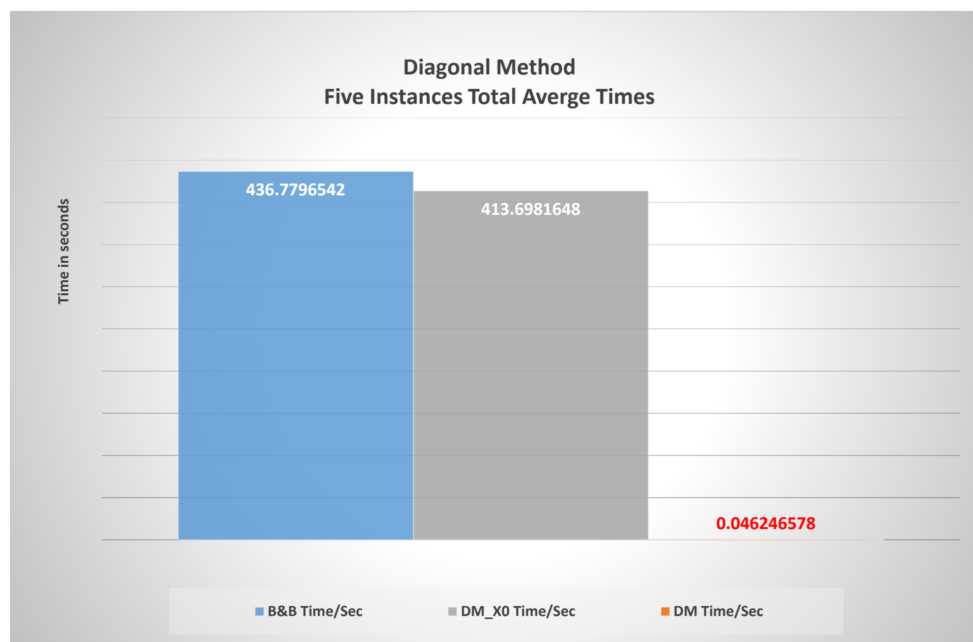

Figure 3.4: DM: Average Total Time Comparisons

3.6 The Diagonals Method: Tie Cases

The tie case always happened when we have random large numbers. The problem arises when there are similarity to the number required selection and a decision will be necessary to decide which number is to be chosen.

The Tie case happened when a specific selection occurs in different positions in one stage or more with same number. The question is; shall we choose the first, the last or a random number from that stage. Mainly the programmers selected the first number in their code to break the tie.

Sometimes this selection is not accurate and will affect the final result.Tie break here is often carried out through random selection. In our case we break the tie by selecting the first minimum number. In case the size of the problem is not large and we have the option to change the selected number and comparing the result, then this will be useful. Example 3.6.1 shows that changing the tie number will give a better solution for this size 4 problem.

Example 3.6.1 Tie Example Size 4

In this problem, we will discus how the tie number will affect the final result.

Let **A**, **B**, **C** and **D** be four matrices representing to the cost of SAP allocations. The alloca-

tion must satisfy SAP conditions that have been explained in the previous sections.

$$\mathbf{A} = \begin{pmatrix} 8 & 6 & 9 & 9 \\ 9 & 0 & 9 & 4 \\ 1 & 2 & 1 & 8 \\ 9 & 5 & 9 & 1 \end{pmatrix}, \ \mathbf{B} = \begin{pmatrix} 4 & 6 & 6 & 6 \\ 9 & 0 & 7 & 1 \\ 7 & 8 & 7 & 7 \\ 9 & 9 & 3 & 0 \end{pmatrix}, \ \mathbf{C} = \begin{pmatrix} 2 & 6 & 4 & 1 \\ 0 & 3 & 3 & 4 \\ 0 & 9 & 7 & 4 \\ 8 & 0 & 7 & 6 \end{pmatrix}, \ \mathbf{D} = \begin{pmatrix} 7 & 6 & 9 & 7 \\ 7 & 1 & 3 & 2 \\ 2 & 1 & 5 & 5 \\ 6 & 4 & 2 & 6 \end{pmatrix}.$$

Rearrange the above matrices in descending order according to their maximum diagonals, we have;

$$\mathbf{B} = \begin{pmatrix} \textcircled{4} & 6 & 6 & 6 \\ 9 & \textcircled{0} & 7 & 1 \\ 7 & 8 & \textcircled{7} & 7 \\ 9 & 9 & 3 & \textcircled{0} \end{pmatrix}, \ \mathbf{A} = \begin{pmatrix} 8 & 6 & 9 & 9 \\ 9 & 0 & 9 & 4 \\ 1 & 2 & 1 & 8 \\ 9 & 5 & 9 & 1 \end{pmatrix}, \ \mathbf{C} = \begin{pmatrix} 2 & 6 & 4 & 1 \\ 0 & 3 & 3 & 4 \\ 0 & 9 & 7 & 4 \\ 8 & 0 & 7 & 6 \end{pmatrix}, \ \mathbf{D} = \begin{pmatrix} 7 & 6 & 9 & 7 \\ 7 & 1 & 3 & 2 \\ 2 & 1 & 5 & 5 \\ 6 & 4 & 2 & 6 \end{pmatrix}.$$

Applying the Hungarian method to the first matrix **B**, the allocation will be the set of circled numbers {4,0,7,0}. To select the minimum numbers of this set we have two options either to select the zero costs $c_{2,2}$ or $c_{4,4}$ from matrix B. If we select the first cell $c_{2,2}$ and proceed, the initial basic feasible solution is 8, while if we select the other cell $c_{4,4}$ the optimal solution is 6 which is better. Since the size of the problem is not large, it is possible to select and control the minimum cost but it will be more difficult and complicated when the size of the problem is large.

Example 3.6.2 Tie Example Size 3

In this example we will discuss the tie situation for size 3 problem. The tie is more

complicated although the size of the problem is 3.

Let $\mathbf{C} = \{\mathbf{c_{ijk}}\}$ be the cost matrix for the SAP.

$$\mathbf{C} = \begin{pmatrix} 10 & 09 & 09 & & 09 & 09 & 10 & & 15 & 07 & 08 \\ 09 & 06 & 09 & & 10 & 09 & 07 & & 10 & 11 & 06 \\ 12 & 09 & 06 & & 27 & 12 & 09 & & 13 & 12 & 08 \end{pmatrix}.$$

Let $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ denote the cost of the three factories F_1, F_2 and F_3 such that,

$$\mathbf{C_1} = \begin{pmatrix} 10 & 09 & 09 \\ 09 & 06 & 09 \\ 12 & 09 & 06 \end{pmatrix}.$$

Calculate the UDL and LDL of the first factory from the matrix $\mathbf{C_1}$ then select the maximum value and denote it d_1. Obtain the optimum allocation cost using the Hungarian method. The outcomes are as follows.

LDL = 22 and UDL = 27, d_1 = Max{LDL, UDL} = 27.

The optimum allocation cost = 10 + 6 + 6 = 22.

Applying the same procedures on the factory F_2 from the matrix $\mathbf{C_2}$ we have,

$$\mathbf{C_2} = \begin{pmatrix} 9 & 9 & 10 \\ 10 & 9 & 7 \\ 27 & 12 & 9 \end{pmatrix}.$$

LDL = 27 and UDL = 46, d_2= Max{LDL, UDL} = 46.

The optimum allocation cost = 9 + 10 + 9 = 28.

Applying the same procedures on factory F_3 from the matrix $\mathbf{C_3}$ we have,

$$\mathbf{C_3} = \begin{pmatrix} 15 & 7 & 8 \\ 10 & 11 & 6 \\ 13 & 12 & 8 \end{pmatrix}.$$

LDL = 34 and UDL = 32, d_3 = Max{LDL, UDL}= 34.

The optimum allocation cost = 7 + 10 + 8 = 25.

Arrange the factories cost matrices $\mathbf{C_1}, \mathbf{C_2}$ and $\mathbf{C_3}$ in descending order with respect to their diagonal values as follows.

$$d_1 = 27, \; d_2 = 46 \; \text{ and } \; d_3 = 34.$$

The descending order of the diagonal lines will be

$$d_2 = 46, d_3 = 34, d_1 = 27.$$

Hence the descending order of the factories costs is $\{\mathbf{C_2}, \mathbf{C_3}, \mathbf{C_1}\}$ accordingly.

Rename the factories as $\mathbf{C_2} = \mathbf{C}_1^{'}$, $\mathbf{C_3} = \mathbf{C}_2^{'}$ and $\mathbf{C_1} = \mathbf{C}_3^{'}$ and arrange them in descending order,

$$\mathbf{C}_1^{'} = \begin{pmatrix} 09 & 09 & 10 \\ 10 & 09 & 07 \\ 27 & 12 & 09 \end{pmatrix}, \mathbf{C}_2^{'} = \begin{pmatrix} 15 & 7 & 8 \\ 10 & 11 & 6 \\ 13 & 12 & 8 \end{pmatrix}, \mathbf{C}_3^{'} = \begin{pmatrix} 10 & 09 & 09 \\ 09 & 06 & 09 \\ 12 & 09 & 06 \end{pmatrix}.$$

Start from factory $\mathbf{C}_1^{'}$,

$$\mathbf{C}_1^{'} = \begin{pmatrix} 09 & 09 & 10 \\ 10 & 09 & 07 \\ 27 & 12 & 09 \end{pmatrix}.$$

Apply the Hungarian method, to get the allocations and values in factory $\mathbf{C}_1^{'}$ as follows.

$$\mathbf{C}_1^{'}(\mathbf{1},\mathbf{1},\mathbf{1}) = \mathbf{9},\ \mathbf{C}_1^{'}(\mathbf{2},\mathbf{2},\mathbf{1}) = \mathbf{9} \text{ and } \mathbf{C}_1^{'}(\mathbf{3},\mathbf{3},\mathbf{1}) = \mathbf{9}.$$

The assignment values row vector = [1 2 3] = [09 09 09]. The minimum assignment value is 09 and there are three values that have the same minimum value. In such a case we say that the problem has a tie. We can not judge which assignment value is the best without solving the problem and try all the three values. If we start from the first minimum value $\mathbf{C}_1^{'}(\mathbf{1},\mathbf{1},\mathbf{1})$ and allocate the minimum assignment value 09 to job J_1, machine M_1 in factory $F_1^{'}$ then delete the first row and the first column from both $\mathbf{C}_2^{'}$ and $\mathbf{C}_3^{'}$. The remaining matrices are as follows;

$$\mathbf{C}_2^{'} = \begin{pmatrix} 11 & 06 \\ 12 & 08 \end{pmatrix},\ \ \mathbf{C}_3^{'} = \begin{pmatrix} 06 & 09 \\ 09 & 06 \end{pmatrix},$$

or $\mathbf{C}_2^{'}\mathbf{C}_3^{'}$ the concatenated 2×4 matrix

$$\mathbf{C}_2^{'}\mathbf{C}_3^{'} = \begin{pmatrix} 11 & 06 & & 06 & 09 \\ 12 & 08 & & 09 & 06 \end{pmatrix}.$$

Since only two factories remain, as mentioned in step 10, select the minimum cost from

the four possible choices of the two factories $\mathbf{C}_2'\mathbf{C}_3'$ as follows,

$\{11 + 06, 06 + 09, 06 + 08, 09 + 12\}$.

Hence, 06 + 08 is the minimum cost allocated to the last two remaining factories, where 06 is the minimum cost in $\mathbf{C}_3'$ and 08 is the minimum cost in $\mathbf{C}_2'$. The optimal allocation in all the factories are as follows;

Factory $\mathbf{C}_1'$: Job J_1 : Machine $M_1 = 09$.

Factory $\mathbf{C}_2'$: Job J_3 : Machine $M_3 = 08$.

Factory $\mathbf{C}_3'$: Job J_2 : Machine $M_2 = 06$.

Total minimum cost = 23.

Following the same procedure for the second minimum assignment value, we have the value $\mathbf{C}_1'(\mathbf{2},\mathbf{2},\mathbf{2}) = \mathbf{09}$. Allocate the minimum assignment value 09 to job J_2, machine M_2 in factory $\mathbf{C}_1'$ then delete the second row and the column from both $\mathbf{C}_2'$ and $\mathbf{C}_3'$. The remaining matrices are as follows,

$$\mathbf{C}_2' = \begin{pmatrix} 15 & 08 \\ 13 & 08 \end{pmatrix}, \quad \mathbf{C}_3' = \begin{pmatrix} 10 & 09 \\ 12 & 06 \end{pmatrix},$$

or $\mathbf{C}_2'\mathbf{C}_3'$ the concatenated 2×4 matrix,

$$\mathbf{C}_2'\mathbf{C}_3' = \begin{pmatrix} 15 & 08 & & 10 & 09 \\ 13 & 08 & & 12 & 06 \end{pmatrix}.$$

Since only two factories remain, as mentioned in step 10, select the minimum cost from the four possible choices of the two factories $\mathbf{C}_2'\mathbf{C}_3'$as follows,

$\{15 + 06, 08 + 12, 10 + 08, 09 + 13\}$.

Hence, 10 + 08 is the minimum cost allocated to the last two remaining factories, where 10 is the minimum cost in $\mathbf{C}_3'$ and 08 is the minimum cost in $\mathbf{C}_2'$. The minimum allocations in all the factories are as follows;

Factory $\mathbf{C}_3'$: Job J_1 : Machine M_1 : = 10.

Factory $\mathbf{C}_1'$: Job J_2 : Machine M_2 : = 09.

Factory $\mathbf{C}_2'$: Job J_3 : Machine M_3 : = 08.

Total minimum cost = 27.

Following the same procedure for the third minimum assignment value, we have the value $\mathbf{C}_1'(\mathbf{3}, \mathbf{3}, \mathbf{3}) = \mathbf{09}$, allocate the minimum assignment value 09 to J_3, M_3 in factory $\mathbf{C}_1'$ then delete the third row and the third column from both $\mathbf{C}_2'$ and $\mathbf{C}_3'$. The remaining matrices are as follows,

$$\mathbf{C}_2' = \begin{pmatrix} 15 & 07 \\ 10 & 11 \end{pmatrix}, \quad \mathbf{C}_3' = \begin{pmatrix} 10 & 09 \\ 09 & 06 \end{pmatrix}$$

or the $\mathbf{C}_2'\mathbf{C}_3'$ concatenated 2×4 matrix

$$\mathbf{C}_2'\mathbf{C}_3' = \begin{pmatrix} 15 & 07 & & 10 & 09 \\ 10 & 11 & & 09 & 06 \end{pmatrix}.$$

Since two factories are left, as mentioned in step 9, select the minimum cost from the four possible choices of the two factories $\mathbf{C}_2'$ and $\mathbf{C}_3'$ as follows;

$\{15 + 06, 07 + 09, 10 + 11, 09 + 10\}$.

Hence, 07 + 09 is the minimum cost allocated to the last two remaining factories, where

09 is the minimum cost in $\mathbf{C}_3^{'}$ and 07 is the minimum cost in $\mathbf{C}_2^{'}$. The minimum allocation in all the factories are as follows;

Factory $\mathbf{C}_1^{'}$: Job J_3 : Machine $M_3 = 09$.

Factory $\mathbf{C}_2^{'}$: Job J_1 : Machine $M_2 = 07$.

Factory $\mathbf{C}_3^{'}$: Job J_2 : Machine $M_1 = 09$.

Total minimum cost = 25.

We have obtained three solutions because of the tie problem. Because we cannot avoid the tie, our code starts with the first minimum number. The tie can happen at any stage of the problem.

3.7 Summary

We have discussed the Hungarian assignment problem and stated an algorithm to solve the two dimensional assignment problem. We have explained the target, the structure and stated an algorithm called the DM. Definitions and numerical examples are given to explain the procedures of the DM. A case study of the effect of tie allocation problem on the optimal solution is considered and explained with examples. A case study of the hybridisation of the DM and comparisons of the costs and times between the DM and the B&B methods are studied and explained with graphs and tables.

Chapter 4

Further Heuristic Approaches to SAP

4.1 Introduction

In the previous chapter we have solved the SAP using DM. We have dealt with calculating the upper and lower diagonal lines of the problem. Then we have reduced the problem by deleting the allocated job, machine and factory from our calculations. Although the basic feasible solution is not always close to the optimum, we benefit from the speed and efficiency of the algorithm to solve the problem and get approximate solution for higher size problems.

In this chapter we study three more heuristic methods. The first is the Average Cost Method (ACM), the second is the Addition Method (AM) and the third is the Multiplication Method (MM). Algorithms, examples and case studies were given for all the three methods. Our target is to find a better approach to solve SAP. All the methods used guarantee a basic feasible solution for higher size SAP. We have used different types of normal random numbers generates using MATLAB. For example see the tables listed in

this chapter and Table 4.1

The average cost method will approach the solution in a different way to DM. We took in consideration a simple technique to do the allocations. The basic idea depends on connecting the cost of the allocation of the selected job and add it to the average cost of the other non-selected jobs, machines and factories.

The addition method relies on adding the costs for each worker in each factory, then forming a two dimensional matrix to which we apply the Hungarian method for allocation.

The multiplication method is similar to the addition method, the only difference being to multiply the cost of each worker in the factories to form a new two dimensional matrix to which we apply the Hungarian method. All these three methods are described through algorithms, examples, case studies and graphs.

4.2 The Average Cost Method

The Average Cost Method (ACM) approaches the solution in a different way to solve SAP. We took in consideration to use simple techniques to do the allocations. The basic idea depends on connecting the cost of the allocation of the selected job and add it to the average cost of the other non-selected jobs, machines and factories. This method can be considered as a basic stage of a dynamic model. Algorithm 7 explains the approach and how to apply it.

The average cost method as its name says dealings with calculating an average of the remaining unallocated costs. The idea is to start from selecting the first cost c_{111} allocated to the first worker to do the first job J_1 on the first machine M_1 in the first factory F_1. Then

we calculate the average cost for all the non allocated costs after deleting the first row and the first column from all the non-allocated jobs, machines and factories. We added the selected cost and the average cost. Then repeat this process for all the n^3 costs we have and select the minimum sum. The method is explained in details through an algorithm, examples, case study and graphs.

Algorithm 7: The Average Cost Method

Step 1: Initialisation:
Set the size of the problem to be a non negative integer number n.
Let $A = 0$, where A is the average cost for the non-selected indices.
Let the initial cost $c_{ijk} = 0$, where (i, j and $k = 1, 2, ..., n$).
Let S be the sum of the cost c_{ijk} and the average cost A, where $S = 0$ or $S = c_{ijk} + A$.

Step 2: Calculating the average cost A:
The following points explain how to calculate and select the average cost A,

1. Start the first allocation, job J_1 and select the cost $c_{ijk} = c_{111}$, where c_{111} is the cost of allocating the first job to the first machine in the first factory.
2. Delete the first row from each job.
3. Delete the first column from each factory.
4. Remove the first factory that c_{111} is related to.
5. Calculate A the average of the remaining costs RC.
6. Calculate the sum S; by adding the cost c_{111} to A.
7. Repeat 1-6 for all j^{th} machine of the first job in the kth factories.
8. Repeat 1-7 for all the workers who do the jobs of SAP.
9. Select the minimum S. If there is a tie, select the first one or select S with minimum cost c_{ijk}

Step 3: Allocation:

1. Allocate c_{ijk} related to the selected S from step 2, points (1-9).
2. Delete the ith row and all the jth columns crossing the allocated cost c_{ijk} from SAP, remove the ith factory F_i.
3. Terminate if all jobs are allocated. Calculate the total sum of all allocated cost. Otherwise, go to Step 2.

The following Example 4.2.1 explains how to apply Algorithm 7 to solve SAP.

Example 4.2.1 Let us consider a SAP with three jobs J_1, J_2 and J_3, three machines M_1, M_2 and M_3 and three factories F_1, F_2 and F_3. Let $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ be the costs of allocation of the jobs, machines and factories for SAP respectively as follows.

$$\mathbf{C_1} = \begin{pmatrix} 8 & 9 & 2 \\ 9 & 6 & 5 \\ 1 & 0 & 9 \end{pmatrix}, \quad \mathbf{C_2} = \begin{pmatrix} 9 & 9 & 1 \\ 1 & 4 & 4 \\ 9 & 8 & 9 \end{pmatrix}, \quad \mathbf{C_3} = \begin{pmatrix} 7 & 0 & 6 \\ 9 & 8 & 7 \\ 6 & 9 & 7 \end{pmatrix}.$$

Suppose that $C = \{c_{ijk}\}$, where (i, j and $k = 1, 2, 3$) be the allocation costs for the previous three jobs, machines and factories as it is shown in the following matrix;

$$\mathbf{C} = \begin{pmatrix} 08 & 09 & 02 & 09 & 09 & 01 & 07 & 0 & 06 \\ 09 & 06 & 05 & 01 & 04 & 04 & 09 & 08 & 07 \\ 01 & \boxed{0} & 09 & 09 & 08 & 09 & 06 & 09 & 07 \end{pmatrix}.$$

The target is to find the minimum allocation cost applying the ACM. To solve the example, apply Algorithm 7. Select the first cost $c_{111} = 08$ from C then delete row 1, columns 1, 4 and 7 from $\mathbf{C}$ and remove all the remaining cost $\mathbf{C_1}$ of factory F_1, we have the remaining costs;

$$\mathbf{RC} = \begin{pmatrix} 04 & 04 & 08 & 07 \\ 08 & 09 & 09 & 07 \end{pmatrix}.$$

Calculate the average of the remaining costs, $A = RC/8$, hence $A = 56/8 = 7$. Repeat the same process and calculate the average cost A for all the costs that belong to $\mathbf{C}$. Let

AA = {$\mathbf{A_{ij}}$} be the outcome matrix of the corresponding average $A's$ values for **C**.

$$\mathbf{AA} = \begin{pmatrix} 7.000 & 6.500 & 6.750 & 6.375 & 6.375 & 6.625 & 5.875 & 5.875 & 4.750 \\ 6.125 & 6.750 & 7.125 & 5.250 & 5.250 & 5.750 & 6.000 & 6.000 & 6.625 \\ 4.875 & 5.500 & 5.875 & 5.375 & 5.375 & 6.625 & 4.785 & 4.875 & 6.875 \end{pmatrix}$$

To find the sum **S** for all the costs, simply add the two matrices **C** and **AA**, we have the following matrix;

$$\mathbf{S} = \begin{pmatrix} 15.000 & 15.500 & 8.750 & 15.375 & 15.375 & 7.625 & 12.875 & 5.875 & 10.750 \\ 15.250 & 12.750 & 12.125 & 6.250 & 9.250 & 9.750 & 15.000 & 14.000 & 13.625 \\ 5.875 & \bigcirc\!\!\!\!\!\!5.500 & 14.875 & 14.375 & 13.375 & 15.625 & 10.875 & 13.875 & 13.875 \end{pmatrix}.$$

Select the minimum number (circled) 5.500, if there is a tie (more than one minimum number), break the tie by selecting the first minimum number. In this example we do not have a tie. Hence the job J_3 is scheduled to the machine M_2 in the factory F_1, the allocation cost related to it is the boxed 0 in the above array **C**. The next step is to remove the first factory, the third job and the second machine. Repeat the steps 1, 2 and 3 of Algorithm 7. We have C', AA' and S' as follows.

$$\mathbf{C'} = \begin{pmatrix} 9 & 1 & 7 & \boxed{6} \\ 1 & 4 & 9 & 7 \end{pmatrix}, \ \mathbf{AA'} = \begin{pmatrix} 7 & 9 & 4 & 1 \\ 6 & 9 & 1 & 9 \end{pmatrix}, \ \mathbf{S'} = \begin{pmatrix} 16 & 10 & 11 & \textcircled{7} \\ 7 & 13 & 10 & 16 \end{pmatrix}$$

The minimum cost of **S′** is the circled 7 and the minimum related cost in **C′** is the boxed 6, which means that job J_1 is allocated to machine M_3 in factory F_3. Apply steps 1, 2 and 3

of Algorithm 7 and we will have the final step with the following outputs:

$$\mathbf{C}'' = \boxed{1}, \ \mathbf{AA}'' = 6, \ \mathbf{S}'' = 7.$$

The final allocation is that job J_2 is allocated to the machine M_1 in factory F_2 and the cost of the allocation is the boxed 1. Hence the total minimum cost is $0 + 6 + 1 = 7$.

4.3 Case Study

Here, we have selected five instances randomly with different sizes problems as it is shown in Table 4.1. All the tables related to the average calculations of this table are listed below. For each five instances with a specific size started from dimension $4 \times 4 \times 4$ to $26 \times 26 \times 26$ we have calculated the minimum cost and the possible allocations using the B&B method, the ACM and the Hybridisation techniques.

We have considered the outcome costs of the ACM as an initial basic feasible solution to start the B&B. This hybridisation techniques have been used to see if there is any effect on the execution time.

From Table 4.1 and the Figures 4.1 to 4.4, it is clear that the cost of the ACM is larger than that of B&B for all the 12 tables with five instances listed in this section, but the costs of the hybridisation of ACM is equal to all the costs of the B&B. It is clear from Figure 4.2 that the time of the ACM is considerably small comparing to both the B&B and the hybridisation ACM, also it is noticeable that the time of the hybridisation ACM is less than that of the B&B. The data in both Figures 4.3 and 4.4 are obtained from Table 4.1, the average total costs and times are measured for the B&B, the ACM and the hybridisation

n	B&B Cost	Sec	ACM Cost	Sec	ACM_X0 Cost	Sec
4	49	0.0373206	70.8	0.00564684	49	0.03449
6	34.2	0.2414712	106	0.0216724	34.2	0.254528
8	24.2	0.753232	169	0.0512444	24.2	0.757932
10	22.6	2.8664	166.6	0.145536	22.6	2.87816
12	14.4	4.00974	300.2	0.303232	14.4	4.06066
14	11.2	21.82838	352	0.586508	11.2	21.65406
16	9.8	54.77274	326.4	1.17676	9.8	54.85742
18	8.2	83.1472	443	2.16982	8.2	82.5324
20	4.6	258.3854	457	4.00552	4.6	256.1862
22	2.4	545.421	588.8	7.45002	2.4	521.2882
24	2.2	2301.196	643.8	12.0346	2.2	2288.138
26	1.4	5781.52	785.2	18.9008	1.4	5755.1
Average	**15.35**	**754.514907**	**367.4**	**3.90427997**	**15.35**	**748.9785042**

Table 4.1: Case Study: ACM Hybridisation

of ACM in all the tables listed in this section. Figure 4.3 shows that the Average total costs of the B&B is equal to the hybridisation of ACM but the average total costs of ACM is too large. Figure 4.4 shows that the time of the hybridisation of ACM is slightly less than the B&B, it is clear that the time of ACM is considerably smaller than both the B&B and the hybridisation of ACM. Hence the ACM is running considerably fast and the hybridisation of ACM is running in less time than the B&B with same outcomes.

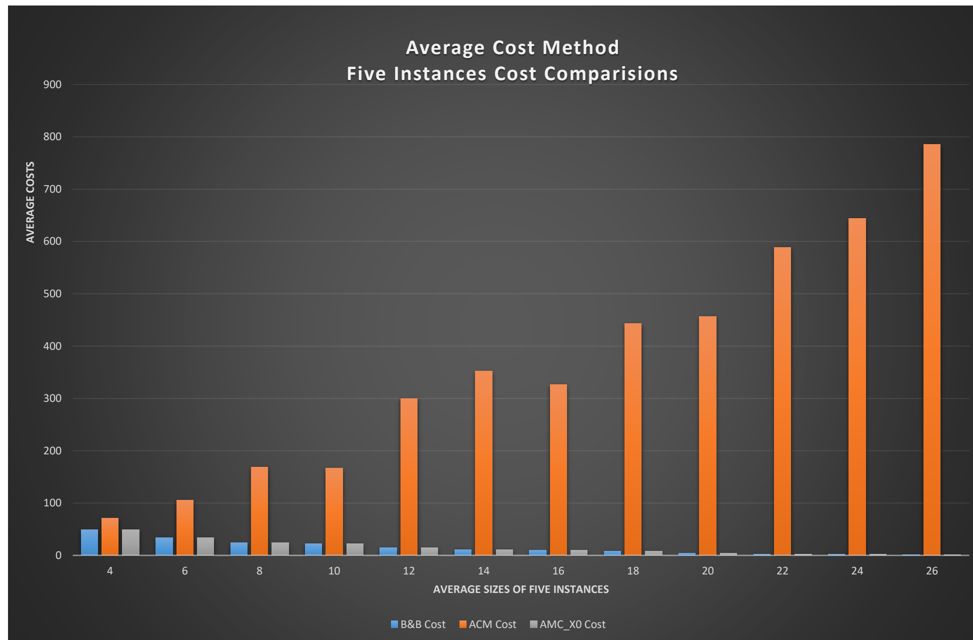

Figure 4.1: ACM: Cost Comparisons

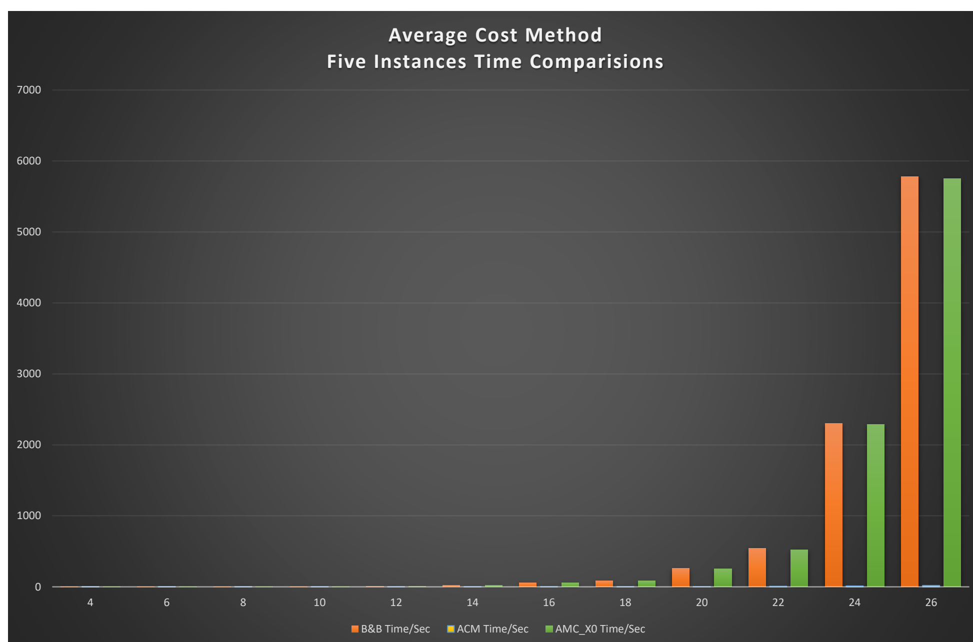

Figure 4.2: ACM: Time Comparisons

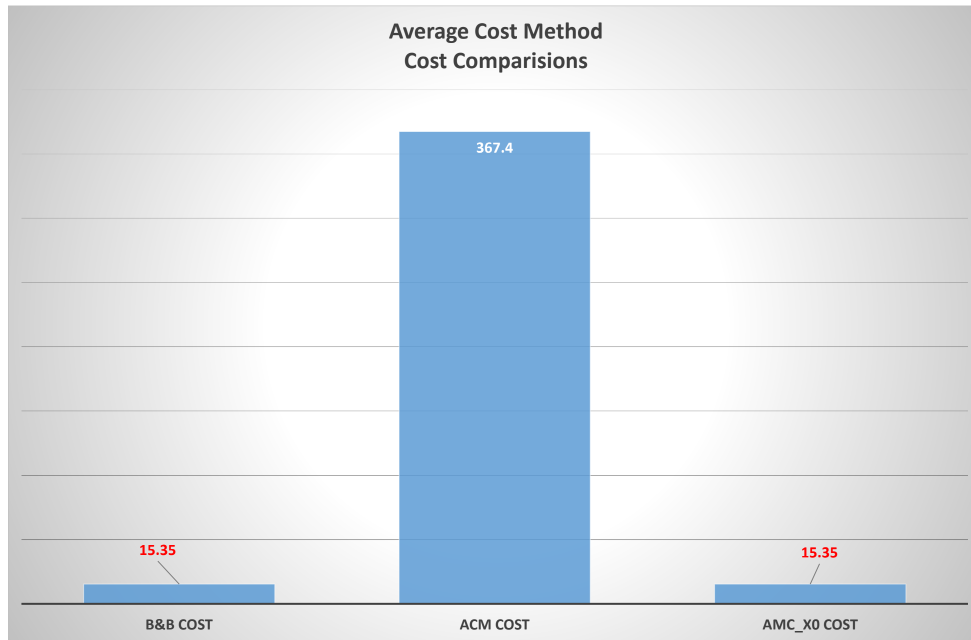

Figure 4.3: ACM: Average Total Cost Comparisons

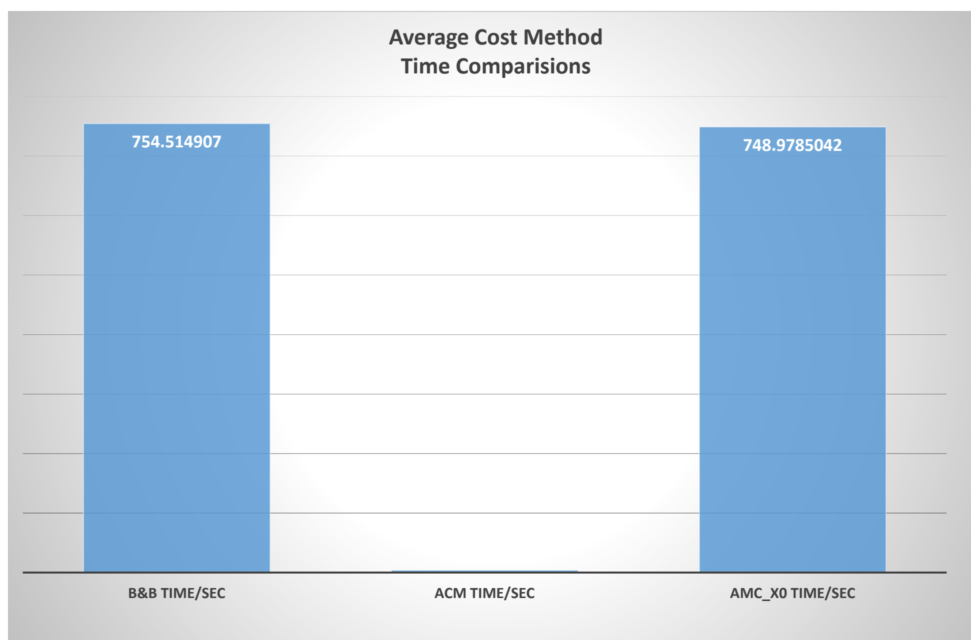

Figure 4.4: ACM: Average Total Time Comparisons

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	4	40	0.022237	40	0.0041906	40	0.022084
2	4	57	0.033655	84	0.0039938	57	0.035617
3	4	33	0.060169	75	0.0089276	33	0.042337
4	4	44	0.025459	44	0.00609	44	0.026355
5	4	71	0.045083	111	0.0050322	71	0.046057
Average	**4**	**49**	**0.037321**	**70.8**	**0.00564684**	**49**	**0.03449**

Table 4.2: Average Method: Five instances average comparison ($4 \times 4 \times 4$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	6	24	0.16955	69	0.016163	24	0.17053
2	6	28	0.097782	160	0.018943	28	0.084838
3	6	21	0.095614	60	0.036119	21	0.080912
4	6	56	0.71314	166	0.016051	56	0.80303
5	6	42	0.13127	75	0.021086	42	0.13333
Average	**6**	**34.2**	**0.241471**	**106**	**0.0216724**	**34.2**	**0.254528**

Table 4.3: Average Method: Five instances average comparison ($6 \times 6 \times 6$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	8	30	1.1826	130	0.04812	30	1.1488
2	8	25	0.28687	163	0.051687	25	0.28905
3	8	24	0.16907	158	0.052278	24	0.17619
4	8	24	1.4897	123	0.051323	24	1.5225
5	8	18	0.63792	271	0.052814	18	0.65312
Average	**8**	**24.2**	**0.753232**	**169**	**0.0512444**	**24.2**	**0.757932**

Table 4.4: Average Method: Five instances average comparison ($8 \times 8 \times 8$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	10	23	1.6918	179	0.12193	23	1.6918
2	10	24	2.1254	143	0.13554	24	2.1039
3	10	16	2.1423	152	0.15774	16	2.0749
4	10	25	5.3332	163	0.17689	25	5.4463
5	10	25	3.0393	196	0.13558	25	3.0739
Average	**10**	**22.6**	**2.8664**	**166.6**	**0.145536**	**22.6**	**2.87816**

Table 4.5: Average Method: Five instances average comparison ($10 \times 10 \times 10$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	12	14	2.7462	241	0.28215	14	2.7456
2	12	14	3.2458	352	0.29756	14	3.2701
3	12	16	8.1875	291	0.29327	16	8.3561
4	12	15	3.7642	262	0.29804	15	3.7218
5	12	13	2.105	355	0.34514	13	2.2097
Average	**12**	**14.4**	**4.00974**	**300.2**	**0.303232**	**14.4**	**4.06066**

Table 4.6: Average Method: Five instances average comparison ($12 \times 12 \times 12$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	14	17	37.995	291	0.58292	17	34.672
2	14	14	41.427	315	0.57463	14	44.31
3	14	8	15.635	342	0.5901	8	15.3
4	14	7	1.7359	367	0.59605	7	1.5893
5	14	10	12.349	445	0.58884	10	12.399
Average	**14**	**11.2**	**21.82838**	**352**	**0.586508**	**11.2**	**21.65406**

Table 4.7: Average Method: Five instances average comparison ($14 \times 14 \times 14$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	16	13	109.74	394	1.1144	13	108.81
2	16	10	100.38	218	1.1844	10	101.97
3	16	8	3.992	285	1.2069	8	4.06
4	16	5	7.7437	364	1.2012	5	7.8311
5	16	13	52.008	371	1.1769	13	51.616
Average	**16**	**9.8**	**54.77274**	**326.4**	**1.17676**	**9.8**	**54.85742**

Table 4.8: Average Method: Five instances average comparison ($16 \times 16 \times 16$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	18	10	45.768	398	2.1274	10	45.571
2	18	8	61.176	399	2.1738	8	62.056
3	18	5	103.81	457	2.1892	5	103.5
4	18	11	175	406	2.1714	11	171.77
5	18	7	29.982	555	2.1873	7	29.765
Average	**18**	**8.2**	**83.1472**	**443**	**2.16982**	**8.2**	**82.5324**

Table 4.9: Average Method: Five instances average comparison ($18 \times 18 \times 18$)

	n	B&B cost	Time/sec	Average cost	Time/sec	B&B_X0 cost	Time/sec
1	20	8	417.81	421	3.8634	8	398.23
2	20	2	127.07	427	3.9904	2	126.06
3	20	5	350.17	572	3.8909	5	345.04
4	20	3	56.967	468	4.3926	3	56.761
5	20	5	339.91	397	3.8903	5	354.84
Average	**20**	**4.6**	**258.3854**	**457**	**4.00552**	**4.6**	**256.1862**

Table 4.10: Average Method: Five instances average comparison ($20 \times 20 \times 20$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	22	3	2399.9	573	6.9149	3	2285.6
2	22	2	112.01	570	6.7059	2	112.32
3	22	3	154.87	411	8.6768	3	150.51
4	22	1	12.142	641	8.183	1	12.078
5	22	3	48.183	749	6.7695	3	45.933
Average	**22**	**2.4**	**545.421**	**588.8**	**7.45002**	**2.4**	**521.2882**

Table 4.11: Average Method: Five instances average comparison ($22 \times 22 \times 22$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	24	3	3810.1	829	11.7	3	3731.6
2	24	2	305.78	635	11.41	2	295.99
3	24	1	3730.4	555	13.746	1	3602
4	24	2	2658	636	11.552	2	2797.1
5	24	3	1001.7	564	11.765	3	1014
Average	**24**	**2.2**	**2301.196**	**643.8**	**12.0346**	**2.2**	**2288.138**

Table 4.12: Average Method: Five instances average comparison ($24 \times 24 \times 24$)

	n	B&B cost	Time/sec	AM cost	Time/sec	B&B_X0 cost	Time/sec
1	26	1	5769.7	742	18.914	1	5765.8
2	26	2	5807.8	648	18.887	2	5822.7
3	26	1	5721.3	522	18.917	1	5716.8
4	26	1	5804.4	1007	18.893	1	5735.1
5	26	2	5804.4	1007	18.893	2	5735.1
Average	**26**	**1.4**	**5781.52**	**785.2**	**18.9008**	**1.4**	**5755.1**

Table 4.13: Average Method: Five instances average comparison ($26 \times 26 \times 26$)

4.4 The Addition Method

In this section we have applied a new approach to find the basic initial solution to SAP. The problem can be solved based on two stages. The first stage is to convert the SAP from three dimensional problem into two dimensions linear problem by using addition operator.

The methodology of the Addition Method (AM) is simply adding the costs of allocation machines for each worker in each factory while in the second stage we applied the well known Hungarian method, [42]. This method is good for large dimension problems and it is significantly faster compared to other methods. Algorithm 8 described the procedure of AM.

Algorithm 8: The Addition Method

1: Input n the number of jobs, machines and factories, where $n \geq 2$.
2: Input C_{ijk} the matrix of the allocation costs, where (i, j and $k = 1, 2, ..., n$).
3: Add the costs of each of the n machines in all the factories allocated to all the jobs. $n \times n$ costs matrix is formulated.
4: Apply the Hungarian assignment method to the $n \times n$ matrix formulated from (3).
5: Formulate another linear $n \times n$ dimensional costs matrix of the allocated machines in the factories selected from (4).
6: By applying the Hungarian assignment method again, we will reach the required allocation of jobs, machines and factories.
7: Sum the selected allocations total costs. Stop.

The following Example 4.4.1 explains how to apply AM to SAP.

Example 4.4.1 Let us consider the size of the problem as $n = 4$. Let C be the allocation costs matrix for all the jobs, machines and factories as shown in the following matrix;

$$\mathbf{C} = \begin{pmatrix} 8 & 6 & 9 & 9 & \quad 4 & 6 & 6 & 6 & \quad 2 & 6 & 4 & 1 & \quad 7 & 6 & 9 & 7 \\ 9 & 0 & 9 & 4 & \quad 9 & 0 & 7 & 1 & \quad 0 & 3 & 3 & 4 & \quad 7 & 1 & 3 & 2 \\ 1 & 2 & 1 & 8 & \quad 7 & 8 & 7 & 7 & \quad 0 & 9 & 7 & 4 & \quad 2 & 1 & 5 & 5 \\ 9 & 5 & 9 & 1 & \quad 9 & 9 & 3 & 0 & \quad 8 & 0 & 7 & 6 & \quad 6 & 4 & 2 & 6 \end{pmatrix}.$$

Our target is to obtain an optimum or near optimum solution for a SAP allocation. To solve this example, apply step (3) to convert the problem from three dimensions to two; we will have **S** as a 4×4 matrix. We have added the costs of workers on the machines for

every job at each factory as follow;

$$\mathbf{S} = \begin{pmatrix} 32 & 22 & 13 & 29 \\ 22 & 17 & 10 & 13 \\ 12 & 29 & 20 & 13 \\ 24 & 21 & 21 & 18 \end{pmatrix}.$$

Using step 5 and applying the Hungarian method we will have the allocation in boxed figures as follows;

$$\mathbf{S} = \begin{pmatrix} 32 & 22 & \boxed{13} & 29 \\ 22 & 17 & 10 & \boxed{13} \\ \boxed{12} & 29 & 20 & 13 \\ 24 & \boxed{21} & 21 & 18 \end{pmatrix}.$$

These boxed figures represent the original allocation of jobs to the machines in the factories as it is shown as follows;

$$\mathbf{C} = \begin{pmatrix} 8 & 6 & 9 & 9 & 4 & 6 & 6 & 6 & \boxed{2} & \boxed{6} & \boxed{4} & \boxed{1} & 7 & 6 & 9 & 7 \\ 9 & 0 & 9 & 4 & 9 & 0 & 7 & 1 & 0 & 3 & 3 & 4 & \boxed{7} & \boxed{1} & \boxed{3} & \boxed{2} \\ \boxed{1} & \boxed{2} & \boxed{1} & \boxed{8} & 7 & 8 & 7 & 7 & 0 & 9 & 7 & 4 & 2 & 1 & 5 & 5 \\ 9 & 5 & 9 & 1 & \boxed{9} & \boxed{9} & \boxed{3} & \boxed{0} & 8 & 0 & 7 & 6 & 6 & 4 & 2 & 6 \end{pmatrix}.$$

The next step is to select the allocated costs and arrange a linear 4×4 matrix **SS**. The step will convert the problem from three dimensions into two. Now we can apply the Hungarian assignment method again and we have converted the matrix SS with circled

allocation costs as follows;

$$\mathbf{SS} = \begin{pmatrix} \textcircled{2} & 6 & 4 & 1 \\ 7 & \textcircled{1} & 3 & 2 \\ 1 & 2 & \textcircled{1} & 8 \\ 9 & 9 & 3 & \textcircled{0} \end{pmatrix}.$$

The allocation of jobs, machines and factories are,

Factory F_1 : Job J_3 : Machine $M_3 = 1$.

Factory F_2 : Job J_4 : Machine $M_4 = 0$.

Factory F_3 : Job J_1 : Machine $M_1 = 2$.

Factory F_4 : Job J_2 : Machine $M_2 = 1$.

The total minimum cost = 4.

The final circled allocations are assigned in **C** costs matrix as follows;

$$\mathbf{C} = \begin{pmatrix} 8 & 6 & 9 & 9 & 4 & 6 & 6 & 6 & \textcircled{2} & 6 & 4 & 1 & 7 & 6 & 9 & 7 \\ 9 & 0 & 9 & 4 & 9 & 0 & 7 & 1 & 0 & 3 & 3 & 4 & 7 & \textcircled{1} & 3 & 2 \\ 1 & 2 & \textcircled{1} & 8 & 7 & 8 & 7 & 7 & 0 & 9 & 7 & 4 & 2 & 1 & 5 & 5 \\ 9 & 5 & 9 & 1 & 9 & 9 & 3 & \textcircled{0} & 8 & 0 & 7 & 6 & 6 & 4 & 2 & 6 \end{pmatrix}.$$

The total cost is the summation of the circled costs $1 + 0 + 2 + 1 = 4$.

4.5 Tie Cases

In this section we consider the different tie cases that play important roles and affect the results of the solution of SAP. One of the points is the nature of the data used, if the data are homogeneous or not, or there are extreme values among the data either very small or very large this will divert the results. For example, if we look at the allocation costs, we will notice that some of the costs are extremely large and some times the data look like similar or very close to each other. This situation makes it very difficult to decide which allocation cost to select. If this happened we call it a tie case or a tie situation.

Let us consider the $4\times4\times4$ dimensions SAP problem. As it has been discussed before, the allocation costs is given as follows;

$$\mathbf{C} = \begin{pmatrix} 3 & 4 & 2 & 3 & & 2 & 4 & 4 & 2 & & 1 & 4 & 4 & 4 & & 2 & 2 & 4 & 4 \\ 2 & 6 & 2 & 2 & & 3 & 4 & 2 & 3 & & 3 & 3 & 3 & 3 & & 3 & 1 & 4 & 5 \\ 3 & 4 & 3 & 2 & & 5 & 2 & 1 & 5 & & 3 & 4 & 3 & 3 & & 4 & 2 & 3 & 3 \\ 6 & 2 & 4 & 1 & & 2 & 5 & 2 & 3 & & 3 & 5 & 2 & 2 & & 3 & 4 & 3 & 2 \end{pmatrix}.$$

Apply Algorithm 8 by adding the allocation costs of the machines in each factory and applying the Hungarian method, we have different results as it is explained in the following tie cases;

4.5.1 Tie problem: Case 1

$$\mathbf{S} = \begin{pmatrix} \enclose{circle}{12} & 12 & 13 & 12 \\ 12 & \enclose{circle}{12} & 12 & 13 \\ 12 & 13 & 13 & \enclose{circle}{12} \\ 13 & 12 & \enclose{circle}{12} & 12 \end{pmatrix}, \quad \mathbf{SS} = \begin{pmatrix} 3 & 4 & \enclose{circle}{2} & 3 \\ \enclose{circle}{3} & 4 & 2 & 3 \\ 4 & \enclose{circle}{2} & 3 & 3 \\ 3 & 5 & 2 & \enclose{circle}{2} \end{pmatrix}.$$

$$\mathbf{C} = \begin{pmatrix} 3 & 4 & \enclose{circle}{2} & 3 & 2 & 4 & 4 & 2 & 1 & 4 & 4 & 4 & 2 & 2 & 4 & 4 \\ 2 & 6 & 2 & 2 & \enclose{circle}{3} & 4 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 1 & 4 & 5 \\ 3 & 4 & 3 & 2 & 5 & 2 & 1 & 5 & 3 & 4 & 3 & 3 & 4 & \enclose{circle}{2} & 3 & 3 \\ 6 & 2 & 4 & 1 & 2 & 5 & 2 & 3 & 3 & 5 & 2 & \enclose{circle}{2} & 3 & 4 & 3 & 2 \end{pmatrix}.$$

The solution is the sum of all the allocated costs $2 + 3 + 2 + 2 = 9$.

4.5.2 Tie problem: Case 2

$$\mathbf{S} = \begin{pmatrix} 12 & 12 & 13 & \enclose{circle}{12} \\ 12 & 12 & \enclose{circle}{12} & 13 \\ \enclose{circle}{12} & 13 & 13 & 12 \\ 13 & \enclose{circle}{12} & 12 & 12 \end{pmatrix}, \quad \mathbf{SS} = \begin{pmatrix} 2 & \enclose{circle}{2} & 4 & 4 \\ 3 & 3 & \enclose{circle}{3} & 3 \\ 3 & 4 & 3 & \enclose{circle}{2} \\ \enclose{circle}{2} & 5 & 2 & 3 \end{pmatrix}.$$

$$\mathbf{C} = \begin{pmatrix} 3 & 4 & 2 & 3 & 2 & 4 & 4 & 2 & 1 & 4 & 4 & 4 & 2 & \enclose{circle}{2} & 4 & 4 \\ 2 & 6 & 2 & 2 & 3 & 4 & 2 & 3 & 3 & 3 & \enclose{circle}{3} & 3 & 3 & 1 & 4 & 5 \\ 3 & 4 & 3 & \enclose{circle}{2} & 5 & 2 & 1 & 5 & 3 & 4 & 3 & 3 & 4 & 2 & 3 & 3 \\ 6 & 2 & 4 & 1 & \enclose{circle}{2} & 5 & 2 & 3 & 3 & 5 & 2 & 2 & 3 & 4 & 3 & 2 \end{pmatrix}.$$

The solution after applying the Hungarian method is the sum of all the allocated costs

$2 + 3 + 2 + 2 = 9.$

4.5.3 Tie problem: Case 3

$$\mathbf{S} = \begin{pmatrix} 12 & \enclose{circle}{12} & 13 & 12 \\ \enclose{circle}{12} & 12 & 12 & 13 \\ 12 & 13 & 13 & \enclose{circle}{12} \\ 13 & 12 & \enclose{circle}{12} & 12 \end{pmatrix}, \quad \mathbf{SS} = \begin{pmatrix} \enclose{circle}{2} & 4 & 4 & 2 \\ 2 & 6 & \enclose{circle}{2} & 2 \\ 4 & \enclose{circle}{2} & 3 & 3 \\ 3 & 5 & 2 & \enclose{circle}{2} \end{pmatrix}.$$

$$\mathbf{C} = \begin{pmatrix} 3 & 4 & 2 & 3 & \enclose{circle}{2} & 4 & 4 & 2 & 1 & 4 & 4 & 4 & 2 & 2 & 4 & 4 \\ \enclose{circle}{2} & 6 & 2 & 2 & 3 & 4 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 1 & 4 & 5 \\ 3 & 4 & 3 & 2 & 5 & 2 & 1 & 5 & 3 & 4 & 3 & 3 & 4 & \enclose{circle}{2} & 3 & 3 \\ 6 & 2 & 4 & 1 & 2 & 5 & 2 & 3 & 3 & 5 & 2 & \enclose{circle}{2} & 3 & 4 & 3 & 2 \end{pmatrix}.$$

The solution after applying the Hungarian method is the sum of all the allocated costs $2 + 2 + 2 + 2 = 8.$

4.5.4 Tie problem: Case 4

$$\mathbf{S} = \begin{pmatrix} 12 & \enclose{circle}{12} & 13 & 12 \\ 12 & 12 & \enclose{circle}{12} & 13 \\ \enclose{circle}{12} & 13 & 13 & 12 \\ 13 & 12 & 12 & \enclose{circle}{12} \end{pmatrix}, \quad \mathbf{SS} = \begin{pmatrix} \enclose{circle}{2} & 4 & 4 & 2 \\ 3 & \enclose{circle}{3} & 3 & 3 \\ 3 & 4 & 3 & \enclose{circle}{2} \\ 3 & 4 & \enclose{circle}{3} & 2 \end{pmatrix}.$$

$$\mathbf{C} = \begin{pmatrix} 3 & 4 & 2 & 3 & \textcircled{2} & 4 & 4 & 2 & 1 & 4 & 4 & 4 & 2 & 2 & 4 & 4 \\ 2 & 6 & 2 & 2 & 3 & 4 & 2 & 3 & 3 & \textcircled{3} & 3 & 3 & 3 & 1 & 4 & 5 \\ 3 & 4 & 3 & \textcircled{2} & 5 & 2 & 1 & 5 & 3 & 4 & 3 & 3 & 4 & 2 & 3 & 3 \\ 6 & 2 & 4 & 1 & 2 & 5 & 2 & 3 & 3 & 5 & 2 & 2 & 3 & 4 & \textcircled{3} & 2 \end{pmatrix}.$$

The solution after applying the Hungarian method is the sum of all the allocated costs $2 + 3 + 2 + 3 = 10$.

4.5.5 Tie problem: Case 5

$$\mathbf{S} = \begin{pmatrix} \textcircled{12} & 12 & 13 & 12 \\ 12 & 12 & \textcircled{12} & 13 \\ 12 & 13 & 13 & \textcircled{12} \\ 13 & \textcircled{12} & 12 & 12 \end{pmatrix}, \quad \mathbf{SS} = \begin{pmatrix} 3 & 4 & \textcircled{2} & 3 \\ 3 & 3 & 3 & \textcircled{3} \\ 4 & \textcircled{2} & 3 & 3 \\ \textcircled{2} & 5 & 2 & 3 \end{pmatrix}.$$

$$\mathbf{C} = \begin{pmatrix} 3 & 4 & \textcircled{2} & 3 & 2 & 4 & 4 & 2 & 1 & 4 & 4 & 4 & 2 & 2 & 4 & 4 \\ 2 & 6 & 2 & 2 & 3 & 4 & 2 & 3 & 3 & 3 & 3 & \textcircled{3} & 3 & 1 & 4 & 5 \\ 3 & 4 & 3 & 2 & 5 & 2 & 1 & 5 & 3 & 4 & 3 & 3 & 4 & \textcircled{2} & 3 & 3 \\ 6 & 2 & 4 & 1 & \textcircled{2} & 5 & 2 & 3 & 3 & 5 & 2 & 2 & 3 & 4 & 3 & 2 \end{pmatrix}.$$

The solution after applying the Hungarian method is the sum of all the allocated costs $2 + 3 + 2 + 2 = 9$.

As we can see from the five tie problem cases, there are different results and the best solution is tie problem case 3 where the minimum allocation cost is 8. But unfortunately non of the cases reach the optimum solution as the method failed to reach the best result because the nature of the input data. Although the problem is small size, we have 5 cases.

This method is good if the problem's size is large. The data will disperse in a large space. Although we have five options but the optimum was not obtained. The best solution is as follows;

$$\mathbf{S} = \begin{pmatrix} 12 & 12 & \textcircled{13} & 12 \\ 12 & 12 & 12 & \textcircled{13} \\ 12 & \textcircled{13} & 13 & 12 \\ \textcircled{13} & 12 & 12 & 12 \end{pmatrix}, \quad \mathbf{SS} = \begin{pmatrix} \textcircled{1} & 4 & 4 & 4 \\ 3 & \textcircled{1} & 4 & 5 \\ 5 & 2 & \textcircled{1} & 5 \\ 6 & 2 & 4 & \textcircled{1} \end{pmatrix}.$$

$$\mathbf{C} = \begin{pmatrix} 3\ 4\ 2\ \ 3 & 2\ 4\ \ 4\ \ 2 & \textcircled{1}\ 4\ 4\ 4 & 2\ \ 2\ \ 4\ 4 \\ 2\ 6\ 2\ \ 2 & 3\ 4\ \ 2\ \ 3 & 3\ \ 3\ 3\ 3 & 3\ \textcircled{1}\ 4\ 5 \\ 3\ 4\ 3\ \ 2 & 5\ 2\ \textcircled{1}\ 5 & 3\ \ 4\ 3\ 3 & 4\ \ 2\ \ 3\ 3 \\ 6\ 2\ 4\ \textcircled{1} & 2\ 5\ \ 2\ \ 3 & 3\ \ 5\ 2\ 2 & 3\ \ 4\ \ 3\ 2 \end{pmatrix}.$$

The optimum solution is the sum of all the allocated costs $1 + 1 + 1 + 1 = 4$.

4.6 Case Study: Different Functions for the Addition Method

In this case study we have used two different functions $\mathbf{A_1}$ and $\mathbf{A_2}$ with random selecting data. We applied the AM starting from size 50 up to 800. The AM reaches an optimum solution and total cost is zero in fast time, the two functions used to generate two sets of different random unbiased data. The first function $\mathbf{A_1} = \mathbf{fix(10 * rand(n, n, n))}$ generate random data using MATLAB code, the second function $\mathbf{A_2} = \mathbf{fix(a + (b - a).*rand(n, n, n))}$ generate random numbers, where $a = 0,\ b = 10$.

Table 4.14 shows that when the size of the problem starts from $n = 50$ we reach near

optimum solution, but at size $n = 100$ up to $n = 800$ we reached the optimum solution and the total cost is zero from both unbiased functions $\mathbf{A_1}$ and $\mathbf{A_2}$ with considerable times 1113.1 and 1101.6 seconds for both functions.

In Table 4.15 we used another two functions to generate unbiased random data with different range of decimal numbers, $\mathbf{A_3} = (\mathbf{100} * \mathbf{rand}(\mathbf{n}, \mathbf{n}, \mathbf{n}))$ is the first function and $\mathbf{A_4} = \mathbf{fix}(\mathbf{a} + (\mathbf{b} - \mathbf{a}). * \mathbf{rand}(\mathbf{n}, \mathbf{n}, \mathbf{n}))$ is the second function where $a = 0,\ b = 100$. Because we used real numbers it is obvious that with increasing the size of the problem, the total cost decreases and that is clear from the random distributed function $\mathbf{A_4}$. We have reached optimum or near solution at problem size 400, the cost is 2 and the time is 47.4575 seconds.

The four functions that generate random numbers simply spread out the uniformly generated numbers in different intervals and with larger ranges the chance of having Tie cases is possible.As it is shown in Table 4.14 and Table 4.1.

	$\mathbf{A_1}$		$\mathbf{A_2}$	
n	Total Cost	Time (Sec)	Total Cost	Time (Sec)
50	1	0.0716	1	0.0709
100	0	0.2516	0	0.2578
150	0	0.7901	0	0.7920
200	0	2.4401	0	2.4200
250	0	6.0753	0	6.0536
300	0	12.4541	0	12.2679
350	0	22.4312	0	22.5187
400	0	47.5084	0	47.4077
450	0	85.9222	0	85.1503
500	0	139.6855	0	138.2706
550	0	211.0530	0	209.8084
600	0	307.1052	0	303.1334
650	0	432.1390	0	441.6483
700	0	599.0299	0	603.1973
750	0	798.3316	0	827.4262
800	0	1113.1	0	1101.6

Table 4.14: AM: The $\mathbf{A_1}$ and $\mathbf{A_2}$ total cost comparisons

	$\mathbf{A_3}$		$\mathbf{A_4}$	
n	Total Cost	Time(Sec)	Total Cost	Time(Sec)
50	108.2962	0.0845	78.0000	0.0793
100	103.3870	0.2906	55.0000	0.2785
150	120.1544	0.9324	49.0000	0.8483
200	109.8225	2.6417	24.0000	2.6401
250	128.3742	6.4571	26.0000	6.6330
300	128.6120	13.0053	16.0000	13.0731
350	124.5820	23.1420	8.0000	23.2739
400	133.4406	48.4694	2.0000	47.4575
450	126.9781	85.5552	1.0000	85.2628
500	129.9565	138.1425	2.0000	137.9296
550	128.6686	209.0964	0	213.7691
600	132.6624	312.9279	1.0000	316.2109
650	140.2924	448.8763	0	436.7286
700	141.5285	583.4882	0	591.5969

Table 4.15: AM: The $\mathbf{A_3}$ and $\mathbf{A_4}$ total cost comparisons.

We used other two different random Poisson distribution functions B_1 and B_2. Monte Carlo simulation applied for 20 instances started from $n = 10$ up to $n = 500$ with different intervals. The average results that we obtained are more accurate and reliable than the previous results generated from single instance of A_1, A_2, A_3 and A_4 functions. The two functions B_1 and B_2 that generate the random numbers simply spread out the Poisson generated numbers in different mean (λ) and with large size problems up to $n = 500$, the result converges to small cost. ($\lambda = 3$) applied in Poisson formula of the results in Table 4.16 and ($\lambda = 5$) in Table 4.17. The mathematical formula for Poisson distribution function is as follows;

$$f(x \mid \lambda) = \frac{\lambda^x}{x!} e^{-\lambda}; \quad x = 0, 1, \ldots, \infty. \tag{4.1}$$

Where the mean is λ and the standard deviation is $\sqrt{\lambda}$.

n	Instances	$\mathbf{B_1}$ Average Cost	$\mathbf{B_1}$ Average Time (Sec)
10	20	5.85	0.0042106
20	20	5.7425	0.0081049
30	20	3.3871	0.012325
40	20	2.6694	0.023673
50	20	1.6835	0.03754
60	20	0.63417	0.056388
70	20	0.53171	0.081303
80	20	0.62659	0.11717
90	20	0.23133	0.16443
100	20	0.061566	0.21835
110	20	0.0030783	0.28375
120	20	0.050154	0.37438
130	20	0.0025077	0.48059
140	20	0.00012538	0.61178
150	20	0.050006	0.78012
160	20	0.00012502	0.99111
170	20	6.2508 exp(−06)	1.2447
180	20	3.1254 exp(−07)	1.627
190	20	1.5627 exp(−08)	2.0574
200	20	0	2.5709
250	20	0	6.4788
300	20	0	13.23
350	20	0	23.534
400	20	0	49.101
450	20	0	87.738
500	20	0	144.200

Table 4.16: AM: The Poisson $\mathbf{B_1}$ comparisons, $\lambda = 3$ for 20 instances.

n	Instances	$\mathbf{B_2}$ Average Cost	$\mathbf{B_2}$ Average Time (Sec)
10	20	16.300	0.0097539
20	20	25.265	0.0095567
30	20	31.863	0.016358
40	20	34.943	0.03632
50	20	38.847	0.05577
60	20	41.292	0.067569
70	20	42.865	0.10935
80	20	45.693	0.13912
90	20	46.235	0.19125
100	20	47.012	0.27391
110	20	47.001	0.35474
120	20	49.450	0.45579
130	20	49.673	0.56916
140	20	51.034	0.70500
150	20	49.252	0.90868
160	20	50.463	1.10020
170	20	48.873	1.35250
180	20	46.994	1.65290
190	20	46.500	2.05740
200	20	45.175	2.78240
250	20	37.900	6.84280
300	20	30.195	13.5870
350	20	23.860	25.1420
400	20	19.943	51.4130
450	20	15.497	87.1970
500	20	12.025	142.680

Table 4.17: AM: The Poisson $\mathbf{B_2}$ comparisons, $\lambda = 5$ for 20 instances.

4.7 The Multiplication Method

The Multiplication Method (MM) is another approach to find the basic initial solution to SAP. It is similar to the previous AM. The only difference is applying the multiplication instead of the addition operator. The problem can be solved based on two stages. The first stage is to convert SAP from three dimensional into two dimensions linear problem . We will simply multiply the costs of allocation machines in each factory while in the second stage we will use the well known Hungarian method, [42].

In the first stage will convert the problem from SAP into a linear two dimensional problem. In the second stage applying the Hungarian method twice to solve the problem and reach an optimum or near solution. This method is suitable for small and medium size problems, because of the nature of the multiplication operator, the multiplication process effect will increase significantly when the size of the problem become larger. MM is relatively faster compared to other methods. Algorithm 9 shows the procedures of MM;

Algorithm 9: The Multiplication Method

1: Input n to be the number of jobs, machines and factories, where $n \geq 2$.
2: Input C_{ijk} to be the matrix of the allocation costs, where (i, j and $k = 1, 2, ..., n$).
3: Multiply the costs of all n machines in each factory allocated to all the jobs. Formulate an $n \times n$ costs matrix.
4: Apply the Hungarian method to the $n \times n$ matrix formulated from 3.
5: Formulate another linear $n \times n$ costs matrix of the allocated machines in the factories selected from 4.
6: By applying the Hungarian method again, we will reach the required allocation of jobs, machines and factories.
7: Sum the allocation total costs. Stop.

Example 4.7.1 explains how to apply the MM to solve the SAP.

Example 4.7.1 Let the dimension of the problem is $n = 4$, the objective function is subject to SAP constraints. Let us assume that **C** be the allocation costs matrix for all the jobs, machines and factories as shown in the following **C** cost matrix;

$$\mathbf{C} = \begin{pmatrix} 81 & 63 & 95 & 95 & & 42 & 65 & 67 & 65 & & 27 & 69 & 43 & 18 & & 70 & 65 & 95 & 75 \\ 90 & 09 & 96 & 48 & & 91 & 03 & 75 & 17 & & 04 & 31 & 38 & 48 & & 75 & 16 & 34 & 25 \\ 12 & 27 & 15 & 80 & & 79 & 84 & 74 & 70 & & 09 & 95 & 76 & 44 & & 27 & 11 & 58 & 50 \\ 91 & 54 & 97 & 14 & & 95 & 93 & 39 & 03 & & 82 & 03 & 79 & 64 & & 67 & 49 & 22 & 69 \end{pmatrix}.$$

Our target is to reach an optimum or near optimum for allocation of SAP. We applied Algorithm 9 step 3. Multiply the costs of each row (Machines) in each square matrix (Factories), the outcomes are as follow;

$$\mathbf{S} = \begin{pmatrix} 46054575 & 11889150 & 1441962 & 32418750 \\ 3732480 & 348075 & 226176 & 1020000 \\ 388800 & 34374480 & 2859120 & 861300 \\ 6673212 & 1033695 & 1243776 & 4983594 \end{pmatrix}.$$

Using step 5 and applying the Hungarian method we will have the allocation in the boxed numbers as follows;

$$\mathbf{S} = \begin{pmatrix} 46054575 & 11889150 & \boxed{1441962} & 32418750 \\ 3732480 & 348075 & 226176 & \boxed{1020000} \\ \boxed{388800} & 34374480 & 2859120 & 861300 \\ 6673212 & \boxed{1033695} & 1243776 & 4983594 \end{pmatrix}.$$

These boxed numbers represent the original allocation of jobs to the machines in the factories as it is shown as follows;

$$\mathbf{C} = \begin{pmatrix} 81 & 63 & 95 & 95 & 42 & 65 & 67 & 65 & \boxed{27} & \boxed{69} & \boxed{43} & \boxed{18} & 70 & 65 & 95 & 75 \\ 90 & 09 & 96 & 48 & 91 & 03 & 75 & 17 & 04 & 31 & 38 & 48 & \boxed{75} & \boxed{16} & \boxed{34} & \boxed{25} \\ \boxed{12} & \boxed{27} & \boxed{15} & \boxed{80} & 79 & 84 & 74 & 70 & 09 & 95 & 76 & 44 & 27 & 11 & 58 & 50 \\ 91 & 54 & 97 & 14 & \boxed{95} & \boxed{93} & \boxed{39} & \boxed{03} & 82 & 03 & 79 & 64 & 67 & 49 & 22 & 69 \end{pmatrix}$$

The next step is to select the boxed cost and arrange a linear 4×4 matrix cost **SS**.

Applying the Hungarian assignment method again, we will have the following matrix in circled allocations.

$$\mathbf{SS} = \begin{pmatrix} \textcircled{27} & 69 & 43 & 18 \\ 75 & \textcircled{16} & 34 & 25 \\ 12 & 27 & \textcircled{15} & 80 \\ 95 & 93 & 39 & \textcircled{03} \end{pmatrix}.$$

The allocation jobs, machines and factories are as follows;

Factory F_1 : Job J_3 : Machine $M_3 = 15$.

Factory F_2 : Job J_4 : Machine $M_4 = 03$.

Factory F_3 : Job J_1 : Machine $M_1 = 27$.

Factory F_4 : Job J_2 : Machine $M_2 = 16$.

The final circled allocations are assigned in **C** costs matrix as follows;

$$\mathbf{C} = \begin{pmatrix} 81 & 63 & 95 & 95 & 42 & 65 & 67 & 65 & \textcircled{27} & 69 & 43 & 18 & 70 & 65 & 95 & 75 \\ 90 & 09 & 96 & 48 & 91 & 03 & 75 & 17 & 04 & 31 & 38 & 48 & 75 & \textcircled{16} & 34 & 25 \\ 12 & 27 & \textcircled{15} & 80 & 79 & 84 & 74 & 70 & 09 & 95 & 76 & 44 & 27 & 11 & 58 & 50 \\ 91 & 54 & 97 & 14 & 95 & 93 & 39 & \textcircled{03} & 82 & 03 & 79 & 64 & 67 & 49 & 22 & 69 \end{pmatrix}.$$

The total cost is the summation of the circled costs = 15 + 03 + 27 + 16 = 61.

4.8 Summary

We have designed new heuristic approaches to solve SAP. They are the Average Cost Method (ACM), the Addition Method (AM) and the Multiplication Method (MM). The algorithms have been presented, illustrated and tested on some small size problems. Comparisons tables of the AD with large size problems by using Poisson distribution with random generating numbers explains the convergence of the total cost to zero which indicate that the solution is either optimal or near optimal solution. Moreover, a review of the literature relevant to the problem has been presented. The main issue is to understand why the method works at all, how reliable it is, how robust, accurate and efficient it is compared to other approaches.

Chapter 5

The Genetic Algorithm

5.1 Introduction

The Genetic Algorithm (GA) is a directed search algorithm based on the mechanics of biological evolution. Developed by John Holland, [2], the idea is to use genes and chromosomes as encoding techniques. The following definitions are useful to understand the terms used in genetic algorithms.

Definition 5.1.1 Fitness Function

Fitness function is used to rank solutions against each other. It is often the objective function of the optimisation problem in hand.

Definition 5.1.2 Elitism

Elitism is a method used to guarantee keeping the good or the best chromosome that we need to use them in the next generated population.

When we apply crossover and mutation on the chromosomes, there is a chance that

we will miss selection of the good or best chromosomes. To avoid the loss of the best chromosomes, while generating new population, we are using elitism method to select the good chromosomes and copying them to the next generation. The rest of chromosomes are done in classical way. Elitism can increases the performance of the GA to be fast, because it prevents losing the best found solution.

Definition 5.1.3 Reproduction

Parents are selected at random with selection chances biased in relation to chromosome evaluations.

The cycle of the reproduction of a population from the parents and children explained in the following relationship;

$$\begin{array}{ccc} \text{Reproduction} \longrightarrow & \text{Children / Parents} \\ & \downarrow \uparrow \\ & \text{Population.} \end{array}$$

Definition 5.1.4 Modifications

The affects happened on chromosomes by changing the structure of them when applying mutation or crossover (recombination).

The chromosomes and mutations are playing an important rule in changing and improving the structure of the children, generation after generation. The following relationship shows the modification of the children.

Children $\longrightarrow$ Modification (chromosomes and mutation)

$\downarrow$

Modified Children.

Definition 5.1.5 Evaluation Operator

Evaluation is the operator that link between the fitness function that we wanted to solve and the GA coding. The evaluator is responsible to assign the chromosome of the child a fitness measure.

The following relationship of modifying the children using the evaluation operator as follows;

Modified Children

$\downarrow$

Evaluation (assign a fitness measure) $\longrightarrow$ Evaluated Children.

Definition 5.1.6 Roulette Wheel Selection

Parents are selected according to their fitness values. The better chromosomes have more chances to be selected. Imagine a roulette wheel where all chromosomes in the population are placed and the area each chromosome has corresponds to its fitness function, like on a circle divided into different pie sectors, then a marble is thrown there and it selects the chromosome. Chromosomes with higher rank fitness will be selected more times.

Definition 5.1.7 Steady State Selection

Some population with good chromosomes replaced other with bad chromosome in each generation.

The following points are representing the basic outline of the genetic algorithm to solve SAP.

1. **Initialisation**

   Initialisation is the starting phase to create genes and chromosomes. After defining the size n of SAP, generate random population of size N chromosomes. Each chromosome has a number of genes. N is not fixed and its variation depends on the size n of the problem. If n is large the population N is large too and $N > n > 0$.

2. **Fitness Function**

   The fitness function is to evaluate the function $f(x)$ of each chromosome x in the population. The fitness function for SAP is to minimise $f(x)$ subject to the constraints as it was mentioned in Chapter 2, Equations 2.6 to 2.10. $C = \{c_{ijk}\}$ is the cost or the weight of allocating.

3. **New Population**

   Until the mating in the pool is completed, the new population can be created by repeating the process using one or more of the following operators;

   - **Selection**

     Selection is a procedure to select chromosomes (parents) in the population for reproduction, selecting two chromosomes according to their fitness. The process determines which to be kept and allowed to reproduce and which to be removed. There are different techniques to implement selection in the genetic algorithms, we used roulette wheel, rank and steady state selection operators. There are different selection's operators such as;

(a) Tournament.

(b) Roulette Wheel.

(c) Proportionate.

(d) Rank.

(e) Steady State.

(f) Others.

- **Crossover**

  It is a second operator to select randomly two parents with a crossover probability from the population, either to select a single point or several points of the chromosome from each parent. Crossover two parents to form new children. For two points crossover; select randomly a segment size of genomes from both parents, crossover them to form new offspring (children). If no crossover performed, it means that offspring is an exact copy of parents.

- **Mutation**

  The mutation is a third operator to select randomly a chromosome from the population. With a mutation probability, mutate a new offspring by selecting randomly two genomes from a parent chromosome from the population and swap or flip their positions in the same chromosome.

- **Accepting**

  By accepting the procedure of mating it means we accept to place new offspring in a new population.

4. **Replace**

Replace the old generation by a new generated population for further run of the algorithm.

5. **Test**

   The running process is tested to see if the conditions have been satisfied. It will terminate and stop if it is satisfied and return the best solution in current population.

6. **Repeat**

   Repeat the process if it is not satisfied to find better fitness function.

5.2 The Genetic Algorithm for SAP

The fitness function is to optimise the total cost of SAP subject to the constraints where the allocation costs are known. To simplify the problem, presume that one allocation is successful; mathematically it is **ON** and it is represented by **1**, if it is not allocated it is **OFF** and it represented by **0**. Suppose a size 3 SAP problem with costs as shown in the following Table 5.1.

Factories	F_1			F_2			F_3		
Machines	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3
J_1	4	3	2	1	2	3	1	3	2
Jobs J_2	1	2	3	4	5	3	4	2	1
J_3	1	1	2	6	1	2	1	3	1

Table 5.1: GA: SAP costs matrix.

Rewrite Table 5.1 as it is shown in Table 5.2 where the allocations for a_1 = {4,3,2 }, a_2 = {1,2,3 } and a_3 = {1,3,2 }, same process applied to b_1 = {1,2,3 }, b_2 = {4,5,3 } and b_3 = {4,2,1 }, finally c_1 = {1,1,2 }, c_2 = {6,1,2 } and c_3 = {1,3,1 }.

Factories	F_1	F_2	F_3
Machines	M_1 M_2 M_3	M_1 M_2 M_3	M_1 M_2 M_3
J_1	a_1	a_2	a_3
Jobs J_2	b_1	b_2	b_3
J_3	c_1	c_2	c_3

Table 5.2: GA: Symbolised Costs.

To solve SAP using the GA, SAP can be represented with a suitable fitness function and generating a set of population. Let assume the random permuted number $n = 1, 3, 2$ or a_1, b_3, c_2 and Table 5.3 shows the costs related to each allocation. Furthermore, the

Factories	F_1	F_2	F_3
Machines	M_1 M_2 M_3	M_1 M_2 M_3	M_1 M_2 M_3
J_1	$a_1 = 4, 3, 2$	-	-
Jobs J_2	-	-	$b_3 = 4, 2, 1$
J_3	-	$c_2 = 6, 1, 2$	-

Table 5.3: GA:Chromosome Allocations.

selected chromosome from Table 5.3 can be re-arranged as a two dimensional matrix, as in Table 5.4. Selecting the second random permuted number, say $n = 1, 3, 2$ means we are

	M_1	M_2	M_3
J_1	4	3	2
J_2	4	2	1
J_3	6	1	2

Table 5.4: GA: Chromosomes Matrix.

allocating the machines in the nominated factories (1,3,2). The new machine allocations can be written as binary digit as follows; {100,001,010} which means J_1 allocated to M_1 in F_1, J_2 allocated to M_3 in F_3 and finally J_3 is allocated to M_2 in F_2 or the final binary allocation is {**(1,1,1), (2,3,3), (3,2,2)**} and Table 5.5 shows that we have assigned the allocation. The total sum is $4 + 1 + 1 = 6$ and this is only one solution of many other solutions we need to go through them.

Factories	F_1			F_2			F_3		
Machines	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3
Jobs J_1	4	-	-	-	-	-	-	-	-
Jobs J_2	-	-	-	-	-	-	-	-	1
Jobs J_3	-	-	-	-	1	-	-	-	-

Table 5.5: GA: Job Allocations.

The important question is, how do we assign the jobs to the machines in the factories? It is important to understand the idea of this allocation as it plays an important role to implement the fitness function. We used two methods to do the assignment of the jobs to the machines in the factories and implementing the fitness function. These two methods will be explained in more details with examples in the next section.

5.3 Implementing the Fitness Function

Two methods have been used to implement the fitness function. The first method depends totally on selecting random permuted numbers twice to do allocation while the second used random permuted numbers and the Hungarian method to do the allocation. To implement the fitness function we will use random permutation encoding numbers which is defined and explained as follows;

Definition 5.3.1 Randomly Permuted Encode Function

The Randomly Permuted Encode Function (RPEF) is used in scheduling or allocation problems, such as the travelling salesman problem. It is useful for ordering problems. We will apply random permutation encoding in calculating the initial solution for SAP. We consider every chromosome as a string representing the factories or machines.

The following generating chromosomes are representing two strings of 12 random

permuted numbers.

Chromosome A: 08 02 10 07 01 04 09 05 12 11 06 03

Chromosome B: 04 07 11 02 10 03 09 05 06 01 08 12

5.4 Random Permuted Numbers Method to Solve SAP

The basic steps of the random permuted numbers method (RPNM) are as follows;

1. **Initialisation:** Initialise the problem and select the number of machines = n, the population = N where, $N > n$ and both n and N are non negative integer numbers.

2. **Randomly Permuted Numbers:** Generate random permute n numbers.

3. **Allocation:** Allocate the first random permuted number to the first job in the first factory and the second random permuted number to the second job in the second factory, continue until finishing all the permuted numbers. Assign the first set of machines to the first job in the first permuted factory, continue the same procedure and allocate all the sets of machines accordingly.

4. **Machine allocations:** Generate another random permuted n numbers to select the first permute number of n and fix it to the machine to be allocated to the first job in the first factory. Continue for all the remaining machines.

5. **Total cost** Calculate the total cost by adding the cost of each machine allocated to the job in the factory.

Example 5.4.1 In this example we will explain how to calculate the fitness function applying method (RPNM), let the number of machines to be allocated in each factory is

$n = 4$. There are four workers allocated to do the four jobs on the four machines in the four factories such that each worker is allowed to do one job only on one machine in one factory. The cost of doing the jobs on the machines in the factories are given in Table 5.6.

Factories	F_1				F_2				F_3				F_4			
Machines	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4
J_1	8	6	9	9	4	6	6	6	2	6	4	1	7	6	9	7
Jobs J_2	9	0	9	4	9	0	7	1	0	3	3	4	7	1	3	2
J_3	1	2	1	8	7	8	7	7	0	9	7	4	2	1	5	5
J_4	9	5	9	1	9	9	3	0	8	0	7	6	6	4	2	6

Table 5.6: GA: SAP Size 4 Problem

Applying the basic steps of the random permuted function. Let us assume the numbers (1 4 3 2) be the generated random permuted numbers for the machines is $n = 4$. Applying the allocation step, the first permuted number is 1 so we shall allocate all the machines in factory F_1 to the job J_1. Which means that J_1 can be allocated to any machine in factory F_1. As from Table 5.7 , the costs of allocating J_1 to do jobs in factory F_1 are $\{8, 6, 9, 9\}$, repeat the same procedure for the second random permuted number 4 and the costs are $\{7, 1, 3, 2\}$, for the third random permuted number 3 the costs are $\{0, 9, 7, 4\}$ and finally the fourth random permuted number 2 the costs are $\{9, 9, 3, 0\}$. The allocation positions are explained in the following Table 5.7 and Table 5.8.

Machines	M_1	M_2	M_3	M_4	F_i
J_1	8	6	9	9	F_1
Jobs J_2	7	1	3	2	F_4
J_3	0	9	7	4	F_3
J_4	9	9	3	0	F_2

Table 5.7: GA: Machine Allocations.

Next step is to start the allocations to achieve this, we will generate another n random permuted numbers, let us say {3 1 4 2}. From Table 5.8, the distribution of the machines will be the third machine M_3 is allocated to the first job J_1 in factory F_1 and the cost of allocation is 9. Also, the first

Factories	F_1				F_2				F_3				F_4			
Machines	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4
J_1	8	6	9	9	4	6	6	6	2	6	4	1	7	6	9	7
Jobs J_2	9	0	9	4	9	0	7	1	0	3	3	4	7	1	3	2
J_3	1	2	1	8	7	8	7	7	0	9	7	4	2	1	5	5
J_4	9	5	9	1	9	9	3	0	8	0	7	6	6	4	2	6

Table 5.8: GA: SAP Allocation Costs Matrix.

machine M_1 is allocated to job J_2 in factory F_4 and the cost of allocation is 7. Similarly, the second machine M_4 is allocated to job J_3 in the factory F_3 and the cost of allocation is 4. Finally, the second machine M_2 is allocated to job J_4 in factory F_2 and the cost of allocation is 9. The fitness function is equal to the total sum of the allocation costs $(9, 7, 4, 9)$, hence $f(x) = 29$. Table 5.9 and Table 5.10 show the circled costs of the allocated machined to the jobs in the factories. Until now we have

Machines	M_1	M_2	M_3	M_4	F_i
J_1	8	6	9	9	F_1
Jobs J_2	7	1	3	2	F_4
J_3	0	9	7	4	F_3
J_4	9	9	3	0	F_2

Table 5.9: GA: The SAP Allocation.

Factories	F_1				F_2				F_3				F_4			
Machines	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4
J_1	8	6	9	9	4	6	6	6	2	6	4	1	7	6	9	7
Jobs J_2	9	0	9	4	9	0	7	1	0	3	3	4	7	1	3	2
J_3	1	2	1	8	7	8	7	7	0	9	7	4	2	1	5	5
J_4	9	5	9	1	9	9	3	0	8	0	7	6	6	4	2	6

Table 5.10: GA: Circled Allocation Costs

calculated only one fitness function from the population. The total cost or the fitness value is the sum of the allocated machine cost and it is equal to 29. Repeat the same procedures to calculate all other fitness function for all the N population strings. After calculating all the population fitness function costs, we rank all the costs obtained by arranging them in ascending or descending order.

Next step is to apply the process of the roulette wheel operator to find the best solution among all the fitness function costs.

5.5 Random Permuted Numbers and the Hungarian Method

The basic steps of the random permuted numbers and the Hungarian method are explained in the following Algorithm 10.

Algorithm 10: Randomly Permute Number and the Hungarian Method

1: Initialisation
Initialise the problem and select n factories, N population, where $N > n > 0$.
2: Permute Function
Generate n randomly permuted factory numbers.
3: Allocation
Allocate the first random number to the first job in the first factory and the second number to the second job in the second factory, continue until finishing all the permuted numbers. Assign the first set of machines to the first job in the first permuted factory, continue the same procedure and allocate all the sets of machines accordingly.
4: Hungarian Assignment
Assemble all the sets of machines from the previous 3, apply the Hungarian method to the $n \times n$ selected set of Factories.
5: Total cost
Calculate the total cost of all the allocated machines by adding the cost allocation. Stop.

As mentioned before, this is only one set of allocation for only one fitness cost. Selecting a set of these chromosomes each time will enable us to generate the required population. The next step is to find out the other sets of allocations and calculate the related cost to each allocation. Then, sort all the costs in ascending or descending order and rank the chromosomes accordingly in order to select the best or the minimum cost.

The selection of the best solution depends on some factors such as the number of population, generation numbers, the type of chromosomes crossover (single or multiple),

the mutation of the chromosomes and the number of elites. All these factors and their percentages might affect the result and change the values of the fitness function, either to improve or worsen the final solution.

Example 5.5.1 Using Example 5.4.1 , we will explain how to calculate the fitness function applying the random permuted numbers and Hungarian assignment method. Apply the first two steps and Table 5.7 from the previous example and applying the Hungarian method we have the following solution as shows in Table 5.11, the circled numbers represent the allocation of machines to jobs in the permuted factories. It is clear from

Machines	M_1	M_2	M_3	M_4	F_i
J_1	8	6	(9)	9	F_1
Jobs J_2	7	(1)	3	2	F_4
J_3	(0)	9	7	4	F_3
J_4	9	9	3	(0)	F_2

Table 5.11: The SAP allocation.

Table 5.11 that machine M_3 is allocated to job J_1 in the first random permuted selected factory F_1, machine M_2 is allocated to job J_2 in the factory F_4, machine M_1 is allocated to job J_3 in the factory F_3 and machine M_4 is allocated to job J_4 into the factory F_2. The total cost is 10.

5.6 The Genetic Algorithm to Solve SAP

In this section we have used the GA method to solve the SAP with size $n = 3$.

let $\mathbf{C} = \{\mathbf{c_{ijk}}\}$ be the cost of allocation as follows;

$$\mathbf{C} = \begin{pmatrix} 81 & 91 & 27 & 96 & 95 & 14 & 79 & 03 & 67 \\ 90 & 63 & 54 & 15 & 48 & 42 & 95 & 84 & 75 \\ 12 & 09 & 95 & 97 & 80 & 91 & 65 & 93 & 74 \end{pmatrix}.$$

The matrix cost is divided into three matrices, the first matrix is the cost of the first factory, the second cost of the second factory and the third of the third factory. To initialise the problem let us presume the following input data.

Population = 10.

Generations = 20

Crossovers = 7.

Mutation = 1.

Elite = 1.

Population permute numbers			Hungarian assignment			Fitness value
3	1	2	2	3	1	154
2	3	1	3	2	1	110
1	2	3	3	1	2	135
2	1	3	3	2	1	142
1	3	2	3	1	2	202
3	1	2	2	3	1	154
2	1	3	3	2	1	142
3	1	2	2	3	1	154
3	2	1	2	3	1	57
1	2	3	3	1	2	135

Table 5.12: GA: Population numbers, Hungarian method and fitness value

Re-arranging the fitnesses values in descending order and repeating the process according the number of generations, $G = 20$. The final solution is as follows;

The population permuted is 3 2 1.

The Hungarian assignment cost is 2 3 1.

The Optimum cost is 57.

The final allocations and the costs are as it is explained in Table 5.13 Hence the final allocations are;

Factories		F_1			F_2			F_3		
Machines		M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3
	J_1	-	-	-	-	-	-	79	(3)	67
workers	J_2	-	-	-	15	48	(42)	-	-	-
	J_3	(12)	09	95	-	-	-	-	-	-

Table 5.13: GA: The Final Allocations.

First worker to do J_1 on M_2 in $F_3 = 03$.

Second worker to do J_2 on M_3 in $F_2 = 42$.

Third worker to do J_3 on M_1 in $F_1 = 12$.

The optimum value is 57, it is the same value comparing to the classical B&B method, the time was 0.29968 second while the execution time using B&B method was 0.55827 second. MATLAB code used on the same Lenovo laptop device.

For the permuted method without applying the Hungarian method, we have solved the same problem applying the random permuted function twice with the following data initialisation,

Population number = 10.

Generation number = 5.

Crossover number = 5.

Mutation number = 2.

Elite number = 2.

We have reached the same optimum value 57 and same allocation as the previous method, the execution time is 0.098029 second while in B&B method is 0.57638 second.

5.7 Summary

We have solved the SAP using GA. Two methods have been used, the first method depends on generating a randomly permuted number function twice. The first random number is required to select the factories allocated to the workers to do the jobs and the second random number is required to select the machines from the factories to be allocated to the jobs, these allocations will enable us to calculate the fitness value.

The second method also generates a random permuted number function to select the factories allocated to the workers to do the jobs as in the first method. But in the second stage we have applied the Hungarian method to assign the required machine in the factory to do the job, and to enable us to calculate the fitness value. We used crossover and mutation in both methods.

The crossover we have applied depends on selecting a segment of a random chromosome, the effect is to change and improve the result by generating a new offspring while the mutation will effect on the solution to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. Mutation changes one or more gene values in a chromosome from its initial state. Changes on the probabilities

number of the mutation might effect on the result of the problem as it was shown in (Section 5.6).

Both methods are approaching good results for small size dimension problems as it was shown in (Section 5.6), we have obtain optimum solution within 0.098029 second and this result was better than the result obtained by B&B. Applying the Hungarian method is more reliable than applying the random permuted number because it is biased and it relies on the techniques used by the Hungarian method but the disadvantage is costing some extra time and iterations. The other method is unbiased but needs large population size and large number of generations especially in higher dimensions.

Chapter 6

SAP with Monge Matrices

6.1 Introduction

In this chapter we have studied two dimensional and multidimensional Monge array sequences and some related properties. We also studied Supnick symmetric matrices and the relation with Monge. We will discuss their related researches and the mathematical formulation for solving the three dimensional assignment problem especially when the cost conditions are restricted. The following general definitions are from Hoffman, [71], Aggarawal, [72] and Supnick, [73].

The French scientist Monge, [74] studied the transportation problem. Hoffman, [71] mentioned and named the property that satisfied the $m \times n$ array used by Monge to solve the transportation problem. There is a plenty wealth of research work and applications related to the axial and the planar three dimensional assignment problems using the two dimensions and d-dimensional Monge array properties.

In this chapter we have dealt with a special case of multidimensional Monge, we have

relaxed the constraints of the axial solid assignment problems using Monge properties and applying an algorithm to find an exact solution for SAP.

Definition 6.1.1 An $m \times n$ two-dimensional array of real matrix $\mathbf{A} = \{a_{i,j}\}$, is called a Monge array if it satisfies the following property:

$$a_{i_1,j_1} + a_{i_2,j_2} \leq a_{i_1,j_2} + a_{i_2,j_1}, \tag{6.1}$$

for all rows i_1 and i_2 and columns j_1 and j_2 satisfying $1 \leq i_1 \leq i_2 \leq m$ and $1 \leq j_1 \leq j_2 \leq n$,

Definition 6.1.2 An $m \times n$ matrix $\mathbf{A}$ with unspecified elements is called incomplete Monge matrix if and only if for all $1 \leq i \leq r \leq m,\ 1 \leq j \leq s \leq n,$ the following holds:
If all four entries a_{ij}, a_{rs}, a_{is} and a_{rj} are specified, then they fulfil the Monge property

$$a_{ij} + a_{rs} \leq a_{rj} + a_{is}. \tag{6.2}$$

Incomplete Monge matrices are a generalisation of standard Monge matrices, the values of some entries are not specified and the Monge property only must hold for all specified entries [75].

Definition 6.1.3 An $m \times n$ two dimensional array $\mathbf{A} = \{a_{i,j}\}$ is inverse Monge if

$$a_{i,j} + a_{i+1,j+1} \geq a_{i,j+1} + a_{i+1,j},$$

for all i and j, $1 \leq i < m$ and $1 \leq j < n$.

Definition 6.1.4 An $m \times n$ two-dimensional array of real matrix $\mathbf{A} = \{a_{i,j}\}$, is called a monotone if the maximum value in its ith row lies below or to the right of the maximum value in its $(i-1)^{th}$ row.

An array $\mathbf{A}$ is called totally monotone if every 2×2 sub-array (*i.e.*, every 2×2 minor) is monotone.

Definition 6.1.5 Aggarwal and Park, [76] generalized the property of Monge for n dimensions as follows.

For $d \geq 2$, and $n_1 \times n_2 \times ... \times n_d$ d-dimensional array $\mathbf{A} = \{a_{i_1,i_2...i_d}\}$ has the Monge property if for each pair of entries $a_{i_1,i_2...i_d}$ and $a_{j_1,j_2...j_d}$ we have,

$$a_{s_1,s_2...s_d} + a_{t_1,t_2...t_d} \leq a_{i_1,i_2...i_d} + a_{j_1,j_2...j_d} \tag{6.3}$$

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

The following are several useful properties of Monge sequences.

1. Definition 6.1.2 is equivalent to the statement, a matrix $\mathbf{A}$ is a Monge array if

$$a_{i,j} + a_{i+1,j+1} \leq a_{i,j+1} + a_{i+1,j} \text{ for all } 1 \leq i < m \text{ and } 1 \leq j < n.$$

2. If a sub array is selected from a certain row and a certain column from a Monge array, then the selected subset will be a Monge array.

Definition 6.1.6 Supnick Matrix

A symmetric $n \times n$ matrix $\mathbf{A}$ is called Supnick if

$$a_{i,j} + a_{k,l} \leq a_{i,k} + a_{j,l} \leq a_{i,l} + a_{j,k}, \tag{6.4}$$

for all $1 \leq i < j < k < l \leq n$.

6.2 Related Research Works

Burkard et al., [77] studied in their perspectives of Monge properties in optimisation, they discussed both Monge matrices and arrays and their applications. They had put many observations and fundamental properties of Monge arrays. Although many hard problems had been solved in polynomial time when applying Monge properties with some restrictions but still some problems need more attention to solve them. Burkard et al. mentioned that they were not aware that; the problem of given a cost matrix $\mathbf{C}$ which is not Monge and an instance I of a hard optimisation problem $\mathbf{P}$, find a Monge matrix $\mathbf{C}'$ is as close as possible to the optimal solution of the modified instance $\mathbf{I}'$.

Derigs et al., [78] presented a graphical bipartite two dimension problem using Monge properties and observations, the idea of the method was to build an equivalent problem to an arbitrary assignment and apply the Monge conditions.

Arora and Puri, [79] proposed a lexi-search approach to find an optimal solution for the assignment problem. Optimum allocation time to assign m persons to n jobs where $m < n$. This condition will allow one person to do more than one job and to start immediately after finishing the first job in any order. The method and the results can lead to the relation of the three dimensional assignment problem. Malhotra, [80].

Rudolf, [81] answered the question of possibility of finding d permutations in a such a way that the permuted array becomes a Monge array. An algorithm had been done to construct such permutation. The permuted method was helpful to solve the d- dimensional axial assignment problem efficiently.

Custic et al., [82] studied the planar 3-dimensional assignment problems with Monge-like cost arrays. He proved that p layer-planar 3-dimensional assignment problems (p-3PAP) is **NP**-hard for every fixed $p \geq 2$. They had constructed a dynamic programming algorithm for the P planar layer assignment problem (P-PLAP) on layered Monge arrays.

6.3 Monge Minimisation Algorithm

The benefit of using Monge array is the nature of the structure of the array. The Monge structures allow us to use certain entries without the need of using the whole array and this by itself will reduce the number of iterations. For example if we want to search for the smallest number in $m \times n$ entries Monge array then we need only $O(m+n)$ operations, [83]. There are many useful algorithms which can benefit from working on Monge arrays.

Example 6.3.1 Consider a convex polygon, remove two sides of this polygon to get two chain of points P with m vertices and Q with n vertices. Let the vertices $p_1, ..., p_m$ denote the vertices of P in clockwise order and let $q_1, ..., q_n$ denote the vertices of Q in counter-clockwise order. The aim is to find the vertex p_i of P and a vertex q_j of Q to minimise the Euclidean distance $d(p_i, q_j)$ separating p_i and q_j. This problem can be solved using Monge array as follows.

Consider $\mathbf{A} = \{a_{ij}\}$ to be an $m \times n$ array where $a_{ij} = d(p_i, q_j)$. This array is Monge because

if we consider any two rows i_1 and i_2 and two columns j_1 and j_2 such that $1 < i_1 < i_2 < m$ and $1 < j_1 < j_2 < n$. The entries $a\{i_1, j_1\}$ and $a\{i_2, j_2\}$ correspond to the opposite sides of the quadrilateral formed by $p_{i_1}, p_{i_2}, q_{j_1}$ and q_{j_2} and the entries $a\{i_1, j_2\}$ and $a\{i_1, j_2\}$ correspond to the diagonal. By the quadrangle inequality (the sum of the lengths of the diagonal of any quadrilateral is greater than the sum of the lengths of any pair of opposite sides), we have

$$d(p_{i_1}, p_{j_1}) + d(p_{i_2}, q_{j_2}) \leq d(p_{i_1}, q_{j_2}) + d(p_{i_2}, q_{j_1}).$$

Thus, **A is Monge, and we have reduced to the problem of finding the smallest entry in a Monge array.**

Gerhard J Woeginger, [84] and Burkard, [77], applied Monge array to solve some problems related to the travelling salesman problem. They use the definition of Supnick matrix with relation to Monge and use the properties of Monge array.

Burkard et al., [77] defined a Supnick matrix **A**= $\{a_{ij}\}$ as a symmetric Monge matrix which satisfies the property,

$$a_{i,j} + a_{r,s} \leq a_{i,s} + a_{r,s}, \tag{6.5}$$

for all i, j, r and s with $1 \leq i < r \leq n$ and $1 \leq j < s \leq n$. Hence a Supnick matrix satisfies the inequality Equation 6.5 and $a_{i,j} = a_{j,i}$ for all i and j. Also the sum matrix $\mathbf{A_{ij}} = a_i + a_j$ is a Supnick matrix for all real numbers $a_1, a_2, ..., a_n$, as the sum matrix $\mathbf{A_{ij}}$ satisfies the inequality 6.4.

Supnick, [73] proved that with distance matrices the optimum travelling salesman problem (*TSP*) tour is easy to find when the distance matrix is Supnick, the following

theorem gives the shortest *TSP* and the longest *TSP* tours.

Theorem 6.3.1 Let $\mathbf{A} = \{a_{ij}\}$ be an $n \times n$ Supnick matrix. The shortest and the longest *TSP* tour are given by the following permutation respectively,

$$\sigma^{min} = \langle 1, 3, 5, 7, 9, 13, ..., 14, 12, 10, 8, 6, 4, 2 \rangle. \tag{6.6}$$

$$\sigma^{max} = \langle n, 2, n-2, 4, n-4, 6, ..., 5, n-3, 3, n-1, 1 \rangle. \tag{6.7}$$

The short permutation σ^{min} is to start visiting the odd cities in increasing order and then the even in decreasing order. This distribution is an optimum solution for all the instances of Supnick *TSP*.

Definition 6.3.1 An $n \times n$ matrix $\mathbf{A} = \{a_{ij}\}$ is a relaxed Supnick matrix, if

$$a_{ij} + a_{i+1,j+1} \leq a_{i,j+1} + a_{i+1,j}, \text{ for all } 1 \leq i < j - 1 \leq n - 2. \tag{6.8}$$

Deineko et al., [85], had shown that the *TSP* on a relaxed Supnick matrix, which can also be viewed as a one-sided Monge matrix, is **NP**-hard.

6.4 Monge Array

In this section we have listed some basic graph theory definitions, to explain the graph theory related to Monge sequences. The following definitions are borrowed from the book, Introduction to graph theory , of Wilson, [86].

A simple graph $\mathbf{G} = (V, E)$ consists of a non-empty set representing vertices, V, and a

set of unordered pairs of elements of V representing edges, E.

A simple graph has no arrows, no loops, and cannot have multiple edges joining vertices.

Two vertices V_1, V_2 of a graph are said to be adjacent if $\{V_1, V_2\}$ is an edge of the graph. The diameter *dim* of a graph **G** is the maximal distance d between any two points on the graph. If the graph is not connected, its diameter is infinite.

Definition 6.4.1 A graph is bipartite if its set of vertices can be split into two parts V_1, V_2, such that every edge of the graph connects a V_1 vertex to a V_2 vertex. More generally, a graph is bipartite if and only if all cycles in the graph have even length.

Definition 6.4.2 A graph in which every pair of vertices is adjacent. Such a graph is sometimes called K_n, where n is the number of vertices. For example, a triangle is a complete graph namely K_3, but no other polygon is.

A directed graph in which each graph edge is replaced by a directed graph edge, also called a digraph. A directed graph having no multiple edges or loops.

An acyclic digraph is a directed graph containing no directed cycles, also known as a directed acyclic graph or a **DAG**. Every finite acyclic digraph has at least one node of out degree 0.

Definition 6.4.3 A Hamiltonian path in a graph **G** is a path that goes through each vertex of G once. If the initial and final vertices are adjacent then the path can be completed to a Hamiltonian circuit. The paths and circuits can be hard to find, or even to tell whether they exist on a given graph **G**.

Aggarwal et al., [87], suggested an efficient algorithm for finding a minimum weight K-link path in graph with Monge property and applications, they called the Graph **G** =

(V, E) a K-linked path if the path contain exactly K arcs, where G be a weighted, complete, directed acyclic graph (**DAG**), with the vertex set $V = \{v_1, v_2, ..., v_n\}$, and denoted $w(i, j)$ be the weight for $1 \le i \le j$ associated with the arc (i, j). A weighted **DAG**, **G**, satisfies the concave Monge condition if $w_{i,j} + w_{i+1,j+1} \le w_{i,j+1} + w_{i+1,j}$ holds for all $1 < i + 1 < j < n$, and satisfies the convex Monge condition if the inequality is reversed.

The application of geometric path related to Monge sequences graphs using the minimum K-link path method has several applications which were discussed by Aggarwal et al. and the complexity of each application also had been explained. Various type of applications will be discussed in the following sections of this chapter.

6.5 Monge Sequence Special Case for SAP

In this section we have studied a special case of SAP with Monge sequence. We solve it for the three dimensional assignment problems (SAP) after relaxing the constraints of the problem to satisfy Monge sequence requirements. We have found an exact solution for n dimensional assignment problems. The cost matrix **A**= $\{a_{ij}\}$ for (i and $j = 1, 2, .., n,$) and for all k layers where $k = 1, 2, ..., n$, was restricted to be totally monotone Monge sequence with a constant cost differences $\delta_{ij} = (a_{i,j+1} - a_{i,j})$ for all layers $k = 1, 2, ..., n$ and $1 \le i \le n$ and $1 \le j < n$.

We have constructed two formulas. The first one is to calculate the exact solution for the three dimensional assignment problems SAP with the relaxation of the cost differences δ_{ij}, $1 \le i \le n$ and $1 \le j < n$ for any finite number n of dimensions. The second formula is to generate a totally monotone Monge sequence cost matrix while fixing the total cost. The

MATLAB is used to program the two algorithms. Some useful observations discussed relate to the Hungarian method.

To simplify the problem, consider SAP with size n . It is also proposed that the total cost for the first factory is less than the total cost of the second factory and so on for all the n factories. Same amount of wages are given for doing the required n jobs when they finished all the tasks. The mathematical formulation of this problem is similar to the d-dimensional Monge sequences. The mathematical formulation was explained by Burkard et al., [77] and Rudolf, [88].

In this section we have used the same mathematical notation and the formulations used by Burkard.

Let $\mathbf{C}$ be an $n_1 \times n_2 \times ... \times n_d$ array, for $d \geq 3$ and define $N_k = \{1, 2, ..., n_k\}$, where $k = 1, 2, ..., d$. S is called a d-dimensional sequence subject to the array C if the ordered elements are formulated in the Cartesian product $N = N_1 \times ... \times N_d$. Furthermore, let $I = \{(i_1^k, i_2^k, ..., i_d^k)$ such that $i_l^k \in N_l$, where $1 \leq l \leq d, 1 \leq k \leq q\}$ be a set of N of cardinality q and define $L_k(I) = \{i_l^1, ..., i_l^q\}$ as the list of all integers which occurs as l-th coordinate of an element in I.

A set $I \subseteq N$ is said to be feasible with respect to d-tuples $(i_1, i_2, ..., i_d) \in N$ if and only if for all $k = 1, 2, ..., q$ the entry i_k is contained at least once in the list $L_k(I)$. If I is feasible but no proper subset of I is feasible, then I is said to be minimal. Finally, Let $M(I) = \{(i_1, i_2, ..., i_d)$ such that $i_k \in L_k(I)\}$, i.e. $M(I)$ contains all d-tuples $(i_1, i_2, ..., i_d) \in N$ which can be formed by choosing i_k from the list $L_k(I)$ for all $k = 1, 2, ..., d$.

Comparing the previous formulation with the following d-dimensional Monge sequence, a sequence C is called a d- dimensional Monge sequence with respect to the array C if the

following condition is satisfied, Rudolf, [88].

Let $(i_1^I, i_2^I, ..., i_d^I) \in S$. Then for each set $I = \{(i_1^k, i_2^k, ..., i_d^k)$ such that $i_l^k \in N_l, 1 \leq l \leq d, 1 \leq k \leq q\}$ which is minimal with respect to $(i_1^I, i_2^I, ..., i_d^I)$ we need to have that whenever $(i_1^I, i_2^I, ..., i_d^I)$ is the element which occurs first in S among all element of $M(I)$ then,

$$c[i_1^I, i_2^I, ..., i_d^I] + \min_{\phi_2,...,\phi_d} \{\sum_{k=2}^{q} c[i_1^k, i_2^{\phi_2(k)}, ..., i_d^{\phi_d(k)}]\} \leq \sum_{(j_1,j_2,...,j_d)\in I} c[j_1, j_2, ..., j_d] \quad (6.9)$$

has to hold, where $\phi_2, ..., \phi_d$ are bijections acting on the set $\{2, ..., q\}$.

The following theorem is the generalization of Hoffman classical result on the Hitchcock transportation problem.

Theorem 6.5.1 The lexicographical greedy algorithm determines an optimal solution of the axial d-dimensional transportation problem (dTP) for all feasible right- hand side vectors $a^k, k = 1, 2, ..., d$ if and only if the underlying cost array C is a Monge array.

The axial d-dimensional assignment problem (dAP) is a special case of the (dTP) where in (dAP) $a_i^k = 1$ for all $k = 1, 2, ..., n_k$, in addition to that all the variables $x_{i_{1,2,...,d}}$ are integral. In general (dAP) is an **NP**-hard problem but if the cost array C is a Monge array, then the solution is optimal according to Theorem 6.5.1.

6.6 Special Case of the d-Dimensional Assignment Problem

Given n disjoint n-sets array $\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_n}$, each matrix $\mathbf{A_i}$ with totally monotone Monge sequence has $n \times n$ elements. Let a weight function $\omega : \mathbf{A_1} \times \mathbf{A_2} \times ... \times \mathbf{A_n} \rightarrow \mathbf{R}^+$, asks for

a collection of n-sets $\mathbf{M} \subseteq \mathbf{A_1} \times \mathbf{A_2} \times ... \times \mathbf{A_n}$ such that each element of each set appears in exactly one n-set, and the function ω is minimized. The following Theorem 6.6.1 gives a useful formula for calculating the optimum solution.

Theorem 6.6.1 The optimal cost $\mathbf{C}$ for a totally monotone Monge sequence $\mathbf{A} = \{\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_n}\}$ d-dimensional assignment problem is,

$$\mathbf{C} = \mathbf{d} \times \mathbf{a_{11}} + \delta \times (\mathbf{d^3} + \mathbf{d^2} - \mathbf{2d})/\mathbf{2}, \tag{6.10}$$

where, a_{11} is an initial cost $\in \mathbf{A_1}$, d is the dimensional number of the problem, and δ is the differences between two costs such that $\delta = (a_{i,j+1} - a_{ij})$, for all $1 \leq i \leq n$ and $1 \leq j < n$.

Proof of Theorem 6.6.1

Let $\mathbf{C_1} = \mathbf{a_{11}}$ be the initial cost of the $\mathbf{A_l}$ matrix, we can write $\mathbf{C_j}$ as follows,

$$\mathbf{C_j} = \mathbf{C_{j-1}} + \delta \times (\mathbf{j} - \mathbf{1})(\mathbf{d} + \mathbf{2}), \text{ for all } \mathbf{j} \geq \mathbf{2}. \tag{6.11}$$

$\mathbf{C_1} = \mathbf{a_{11}}, \mathbf{C_2} = \mathbf{C_1} + \delta \times (\mathbf{2} - \mathbf{1})(\mathbf{d} + \mathbf{2}), ..., \mathbf{C_d} = \mathbf{C_{d-1}} + \delta \times (\mathbf{d} - \mathbf{1})(\mathbf{d} + \mathbf{2})$,

adding $\mathbf{C_1}, \mathbf{C_2}, ..., \mathbf{C_d}$ we have

$$C = \sum_{l=1}^{d} (a_{11} + \delta \times (l - 1)(d + 2)). \tag{6.12}$$

Simplifying the above equation we have,

$C = (a_{11} + \delta \times (1 - 1)(d + 2)) + (a_{11} + \delta \times (2 - 1)(d + 2)) + \cdots + (a_{11} + \delta \times (d - 1)(d + 2))$.

$C = d \times a_{11} + \delta \times (d + 2) \times \{1 + 2 + \cdots + (d - 1)\}$.

$C = d \times a_{11} + \delta \times (d+2) \times \{(1 + (d-1)) \times (d-1)/2\}.$

$C = d \times a_{11} + \delta \times d \times (d+2) \times (d-1)/2.$

$C = d \times a_{11} + \delta \times (d^3 + d^2 - 2d)/2.$

The following example shows how to find the total optimum cost for 3-set of totally monotone matrices each of them has 3×3 dimension.

Example 6.6.1 Let $\mathbf{A_1}, \mathbf{A_2}$ and $\mathbf{A_3}$ be three totally monotone Monge sequences with 3×3 dimension. Applying Theorem 6.6.1 to find the optimum SAP allocation.

$$\mathbf{A_1} = \begin{pmatrix} 02 & 04 & 06 \\ 08 & 10 & 12 \\ 14 & 16 & 18 \end{pmatrix}, \ \mathbf{A_2} = \begin{pmatrix} 04 & 06 & 08 \\ 10 & 12 & 14 \\ 16 & 18 & 20 \end{pmatrix}, \ \mathbf{A_3} = \begin{pmatrix} 06 & 08 & 10 \\ 12 & 14 & 16 \\ 18 & 20 & 22 \end{pmatrix}.$$

It is clear that $a_{11} = 2$, $d = 3$ and $\delta = 2$, inserting these data in the Equation 6.10 we have the optimum solution $C = 36$. It is also obvious that if we select any different combination of three elements in the matrices, we will have the same result.

6.7 Observations:

1. Applying the Hungarian method to Example 6.6.1, the minimum assignment allocation cost for $\mathbf{A_1} = \mathbf{30}$ and all other combinations will give us the same minimum allocation cost 30.

2. The minimum cost of $\mathbf{A_1}$ is equal to the maximum cost of $\mathbf{A_1}$.

3. Both previous observations are applied to both $\mathbf{A_2}$ and $\mathbf{A_3}$.

4. The total cost of $\mathbf{A_1} < \mathbf{A_2} < \mathbf{A_3}$, where $\mathbf{A_1} = \mathbf{30}$, $\mathbf{A_2} = \mathbf{36}$ and $\mathbf{A_3} = \mathbf{42}$.

5. It is clear that the differences between the optimum cost of the three matrices $\mathbf{A_1}, \mathbf{A_2}$ and $\mathbf{A_3}$ is constant and for this example is equal to 6.

From observation (5) we can conclude that the differences between the optimum cost of the three matrices $\mathbf{A_1}, \mathbf{A_2}$ and $\mathbf{A_3}$ can be written as follows;

Lemma 1 Let $\mathbf{A} = \{a_{i,j}\}$ where (i and $j = 1, 2, ..., n$) be totally monotone Monge sequence. Let z_1, z_2 and z_3 be the optimum allocation costs for $\mathbf{A_1}, \mathbf{A_2}$ and $\mathbf{A_3} \in \mathbf{A}$ respectively then,

$$z_3 - z_2 = z_2 - z_1 = \delta \times d, \tag{6.13}$$

where d is the dimension number of the array and $\delta = a_{i,j+1} - a_{ij}$ for all $1 \leq i \leq n$ and $1 \leq j < n$.

The next Theorem 6.7.1 is the generalization of Theorem 6.6.1 to be used for selecting an arbitrary $\mathbf{A_l}$. It is not necessary to be in sequenced order, while the optimal cost **C** remain not changed and it is a totally monotone Monge sequence d-dimensional assignment problem.

Theorem 6.7.1 The optimum cost for a totally monotone Monge sequence d-dimensional assignment problem is,

$$C = d \times a_{11} + \delta \sum_{l=1}^{d-1} l m_{l+1}, \tag{6.14}$$

where a_{11} is an initial cost $\in \mathbf{A_1}$, d is the dimensional number of the problem, δ is the differences between two costs such that $\delta = (a_{i,j+1} - a_{ij})$ for all $1 \leq i \leq n$ and $1 \leq j < n$ and

$m_l = (a_{ll} - a_{11})/(\delta(l-1))$ where $2 \le l \le d$.

Proof of Theorem 6.7.1

let $C_1 = a_{11}$ be the initial cost of the A_l matrix, we can write C_l as follows,

$$C_j = a_{11} + (j-1) \times \delta \times m_l, \tag{6.15}$$

where $m_l = (a_{ll} - a_{11})/(\delta \times (l-1))$, for $l \ge 2$. Applying Equation 6.15, for $j = 1, 2, ..., d$, we have,

$$C_1 = a_{11}, C_2 = a_{11} + \delta \times m_2, C_3 = a_{11} + 2 \times \delta \times m_3, ..., C_d = a_{11} + (d-1) \times \delta \times m_d. \tag{6.16}$$

By adding all the terms of Equation 6.16, we will have;

$C = d \times a_{11} + \delta \sum_{l=1}^{d-1} l m_{l+1}$.

The following Example 6.7.1 shows how to find the total optimum cost for three set of totally monotone matrices each of them has 3×3 dimension but they are not sequences in order.

Example 6.7.1 Let $\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_6}$ be six different arrays, each of them is a totally monotone Monge sequence array. We will select any three arrays and try to apply Theorem 6.7.1

$$\mathbf{A_1} = \begin{pmatrix} 02 & 04 & 06 \\ 08 & 10 & 12 \\ 14 & 16 & 18 \end{pmatrix}, \mathbf{A_2} = \begin{pmatrix} 14 & 16 & 18 \\ 20 & 22 & 24 \\ 26 & 28 & 30 \end{pmatrix}, \mathbf{A_3} = \begin{pmatrix} 40 & 42 & 44 \\ 46 & 48 & 50 \\ 52 & 54 & 56 \end{pmatrix},$$

$$\mathbf{A_4} = \begin{pmatrix} 48 & 50 & 52 \\ 54 & 56 & 58 \\ 60 & 62 & 64 \end{pmatrix}, \quad \mathbf{A_5} = \begin{pmatrix} 56 & 58 & 60 \\ 62 & 64 & 66 \\ 68 & 70 & 72 \end{pmatrix}, \quad \mathbf{A_6} = \begin{pmatrix} 66 & 68 & 70 \\ 72 & 74 & 76 \\ 78 & 80 & 82 \end{pmatrix}.$$

Let the matrix **A** be the concatenated of the matrices $\mathbf{A_1}$, $\mathbf{A_3}$ and $\mathbf{A_6}$, we will calculate SAP allocation to **A**. From the initial array $\mathbf{A_1}$ we have the cost $a_{11} = 2$, $\delta = 2$ and the dimension of the array (number of matrices) is $d = 3$. We will calculate m_2 for the array $\mathbf{A_3}$ and m_3 for the array $\mathbf{A_6}$ using the formula in equation 6.15, $m_l = (a_{ll} - a_{11}/(\delta \times (l-1))$ as follows, $m_2 = (48-2)/(2 \times (2-1)) = 23$ and $m_3 = (82-2)/(2 \times (3-1)) = 20$.

Using Equation 6.16 of Theorem 6.7.1 and applying the above data, we have the optimum costs $C = (3 \times 2) + 2 \times (23 + 2 \times 20) = 132$.

6.8 Summary

The ordinary or 2-dimensional assignment is in the **P**-class. SAP, which is an extension of it, is **NP**-hard. It is commonly solved using exact methods of integer programming such as Branch-and-Bound. However, in general, it is intractable and only approximate solutions are found in reasonable time. Here, we find an exact solution to the problems with the condition of Monge sequence property applied to the pay off cost matrix.

We have established two theorems and proved them. Several useful observations related to theorems and a lemma were discussed in this chapter. The relaxation of constraints of SAP has been discussed and explained with examples especially when we relaxed the objective function and make the cost of SAP to be fixed for all the workers doing jobs in the factories. We have shown that applying Monge properties with relaxation of the

constraints; SAP can be solved for large size problem in linear polynomial time.

We explained the problem by using examples and algorithms. Some useful ideas and observations also have been discussed.

Chapter 7

Conclusion

7.1 Conclusion

This thesis proposes combinatorial new heuristic approaches to solve SAP. The algorithms have been presented, illustrated and tested on some small size problems. Hybridisation of the heuristic approach with B&B has been implemented. The mathematical background and the literature relevant to the problem were considered in Chapter 2. We explained the 2-dimensional assignment problem and the well-known Hungarian method. We have given the characteristics of the assignment problem and explained all the points.

We focused especially, on the first type of the 3-dimensional solid assignment problem which is an extension of the 2-dimensional assignment problem. The second type is called the planar 3-dimensional problem. The mathematical formulation and the differences among all the assignment models were explained in this thesis. The classical B&B is a common method to find the exact solution for limited size problems SAP up to $n = 26$.

We have explained several methods in Chapter 2 such as the (0,1) integer B&B tech-

nique, primal-dual implicit enumeration and some special approaches. We have used MATLAB source code to implement the methods and solve instances of the problems considered in this thesis.

In Chapter 3, we presented DM which it solved SAP. The main issue is to understand why the method works at all, how reliable it is, how accurate and efficient it is compared to other approaches. The algorithm of the DM starts by re-arranging the n factories in an ascending or descending order, then the allocation of the factories is carried out individually one by one. The importance of this procedure is the reduction of the size of the problem by one in each iteration during the execution of the program. This reduction of the size makes it running fast and completing the execution of the source code in a short time.

The large size of solving SAP is another benefit of the DM. As we have explain that SAP was solved for size $n = 26$, we have managed to solve SAP for size up $n = 500$ in 516.558064 seconds as it was shown in Chapter 3 Section 3.5, the fast solution and the large size problem are useful achievement.

Three more heuristic approaches have been considered in Chapter 4. The first is called the Average Cost Method (ACM). The ACM technique is simple but not as efficient as the DM especially when the size of the problem is large comparing to a small size problem, costly and large execution time.

The second and third approaches considered in Chapter 4 are called the Addition Method and the Multiplication Method. The basic idea in both methods is to convert SAP from three dimensions to two dimensions. Then apply the Hungarian assignment problem twice to reach the allocation. To achieve this idea in the AM problem, we add

all the allocation costs in each factory for each worker in order to achieve a square matrix with $n \times n$ dimension. Then by applying the Hungarian assignment method for the first time it will guarantee the allocation of the machines in the factories while applying the Hungarian assignment method for the second time it will guarantee the allocation of the workers with their nominating cost to do the jobs on the machines in the factories.

The same process is applied in the Multiplication Method. The only difference is we multiply all the allocation costs in each factory for each worker instead of adding them.

The advantages of AM are that it is fast and can solve large size problems up to $n = 800$. The algorithm can converge to the minimum cost easily sometimes. The disadvantage is, it can be effected by the nature of the random numbers or costs used (zeros, small or large number). Also when we used the addition operation in each row vector n times this might lead to a tie and make the selection ineffective.

The advantages of the MM are that it is fast and it can solve large size problems up to $n = 800$. If the cost matrix have many zeros this will accelerate the procedure and the results are accurate. The disadvantage is that if the cost numbers are large and do not have zero costs it will only work on limited size problems.

In Chapter 5 we have solved the SAP using GA method. Two methods have been used, the first depends on generating randomly permuted function twice. The second GA method is also generates a random permuted function only once. Both methods find good results for the small size problems.

In Chapter 6 we have dealt with a special case SAP having of Monge matrix. We have used the Monge properties and relaxed the constraints of SAP. We considered an algorithm to solve SAP for a large size problem and managed to find an exact minimum

allocation.

We have exploited Monge properties and some useful conditions. The algorithm is not complicated and is fast. We established two theorems related to Monge sequence and proved them. We explained the problems and illustrated them. Some useful ideas and observations have also been discussed. This special case of SAP is linear and can be solved in polynomial time.

For the future work there are a lot of potentials ideas to extend the research work of this theses. For example, applying the Genetic Algorithm for all the approaches we discussed. The SAP can be investigated and solved in conduction with the methods outlined in the thesis to reach even better results. Using the GA with different approaches of the crossover, roulette wheel, tournament election, mutation , single or multi points crossover chromosomes.

As we have seen that all the methods constructed in this thesis can solve SAP with high dimensions up to size $n = 800$ (memory constraint) within seconds and a feasible solution is guaranteed in all the methods. While the Branch-and-Bound can only solve a limited size problem with longer execution time.

It is important to investigate and research how to minimise the impact of the Tie cases on reaching an optimum cost solution. We have discussed this problem and explained the effects of the Tie on the objective solution and how the solution divert far from the objective solution because of the Tie and the unwise selection of the cost when the Tie happened.

In this thesis we have discussed the first type of the 3-dimensional Solid Assignment Problem (SAP) in details. It is important if to apply all the constructed methods to solve

SAP type two; the Planar Solid Assignment Problem. We have discussed the mathematical formulation of this method.

It is observed that if the Tie case occurs, then a major impact on the optimal solution might occur. More thorough research works are required to study this phenomena in such a way to avoid or prevent the solution from diversion. Avoid the Tie will help to obtain an optimal or near optimal solution.

Bibliography

[1] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al., "Optimization by simulated annealing," science, vol. 220, no. 4598, pp. 671–680, 1983.

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

[3] H. Izakian, B. T. Ladani, A. Abraham, V. Snasel, et al., "A discrete particle swarm optimization approach for grid job scheduling," International Journal of Innovative Computing, Information and Control, vol. 6, no. 9, pp. 1–15, 2010.

[4] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on, vol. 5, pp. 4104–4108, IEEE, 1997.

[5] X. Yang, "Introduction to mathematical optimization," From Linear Programming to Metaheuristics, 2008.

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

[7] G. C. Goodwin, M. M. Seron, and J. A. De Doná, Constrained control and estimation: an optimisation approach. Springer Science & Business Media, 2006.

[8] R. E. Burkard and E. Cela, "Linear assignment problems and extensions," in Handbook of combinatorial optimization, pp. 75–149, Springer, 1999.

[9] H. A. Taha, Operations Research: An Introduction (For VTU). Pearson Education India, 1982.

[10] E. L. Lawler, "A note on the complexity of the chromatic number problem," Information Processing Letters, vol. 5, no. 3, pp. 66–67, 1976.

[11] W. L. Winston and J. B. Goldberg, Operations research: applications and algorithms, vol. 3. Thomson Brooks/Cole Belmont, 2004.

[12] M. R. Garey and D. S. Johnson, "A guide to the theory of np-completeness," WH Freemann, New York, vol. 70, 1979.

[13] S. Arora and B. Barak, Computational complexity: a modern approach. Cambridge University Press, 2009.

[14] S. Mertens, "Computational complexity for physicists," Computing in Science & Engineering, vol. 4, no. 3, pp. 31–47, 2002.

[15] O. Goldreich, "Introduction to complexity theory," Lecture Notes for a Two-Semester course [1999], http://www. wisdom. weizmann. ac. il/mathusers/oded/cc99. html, 1997.

[16] G. J. Woeginger, "Exact algorithms for np-hard problems: A survey," in Combinatorial OptimizationEureka, You Shrink!, pp. 185–207, Springer, 2003.

[17] R. M. Karp, Reducibility among combinatorial problems. Springer, 1972.

[18] A. Ćustić, B. Klinz, and G. J. Woeginger, "Geometric versions of the 3-dimensional assignment problem under general norms," arXiv preprint arXiv:1409.0845, 2014.

[19] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," Operations research, vol. 14, no. 4, pp. 699–719, 1966.

[20] R. Bellman, "Dynamic programming and lagrange multipliers," Proceedings of the National Academy of Sciences, vol. 42, no. 10, pp. 767–769, 1956.

[21] B. Korte and J. Vygen, "Network flows," in Combinatorial Optimization, pp. 153–184, Springer, 2002.

[22] T. H. Cormen, Introduction to algorithms. MIT press, 2009.

[23] R. S. Garfinkel, G. L. Nemhauser, et al., Integer programming, vol. 4. Wiley New York, 1972.

[24] J. E. Mitchell, "Branch-and-cut algorithms for combinatorial optimization problems," Handbook of applied optimization, pp. 65–77, 2002.

[25] A. Chung, R. Johnson, and T. Donnelly, "Calculation of the (γ, n) and (γ, p) cross sections for 4he in a continuum shell-model approximation," Nuclear Physics A, vol. 235, no. 1, pp. 1–10, 1974.

[26] E.-G. Talbi, Metaheuristics: from design to implementation, vol. 74. John Wiley & Sons, 2009.

[27] D. P. Williamson and D. B. Shmoys, The design of approximation algorithms. Cambridge university press, 2011.

[28] X.-S. Yang, Nature-inspired metaheuristic algorithms. Luniver press, 2010.

[29] R. Martí and G. Reinelt, The linear ordering problem: exact and heuristic methods in combinatorial optimization, vol. 175. Springer Science & Business Media, 2011.

[30] K. Sörensen and F. W. Glover, "Metaheuristics," in Encyclopedia of operations research and management science, pp. 960–970, Springer, 2013.

[31] S. Arora, "Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems," Journal of the ACM (JACM), vol. 45, no. 5, pp. 753–782, 1998.

[32] M. Yagiura and T. Ibaraki, "On metaheuristic algorithms for combinatorial optimization problems," Systems and Computers in Japan, vol. 32, no. 3, pp. 33–55, 2001.

[33] A. Salhi and J. A. V. Rodríguez, "Tailoring hyper-heuristics to specific instances of a scheduling problem using affinity and competence functions," Memetic Computing, vol. 6, no. 2, pp. 77–84, 2014.

[34] M. Mitchell, An introduction to genetic algorithms. MIT press, 1998.

[35] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on, pp. 39–43, IEEE, 1995.

[36] R. Bosch and M. Trick, "Integer programming," in Search methodologies, pp. 67–92, Springer, 2014.

[37] F. Commoner, "A sufficient condition for a matrix to be totally unimodular," Networks, vol. 3, no. 4, pp. 351–365, 1973.

[38] M. A. Trick, "A tutorial on integer programming," The Operations Research Faculty of GSIA, 1997.

[39] A. W. Tucker, "Dual systems of homogeneous linear relations," Linear inequalities and related systems, vol. 38, pp. 3–18, 1956.

[40] R. E. Burkard, M. Dell'Amico, and S. Martello, Assignment Problems, Revised Reprint. Siam, 2009.

[41] H. W. Kuhn, "The hungarian method for the assignment problem," Naval research logistics quarterly, vol. 2, no. 1-2, pp. 83–97, 1955.

[42] J. Munkres, "Algorithms for the assignment and transportation problems," Journal of the Society for Industrial and Applied Mathematics, vol. 5, no. 1, pp. 32–38, 1957.

[43] M. L. Fisher and R. Jaikumar, "A generalized assignment heuristic for vehicle routing," Networks, vol. 11, no. 2, pp. 109–124, 1981.

[44] R. Nauss, "The elastic generalized assignment problem," Journal of the Operational Research Society, vol. 55, no. 12, pp. 1333–1341, 2004.

[45] S. Martello and P. Toth, "The bottleneck generalized assignment problem," European Journal of Operational Research, vol. 83, no. 3, pp. 621–638, 1995.

[46] S. O. Krumke and C. Thielen, "The generalized assignment problem with minimum quantities," European Journal of Operational Research, vol. 228, no. 1, pp. 46–55, 2013.

[47] F. Rajabi-Alni, "Two exact algorithms for the generalized assignment problem," arXiv preprint arXiv:1303.4031, 2013.

[48] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of np-completeness. 1979," San Francisco, LA: Freeman, 1979.

[49] W. P. Pierskalla, "Letter to the editorthe multidimensional assignment problem," Operations Research, vol. 16, no. 2, pp. 422–431, 1968.

[50] E. Balas and M. J. Saltzman, "An algorithm for the three-index assignment problem," Operations Research, vol. 39, no. 1, pp. 150–161, 1991.

[51] Y. Crama and F. C. Spieksma, "Approximation algorithms for three-dimensional assignment problems with triangle inequalities," European Journal of Operational Research, vol. 60, no. 3, pp. 273–279, 1992.

[52] G. Gwan and L. Qi, "On facets of the three-index assignment polytope," Australasian J. Combinatorics, vol. 6, pp. 67–87, 1992.

[53] E. Balas and M. J. Saltzman, "Facets of the three-index assignment polytope," Discrete Applied Mathematics, vol. 23, no. 3, pp. 201–229, 1989.

[54] L. Qi and E. Balas, A new class of facet-defining inequalities for the three-index assignment polytope. Carnegie-Mellon University. Graduate School of Industrial Administration, 1990.

[55] R. Burkard, R. Rudolf, and G. Woeginger, "Three-dimensional axial assignment problems with decomposable cost coefficients," Discrete Applied Mathematics, vol. 65, no. 1-3, p. 123, 1996.

[56] G. Hardy, "Je littlewood et g. polya-inequalities," 1952.

[57] Y. Cao, "Hungarian algorithm for linear assignment problems (v2. 1), 2008," 2011.

[58] P. M. Hahn, Y.-R. Zhu, M. Guignard, and J. M. Smith, "Exact solution of emerging quadratic assignment problems," International Transactions in Operational Research, vol. 17, no. 5, pp. 525–552, 2010.

[59] P. Pandian and K. Kavitha, "Sensitivity analysis in solid transportation problems," Applied Mathematical Sciences, vol. 6, no. 136, pp. 6787–6796, 2012.

[60] T. E. Easterfield, "A combinatorial algorithm," Journal of the London Mathematical Society, vol. 1, no. 3, p. 219, 1946.

[61] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," Journal of the ACM (JACM), vol. 19, no. 2, pp. 248–264, 1972.

[62] E. Dinic and M. Kronrod, "An algorithm for the solution of the assignment problem," in Soviet Math. Dokl, vol. 10, pp. 1324–1326, 1969.

[63] F. C. Spieksma, "Multi index assignment problems: complexity, approximation, applications," in Nonlinear Assignment Problems, pp. 1–12, Springer, 2000.

[64] W. P. Pierskalla, "The tri-substitution method for the three-dimensional assignment problem," CORS Journal, vol. 5, pp. 71–81, 1967.

[65] F. C. Spieksma and G. J. Woeginger, "Geometric three-dimensional assignment problems," European Journal of Operational Research, vol. 91, no. 3, pp. 611–618, 1996.

[66] A. Frieze, "Complexity of a 3-dimensional assignment problem," European Journal of Operational Research, vol. 13, no. 2, pp. 161–164, 1983.

[67] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems," in 50 Years of Integer Programming 1958-2008, pp. 105–132, Springer, 2010.

[68] D. L. Poole and A. K. Mackworth, Artificial Intelligence: foundations of computational agents. Cambridge University Press, 2010.

[69] E. Balas, "An additive algorithm for solving linear programs with zero-one variables," Operations Research, vol. 13, no. 4, pp. 517–546, 1965.

[70] D. Anuradha and P. Pandian, "A new method for finding an optimal solution to solid assignment problems," International Journal of Engineering Research and Applications, vol. 2, pp. 1614–1618, 2012.

[71] A. Hoffman, "On simple linear programming problems," in Proceedings of Symposia in Pure Mathematics, vol. 7, pp. 317–327, World Scientific, 1963.

[72] A. Aggarwal and J. Park, "Notes on searching in multidimensional monotone arrays," in Foundations of Computer Science, 1988., 29th Annual Symposium on, pp. 497–512, IEEE, 1988.

[73] F. Supnick, "Extreme hamiltonian lines," Annals of Mathematics, pp. 179–201, 1957.

[74] G. Monge, Mémoire sur la théorie des déblais et des remblais. De l'Imprimerie Royale, 1781.

[75] V. Deineko, R. Rudolf, and G. J. Woeginger, "On the recognition of permuted supnick and incomplete monge matrices," vol. 33, pp. 559–569, 08 1996.

[76] A. Aggarwal, D. Kravets, J. Park, and S. Sen, "Parallel searching in generalized monge arrays with applications," in Proceedings of the second annual ACM symposium on Parallel algorithms and architectures, pp. 259–268, ACM, 1990.

[77] R. Burkard, B. Klinz, and R. Rudolf, "Perspectives of monge properties in optimization," Discrete Applied Mathematics, vol. 2, no. 70, pp. 95–161, 1996.

[78] U. Derigs, O. Goecke, and R. Schrader, "Monge sequences and a simple assignment algorithm," Discrete applied mathematics, vol. 15, no. 2, pp. 241–248, 1986.

[79] S. Arora and M. Puri, "A variant of time minimizing assignment problem," European Journal of Operational Research, vol. 110, no. 2, pp. 314–325, 1998.

[80] R. Malhotra, H. Bhatia, and M. Puri, “The three dimensional bottleneck assignment problem and its variants,” Optimization, vol. 16, no. 2, pp. 245–256, 1985.

[81] R. Rudolf, “Recognition of d-dimensional monge arrays,” Discrete Applied Mathematics, vol. 52, no. 1, pp. 71–82, 1994.

[82] A. Ćustić, B. Klinz, and G. J. Woeginger, “Planar 3-dimensional assignment problems with monge-like cost arrays,” arXiv preprint arXiv:1405.5210, 2014.

[83] A. Aggarwal and J. Park, “Sequential searching in multidimensional monotone arrays, to appear in j,” Algorithms.

[84] J. Gerhard, “Some problems around travelling salesmen, dart boards, and euro-coins,” Bulletin of the EATCS no, vol. 90, pp. 43–52, 2006.

[85] V. Deineko and A. Tiskin, “One-sided monge tsp is np-hard,” 2006.

[86] R. J. Wilson, “Introduction to graph theory,” 1986.

[87] A. Aggarwal, B. Schieber, and T. Tokuyama, “Finding a minimum-weightk-link path in graphs with the concave monge property and applications,” Discrete & Computational Geometry, vol. 12, no. 1, pp. 263–280, 1994.

[88] R. Rudolf, “On monge sequences in d-dimensional arrays,” Linear algebra and its applications, vol. 268, pp. 59–70, 1998.