London South Bank University

Solution Approaches to the Three-Index Assignment Problem

Mohamed Mehbali

https://orcid.org/0000-0002-4920-5221

A thesis submitted in partial fulfilment of the
requirements of London South Bank University,
for the degree of Doctor of Philosophy.
June 2024

Acknowledgements

First and foremost, I thank Allah Almighty for providing me with the will, motivation, strength, guidance, health, and capabilities required to complete this thesis.

I would like to express my sincere gratitude to all those who supported me to achieve this goal throughout my PhD journey.

I extend my sincere gratitude to my dedicated supervisors, Dr. Carrie Rutherford and Prof. Roy Cerqueti, for their unwavering help throughout my PhD research project. Their invaluable guidance was instrumental in advancing my thesis to completion. I would also like to express my appreciation to Dr. Valerio Ficcadenti, my progress reviewer, who later became my second supervisor for his multiple constructive feedback and valuable guidance. Additionally I thank Dr. Nouredin Sadawi, a computing expert, for his kindness in agreeing to review my code and help me set up a GitHub account. Furthermore, I am thankful to my line managers, Dr. Lesley Roberts, Argyrios Georgopoulos, and Marc Griffith, for their invaluable support. I extend my heartfelt thanks to my academic developer colleagues at LSBU and all CRIT staff for their support.

I would like to thank my family including my wife, my children and grandchildren who have accompanied me in this journey and motivated me to move forward.

Declaration

I declare that this PhD thesis titled "**Solution Approaches to the Three-index Assignment Problem**" is my work, the result of my original effort, and that there is no portion of my thesis has been submitted, for any degree, diploma, or other qualification at London South Bank University or any other University, and all sources used have been acknowledged with references and permissions granted where required.

Abstract

This thesis explores the axial Three-Index Assignment Problem (3IAP), also called the Multidimensional Assignment Problem. The problem consists in allocating n jobs to n machines in n factories, such that exactly one job is executed by one machine in one factory at a minimum total cost. The 3IAP is an extension of the classical two-dimensional assignment problem. This combinatorial optimisation problem has been the subject of numerous research endeavours, and proven NP-hard due to its inextricable nature.

The study adopts an algorithmic approach to develop swift and effective methods for solving the problem, focusing on balancing computational efficiency and solution accuracy. The Greedy-Style Procedure (GSP) is a novel heuristic algorithm for solving the 3IAP, guaranteeing feasible solutions in polynomial time. Specific arrangements of cost matrices can lead to the generation of higher-quality feasible solutions. In addressing the 3IAP, analysing the tie-cases and the matrix ordering led to new variants. Further exploration of cost matrix characteristics has allowed two new heuristic classes to be devised for solving 3IAP. The approach focuses on selecting the best solution within each class, resulting in an optimal or a high-quality approximate solution. Numerical experiments confirm the efficiency of these heuristics, consistently delivering quality feasible solutions in competitive computational times. Moreover, by employing diverse optimisation solvers, we propose and implement two effective methods to achieve optimal solutions for 3IAP in good CPU times.

The study introduces two local search methods based on evolutionary algorithms to solve 3IAP. These approaches explore the solution space through random permutations and the Hungarian method. Building on this, a hybrid genetic algorithm that integrates these local search strategies has been proposed for solving the 3IAP. Implementing the Hybrid Genetic Algorithm (HGA) produces high-quality solutions with reduced computational time, surpassing traditional deterministic approaches. The efficiency of the HGA is demonstrated through experimental results and comparative analyses. On medium to large 3IAP instances, our method delivers comparable or better solutions within a competitive computational time frame.

Two potential future developments and expected applications are proposed at the end of this project. The first extension will examine the correlation between cost matrices and the optimal total cost of the assignment and will investigate the dependence structure of matrices and its influence on optimal solutions. Copula theory and Sklar's theorem can help with this analysis. The focus will be on understanding the stochastic dependence of cost matrices and their multivariate properties. Furthermore, the impact of variations in cost distributions, is often modelled based on economic sectors. The second extension involves integrating variable costs defined by specific probability distributions, enhancing the comprehensive analysis of economic scenarios and their impact on the assignment problem. The study considers various well-defined probability distributions and highlights more practical applications of the assignment problem in real-world economics.

The project's original contribution lies in its algorithmic approach to investigating the 3IAP, which has led to the development of new, fast, and efficient heuristic methods that strategically balance computational speed and the accuracy of the solutions achieved.

Contents

List of Tables

List of Figures

List of Abbreviations

Acronyms	**Meaning**
3IAP	Three-index Assignment Problem
B&B	Branch and Bound method
DM	Diagonals Method
FIFO	First In First Out rule
GA	Genetic Algorithm
GAP	Generalised Assignment Problem
GSP	Greedy-style Procedure
GRASP	Greedy Randomised Adaptive Search Procedure
HGA	Hybrid Genetic Algorithm
INLP	Integer Linear Programming
LAP	Linear Assignment Problem
LIFO	Last In First Out rule
MAP	Multi-index Assignment Problem
RPNM	Random Permuted Numbers' method
QAP	Quadratic Assignment Problem
SPP	Set Partition Problem
TLS	Triple Local Search method
TP	Transportation Problem
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem

Chapter 1

Introduction

This research project focuses on an optimisation problem, aiming to identify the optimal element from a finite set of candidate solutions. An illustrative example is the one-to-one task allocation to a team of agents. Assigning agents to tasks incurs costs, and the primary objective is to minimise the total cost. This problem is well-known in literature and referred to as the two-dimensional assignment problem or Linear Assignment Problem (LAP).

LAP stands as a fundamental model in combinatorial optimisation, with theoretical significance and broad applicability. Examples of LAP practical utility may include the optimisation of staff allocation to enhance performance, allocation of tasks to machines for maximum productivity, or teacher-class assignments for specific objectives. The problem is pivotal in optimisation, providing valuable solutions for various real-world scenarios.

Considering further location constraints, resource availability, or timetabling constraints within assignment problems transforms them into a three-dimensional assignment problems. The introduction of a third 'dimension' makes the problem harder.

The Multi-Index Assignment Problem (MAP) is a natural extension of LAP.

Despite extensive research conducted over the past five decades, MAP remains one of the most challenging combinatorial problems to solve optimally. Indeed, Karp (1972) demonstrated that the multi-dimensional assignment problem becomes NP-hard when the dimension is three or higher.

A polynomial-time algorithm exists for the standard LAP, but MAP remains intractable. As the problem size increases, execution time for MAP grows exponentially. Consequently, exact solutions are only achievable for smaller MAP instances, and approximate solutions are expected in a reasonable time-frame for larger instances.

This research project investigates algorithmic approaches for addressing the Three-Index assignment problem (3IAP), aiming to contribute to its resolution. The study objective is to devise new efficient heuristics outperforming traditional deterministic methods, thereby achieving quality solutions in a reduced computational time, and the ability to handle larger problem instances using state-of-the-art optimisation solver. Furthermore, the project introduces a metaheuristic that integrates the genetic algorithm and a local search method. This approach helps to achieve optimal or near-optimal solutions for large instances of the problem within a reasonable timeframe.

The research project conducts a comprehensive review of prominent publications concerning MAP and subsequently investigates key works on assignment problems. This literature review offers a critical analysis of the primary contributors to MAP solution procedures and contextualises our study within the ongoing discourse on these methodologies.

The thesis is structured as follows. Chapter 1 introduces the assignment problem and defines the Three-Index Assignment Problem (3IAP), highlighting its importance in combinatorial optimisation, linking it to common combinatorial problems and providing mathematical formulations. Chapter 2 reviews existing approaches in the literature for solving the 3IAP and presents its wide-ranging applications in the real world. It then articulates the motivation behind addressing the 3IAP, emphasising its critical practical significance.

Chapter 3 proposes the Greedy-Style Procedure and the Diagonals method, along with their variants. Algorithms are proposed, detailed examples illustrate these methods and extensive numerical experiments are carried out to demonstrate their efficiency. Chapter 4 introduces two innovative heuristic categories for the 3IAP, underpinned by theoretical or empirical validation of their efficacy. These heuristic methods generate a spectrum of results, enabling the identification and selection of the best solutions. The quality of these solutions is computationally assessed. Additionally, the chapter details the design and implementation of two programs derived from optimisation solvers, facilitating the attainment of optimal solutions.

Chapter 5 explores evolutionary algorithms and culminates in a proposal for a hybrid genetic algorithm tailored to 3IAP. This section provides an in-depth description of the algorithm's design and development processes. Further, it discusses the implementation and testing phases, with numerical results presented to validate the hybrid approach performance. Chapter 6 delineates two prospective applications of the 3IAP within the field of economics and suggests potential directions for future research.

In Chapter 7, the conclusion summarises the thesis's key findings and contributions, emphasising the importance and impact of this research. It provides a concise overview of each chapter and discusses the practical implications of this research project and its potential to influence future studies in the field.

1.1 The Problem Description

This section starts with an introduction to the classical two-dimensional assignment or linear assignment problem (LAP), and presents its mathematical formulations.

Subsequently, the connections of LAP with prevalent combinatorial problems will be elucidated.

1.1.1 Linear Assignment Problem

The classical LAP represents a typical combinatorial optimisation problem with various real-life applications. It aims to find a perfect matching within a complete bipartite graph $G = (U \cup V; E)$, where the size of the vertex sets U and V are equal, denoted by $|U| = |V| = n$. Given a weight function defined on the edge set E. The objective is to identify a perfect matching with the optimal total weight. The notion of perfect matching can be formulated by a bijective mapping p between sets U and V, which can equivalently be interpreted as a permutation (Burkard, Dell' Amico, and Martello, 2009).

Assuming that n tasks must be completed by n agents, the corresponding assignment can be represented by a permutation p of $\{1, 2, 3, \cdots, n\}$. If $j = p(i)$ and $c_{ip(i)}$ is the cost of allocating task i to agent j, the problem is to find a permutation with the minimum total cost $\underbrace{min}_{p \in \pi_n} Z = \sum_{i=1}^{n} c_{ip(i)}$, where π_n is the set of all permutations of n elements. Note any permutation p corresponds to a unique $n \times n$-matrix $X_p = (x_{ij})$, defined by:

$$x_{ij} = \begin{cases} 1, & \text{if } j = p(i); \\ 0, & \text{otherwise.} \end{cases}$$

A permutation matrix has one 1-entry per column and per row at exactly n non-zero coefficients.

$$\text{subject to} \quad \begin{cases} \sum_{i=1}^{n} x_{ij} = 1 & \text{for} \quad i = 1, 2, \ldots, n \\ \sum_{j=1}^{n} x_{ij} = 1 & \text{for} \quad j = 1, 2, \ldots, n \\ x_{ij} \in \{0, 1\} & \text{for} \quad i, j = 1, 2, \ldots, n \end{cases} \tag{1.1}$$

The system of the above linear equations (1.1) is called *assignment constraints* and has $n!$ solutions since there are $n!$ permutations of n objects.

The bipartite graph $G = (U \cup V; E)$ below illustrates an assignment of size five. The vertices of the set U correspond to tasks, while those of V represent agents. In this model, each task is allocated to exactly one agent, based on the permutation $p = (2, 4, 5, 1, 3)$, ensuring a direct, one-to-one matching between tasks and agents. The adjacency matrix A of the bipartite graph $G = (U \cup V; E)$ is none other than the matrix associated with the permutation p.

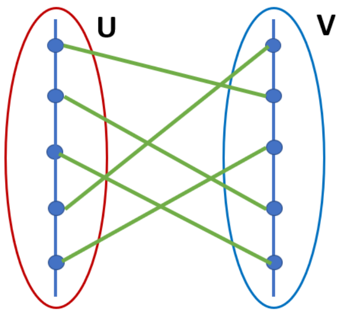

Figure 1.1: Example of a bipartite graph of order 5.

Hitchcock (1941) was the first to formulate LAP as an integer linear program

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

M is the adjacency matrix of the bipartite graph G.

$$\begin{array}{c|ccccc} & task_1 & task_2 & task_3 & task_4 & task_5 \\ \hline agent_1 & 0 & 1 & 0 & 0 & 0 \\ agent_2 & 0 & 0 & 0 & 1 & 0 \\ agent_3 & 0 & 0 & 0 & 0 & 1 \\ agent_4 & 1 & 0 & 0 & 0 & 0 \\ agent_5 & 0 & 0 & 1 & 0 & 0 \end{array}$$

This matrix represents the permutation p.

in a comprehensive mathematical formulation.

The decision variable is defined by $x_{ij} = 1$ if task j is assigned to agent i and 0 otherwise.

$$\begin{aligned}
&\text{Minimize} \quad && Z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} && && (1.2)\\
&\text{subject to} \quad && \sum_{j=1}^{n} x_{ij} = 1, && i = 1, 2, \dots, n && (1.2a)\\
& && \sum_{i=1}^{n} x_{ij} = 1, && j = 1, 2, \dots, n && (1.2b)\\
& && x_{ij} \in \{0, 1\}, && i, j = 1, 2, \dots, n && (1.2c)
\end{aligned}$$

The assignment constraints (1.2a) mean that each task is assigned to exactly one agent. The system of equations (1.2b) indicates that each agent performs exactly one task, and the objective function Z (1.2) evaluates the total cost of such an assignment, where c_{ij} is the cost when task j is executed by agent i. Therefore, the LAP consists of searching for an assignment with minimum total cost. The integer linear program(1.2)–(1.2c) can be also expressed in matrix form.

$$\begin{aligned}
&\text{Minimize} \quad && Z = C \cdot x && (1.3)\\
&\text{subject to} \quad && A.x = \boldsymbol{e} && (1.3a)\\
& && x \in \{0, 1\}^{n^2} && (1.3b)
\end{aligned}$$

where A represents the $2n \times n^2$ constraint matrix, and the objective function (1.3) is $Z = C \cdot x$, where C is the n^2-dimensional row-vector with nonnegative components. Note that the column-vector $\boldsymbol{e}$ has $2n$ components, all equal to 1. Finally, the decision variable $x \in \{0,1\}^{n^2}$.

Definition 1. Given an undirected graph $G = (V; E)$ not necessarily bipartite, the *incidence matrix* $A(G)$ of G is defined by $|V|$ rows and $|E|$ columns with all entries from {0,1}. Each coefficient a_{ve} of $A(G)$ corresponds to a row for vertex $v \in V$, and a column for edge $e \in E$, with

$$a_{ve} = \begin{cases} 1, & \text{if } v \text{ is an endpoint of } e; \\ 0, & \text{otherwise.} \end{cases}$$

The matrix $A(G)$ represents the vertex-edge incidence modelling the graph structure. Further, A in (1.3a) is the incidence-matrix of the complete bipartite graph $G = (U \cup V; E)$.

Definition 2. A matrix A is *totally unimodular* if every square sub-matrix of A has its determinant in {-1, 0, 1}.

Theorem 1.1. (Hoffman-Kruskal, 1956)
Let $A \in \mathbb{Z}^{m\times n}$ be a totally unimodular matrix and b a vector of $\mathbb{Z}^m$. Then, all basic solutions of the linear system $Ax = b$ with $x \geq 0$, are integer.

Corollary 1. The incidence matrix of a bipartite graph is totally unimodular.

Consequently, LAP retains an integer optimal solution even when the integrity constraint (1.3b) of the integer linear program (1.3)–(1.3b) is relaxed. This property is due to the unique *totally unimodular* structure of the constraint matrix. Linear programming has substantially contributed to the development of efficient algorithms for solving an important class of combinatorial problems (Dantzig, 1963 and Taha, 2011).

1.1.2 LAP links with common combinatorial problems

The literature shows a tight connection between the assignment problem and main combinatorial optimisation problems. For instance, LAP is considered a particular case of the transportation problem (TP), as depicted below.

1.1.2.1 Transportation Problem

An industrial challenge often encountered involves the efficient transportation of a particular product from n warehouses to stores while minimising costs. Let a_i denote the total supply of the product available at warehouse i, and b_j represents the total demand for the product at store j. The cost associated with transporting one product unit from warehouse i to store j is c_{ij}; a_i, b_j and c_{ij} are assumed positive. Utilising the nonnegative variable x_{ij} to evaluate the quantity of the product transported from a given warehouse to a specific store, a linear programming model for the transportation, proposed by Monge (1871), is presented below.

$$\text{Minimize} \qquad Z = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \tag{1.4}$$

$$\text{subject to} \qquad \sum_{j=1}^{n} x_{ij} \leq a_i, \qquad i = 1, 2, \ldots, n \tag{1.4a}$$

$$\sum_{i=1}^{n} x_{ij} = b_j, \qquad j = 1, 2, \ldots, m \tag{1.4b}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, 2, \ldots, n; j = 1, 2, \ldots, m \tag{1.4c}$$

The TP objective function minimises total transportation cost, subject to constraints. The first system of inequalities (1.4a) represents supply constraints, ensuring that each warehouse i distribution does not exceed its supply a_i. Conversely, the second system of inequalities (1.4b) represents demand constraints assuring that each store j incoming distribution meets at most its demand b_j. Both coefficients a_i and b_j are assumed nonnegative. The problem involves allocating resources efficiently and balancing supply limitations and demand requirements to optimise overall transportation expenditure.

The TP can be associated with a complete bipartite graph $G = (U \cup V; E)$, where U represents the set of warehouses and V the set of stores. Each variable x_{ij} corresponds to an arc of E from warehouse i to store j. The following linear program formulates the TP in matrix form, shown below.

$$\text{Minimize} \qquad Z = C \cdot x \tag{1.5}$$

$$\text{subject to} \qquad A.x \geq B \tag{1.5a}$$

$$x \geq 0 \tag{1.5b}$$

where A is the incidence-matrix of the bipartite graph $G = (U \cup V; E)$. Thus, A is an $(m+n) \times m.n$ matrix, $B = \begin{pmatrix} -a \\ b \end{pmatrix}$ is an $(m+n)$-column vector and C is a $(m.n)$-row vector. Components of vectors B and C are assumed nonnegative.

In (1.4), summing the n inequalities (1.4a) and the m inequalities (1.4b) result in:

$$\sum_{j=1}^{m} b_j \leq \sum_{j=1}^{m} \sum_{i=1}^{n} x_{ij} \leq \sum_{i=1}^{n} a_i. \tag{1.6}$$

In the conventional transportation problem model, the total supply $\sum_{i=1}^{n} a_i$ is assumed to equal to the total demand $\sum_{j=1}^{m} b_j$; otherwise, a *'dummy'* destination is introduced to absorb the discrepancy, assigning a zero cost for distributions from warehouses to this dummy. This case maintains the original cost structure. Subsequently, a feasible TP solution satisfies all inequality constraints with equality, ensures compliance with conditions (1.4a) and (1.4b) in PL (1.4)–(1.4c), and fulfils all general constraints cast as equations.

1.1.2.2 Problem of Maximum Flow

Let $G(V, E)$ be a directed connected graph with V the set of its vertices and E the set of its edges. Each edge e of E has a nonnegative capacity $c(e)$, if $e \notin E$, then $c(e) = 0$. In addition, G possesses two distinguished vertices, a *source* s with in-degree 0 and a *sink* t with out-degree 0.

Definition 3. For any i vertex of V, $\delta^+(i)$ (or $\delta^-(i)$) denotes all edges of E having their initial (final, *respectively*) node i.

Definition 4. A *flow* is a function f defined from set E into $\mathbb{R}$ such that:

- $0 \leq f(e) \leq c(e), \quad \forall e \in E,$
- $\forall i, j \in V, f(i,j) = -f(j,i)$ skew symmetry, and
- $\sum_{e \in \delta^+(i)} f(e) = \sum_{e \in \delta^-(i)} f(e), \forall i \in V \setminus \{s,t\}$ the flow conservation.

The last equality means that at each vertex $i \in V \setminus \{s,t\}$, the quantity of flow entering i equals the quantity of flow exiting it. This condition is usually known as *Kirchhoff's* first law. The total amount of flow equals $F = \sum_{e \in \delta^+(s)} f(e)$.

The problem is therefore to determine the maximum amount of flow that can be shipped from the source s to the sink t. In other words, the objective is to maximise the outflow from the source s, *i.e.*, $Max(\sum_{e \in \delta^+(s)} f(e))$ or equally to maximise the inflow from the sink t, *i.e.*, $Max(\sum_{e \in \delta^-(t)} f(e))$.

The flow theory has diverse applications in both combinatorics and operational research. The problem of the maximum flow in the graph G can also be formulated by the following linear program.

$$
\begin{aligned}
&\text{Maximize} && Z = \sum_{e\in\delta^+(s)} f(e) && && (1.7)\\
&\text{subject to} && \sum_{e\in\delta^+(i)} f(e) = \sum_{e\in\delta^-(i)} f(e), && \forall i \in V \setminus \{s,t\} && (1.7a)\\
& && f(e) \le c(e), && \forall e \in E && (1.7b)\\
& && f(e) \ge 0, && \forall e \in E && (1.7c)
\end{aligned}
$$

Clearly, to any feasible solution of LP (1.7)–(1.7c) corresponds to a feasible flow for the graph G.

Ford and Fulkerson (1956) notably contributed to network optimisation by introducing an efficient algorithm for the maximum flow problem in a graph. Their method refers to the concept of augmenting paths from a source s to a sink t and the dual of the LP (1.7)–(1.7c). Ford and Fulkerson's algorithm demonstrated remarkable applicability to the TP, extending the principles of the Kuhn-Egervary (Dantzig, Ford, and Fulkerson, 1956) method to a broader context. Due to the problem's combinatorial nature, the algorithm again demonstrates the power of linear programming.

1.1.2.3 Maximum Perfect Matching in a Bipartite Graph.

Definition 5. Given a bipartite graph $G(X \cup Y, E)$ with $X \cup Y$ vertex set and E edge set,

- A *matching* is a subset $M \subseteq E$ such that each $v \in X \cup Y$ is an endpoint of at most one edge in M, and
- A matching M of the bipartite graph G, is said to be *perfect* if M matches all vertices of the graph G. In other words, each $v \in X \cup Y$ is an endpoint of exactly one edge of M.

Kuhn (1955) developed the Hungarian Method, an efficient algorithm for solving the assignment problem, LAP in polynomial time. The method was named in tribute to Hungarian mathematicians Dénes Kőnig and Jenő Egerváry whose initial work laid the groundwork for Kuhn's algorithm. The Hungarian method is particularly adept at constructing maximum perfect matchings in bipartite graphs.

Moreover, the Ford and Fulkerson algorithm has proven to be highly effective for finding maximum perfect matchings in bipartite graphs, a notable parallel to the capabilities of the Hungarian method.

Later, Munkres (1957) refined the Hungarian method by improving its computational complexity from $O(n^4)$ to $O(n^3)$. Since then, the Kuhn-Munkres algorithm has been known as the Hungarian method, a foundational, efficient algorithm in optimisation, preluding many modern methods in combinatorial optimisation (Rajabi-Alni, 2013).

1.1.2.4 Intersection of Two Matroids

Matroid theory is a branch of combinatorial mathematics that extends linear independence in vector spaces and graphs to abstract settings. A matroid consists of a finite set and a collection of subsets of this set, satisfying specific properties analogous to linear independence in vector spaces. The literature contains an interesting definition of LAP in terms of matroid theory, but only a few applications utilise such a formulation. Although not widespread, this interpretation yields significant insights, particularly in complex scenarios with matroidal structures. The matroid-based approach to LAP can provide tools for understanding the properties of feasible solutions and algorithms for finding optimal assignments.

Definition 6. Let E be a finite non-empty set, called ground set. Let us consider $\mathcal{F}$ a collection of subsets $F \subseteq E$, called independent sets. The system $(E, \mathcal{F})$ defines a *matroid* if it satisfies the three following axioms:

- $\varnothing \in \mathcal{F}$
- For each $F' \subseteq F \subseteq E$, if $F \in \mathcal{F}$, then $F' \in \mathcal{F}$.
- For any two elements of $\mathcal{F}$ (*i.e.*, A and B two subsets of E) with $|A| \leq |B|$ there exists an element e of $B \setminus A$, such that $A \cup \{e\} \in \mathcal{F}$.

Definition 7. Let $(E, \mathcal{F})$ be a matroid. A subset $F \subseteq E$, is a *basis* if F is independent maximal (for the inclusion). By analogy, the notion of independence of sets is comparable to that of linear independence of vectors, in linear algebra.

Welsh (1976) published a book offering a comprehensive introduction to matroid theory and its diverse applications.

Given a bipartite graph $G = (U \cup V, E)$. Let $\mathcal{F}_1$ be a collection of all subsets $F \subseteq E$, such that every vertex of U meets at most one edge of E. Similarly, let $\mathcal{F}_2$ be another collection of all subsets $F \subseteq E$, such that every vertex of V meets at most one edge of E. Clearly, the systems $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$ fulfil the above axioms, and define two matroids.

Both matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$ are defined on a common ground set E. A subset $F \subseteq E$ lies in the intersection of $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$ if $F \in \mathcal{F}_1$ and $F \in \mathcal{F}_2$. Consequently, any matching in the bipartite graph G corresponds to a set $F \in \mathcal{F}_1 \cap \mathcal{F}_2$; reciprocally, every set $F \in \mathcal{F}_1 \cap \mathcal{F}_2$ corresponds to a matching of G. Moreover, if U and V are assumed to have the same number of elements n, then every perfect matching of G is associated uniquely with a set $F \in \mathcal{F}_1 \cap \mathcal{F}_2$, with $|F| = n$. Furthermore, assuming every $e \in E$ has a cost $c(e)$, the LAP can be considered as a special matroid intersection problem. LAP becomes a search for a set $F \in \mathcal{F}_1 \cap \mathcal{F}_2$ with minimum cost *i.e.*, $\underbrace{\mathit{minimum}}_{F \in \mathcal{F}_1 \cap \mathcal{F}_2} \sum_{e \in F} c(e)$.

1.1.3 Multi-Index Assignment Problem

The axial Multi-Index (or Multi-Dimensional) Assignment Problem (MAP) generalises the standard LAP, extending it into higher dimensions. The MAP of dimension k for ($k \geq 3$), is defined as follows. Let $U_1, U_2, \ldots, U_k$ be k mutually disjoint sets, each of cardinality n. Let us consider the complete k-partite hypergraph $G(V, E)$ where $V = \bigcup_{i=1}^{k} U_i$ denotes the vertex set, and E the edge set. The problem is to determine n cliques, each of order k, such that every clique intersects every set U_i ($i = 1, 2, \ldots, k$) exactly at one vertex, in other words, to find a perfect matching in the hypergraph $G(V, E)$. The MAP thus involves partitioning the vertex set V into n pairwise disjoint cliques. If a cost function c is defined on the edge set E, the objective aims to find such a partition of minimum cost. The next section shows how LAP results are extended to the MAP case.

The research project focuses on studying the axial Three-Index Assignment Problem (3IAP), a particular case of the MAP of dimension three. Also referred to as the Three-Dimensional or Solid Assignment Problem, 3IAP aims to minimise the total cost involved in assigning n jobs (tasks) to n machines (agents) within n factories (locations). The 3IAP is characterised by three bijections, such that the formulation ensures that each job is assigned to exactly one machine and each machine is assigned to one factory. Due to its wide-ranging practical applications, the 3IAP attracts more research interest than the general MAP.

Pierskalla (1967) is the first to introduce the 3IAP as a natural extension of the classical assignment problem (LAP), drawing inspiration from Haley's research on the multi-index transportation problem, as outlined by (Schell, 1955) and Haley's foundational work on the solid Transportation Problem (TP) (Hayley, 1962, 1963, and 1965). Later, Pierskalla (1968) extended the 3IAP into the broader MAP. Comprehensive insights into 3IAP are presented in the (Spieksma, 2000) survey, while Burkard *et. al.*, (2009) provide a detailed exploration of the MAP.

1.1.4 Mathematical Formulations

The 3IAP is a special form of the assignment problem that extends the traditional two-dimensional assignment model to three dimensions. In graph theory, the 3IAP involves a complete tri-partite hypergraph, $G = (I \cup J \cup K, E)$, consisting of three mutually disjoint vertex sets I, J, and K each with n vertices. The hypergraph edges E are represented by triplets (i, j, k) where $i \in I$, $j \in J$, and $k \in K$. The problem aims to determine a collection of n such triplets, ensuring every vertex from I, J, or K is uniquely included in one triplet, constituting n disjoint cliques of size three. The objective is to minimise the total cost, as dictated by a cost function defined over the edge set. Thus, the 3IAP searches for the minimum perfect matching in the hypergraph G. Figure 1.2 illustrates a 3IAP example represented by a tripartite hypergraph of size 5 whose edges are partitioned into five three-coloured triangles.

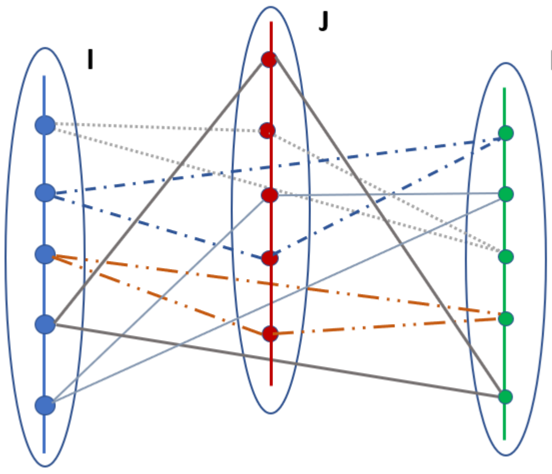

Figure 1.2: Example of a tri-partite graph of size 5.

1.1.4.1 First Formulation

The 3IAP is defined in the literature in various ways. Burkard and Çela (1999) present a combinatorial formulation which involves three sets I, J, and K of n elements each. The 3IAP can be described using two bijective mappings or permutations (a) p between I and J, and (b) q between J and K. If c_{ijk} is the cost associated with the triplet (i, j, k) of $I \times J \times K$, the 3IAP aims to find an assignment of minimum total cost. The problem seeks two permutations p and q of $\{1, 2, \ldots, n\}$, realising the minimum total cost, *i.e.*, $\underbrace{min}_{p,q\in\pi_n} Z = \sum_{i=1}^{n} c_{ip(i)q(i)}\}$.

The axial 3IAP is then defined as:

$$\text{Minimize} \qquad Z = \sum_{i=1}^{n} c_{ip(i)q(i)} \tag{1.8a}$$

$$\text{subject to} \qquad p, q \quad \text{two permutations of} \quad \pi_n. \tag{1.8b}$$

where π_n is the set of all permutations of n elements.

The final section of the current project introduces a new genetic algorithm for 3IAP, heavily based on this formulation.

1.1.4.2 Second Formulation

The earliest mathematical formulation of the axial 3IAP is the following linear program (Pierskalla, 1967). Given three disjoint sets I, J, K and a cost function $c : I \times J \times K \longmapsto \mathbb{R}_+$; the decision variable is binary, $x_{ijk} = 1$ if the job i is assigned to machine j in factory k, and 0 otherwise.

$$\text{Minimize} \qquad Z = \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} c_{ijk}.x_{ijk} \tag{1.9}$$

$$\text{subject to} \qquad \sum_{j=1}\sum_{k=1} x_{ijk} = 1, \qquad i \in I \tag{1.9a}$$

$$\sum_{i=1}\sum_{k=1} x_{ijk} = 1, \qquad j \in J \tag{1.9b}$$

$$\sum_{i=1}\sum_{j=1} x_{ijk} = 1, \qquad k \in K \tag{1.9c}$$

$$x_{ijk} \in \{0, 1\}, \qquad i \in I, j \in J, k \in K \tag{1.9d}$$

Here, the model (1.9)–(1.9d) represents an integer linear program where the nonnegative coefficient c_{ijk} may represent the cost of assigning job i to machine j in factory k. If the sets I, J, and K have different cardinalities, the problem is considered unbalanced, in this case simply add zero cost to the rows and columns as appropriate to balance the problem. A balanced 3IAP resembles the model (1.9)–(1.9d) with the supplementary condition on cardinalities $|I| = |J| = |K| = n$, thus the following linear program is written.

$$\begin{aligned}
\text{minimize} \qquad & Z = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}.x_{ijk} && && (1.10)\\
\text{subject to} \qquad & \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1, && i = 1, 2, \ldots, n && (1.10a)\\
& \sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1, && j = 1, 2, \ldots, n && (1.10b)\\
& \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1, && k = 1, 2, \ldots, n && (1.10c)\\
& x_{ijk} \in \{0, 1\}, && i, j, k = 1, 2, \ldots, n && (1.10d)
\end{aligned}$$

The objective function Z (1.10) of the integer linear program evaluates the total cost of the assignment. Linear equalities (1.10a)–(1.10c) define the problem constraints, while (1.10d) specifies the binary decision variable. If the constraint (1.10d) is substituted with the inequality:

$$x_{ijk} \geq 0 \qquad \forall i \in I, \forall j \in J, \forall k \in K.$$

then the above LP model becomes a continuous linear program which constitutes a relaxation of the original problem.

1.1.4.3 Third Formulation

The formulation of the 3IAP model corresponds to a linear program expressed in matrix form.

$$\text{minimize} \qquad Z = \boldsymbol{C} \cdot x \tag{1.11}$$

$$\text{subject to} \qquad A_n.x = \boldsymbol{e} \tag{1.11a}$$

$$x \in \{0,1\}^{n^3} \tag{1.11b}$$

The axial 3IAP is formulated as an integer linear program expressed in matrix form. Let us assume $n \geq 3$, the binary decision variable is a vector of $\mathbb{R}^{n^3}$, Equation (1.11b). The objective function is $Z = \boldsymbol{C} \cdot x$ (1.11), where $\boldsymbol{C}$ is a n^3-dimensional row-vector with nonnegative components, representing the cost associated with every potential triple in the hypergraph. Equation (1.11a) outlines the constraint matrix A_n, of rank $3n-2$, which has a specific structure of n^3 columns and $3n$ rows, and all its coefficients are 0 or 1. The vector $\boldsymbol{e}$ in Equation (1.11a) has all its components equal to 0 or 1. All 3IAPs of size n share the same constraint matrix A_n; distinct 3IAPs differ only in their objective functions. Investigating the function objective characteristics for solution approaches would be worthwhile.

The integer linear program (1.11)–(1.11b) holds up to $(n!)^2$ feasible solutions. This high number of potential solutions reflects the combinatorial complexity of the 3IAP, especially as the size n increases. Solving such a problem optimally often requires sophisticated optimisation techniques, especially for larger values of n. The 3IAP remains one of the most exacting combinatorial problems to solve optimally since Karp (1972) proved that the MAP is *NP*-hard for any dimension n greater than or equal to 3.

1.1.5 3IAP links with common combinatorial problems

The formulation of the axial 3IAP as a linear program underlines its significant association with major combinatorial optimisation problems. This connection is evidenced by the problem's structure, involving a combination of a linear objective function, a constraint matrix and binary decision variables, which are the hallmarks of many classic combinatorial problems. Such formulations are instrumental in solving complex optimisation problems, often encountered in diverse fields such as operations research, computer science, applied mathematics, industry, and economics, highlighting the 3IAP's relevance and applicability in diverse problem-solving contexts.

1.1.5.1 Three-Index Transportation Problem

By analogy with LAP, the 3IAP can be regarded as a particular variant of the Three-Index transportation problem, an extension of the traditional transportation problem (TP). This similarity is evident when the 3IAP is modified such that the linear equations (1.10a)–(1.10c) have arbitrary nonnegative values on the right-hand side and the sets I, J, and K differ in cardinality. Additionally, relaxing the integrity constraint (1.10d) transforms the problem into the more general Three-Index transportation problem.

1.1.5.2 Set Partitioning Problem

The Set Partitioning Problem (SPP) is a classic problem in combinatorial optimisation and operations research, which entails dividing items into distinct subsets to meet specific criteria, often for optimising outcomes. SPP extensive application across multiple sectors, underlines its importance, as evidenced by the substantial academic literature. An SPP can be stated as follows.

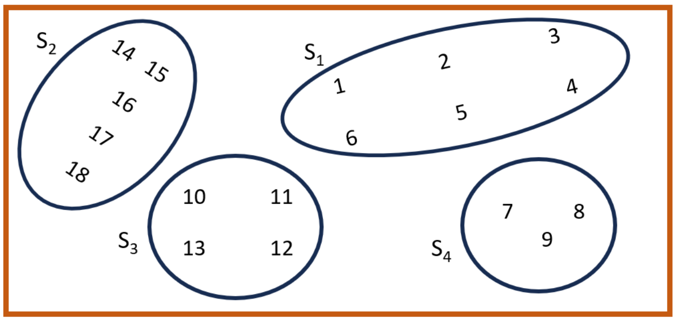

Figure 1.3: Example of a set partition.

Definition 8. Let $E = \{1, 2, \dots n\}$ be a set of items, and $S = \{S_1, S_2, \dots, S_m\}$ a set of subsets of E. Let I be a set of indices $\{1, 2, \dots m\}$ then I defines a *partition* of E if and only if:

- $\bigcup_{i \in I} S_i = E$
- $S_i \cap S_j = \varnothing, \quad \forall i, j \in I, \quad i \neq j.$

Example: Let $E = \{1, 2, \dots, 18\}$ be a set of 18 items and $S = \{S_1, S_2, \dots, S_m\}$ a set of subsets of E. Consider $P = \{1, 2, 3, 4\}$ a subset of $\{1, 2, \dots, 18\}$, then Figure 1.3 shows that P is a partition of E.

If a cost function is defined $c : S \longrightarrow \mathbb{R}_+$, then the sum $\sum_{i \in P} c(S_i)$ equals the cost of the partition P. The objective of SPP is to find the partition $P^\star$ of E, of the minimum cost.

Thus, the 3IAP is also considered a particular case of the SPP, whose mathematical formulation is the following.

$$\text{Minimize} \qquad Z = \boldsymbol{C} \cdot x \tag{1.12}$$

$$\text{subject to} \qquad M.x = \boldsymbol{e} \tag{1.12a}$$

$$x \in \{0, 1\}^n \tag{1.12b}$$

where M is the $n \times m$ constraint matrix whose coefficients are in $\{0,1\}$, and the column-vector $\boldsymbol{e}$ in (1.12a), has m components all equal to 1. The decision variable x is a binary vector of dimension n, where each element equals 0 or 1, as displayed in equation (1.12b).

1.1.5.3 Set Packing Problem

If the equality (1.12a) is replaced by an inequality, the formulation of the Set Packing Problem results in another mathematical problem that involves finding the largest collection of sets that do not overlap. It is a combinatorial optimisation problem with applications in various fields such as computer science, operations research, and logistics.

The Set Packing Problem mathematical formulation is:

$$\text{Minimize} \qquad Z = \boldsymbol{C} \cdot x \tag{1.13}$$

$$\text{subject to} \qquad M.x \leq \boldsymbol{e} \tag{1.13a}$$

$$x \in \{0,1\}^n \tag{1.13b}$$

where the $n \times m$ constraint matrix M in (1.13a), is the same as in SPP and the column-vector $\boldsymbol{e}$ is similar to that in (1.12a), having m components all smaller than or equal to 1. The same decision variable x is a binary vector of dimension n, where all components equal 0 or 1, as shown in equation (1.13b).

1.1.5.4 Set Covering Problem

If the equality (1.12a) is substituted by another inequality, this leads to the Set Covering Problem, whose mathematical formulation is the following.

$$\text{Minimize} \qquad Z = \boldsymbol{C} \cdot x \tag{1.14}$$

$$\text{subject to} \qquad M.x \geq \boldsymbol{e} \tag{1.14a}$$

$$x \in \{0,1\}^n \tag{1.14b}$$

Nearly all identical integer linear programs represent these three problems. The set partitioning problem (1.12)–(1.12b), the set packing problem (1.13)–(1.13b), and the set covering problem (1.14)–(1.14b) have been extensively investigated, and useful results have been achieved and published in the literature (Boshetti *et al.*, 2008).

1.1.5.5 Intersection of Three Matroids

Like LAP, the 3IAP also possesses a matroid-based formulation yet to be appropriately explored. A collection F of triplets (i, j, k) from $I \times J \times K$ defines an assignment if and only if they satisfy the following inequalities.

- $|F \cap \{i\}| \geq 1 \quad \forall i \in I,$
- $|F \cap \{j\}| \geq 1 \quad \forall j \in J,$
- $|F \cap \{k\}| \geq 1 \quad \forall k \in K.$

Let $(E, \mathcal{F}_I)$ define a partition matroid on the ground set $E = I \times J \times K$ where the family $\mathcal{F}$ of collection F triplets such that every element of I appears at least once in the collection F. Similarly, the matroids $(E, \mathcal{F}_J)$ and $(E, \mathcal{F}_K)$ are defined on the same ground set E. The 3IAP then returns to the search for a subset $F^\star$ of minimum cost and cardinality n in the intersection of the three matroids $\mathcal{F}_I, \mathcal{F}_J$ and $\mathcal{F}_K$, *i.e.*, $\underbrace{\textit{minimum}}_{F \in \mathcal{F}_I \cap \mathcal{F}_J \cap \mathcal{F}_K} \sum_{e \in F} c(e)$.

The book 'Assignment Problems' by Burkard, Dell' Amico, and Martello (2009) provides a comprehensive analysis of various assignment problems; it is the sole reference that delves into the 3IAP using a matroid-based formulation. Given this unique approach, as future development, there is an opportunity to explore this avenue by investigating the insights and methodologies covered in this seminal work to advance the understanding and problem-solving strategies for the 3IAP applications.

Chapter 2

Literature Review and Applications

This chapter reviews the existing approaches for solving the 3IAP and MAP in general and illustrates their broad applicability. The following section presents diverse real-world applications underscoring the problem's significance across various domains through an examination of its practical implications.

2.1 Applications

The 3IAP, a principal variant of the MAP, has garnered significant scholarly focus over the past thirty years. Its applications have expanded into higher dimensions, notably in multi-target tracking and data association, as exemplified in (Vadrevu and Nagi, 2020), and in addressing distributed multi-object tracking (Chen *et al.*, 2023). Predominantly, MAP applications cover sectors such as data association, educational timetabling, scheduling, satellite-related operations, and biological studies, as evidenced by extensive literature reviews.

2.1.1 Data Association

MAP applications are most prominent in data association, especially in multi-sensor data association. Aligning measurements from one or more sensors with the same object, is a challenge addressed by (Poore, 1994; Andrijich and Caccetta, 2001). Another application of MAP concerns matching information from multiple sensors for tracking elementary particle trajectories. The solution complexity and approach are comparable to those of the five-dimensional assignment problem, as discussed in (Pusztaszeri *et al.*, 1996).

2.1.2 Multi-target Tracking

MAP, with decomposable costs, has been applied to multi-target and multi-sensor tracking and in-plane tracking (Bandelt *et al.*, 2002). This application is also evident in plane tracking, as discussed in the works of (Murphey *et al.*, 1998), (Burkard and Çela 1999), and (Pardalos and Pitsoulis 2000). Comprehensive surveys of assignment problems in multiple-target tracking were conducted by (Poore and Gadaleta, 2006) and later by (Tauer and Nagi 2013), providing detailed insights into the evolution and applications of MAP in these complex tracking scenarios. A recent application Zhang *et al.*, (2023) proposed a multi-target tracking method based on multi-sensor labelled multi-Bernoulli filter in underwater multi-static networks with autonomous underwater vehicles.

2.1.3 Managing Resources

Pierskalla (1967) introduces an early application of MAP to dynamic facility locations, demonstrating its utility in scheduling capital investments like warehouses and factories across different locations and times. Later, Grundel *et al.*, (2004) expand further the scope of MAP, applying it to production planning and resource allocation, illustrating its efficacy in optimising industrial processes and managing resources, thus showcasing its versatility in operational planning.

2.1.4 Multi-surveillance

The literature on MAP reveals its diverse applications, ranging from satellite launching to multi-surveillance (Pierskalla, 1968; Balas and Landweer, 1983). Furthermore, Grundel *et al.*, (2004) delves into its use in real-time radar surveillance, including target tracking systems, image recognition, and air traffic control. Additional applications are evident in computer vision (Veenman *et al.*, 2003) and circuit board assembly optimisation (Spieksma, 2000), highlighting MAP's wide-ranging impact in various advanced technological domains.

2.1.5 Educational Timetabling

MAP has also been effectively applied in educational timetabling for assigning teachers to courses and sessions, first introduced by (Pierskalla, 1967). Frieze and Yadegar (1981) extend this application to schedule teacher trainees. In healthcare, MAP has been employed for scheduling in elderly day-care centres (Lin *et al.*, 2016) and in broader healthcare sector timetabling (Faudzi *et al.*, 2018). Pérez (2017) investigates MAP in solving school timetabling case studies. More recently, Badoni*et al.*, (2023) use a genetic algorithm to tackle university course timetabling, illustrating MAP's continued evolution and relevance in solving complex scheduling problems.

2.1.6 Applications in Biology

Applications of MAP are also encountered in biology. For instance, Arora (2013) proposes a 3IAP model to solve the alignment of three protein-interaction networks. In a recent study, Gabrovšek *et al.*, (2020) investigate the multi-association of three sets of interactions (diseases, drugs, and genes) by formulating the problem as a 3IAP model which they then solved by the multiple Hungarian method.

2.1.7 Applications in Algebra

3IAP has also been applied to solve systems of polynomial equations. Bekker *et al.*, (2005) employ the Combinatorial Optimisation Root Selection (CORS) method for systems with d equations ($d \leq 3$) and n solutions ($n \leq 20$). They deduce that CORS could effectively handle systems comprising up to 5 equations with as many as 30 solutions. However, they acknowledge a substantial computational burden for larger systems. Their observation indicates the potential for further research in polynomial equations with more than 30 roots, expanding the scope of 3IAP applications.

2.1.8 Dynamic Agri-robot

Recent developments have expanded the range of applications for MAP. Bertsekas (2020) studies the constrained multi-agent rollout of MAP, applying the auction algorithm to devise strategies for sub-optimally solving constrained deterministic problems. Lujak *et al.*, (2020) investigate task assignment methods for dynamic agri-robot fleets, formulating these cases as MAPs. Their research includes reviewing dynamic coordination techniques, culminating in applying these methods to the agricultural sector.

2.2 Solution Approaches

Following the definition and formulation of the problem assignment, a concise literature review was compiled, focusing on major solution approaches for the 3IAP to develop efficient algorithms for large-size instances. The review encompasses over 150 scholarly references from the last five decades, including a mix of methodological and theoretical review papers, books, and theses. Compiled and organised using MS Access and MindManager, this review facilitated an efficient and structured analysis, crucial for identifying future directions in the algorithmic resolution of large-scale instances of 3IAP.

After compiling pertinent publications, an exhaustive comparative analysis was performed, focusing on the state-of-the-art in MAP solutions. This critical review spans various methodologies, evaluating each paper's contribution and acknowledging the influence of prominent authors, thereby enriching the understanding of MAP's evolving landscape.

2.2.1 Relaxation Methods

In integer linear programming, a common technique is to relax the integer constraints, transforming the problem into a continuous linear program called *LP-relaxation*. The optimal solution of the LP-relaxation serves as a valid lower bound for the original integer problem. The LP-relaxation and its optimal solution constitute the root node for the branch and bound method (B&B), a widely used integer linear programming.

2.2.1.1 Integer Linear Programming

$$\text{Maximize} \qquad Z = \sum_{i=1}^{n} c_{ij}.x_{ij} \tag{2.1}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} a_{ij}.x_{ij} \leq b_j, \qquad j = 1, 2, \ldots, m \tag{2.1a}$$

$$x_{ij} \in \mathbb{N} \qquad i = 1, 2, \ldots, n; j = 1, 2, \ldots, m \tag{2.1b}$$

The model (2.1)–(2.1b) is an integer linear program of maximising the objective function $Z = C \cdot x$, subject to m inequalities. The variable x takes nonnegative integer values. After relaxing the constraints (2.1b) the following problem is obtained.

$$\text{Maximize} \qquad Z = \sum_{i=1}^{n} c_{ij}.x_{ij} \tag{2.2}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} a_{ij}.x_{ij} \leq b_j, \qquad j = 1, 2, \ldots, m \tag{2.2a}$$

$$x_{ij} \geq 0 \qquad i = 1, 2, \ldots, n; j = 1, 2, \ldots, m; \tag{2.2b}$$

The program (2.2)–(2.2b) represents an LP-relaxation of the previous model, with the key difference being that the variable x here takes nonnegative real values (2.2b). Typically, model (2.2)–(2.2b) is referred to the *'primal'* problem, and it has a corresponding *'dual'* problem, which is also a continuous linear program formulated as follows.

$$\text{Minimize} \qquad W = \sum_{j=1}^{m} b_j y_j \tag{2.3}$$

$$\text{subject to} \qquad \sum_{j=1}^{m} a_{ij}.y_j \geq c_{ij}, \qquad i = 1, 2, \ldots, n \tag{2.3a}$$

$$y_j \geq 0 \qquad j = 1, 2, \ldots, m \tag{2.3b}$$

The dual variables of the program (2.3)–(2.3b) directly correspond to the inequalities of the primal problem (2.2)–(2.2b). Two critical results emerge from these formulations, highlighting the fundamental relationship between a primal problem and its dual in linear programming.

Theorem 2.1. (Weak duality)

If the linear programs (2.2)–(2.2b) and (2.3)–(2.3b) have feasible solutions $\bar{x}$ and $\bar{y}$ respectively, then $Z(\bar{x}) = c.\bar{x} \leq b.\bar{y} = W(\bar{y})$.

This result provides an upper bound for the maximisation problem and a lower bound for its dual.

Theorem 2.2. (Strong duality)

If the linear programs (2.2)–(2.2b) and (2.3)–(2.3b) have finite optimal solutions $x^\star$ and $y^\star$ respectively, then $Z(x^\star) = c.x^\star = b.y^\star = W(y^\star)$.

The latter result derived from duality holds significant practical implications. It can be used as a criterion for determining the optimality of a solution. This insight is particularly valuable in linear programming, as it provides a robust method for assessing whether a proposed solution is optimal under the specified constraints and objective.

2.2.1.2 Lagrangian Relaxation

Let us consider a problem of minimising a function $f(x)$ subject to $x \in S$ and constraint $g(x) \leq 0$, where x is a vector of $\mathbb{R}^n$, S a finite subset of $\mathbb{R}^n$, f and g are two functions defined respectively from $\mathbb{R}^n$ to $\mathbb{R}$, and from $\mathbb{R}^n$ to $\mathbb{R}^m$.

$$\begin{aligned} &\text{Minimize} && f(x) && (2.4) \\ &\text{subject to} && g(x) \leq 0 && (2.4a) \\ & && x \in S && (2.4b) \end{aligned}$$

Let (P) denote the problem (2.4)–(2.4b). A vector $x \in \mathbb{R}^n$ is a feasible solution for the problem (P) if x satisfies constraints (2.4a) and (2.4b).

The inequality constraint $g(x) \leq 0$ can be associated with a nonnegative vector $\lambda \in \mathbb{R}^m$ (called: *Lagrange multiplier*), and the Lagrangian function is defined by: $\mathcal{L}(x, \lambda) = f(x) + \lambda.g(x)$. Hence the problem (P) can be formulated as:

$$\text{Minimize} \qquad \mathcal{L}(x,\lambda) \qquad (2.5)$$

$$\text{subject to} \qquad x \in S \qquad (2.5a)$$

In optimisation, relaxation simplifies problems by removing stringent constraints, leading to two key observations. First, the feasible region of the original problem (P) is a subset of the feasible region of the relaxation, due to fewer constraints (2.5a). Second, for any feasible solution x of (P), x always verifies the inequality $\mathcal{L}(x,\lambda) \leq f(x)$. Next, the dual function $\mathbb{L}$ is defined by: $\mathbb{L}(\lambda) = \underbrace{min}_{x\in S}\mathcal{L}(x,\lambda)$.

As the value of the dual function $\mathbb{L}(\lambda)$ always represents a lower bound for the problem (P), the aim is to find the best possible relaxation bound for (P). Usually the problem (P) is known as a *primal* problem while the following problem (D) is called *dual* and it satisfies:

$$\underbrace{Max}_{\lambda\in\mathbb{R}^+}\mathbb{L}(\lambda) = \underbrace{Max}_{\lambda\in\mathbb{R}^+}(\underbrace{min}_{x\in S}\mathcal{L}(x,\lambda)) \qquad (D)$$

The dual problem (D) is crucial for determining the tightest lower bound for (P) and has three important properties.

Lower bound. The weak duality principle states that for any nonnegative feasible solution λ of the dual problem (D) and any feasible solution x of the primal problem (P), if $\mathbb{L}(\lambda)$ represents the objective function value of (D), and $f(x)$ represents the objective function value of (P), then the inequality $\mathbb{L}(\lambda) \leq f(x)$ invariably holds. In other words, the value of the dual problem is always smaller than or equal to the value of the primal problem for any feasible solution. Conversely, if both solutions are equal, this signifies are optimal for their respective problems.

Concavity.

Definition 9. A subset S of $\mathbb{R}_n$ is *convex* if for any pair of points (x, y) of S, and for any $t \in [0, 1]$, the combination $tx + (1-t)y$ is also a point of S.

Definition 10. A function $f : \mathbb{R}_n \longrightarrow \mathbb{R}$ is convex if its domain is *convex*, for any pair of points (x, y) of S, and for any $t \in [0, 1]$, f verifies:

$$f(tx + (1-t)y) \leq t.f(x) + (1-t).f(y).$$

Definition 11. A function $f : \mathbb{R}_n \longrightarrow \mathbb{R}$ is *concave* if and only if the function $(-f)$ is convex.

Suppose $\mathbb{L}(\lambda)$ represents the value of the objective function of the dual problem (D) for a given λ, then $\mathbb{L}(\lambda)$ is a concave and piecewise linear function.

Sub-gradients.

Definition 12. A vector v is a *sub-gradient* of $\mathbb{L}$ at point λ. For every nonnegative number $\lambda^{'}$, we have:

$$\mathbb{L}(\lambda^{'}) \leq \mathbb{L}(\lambda) + (\lambda^{'} - \lambda).v$$

All sub-gradients of $\mathbb{L}$ at point λ form a convex set denoted $\partial\mathbb{L}(\lambda)$ and called the *sub-differential* of $\mathbb{L}$ in λ.

For every $\lambda \geq 0$, let $Y(\lambda) = \{y_i \in S | f(y_i) + \lambda.g(y_i) = \mathbb{L}(\lambda)\}$ then

1. $y_j \in Y(\lambda)$ and $g(y_j) \in \partial\mathbb{L}(\lambda)$ (*i.e.*, sub-gradient of $\mathbb{L}$ at λ).
2. For every $\lambda > 0$, $g(y_j)$ with $(y_j \in Y(\lambda))$ form a generator set of $\partial\mathbb{L}(\lambda)$ (*i.e.*, any sub-gradient is a convex linear combination of $g(y_j)$ with $(y_j \in Y(\lambda))$.

A sub-gradient method is often used in a Lagrangian relaxation to find the optimum of the dual function $\mathbb{L}$.

In addressing the 3IAP, the Lagrangian relaxation is a prevalent technique, as substantiated by various studies (Gauvrit *et al.*, 1997; Poore, 2000; 2006; Zhang, 2018). The method involves selectively incorporating one or two constraint sets from the 3IAP formulation into the objective function with corresponding multipliers, thereby generating new variants of Lagrangian relaxation. Many possible constraint combinations have been the object of extensive research on various Lagrangian relaxation heuristics, reflecting its broad adoption and effectiveness in 3IAP solutions.

In their seminal study on 3IAP, Frieze and Yadegar (1981) develop a heuristic using Lagrangian relaxation. In their methodology, they relax the final constraint set of the 3IAP formulation, a strategy that proved effective in generating a feasible solution. The Lagrangian relaxation notably facilitates the computation of both upper and lower bounds for the optimal solution, thereby allowing for a robust evaluation of solution quality.

Balas and Saltzman (1991) innovatively applied Lagrangian relaxation to the 3IAP, introducing four different variants:

- In the first variant, all constraints, barring integrity, are incorporated into the objective function. The optimal solution for fixed multipliers is determined by assigning one to each variable with a negative reduced cost and zero to others. The method focuses on recalibrating the objective function to account for constraint violations.

- In the second variant, all constraints, except integrity, are merged into the objective function, replaced by a singular constraint ensuring exactly n variables are set to one. The optimal solution is achieved by selecting the n variables with the smallest reduced costs to be one and setting the remainder to zero. This technique simplifies the constraint framework while prioritising cost optimisation.

- The third variant of Lagrangian relaxation integrated the last two equation sets (J and K) into the objective function. The optimal solution is achieved by assigning one to the variable with the smallest reduced cost for every i of set I.

- The final variant incorporates the first equation set associated with I into the objective function. The relaxed problem is reached by solving an assignment problem across sets J and K, assigning costs based on the minimum reduced cost for each pair (j, k) determined from all triplets containing (j, k) across set I.

Li *et al.*, (2019) explore a novel Lagrangian method for the 3IAP, relaxing three constraint sets. Their approach established a necessary and sufficient condition for achieving a zero-duality gap. The authors claim their approach could reach an optimal solution to the assignment problem with certainty. The method warrants detailed examination for its potential as a significant source of inspiration in optimisation problems.

Developed initially for the 3IAP, certain techniques are later adapted for the Multidimensional Assignment Problem (MAP). Poore and Rijavec (1994), followed by Poore and Robertson (1997), introduce a new class of constructive Lagrangian relaxation algorithms, in solving assignment problems. The authors tackle the d-dimensional assignment problem through a two-phase approach. Initially, they relax the problem, then maximise the related Lagrangian multipliers. The process reduces the problem to a $(d-1)$-dimensional assignment problem. The algorithm iteratively repeats this procedure, progressively decreasing the dimension until it simplifies the case to a two-dimensional assignment problem, *i.e.*, LAP. In the second phase, the authors implement a '*recovery procedure*' to construct a good solution for the original MAP from the optimal solution of the relaxed problems. Their approach, adept at procuring near-optimal solutions for target tracking problems, is expanded to encompass data association across multiple frames (Poore, 2000). Further methods are described in (Poore and Gadaleta, 2006).

2.2.2 Exact Algorithms

In combinatorial optimisation, exact algorithms guarantee optimal solutions with proven optimality. However, their execution times significantly increase as the problem size grows, limiting their practicality to small or medium-sized instances. For larger size instances, where optimality must be balanced with runtime, the reliance shifts towards heuristic methods. Heuristics are preferred for NP-hard problems because they can produce good solutions within reasonable timeframes.

There is a limited literature covering exact algorithms for MAP. Some of these methods prioritise optimal solutions in reduced time but they inherently entail exponential computational complexity, assuming $P \neq NP$, as described in (Garey and Johnson, 1979). Notably, among exponential algorithms, preference leans towards lower complexity, for example, $O(2^n) < O(3^n) < O(n!)$.

Most exact solution methods for MAP are initially designed for solving its three-dimensional version. One of the earliest exact algorithms for MAP was introduced by (Pierskalla,1968). The method implicitly enumerates all feasible solutions via a tree structure. The algorithm applies the (B&B) technique and utilises dual subproblems to generate lower bounds that are straightforward to compute.

Hansen and Kaufman (1973) developed a primal-dual algorithm for solving the 3IAP, resembling the Hungarian method. Later, Fröhlich (1979) implemented a (B&B) approach, incorporating Lagrangian relaxation and sub-gradient optimisation. This method was briefly documented in (Burkard, 1980).

Following Pierskalla's work, numerous studies in the literature referred to algorithms using the (B&B) technique. These algorithms typically employ a branching rule that sets single variables to 0 or 1. However, Balas and Saltzman (1991) introduce a new branching strategy, fixing several variables simultaneously at each branching node. They then integrate facet inequalities at each node, yielding in smaller search trees and terminal nodes being LAPs. Balas and Saltzman (1991) have successfully solved the 3IAP to optimality and tested their algorithm on problem instances up to size $n = 26$.

Burkard and Rudolf (1993) investigate various (B&B) schemes for 3IAP. They experiment with several branching rules, drawing upon insights from (Balas and Saltzman, 1989) research on the facial structure of the three-index assignment polytope. Later, Pasilioa *et al.*, (2005) develop two distinct (B&B) methods for solving the MAP, each based on different tree representations. The authors pointed that their algorithm performance is affected by the size and dimension of the problem.

2.2.3 Approximation Algorithms

While approximation algorithms are tailored to specific geometric forms of versions of 3IAP, they have been widely studied, leading to the development of numerous heuristics. In this context, Spieksma (2000) identifies two fundamental concepts that helped comprehend the nature of approximation algorithms.

1. A $\rho-$*approximation* algorithm for a minimisation (maximisation) problem P is a polynomial-time algorithm that, for all instances, generates a solution with a value at most (at least) respectively, equal to ρ times the value of the optimal solution of P. Note that for minimisation problems $\rho \geq 1$, and maximisation problems $0 \leq \rho \leq 1$.

2. A *polynomial time approximation* scheme for a minimisation (maximisation) problem is a family of polynomial-time $(1+\varepsilon)-$*approximation* algorithms ($(1-\varepsilon)-$*approximation* algorithms) respectively, for any $\varepsilon > 0$.

Crama and Speiksma (1992) explore special cases of the 3IAP formulated in terms of graph theory, particularly the case where edge lengths satisfy the triangular inequality. They consider the costs of a triangle (3-clique) as either the sum of all edge lengths (perimeter $T\Delta$) or the sum of the two shortest edges ($S\Delta$). They demonstrate that both 3IAP variants are *NP*-hard. Nevertheless, they devise heuristics that consistently yield feasible solutions, with the objective function values not exceeding 3/2 (for $T\Delta$) or 4/3 (for $S\Delta$) of the optimal value. Their empirical experiments on randomly generated instances confirmed the high performance of these heuristics.

Later, Bandelt *et al.*, (1994) also consider special cases of the MAP formulated in terms of graph theory. The function cost in these cases is not arbitrary, but is based on elementary costs defined on the edges of the underlying multi-partite graph of the assignment problem. The authors define the cost of a clique (*i.e.*, the sum of the lengths of all edges within the clique). In addition, they consider a tour cost (the minimum cost of a traveling salesman tour within the clique), a tree cost (*i.e.*, the minimum cost of a spanning tree in the clique), and a star cost (*i.e.*, minimum length of a spanning star in the clique). They determine worst-case bounds on the ratio between the cost of solutions derived from simple heuristics and optimal solutions for these special cases.

Despite the *NP*-hard nature of the 3IAP, there are a few special cases that can be solved using polynomial-time algorithms. Burkard, Klinz, and Rudolf (1996) identify such cases when cost coefficients form a 'Monge' array, where for each fixed index i (and similarly for each j or k), we satisfy the following inequality:

$$c_{ilm} + c_{ipq} \leq c_{ilq} + c_{ipm} \quad \text{for} \quad 1 \leq l \leq p \leq n, 1 \leq m \leq q \leq n.$$

However, Burkard, Rudolf, and Woeginger (1996) examine the case of the assignment problem with decomposable costs:
$c_{ijk} = \alpha_i.\beta_j.\gamma_k$ for $i, j, k = 1, 2, \ldots, n$; and $\alpha_i \geq 0, \beta_j \geq 0, \gamma_k \geq 0$ for $i, j, k = 1, 2, \ldots, n$.

The authors confirm that while the minimisation version of the assignment problem with decomposable costs is NP-hard, its maximisation counterpart can be solved in polynomial time. They also identify additional cases of the 3IAP with decomposable costs that are solvable in polynomial time under tightly restricted structures.

In the same year, Spieksma and Woeginger (1996) investigate two additional geometric special cases of the 3IAP. Their study involves three sets I, J, and K, each containing n points in the Euclidean plane. The objective is to partition $I \cup J \cup K$ into n tri-coloured triangles, minimising either the total perimeter or the total area of all the triangles. Both variants of this problem are proven to be NP-hard.

Later, Johnsson *et al.*, (1998) explore the three-matching problem in the Euclidean plane and identified it as a variant of the geometric case of the 3IAP. Assuming the problem size equals n a multiple of three (*i.e.*, $n = 3p$), the authors searched, in the Euclidean plane, for a partition of the n points into p subsets. Their study revolves around subsets of three points or triplets (i, j, k). The cost for each triple is defined by the sum of the lengths of the two shortest sides of the corresponding triangle, and the objective is to minimise the total cost of all triplets. Initially, the authors attempted to devise approximation algorithms for this problem but encountered challenges finding a definitive upper bound for computational complexity. However, they developed alternative heuristics based on tabu-search, simulated annealing, and genetic algorithms.

A decade later, Kuroki and Matsui (2009) investigate geometric special cases of the MAP, focusing on the d-dimensional assignment problem. The problem entails partitioning n nodes of a complete d-partite graph into n disjoint d-cliques to minimise their total weights. The weight of each clique is the sum of the weights of its edges, where the weight of an edge was defined as the square of the Euclidean distance between its endpoints. The authors devise a polynomial approximation algorithm, which improves the known approximation ratio from $(4 - \frac{6}{d})$ to $(\frac{5}{2} - \frac{3}{d})$.

2.2.4 Polyhedral Approach

The polyhedral approach in the axial 3IAP has been widely studied. Notably, Euler (1987) is the first to analyse the structure of the convex hull of feasible solutions for the 3IAP and the role of odd cycles in a class of facets of the polyhedron.

Balas and Saltzman (1989) extensively investigate the facial structure of the three-index assignment polytope. They identify several facet classes and developed an $O(n^4)$ separation algorithm to detect violations in clique facets. This brilliant seminal research inspired numerous researchers. Continuing their exploration, Balas and Saltzman (1991) develop an exact algorithm to solve the 3IAP, integrating a class of facet inequalities into a Lagrangian relaxation approach. They successfully implement their algorithm on problem instances up to $n = 26$, marking a significant achievement in the field.

Following their earlier work, Balas and Qi (1993) develop an efficient $O(n^3)$ separation algorithm for certain facet classes of the 3IAP, including a newly identified class. Given that the 3IAP requires n^3 variables, their algorithm, from a computational complexity perspective, qualifies as linear time; at that point, it was the most efficient. Later, Qi, Balas, and Gwan (1994) advance their work and extend linear time an $O(n^3)$ separation algorithms to additional facet classes, specifically those induced by certain cliques. They then propose a polyhedral approach procedure to solve the axial 3IAP effectively.

Qi and Sun (2000) present an overview summarising the significant advancements in the polyhedral structure analysis of the three-index assignment polytope. They detail various facet classes and propose corresponding separation algorithms. In conclusion, Qi and Sun raise an intriguing question: Do facets exist where the right-hand side of their defining inequalities equals 2?

Magos and Mourtos (2009) present a unified framework for analysing clique facets in axial and planar assignment polytopes. They develop a polyhedral approach-based solution method for the 3IAP, integrating a polynomial-time separation procedure with a class of clique facets from the three-index assignment polytope. This approach remarkably reduces the solution time for 3IAP instances. While some clique facets were previously identified by (Balas and Saltzman, 1989), Magos and Mourtos add improvements to their algorithm.

After reviewing the existing facet classes of the 3IAP, Dokka and Speiksma (2014) identify a new class of facets. Additionally, they affirmatively respond to Qi and Sun's 2000 query by showing that these new facets' defining inequalities have a right-hand side of 2, thereby enriching the polyhedral understanding of the 3IAP.

Kravtsov (2006) investigates the three-index assignment polytope, analysing its non-integer vertices' structure. The author then characterises these vertices and examines their combinatorial properties, enriching the study of the polytope.

Before concluding this section, it is pertinent to highlight the work on multi-index assignment polytopes of higher dimensions. Appa *et al.*, (2004) study the four-index assignment polytope, introducing a branch and cut algorithm for its effective resolution.

Later, Appa *et al.*, (2006) expand the study to the polytope of the MAP in a general context. Their work unifies and generalises earlier results; in particular, they set the dimensions of both axial and planar MAP polytopes. Furthermore, they identify a novel class of clique facets for the axial MAP polytopes and establish a necessary condition for the existence of MAP solutions.

2.2.5 Local search methods

Local search methods are important in discrete optimisation, serving as heuristics for tackling NP-hard combinatorial optimisation problems. Starting from a feasible solution, these methods proceed to a neighbouring solution through incremental local modifications. The process iterates, moving to a neighbouring solution if a potential improvement is achievable otherwise it terminates after a prefixed number of iterations. This iterative approach persists until an optimal solution is found or the fixed iteration limit is reached.

In optimisation neighbourhood-centred methods operate iteratively by exploring the neighbourhoods of the current solution. Various approaches in the literature address diverse neighbourhood classes and sizes. The key principle of these methods is the systematic evaluation of neighbourhoods in a predetermined manner.

Karapetyan and Gutin (2011) distinguish between heuristics construction and local search, noting that the former develops solutions from the ground up with quality constraints, while the latter improves existing solutions, and served as useful post-construction or as part of advanced metaheuristics.

Among the early local search methods for the 3IAP is the variable depth interchange heuristic developed by (Balas and Saltzman, 1991). The method begins with a feasible solution, methodically evaluates all variable interchanges against the objective function, and executes changes that improve cost, repeating the process until no further improvements are possible.

Balas and Saltzman's paper provides detailed analysis and testing of three heuristics (greedy, reduced cost, and max-regret) on randomly generated instances sized $n = 20$ to 70. The results highlight the max-regret method's superiority and its effective combination with variable depth interchange for large 3IAP instances.

Robertson (2001) introduces four greedy randomised adaptive search procedures (GRASP) for MAP, inspired by Balas and Saltzman's 1991 heuristics. These include two constructive methods (randomised reduced cost greedy, max regret) and two local searches (two-assignment-exchange, variable depth exchange), tested on five-index assignment problems ($n = 25$) related to multi-sensor tracking systems, with detailed explanations but no performance metrics.

Bandelt *et al.*, (2002) investigate decomposable cost MAPs, introducing an exponentially sized neighbourhood with polynomial-time optimal solution identification, and develop a corresponding local search method. Their algorithm was empirically tested with constructive heuristics, establishing lower bounds for solution quality assessment. In contrast, Grundel *et al.*, (2004) analyse the asymptotic properties of random MAP instance families.

A few years later, Aiex *et al.*, (2005) introduce GRASP variations incorporating path relinking for the 3IAP, merging a greedy randomised construction process and local search. This approach is tested on established problem instances, and demonstrated enhanced solution quality compared to earlier methods (Balas and Saltzman, 1991), (Burkard *et al.*, 1996), and (Crama and Spieksma, 1992).

2.2.6 Genetic Algorithms

The Genetic Algorithm (GA), inspired by biological evolution and natural selection, is an approximate approach to solving complex optimisation problems. Applied to the 3IAP, GA evolves a population of potential solutions across multiple generations. Combining a local search method with GA leads to enhanced solution quality.

Huang and Lim (2006) develop a new iterative local search procedure for the 3IAP, reducing the problem into a LAP. They then integrate this heuristic within a genetic algorithm; they are the first to use the genetic algorithm to solve 3IAP. Chapter 5 is exploring this method in more detail. Gutin and Karapetyan (2009) introduce a '*memetic*' hybrid algorithm for MAP, merging a genetic algorithm with a local search. Their paper describes the method without numerical results.

Later, Karapetyan and Gutin (2011a) publish on MAP heuristics, expanding existing neighbourhoods and introducing new ones. They rigorously evaluate their local search methods theoretically and empirically then devise an efficient heuristic by combining various neighbourhoods to capitalise on their respective strengths.

Karapetyan and Gutin (2011a) propose new instance families for evaluating

MAP heuristics, with tests indicating a relative error of around 5% in most instances. They conclude that dimensionwise variation heuristics were superior for MAP. In a subsequent 2011b publication, they develop a memetic algorithm for 3AIP using an adjustable population size, similar to Huang and Lim (2006) approach. The size is determined by the full algorithm run-time and the average local search computational time for each specific instance.

Kammerdiner (2017) propose a large-scale neighbourhood search for MAP, while Valencia *et al.*, (2017) recognise Karapetyan and Gutin's (2011b) memetic algorithm as the best for MAP. They show a basic genetic algorithm with dimensionwise variation heuristic to be efficient against advanced memetic algorithms. Pérez (2017) extensively explores MAP, devising a new metaheuristic merging a local search with evolutionary algorithms called MAP_Gurobi, for solving 3IAP large instances. Later, Pal *et al.*, (2018) introduces a MAP heuristic combining genetic algorithms with particle swarm optimisation, albeit with limited numerical testing.

Vadrevu and Nagi (2020) propose a dual-ascent algorithm for MAP, and apply it to multi-target tracking. Concurrently, Sumathi *et al.*, (2020) develop a Lexisearch for 3IAP, finding feasible solutions through intensive search. Medvedev and Medvedeva (2019) propose a probabilistic approach for axial 3IAP, enhancing greedy algorithms with variable randomization. Meanwhile, Sumathi *et al.*, (2020) introduce a robust Bayesian multi-object tracking algorithm for data association. Chen *et al.*, (2023) propose a clustering-based approach for distributed multi-object tracking systems in environments with restricted sensor visibility. Their algorithm integrates kinematic fusion with a specific weighted graph fusion technique.

2.2.7 Other methods

Probabilistic approaches were also proposed for solving the axial 3IAP. Medvedev and Medvedeva (2019), rather than optimise the objective function, minimised its expectation. On the other hand, Afraimovich and Emelin (2021) combine a pair of feasible solutions attempting to find an optimal solution. If the induced graph is disconnected, their algorithm can find a better solution when it exists; otherwise, there is no improvement. This algorithm can be applied as a supplement to heuristics for post-processing the 3IAP approximate solutions. Then, Afraimovich and Emelin (2022) discuss the computational complexity of such combinations.

The current project's contribution lies in its original algorithmic approach. This study presents two rapid heuristic classes for obtaining high-quality approximate solutions, supported by comprehensive numerical experiments, followed by implementing two efficient methods for solving 3IAP optimally. Furthermore, a new hybrid genetic algorithm is developed to address the 3IAP, effectively producing good, feasible solutions within reasonable computational timeframes.

Chapter 3

Proposed Heuristic Solutions for 3IAP

In the previous chapter, the three-index assignment problem was formulated as an integer linear program whose constraint matrix A_n has a particular structure with all coefficients in $\{0, 1\}$. Furthermore, all 3IAPs of size n share the same constraint matrix A_n; two distinct 3IAPs differ only in their objective functions. This chapter investigates the attributes of the objective function, seeking to uncover potential avenues for developing innovative solution approaches for the 3IAP.

The current chapter introduces two novel heuristics for solving the 3IAP. Initially, an efficient algorithm is designed to attain feasible solutions rapidly. Subsequently, two variants are proposed wherein cost matrices are sorted based on their diagonals before applying the proposed procedure.

3.1 Greedy-Style Procedure

The Greedy-Style Procedure (GSP), introduced for the first time, is a heuristic method for 3IAP, consistently yielding feasible solutions. The operation mode of

GSP reminds us of the Greedy algorithm (Kruskal, 1956) for the minimum spanning tree of a graph. GSP plays a pivotal role in this chapter.

Let $\mathcal{M}^{n^2}$ denote the set of all $n \times n$–matrices. Let $I = J = K = \{1, 2, \dots, n\}$ be, for convenience, three sets of n elements, assuming $n \geq 3$. Clearly, the 3IAP objective function Z can be represented by a sequence of n elements of $\mathcal{M}^{n^2}$ each of which is associated with a factory, as Table 3.1 shows. Coefficient c_{ijk} for $\forall i, j, k = 1, 2, \dots, n$ is positive real number evaluating the cost of assigning job i to machine j in factory k.

Table 3.1: Example of 3IAP of size n.

Factories		F_1				F_2				...	F_n			
Machines		m_1	m_2	...	m_n	m_1	m_2	...	m_n	...	m_1	m_2	...	m_n
	j_1	c_{111}	c_{121}	...	c_{1n1}	c_{112}	c_{122}	...	c_{1n2}	...	c_{11n}	c_{12n}	...	c_{1nn}
Jobs	j_2	c_{211}	c_{221}	...	c_{2n1}	c_{212}	c_{222}	...	c_{2n2}	...	c_{21n}	c_{22n}	...	c_{2nn}
	...		...		...		...		...	...		...		...
	j_n	c_{n11}	c_{n21}	...	c_{nn1}	c_{n12}	c_{n22}	...	c_{nn2}	...	c_{n1n}	c_{n2n}	...	c_{nnn}

The GSP heuristic sequentially processes matrices, starting with C_1, the first matrix in the list, applying the Hungarian algorithm to solve the corresponding LAP. GSP selects the minimal cost c_{pq1}, after Hungarian method application, then excising row p and column q from other matrices and removing C_1 from the sequence.

The GSP systematically reduces the size of the 3IAP by one unit after each iteration. The process continues until only two (2×2)–matrices remain, as illustrated in Table 3.2, which are easy to solve. A feasible solution for the original assignment problem arises by summing all selected minima and retrieving their corresponding indices.

3.1.1 GSP Algorithm

Below there is the sequence of steps of the algorithm.

Table 3.2: Example of 3IAP of size 2.

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}

Algorithm 1: GREEDY-STYLE PROCEDURE - GSP

```
Input: A sequence of cost matrices $\langle C_1, C_2, \ldots, C_n \rangle$
Output: A feasible solution for 3IAP.
1  for $k \leftarrow n$ to 2 do
2      while $k \geq 3$ do
3          Apply Hungarian method to $C_k$.
4          From the Hungarian method solution, select the minimum cost, let
             say $c_{pqk}$.
5          Store $c_{pqk}$ and remove matrix $C_k$.
6          Delete row $p$ and column $q$ from all other matrices.
7          $C_{k-1} \leftarrow C_k$
8      end while
9      if $k = 2$ then
10         Solve the last 3IAP of size 2;
11     end if
12     Retrieve all previous allocations.
13     Sum all selected costs.
14 end for
15 return Solution for 3IAP.
```

Theorem 13. *Given an instance of 3IAP of size $n \geq 2$, GSP always returns a feasible solution for the problem, in polynomial time.*

Proof. If $n = 2$ obvious, see Example 1. Assuming $n \geq 3$, at each iteration $k = n, n-1, \ldots, 3$; GSP applies the Hungarian method to the first matrix in the chosen order. From one iteration to the next, GSP reduces the 3IAP size by one unit and the Hungarian method executes $O(n^3)$ operations. As the procedure runs over $n-2$ iterations, the overall computational complexity is at most $O(n^4)$, therefore it is polynomial time. □

Example 1: 3IAP of size 2.

$$C_1 = \begin{array}{c} \\ j_1 \\ j_2 \end{array} \begin{array}{c} \begin{array}{cc} m_1 & m_2 \end{array} \\ \begin{pmatrix} c_{111} & c_{121} \\ c_{211} & c_{221} \end{pmatrix} \end{array} \quad \text{and} \quad C_2 = \begin{array}{c} \\ j_1 \\ j_2 \end{array} \begin{array}{c} \begin{array}{cc} m_1 & m_2 \end{array} \\ \begin{pmatrix} c_{112} & c_{122} \\ c_{212} & c_{222} \end{pmatrix} \end{array}$$

Let C_1 and C_2 be two matrices associated respectively with factory 1 and factory 2; since a job in each factory is performed by one machine, the problem admits four possible solutions from which the optimal solution derives straightforwardly, of cost equal to $min\{c_{111}+c_{222},\ c_{122}+c_{211},\ c_{121}+c_{212},\ c_{112}+c_{221}\}$.

3.1.2 GSP Illustration

Example 2: 3IAP of size 4.

Let C_1, C_2, C_3 and C_4 be cost matrices associated with four factories. Rows correspond to jobs and columns to machines.

$$\mathrm{C}_1 = \begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline j_1 & 87 & 59 & 83 & 41 \\ j_2 & 09 & 47 & 26 & 59 \\ j_3 & 20 & 88 & 95 & 10 \\ j_4 & 53 & 26 & 14 & 75 \end{array} \qquad \mathrm{C}_2 = \begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline j_1 & 66 & 45 & 09 & 51 \\ j_2 & 15 & 69 & 100 & 58 \\ j_3 & 06 & 84 & 60 & 41 \\ j_4 & 13 & 12 & 07 & 05 \end{array}$$

$$\mathrm{C}_3 = \begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline j_1 & 17 & 46 & 34 & 99 \\ j_2 & 20 & 62 & 89 & 04 \\ j_3 & 27 & 72 & 04 & 100 \\ j_4 & 45 & 52 & 67 & 94 \end{array} \qquad \mathrm{C}_4 = \begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline j_1 & 96 & 33 & 42 & 08 \\ j_2 & 16 & 55 & 99 & 63 \\ j_3 & 72 & 60 & 70 & 28 \\ j_4 & 36 & 96 & 63 & 24 \end{array}$$

Clearly the problem requires 64 variables and has up to 576 feasible solutions. If diagonal elements $x_{iii} = 1$, for $\forall i = 1, 2, 3, 4$ and 0 otherwise, it has a trivial solution, and the value of the objective function $Z = 87 + 69 + 4 + 24 = 184$ might constitute an upper bound for the 3IAP optimal solution. For any instance of size n, the restriction of the problem to one factory, the allocation of n tasks to n machines within that factory, reduces it to a two-dimensional assignment problem, solvable in polynomial time.

Let us apply Algorithm 1 to the above example.

Iteration 1. $k = 4$.

Starting with C_1 the first matrix, the Hungarian method produces an optimal solution S_1 to LAP related to C_1, as shown below.

$$\mathrm{C}_1 = \begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline j_1 & 87 & 59 & 83 & 41 \\ j_2 & 09 & 47 & 26 & 59 \\ j_3 & 20 & 88 & 95 & 10 \\ j_4 & 53 & 26 & 14 & 75 \end{array} \Rightarrow \mathrm{S}_1 = \begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline j_1 & - & 59 & - & - \\ j_2 & (9) & - & - & - \\ j_3 & - & - & - & 10 \\ j_4 & - & - & 14 & - \end{array}$$

S_1 represents the Hungarian method solution for C_1.

Step 4: The solution S_1, comprises four costs, one per column and one per row, among which choose the minimum cost, 9 corresponding to the coefficient c_{211}. Therefore, job 2 will be assigned to the machine 1 in factory 1.

Step 5 and 6: Next, delete row 2 and column 1 from matrices C_2, C_3 and C_4, then remove C_1. Hence three submatrices are obtained.

$$C'_2 = \begin{array}{c} \\ j_1 \\ j_3 \\ j_4 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 45 & 09 & 51 \\ 84 & 60 & 41 \\ 12 & 07 & 05 \end{pmatrix} \end{array} \quad C'_3 = \begin{array}{c} \\ j_1 \\ j_3 \\ j_4 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 46 & 34 & 99 \\ 72 & 04 & 100 \\ 52 & 67 & 94 \end{pmatrix} \end{array} \quad C'_4 = \begin{array}{c} \\ j_1 \\ j_3 \\ j_4 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 33 & 42 & 08 \\ 60 & 70 & 28 \\ 96 & 63 & 24 \end{pmatrix} \end{array}$$

Iteration 2. $k = 3$.

Step 3. Applying the Hungarian method to matrix C'_2.

Step 4 and 5. This results in the LAP optimal assignment associated with factory 2, whose lowest value equals 9, the cost of assigning job 1 to machine 3 in factory 2.

$$S_2 = \begin{array}{c} \\ j_1 \\ j_3 \\ j_4 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} - & (9) & - \\ - & - & 41 \\ 12 & - & - \end{pmatrix} \end{array}$$

The Hungarian method solution S_2.

Step 6. Next, delete the first row and the middle column from submatrices C'_3 and C'_4, then eliminate C'_2. Two new submatrices are obtained.

Step 8 and 9. The two submatrices are.

$$C''_3 = \begin{array}{c} \\ j_3 \\ j_4 \end{array}\!\!\begin{array}{c} \begin{array}{cc} m_2 & m_4 \end{array} \\ \begin{pmatrix} 72 & 100 \\ (52) & 94 \end{pmatrix} \end{array} \quad \text{and} \quad C''_4 = \begin{array}{c} \\ j_3 \\ j_4 \end{array}\!\!\begin{array}{c} \begin{array}{cc} m_2 & m_4 \end{array} \\ \begin{pmatrix} 60 & (28) \\ 96 & 24 \end{pmatrix} \end{array}$$

Minimum assignment related to C''_3 and C''_4.

The minimum cost 80 is attained when job 3 is assigned to machine 4 in factory 4, and job 4 is assigned to machine 2 in factory 3.

Step 10 and 11. A feasible solution is found for the original 3IAP when all variables $x_{ijk} = 0$, except $x_{132} = x_{211} = x_{344} = x_{423} = 1$, with a total cost 98.

$$\begin{array}{c|cccc|cccc|cccc|cccc|}
 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 \\
j_1 & - & - & - & - & - & - & 9 & - & - & - & - & - & - & - & - & - \\
j_2 & 9 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
j_3 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 28 \\
j_4 & - & - & - & - & - & - & - & - & - & 52 & - & - & - & - & - & -
\end{array}$$

The GSP solution for the 3IAP.

Cost matrices are endowed with a particular data structure, and the GSP heuristic processes them following their entry sequence by the FIFO principle. Reversing the sequence in which cost matrices are processed, could really impact solution quality. Applying the GSP heuristic to the same matrices in reverse order, thereby implementing the LIFO principle, could produce varied results.

Reversing the order of the 3IAP matrices and applying the GSP heuristic can yield an alternative solution distinct from the initial one. Thus, it is advisable to experiment with both sequences, as one might be better than the other. The inquiry examines the effects of modifying matrices' order before applying GSP, which could result in improved solutions.

Let us apply Algorithm 1 to the same example but with the cost matrices arranged in LIFO order. Commencing with matrix C_4, the Hungarian method produces an optimal solution S_4 to LAP related to C_4, as illustrated below.

Iteration 1. $k = 4$.

$$\mathrm{C}_4 = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 96 & 33 & 42 & 08 \\ 16 & 55 & 99 & 63 \\ 72 & 60 & 70 & 28 \\ 36 & 96 & 63 & 24 \end{pmatrix} \end{array} \Rightarrow \mathrm{S}_4 = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} - & 33 & - & - \\ (16) & - & - & - \\ - & - & - & 28 \\ - & - & 63 & - \end{pmatrix} \end{array}$$

S_4 represents the Hungarian method solution for C_4.

Step 4: From the solution S_4, select the minimum cost, 16 corresponding to the coefficient c_{214}. Thus job 2 will be assigned to machine 1 in factory 4.

Step 5 and 6: Next, delete the second row and first column from matrices C_3, C_2 and C_1, then remove C_4. Thus three submatrices C_3', C_2' and C_1' result, and the procedure continues until the last iteration.

When Algorithm 1 terminates, a feasible solution is found for the original 3IAP where all variables $x_{ijk} = 0$, except $x_{114} = x_{214} = x_{323} = x_{432} = 1$, with a total cost of 73.

$$\begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \begin{array}{c} \begin{array}{cccc|cccc|cccc|cccc} m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 \end{array} \\ \left(\begin{array}{cccc|cccc|cccc|cccc} - & - & - & 41 & - & - & - & - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - & - & - & - & 16 & - & - & - \\ - & - & - & - & - & - & - & - & - & 4 & - & - & - & - & - & - \\ - & - & - & - & - & - & 12 & - & - & - & - & - & - & - & - & - \end{array}\right) \end{array}$$

Another solution for the 3IAP provided by GSP.

The GSP procedure provides two possible solutions for the 3IAP, contingent upon the cost matrices' order. While the second result may not surpass the quality of the first, it remains a viable alternative and the best solution should be selected after testing both.

The next section will explore the effect of matrix order alteration on the quality of solutions generated by the GSP procedure.

3.2 Diagonals Method

Kadhem (2017) studied the 3IAP and introduced an alternative way of sorting cost matrices called the Diagonals Method (DM). The approach organises the cost matrices in ascending or descending order and the DM heuristic often yields at least two solutions for the 3IAP.

Definition 14. Let matrix A be an element of $\mathcal{M}^{n^2}$. The sum of the diagonal (anti-diagonal) coefficients of A is called the '**trace**' ('**anti-trace**' respectively) of A, denoted $tr(A)$ ($atr(A)$ respectively), *i.e.*, $tr(A) = \sum_{i=1}^{n} a_{ii}$ and $atr(A) = \sum_{i=1}^{n} a_{i(n+1-i)}$.

Definition 15. Let matrix A be an element of $\mathcal{M}^{n^2}$. The '**diagonal factor**' of A, denoted $df(A)$, is the maximum of the trace and anti-trace of A , *i.e.* $df(A) = Max(tr(A), atr(A))$.

The diagonal factor of a matrix A, is a new concept introduced for the first time. The diagonal factor plays a pivotal role in the DM heuristic, since it serves to arrange the cost matrices.

The DM heuristic is designed to solve the 3IAP, and has two phases. The initial phase involves sorting the cost matrices in ascending or descending order, followed by a second which invokes the GSP procedure.

3.2.1 DM Illustration

Example 3: Let us re-examine the previous case of 3IAP of size 4.

Let C_1, C_2, C_3 and C_4 be matrices of $\mathcal{M}^{4^2}$ associated with four factories.

$$\mathrm{C}_1 = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \!\!\begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 87 & 59 & 83 & 41 \\ 09 & 47 & 26 & 59 \\ 20 & 88 & 95 & 10 \\ 53 & 26 & 14 & 75 \end{pmatrix} \end{array} \quad \mathrm{C}_2 = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \!\!\begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 66 & 45 & 09 & 51 \\ 15 & 69 & 100 & 58 \\ 06 & 84 & 60 & 41 \\ 13 & 12 & 07 & 05 \end{pmatrix} \end{array}$$

$$\mathrm{C}_3 = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \!\!\begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 17 & 46 & 34 & 99 \\ 20 & 62 & 89 & 04 \\ 27 & 72 & 04 & 100 \\ 45 & 52 & 67 & 94 \end{pmatrix} \end{array} \quad \mathrm{C}_4 = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \!\!\begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 96 & 33 & 42 & 08 \\ 16 & 55 & 99 & 63 \\ 72 & 60 & 70 & 28 \\ 36 & 96 & 63 & 24 \end{pmatrix} \end{array}$$

Phase 1.

Compute the diagonal factors of the matrices.
$df(C_1) = 304$, $df(C_2) = 248$, $df(C_3) = 305$ and $df(C_4) = 245$.
The matrices arranged in ascending order of their diagonal factors, are C_4, C_2, C_1, and C_3.

Phase 2.

Iteration 1. $k = 4$.

Step 3. Starting with C_4 the first matrix, the Hungarian method produces an optimal solution S_4 to LAP corresponding to C_4, as shown below.

$$\mathrm{C}_4 = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \!\!\begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 96 & 33 & 42 & 08 \\ 16 & 55 & 99 & 63 \\ 72 & 60 & 70 & 28 \\ 36 & 96 & 63 & 24 \end{pmatrix} \end{array} \Rightarrow \mathrm{S}_4 = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \!\!\begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} - & 33 & - & - \\ (16) & - & - & - \\ - & - & - & 28 \\ - & - & 63 & - \end{pmatrix} \end{array}$$

S_4 represents the Hungarian method solution for C_4.

Step 4: From the solution S_4, select 16 as the minimum cost, corresponding to the coefficient c_{214}. Thus job 2 will be assigned to machine 1 in factory 4.

Step 5 and 6: Next, delete row 2 and column 1 from matrices C_2, C_1 and C_3, then remove C_4. Then three submatrices C_2', C_1' and C_3' are obtained, and the algorithm continues.

Hence, a feasible solution is achieved for the original 3IAP where all variables are $x_{ijk} = 0$, except $x_{132} = x_{214} = x_{341} = x_{423} = 1$, with a total cost of 87.

$$\begin{array}{c|cccc|cccc|cccc|cccc|}
 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 \\
j_1 & - & - & - & - & - & - & 9 & - & - & - & - & - & - & - & - & - \\
j_2 & - & - & - & - & - & - & - & - & - & - & - & - & 16 & - & - & - \\
j_3 & - & - & - & 10 & - & - & - & - & - & - & - & - & - & - & - & - \\
j_4 & - & - & - & - & - & - & - & - & - & 52 & - & - & - & - & - & -
\end{array}$$

The DM solution for the 3IAP.

Adopting a methodology similar to FIFO and LIFO rules, an alternative solution is anticipated upon reversing the matrices' arrangement or sorting them in descending order of their diagonal factors. In the following, the problem will be re-addressed with cost matrices sorted in descending order according to their diagonal factors, *i.e.*, C_3, C_1, C_2 and C_4.

The Hungarian method applied to matrix C_3 yields optimal solution S_3, highlighting a duplicated minimum cost of 4. The case will be discussed later in further detail. In this iteration, the first coefficient c_{243}, the cost for allocating job 2 to machine 4 in factory 3, is selected as shown below.

$$\mathrm{C}_3 = \begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline j_1 & 17 & 46 & 34 & 99 \\ j_2 & 20 & 62 & 89 & 04 \\ j_3 & 27 & 72 & 04 & 100 \\ j_4 & 45 & 52 & 67 & 94 \end{array} \Rightarrow \mathrm{S}_3 = \begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline j_1 & 17 & - & - & - \\ j_2 & - & - & - & (4) \\ j_3 & - & - & 4 & - \\ j_4 & - & 52 & - & - \end{array}$$

The Hungarian method solution S_3.

Step 5 and 6. Delete matrix C_3, then eliminate row 2 and column 4 from all other matrices, and continue executing the DM algorithm until only two submatrices remain.

Therefore, a second feasible solution is yielded for the original 3IAP when all variables $x_{ijk} = 0$, except $x_{124} = x_{243} = x_{312} = x_{431} = 1$, with a total cost of 57, as displayed below.

$$\begin{array}{c|cccc|cccc|cccc|cccc} & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 \\ \hline j_1 & - & - & - & - & - & - & - & - & - & - & - & - & - & 33 & - & - \\ j_2 & - & - & - & - & - & - & - & - & - & - & - & 4 & - & - & - & - \\ j_3 & - & - & - & - & 6 & - & - & - & - & - & - & - & - & - & - & - \\ j_4 & - & - & 14 & - & - & - & - & - & - & - & - & - & - & - & - & - \end{array}$$

A second DM solution for the same 3IAP.

DM also provides at least two feasible solutions for 3IAP based on the order of the cost matrices. The preferable outcome for the assignment problem is the one with the lowest cost.

3.2.2 DM and Tie-cases.

Definition 16. At step 4 of Algorithm 1, if two or more coefficients are equal to the minimum cost, the situation is called a '***tie-case***'.

A tie-case emerges when the first iteration of the DM algorithm is applied to the previous example, with cost matrices sorted in descending order of their diagonal factors. If the second cost, evaluated at 4 relative to c_{333} is chosen, then DM will usually produce an alternative solution.

$$\mathrm{S}_3 = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \end{array} \begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \left(\begin{array}{cccc} 17 & - & - & - \\ - & - & - & 4 \\ - & - & (4) & - \\ - & 52 & - & - \end{array}\right) \end{array}$$

S_3 is the Hungarian method solution for C_3.

After removing matrix C_3 and deleting row 3 and column 3 from matrices C_1, C_2 and C_4 the execution of the DM algorithm progresses until another feasible solution for the original problem is identified. All variables of the solution are $x_{ijk} = 0$, except $x_{144} = x_{211} = x_{333} = x_{422} = 1$, resulting in a total cost of 33.

$$\begin{array}{c|cccc|cccc|cccc|cccc} & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 & m_1 & m_2 & m_3 & m_4 \\ j_1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 8 \\ j_2 & 9 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\ j_3 & - & - & - & - & - & - & - & - & - & - & 4 & - & - & - & - & - \\ j_4 & - & - & - & - & - & 12 & - & - & - & - & - & - & - & - & - & - \end{array}$$

Another DM feasible solution for 3IAP.

In conclusion, five feasible solutions for the 3IAP are identified; GSP generates two with costs of 98 and 73, whereas DM yields three at 87, 57 and 33, with the latter being optimal.

By rearranging cost matrices in ascending and descending order and then choosing diverse minima in tie-cases, DM reaches several feasible solutions for the same problem, and the lowest total cost result is selected. Kadhem (2017), however, always picked the first minimum emerging in tie-cases despite the presence of multiple candidates.

Selecting the first minimum in tie-cases does not guarantee the best solution. Explicitly enumerating all candidates is impractical and expensive. Therefore, exploring minimum candidates in tie-cases is advised, since the situation generates various solutions. We propose choosing the first, last, or a random candidate in each iteration, leading to three novel variants of the DM algorithm. When a tie-case occurs during an iteration, especially with large instances of the 3IAP, at least three options are available, as illustrated in Figure 3.1.

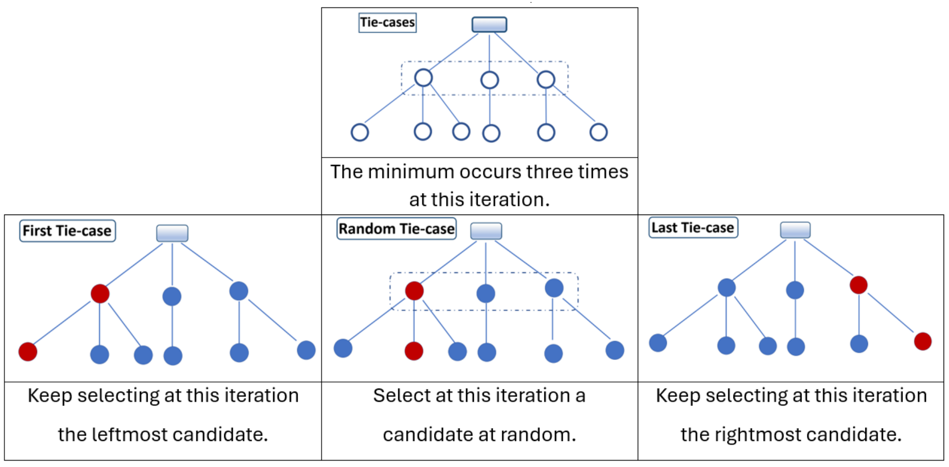
Tie-cases
The minimum occurs three times at this iteration.
First Tie-case
Keep selecting at this iteration the leftmost candidate.
Random Tie-case
Select at this iteration a candidate at random.
Last Tie-case
Keep selecting at this iteration the rightmost candidate.

Figure 3.1: Three types of tie-cases.

The following table summarises the six DM solutions found for the previous example, the 3IAP instance of size 4.

Table 3.3: DM solutions for Example 3.

	First tie-case		**Last tie-case**		**Random tie**	
Sorted in	Cost	Runtime	Cost	Runtime	Cost	Runtime
Ascending	87	0.0312722	87	0.020031	87	0.015622
Descending	57	0.0189335	<u>33</u>	<u>0.015576</u>	33	0.019915

Thus, DM and GSP heuristic produce six potential feasible solutions for each instance by reorganising cost matrices in ascending and descending orders. In cases where multiple results achieve the same total cost, the solution with the lowest execution time is selected. In Table 3.3, the running time is given in seconds. For the previous example, DM reached the minimum cost on two occasions in descending order, and the optimal solution is underlined and attained by choosing the last tie-case. The three variants of each method are considered throughout the rest of this project.

3.2.3 Numerical experiments

Computational tests are conducted on a PC with Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, and 8.79 GB of RAM; the algorithms are coded in Python 3.9 and run on random samples to test the performance of GSP and DM variants.

To evaluate the performance of the GSP and DM variants, numerical experiments are tested on the same dataset as in (Balas and Saltzman, 1991), which will be referred to as Sample 1 throughout the paper. The random sample consists of 60 instances, generated by the uniform probability distribution from 0 to 100, of size $n = 4, 6, 8, \dots, 26$, five instances per size. The data and code used in this section are available online in a Github repository: https://github.com/mmehbali/Three-IAP. The GSP and DM variants provide six feasible solutions for each instance, as displayed in Tables 3.4 and 3.5, respectively; the best solutions found are highlighted.

Table 3.4: GSP Variants applied to a data sample from Balas and Saltzman, 1991.

		FIFO rule						**LIFO rule**					
Size	Instance	First Tie-case		Last Tie-case		Random Tie		First Tie-case		Last Tie-case		Random Tie	
		Cost	Runtime	Cost	Runtime	Cost	Runtime	Cost	Runtime	Cost	Runtime	Cost	Runtime
4	1	46	0.00210214	46	0.00320339	46	0.01134753	127	0.00369668	127	0.00471401	127	0.00643301
	2	89	0.00099730	89	0.00365448	89	0.00510907	73	0.00676274	73	0.00338864	73	0.00595522
	3	54	0.00206923	54	0.00194740	54	0.01150131	59	0.00586843	59	0.00860476	59	0.01399684
	4	83	0.00200605	83	0.00820732	83	0.00541782	75	0.00362301	99	0.00890183	75	0.01168871
	5	50	0.00099754	50	0.00199509	50	0.00624371	77	0.00929666	77	0.00801396	77	0.00879145
6	1	73	0.00567222	73	0.01286340	73	0.01871562	89	0.01606989	89	0.01075602	89	0.00666213
	2	51	0.00423217	51	0.00482130	51	0.02397418	70	0.01296616	70	0.00859118	70	0.00562620
	3	101	0.00940967	123	0.00693583	101	0.02210855	140	0.01097131	140	0.01231647	140	0.01776171
	4	113	0.00232983	113	0.00099707	113	0.01994443	121	0.00728488	121	0.01400566	121	0.00970387
	5	73	0.00742221	73	0.00000000	73	0.02569437	88	0.00583696	88	0.01289344	88	0.01687717
8	1	45	0.00316525	45	0.01435137	45	0.01063967	74	0.01742792	74	0.00908208	45	0.03473115
	2	95	0.01857448	105	0.00316381	95	0.00973868	120	0.01691484	66	0.00710893	95	0.03704453
	3	92	0.02015924	65	0.00099921	65	0.02498555	126	0.01836371	88	0.00699759	46	0.04327226
	4	136	0.02007961	136	0.00321698	136	0.01179981	110	0.01602077	113	0.00893974	136	0.04098654
	5	34	0.01650405	38	0.00310564	34	0.00715566	90	0.01352477	78	0.00738406	34	0.02911496
10	1	99	0.00819159	75	0.02546334	75	0.00769567	141	0.02088284	113	0.02497864	75	0.04487300
	2	93	0.00698781	93	0.00885487	93	0.00899458	38	0.00756812	51	0.01025295	93	0.05185290
	3	113	0.01687169	36	0.01908064	114	0.00824904	55	0.01013374	51	0.00806880	36	0.04388738
	4	121	0.01907325	55	0.01899695	121	0.00902128	93	0.00838351	58	0.01727176	55	0.05068493
	5	93	0.02010012	49	0.02084494	61	0.02093768	112	0.00762677	112	0.01568270	49	0.04832864
12	1	105	0.02376676	47	0.02751851	105	0.01469874	55	0.02929068	123	0.02811289	80	0.04691601
	2	77	0.02067423	127	0.01703501	49	0.02922964	105	0.01340675	51	0.01119018	75	0.02392530
	3	127	0.02186680	119	0.01912832	58	0.02030206	100	0.02878475	68	0.02491188	68	0.03254795
	4	71	0.02586007	76	0.01875687	71	0.04331779	69	0.02734089	69	0.02265501	69	0.03074622
	5	24	0.02119994	42	0.01544404	42	0.02831125	86	0.01444960	108	0.02545428	108	0.02379966
14	1	81	0.02422452	81	0.05414796	81	0.04412675	143	0.03010392	155	0.03524613	125	0.03725243
	2	74	0.02216458	127	0.03516507	60	0.03278971	57	0.03359890	72	0.02438259	85	0.04099655
	3	133	0.02077317	105	0.03572965	97	0.02798080	127	0.03353071	62	0.03149319	67	0.03150654
	4	73	0.02979636	53	0.03472590	73	0.03740811	76	0.03250456	76	0.03215122	76	0.03961515
	5	125	0.02136493	92	0.04097843	79	0.03310132	131	0.03347325	121	0.01786423	121	0.03305411
16	1	89	0.03656673	137	0.04834127	86	0.03751588	107	0.08001876	22	0.02643538	113	0.03915858
	2	97	0.04460144	72	0.04571438	72	0.03697085	100	0.07827020	39	0.03421617	74	0.03585124
	3	74	0.03737926	161	0.04288244	151	0.04281712	45	0.08908629	59	0.03306603	59	0.06250024
	4	81	0.03455424	83	0.04537630	77	0.04227448	102	0.07774377	105	0.03344488	102	0.03752518
	5	121	0.04948711	97	0.04791141	73	0.03916359	122	0.0893116	147	0.05857158	85	0.03489161
18	1	104	0.04776144	68	0.05614901	68	0.06775546	148	0.14999485	171	0.13612795	103	0.04177213
	2	46	0.05033851	78	0.05661201	78	0.04489899	105	0.15546775	113	0.13400340	66	0.04912567
	3	72	0.03997850	63	0.05366135	97	0.03781700	100	0.14067030	145	0.13385487	65	0.06580114
	4	86	0.05907512	88	0.06118488	86	0.08527541	64	0.14927626	84	0.14838052	130	0.06495714
	5	144	0.04507351	92	0.05643296	57	0.05199432	118	0.17069912	106	0.13320255	76	0.04738760
20	1	74	0.06464696	147	0.06724644	19	0.06407595	120	0.15980697	96	0.15749955	90	0.07289219
	2	47	0.07279825	78	0.06670213	77	0.05179119	120	0.19243956	101	0.22532821	80	0.06860566
	3	109	0.07468081	129	0.07346582	76	0.05432439	125	0.16674471	52	0.24699974	45	0.05286050
	4	61	0.05321741	113	0.06722021	51	0.06233215	57	0.16668558	64	0.24061966	130	0.05959034
	5	106	0.06539440	67	0.06606865	80	0.06512165	57	0.18279243	57	0.24754643	92	0.05673695
22	1	71	0.08430076	144	0.08278370	70	0.09581637	124	0.22371101	160	0.20043278	128	0.08588076
	2	79	0.12412429	79	0.08773661	78	0.06870532	45	0.20022154	105	0.21862650	103	0.07362390
	3	105	0.07521343	150	0.09098101	74	0.07457423	90	0.21915984	151	0.22094083	98	0.07908177
	4	60	0.06681585	47	0.09236598	51	0.06897593	81	0.23369527	133	0.20637774	53	0.06499457
	5	85	0.08577561	111	0.09682608	77	0.07283950	88	0.21111965	90	0.18976593	67	0.08724236
24	1	107	0.12582541	54	0.09999275	38	0.09325886	96	0.26368928	87	0.26318669	78	0.16543555
	2	87	0.09863520	59	0.09807944	99	0.10334229	54	0.21050763	52	0.23788238	44	0.08496857
	3	124	0.09705997	156	0.10035801	46	0.08368397	137	0.26320028	11	0.23239470	53	0.09886575
	4	152	0.0918901	109	0.10085797	52	0.08409667	70	0.26605248	103	0.28364468	53	0.09179211
	5	126	0.08877373	46	0.09921670	34	0.09209132	78	0.28938794	90	0.24973154	70	0.09796929
26	1	44	0.14825583	60	0.11530089	80	0.11758876	168	0.09892654	77	0.11169004	72	0.11123443
	2	66	0.17565107	80	0.12044501	72	0.09850121	146	0.12161875	105	0.12804580	45	0.09843779
	3	66	0.18071461	93	0.11276555	64	0.11158848	83	0.10865164	92	0.09544492	71	0.10081863
	4	96	0.11376643	99	0.12015939	88	0.09683895	100	0.11655664	51	0.11103368	77	0.10978484
	5	126	0.10726023	137	0.11393547	74	0.10103488	97	0.10573244	140	0.10838437	66	0.09597945

Table 3.5: DM Variants applied to a data sample from Balas and Saltzman, 1991.

Size	Instance	in Ascending Order						in Descending Order					
		First Tie-case		Last Tie-case		Random Tie		First Tie-case		Last Tie-case		Random Tie	
		Cost	Runtime	Cost	Runtime	Cost	Runtime	Cost	Runtime	Cost	Runtime	Cost	Runtime
	1	120	0.0100050	120	0.0089948	120	0.0156214	101	0.01127505	101	0.0220592	101	0.01894830
	2	64	0.0069816	64	0.0079827	64	0.0120819	51	0.0199466	51	0.0211554	51	0.01592708
4	3	59	0.0079789	59	0.0159271	59	0.0197201	59	0.01990939	59	0.0209439	59	0.01396465
	4	83	0.0075715	83	0.0069814	83	0.0183339	75	0.0219560	99	0.0211179	75	0.01196718
	5	80	0.0069699	80	0.0085332	80	0.0201163	77	0.0214891	77	0.0219414	77	0.01216674
	1	68	0.0130763	68	0.0304453	68	0.0298064	65	0.0339088	65	0.0316148	65	0.0139606
	2	115	0.0101721	115	0.0329139	115	0.0278840	63	0.0406399	63	0.0311301	63	0.0179143
6	3	54	0.0115078	54	0.0289505	54	0.0279279	113	0.0369959	128	0.0324178	113	0.0228801
	4	86	0.0076144	86	0.0339091	86	0.0293858	80	0.0300484	80	0.0309000	80	0.0150967
	5	78	0.0000000	109	0.0309138	78	0.0300243	172	0.0337675	172	0.0289574	172	0.0185933
	1	57	0.0161650	109	0.0423861	57	0.1564496	81	0.0439470	81	0.0466342	81	0.0280113
	2	122	0.0156241	77	0.0446448	68	0.1154072	49	0.0495834	87	0.0433650	49	0.0249772
8	3	63	0.0211608	50	0.0439301	50	0.1348510	84	0.0384920	127	0.0434673	83	0.0317512
	4	75	0.0156887	103	0.0458648	75	0.1018567	90	0.0417278	84	0.0412576	68	0.0189126
	5	8	0.0155582	148	0.0423026	8	0.1683254	56	0.0436890	78	0.0397818	56	0.0235457
	1	129	0.0207374	48	0.0621553	66	0.2348409	84	0.0612285	120	0.0650168	84	0.0291278
	2	74	0.0184803	74	0.0478046	74	0.2250631	52	0.0640123	53	0.0550895	52	0.0185599
10	3	107	0.0168028	107	0.0557406	107	0.2193201	87	0.0725200	121	0.0549350	64	0.0249281
	4	136	0.0156217	82	0.0606716	82	0.2369609	64	0.0661914	84	0.0640216	64	0.0288825
	5	136	0.0156212	104	0.0587394	104	0.2288067	111	0.0539300	113	0.0668213	111	0.0238676
	1	68	0.0249102	66	0.0990796	64	0.0758309	75	0.0793335	57	0.0907576	57	0.0391731
	2	125	0.0230906	122	0.0932608	84	0.0831053	128	0.0953901	77	0.0924416	65	0.0339096
12	3	98	0.0162509	115	0.0947340	79	0.0843966	112	0.0943108	112	0.0961335	112	0.0218928
	4	69	0.0156663	90	0.0980532	69	0.0822625	81	0.0933189	81	0.0928619	81	0.0319502
	5	174	0.0312402	76	0.0936139	76	0.0833957	116	0.0904186	101	0.0932014	101	0.0398910
	1	66	0.0319147	141	0.1309872	35	0.1052270	148	0.1092475	69	0.1226533	69	0.1057429
	2	24	0.0275707	114	0.1384912	53	0.0963893	125	0.1075077	113	0.1115911	31	0.1259358
14	3	111	0.0200613	93	0.1388362	77	0.0976758	104	0.1226351	53	0.1174052	53	0.1195037
	4	118	0.0361946	103	0.1445396	50	0.1257784	61	0.1180797	40	0.1215775	40	0.0935991
	5	43	0.0324488	172	0.1199877	38	0.1163676	50	0.1126332	96	0.1221256	38	0.0962217
	1	59	0.0388928	130	0.1778674	52	0.1228278	107	0.1657932	60	0.1739564	40	0.1268768
	2	92	0.0388606	119	0.2036905	46	0.1422348	129	0.1772234	86	0.1684265	48	0.1388183
16	3	90	0.0389342	125	0.1923089	71	0.1347578	23	0.1709454	149	0.1818943	70	0.1187835
	4	95	0.0390573	91	0.2071362	74	0.1140461	135	0.1625075	70	0.1704474	63	0.1274412
	5	69	0.0397661	96	0.1989081	70	0.1212702	34	0.1775985	61	0.1776381	35	0.1340263
	1	51	0.0450637	87	0.2627711	43	0.1515684	124	0.2528760	136	0.2504299	50	0.1642561
	2	49	0.0458317	71	0.2908456	38	0.1409218	110	0.2543094	70	0.2725921	31	0.1618323
18	3	142	0.0527828	37	0.2641921	52	0.1567829	114	0.2464047	139	0.2352862	33	0.1620760
	4	119	0.0489056	130	0.2554598	78	0.1542656	82	0.2425082	80	0.2451539	55	0.1572001
	5	47	0.0526750	84	0.2798693	73	0.1613507	104	0.2343776	114	0.2317057	61	0.1522055
	1	105	0.0627451	94	0.3496108	47	0.0634198	64	0.3394494	40	0.3405883	26	0.1952128
	2	30	0.0549486	82	0.3579905	34	0.0693061	119	0.3286281	49	0.3341446	49	0.2078474
20	3	122	0.0533302	117	0.3564801	65	0.0708215	94	0.3489153	126	0.3453660	67	0.2066286
	4	53	0.0653865	88	0.3614526	60	0.0738537	91	0.3345735	115	0.3355937	56	0.1855583
	5	122	0.0502832	90	0.3462441	30	0.0708125	80	0.3105972	76	0.3217070	37	0.1937101
	1	47	0.0782130	164	0.4508257	47	0.0881426	154	0.4368594	140	0.4384162	51	0.2324278
	2	88	0.0697444	175	0.4765818	33	0.0754585	112	0.4382422	102	0.4250071	37	0.2334025
22	3	75	0.0625374	125	0.4731760	37	0.0791652	108	0.4618361	73	0.4435165	49	0.2347534
	4	94	0.0621159	106	0.4753542	71	0.0879195	113	0.4583137	96	0.4565649	47	0.2089188
	5	113	0.0863214	46	0.4623990	52	0.0848382	69	0.4483247	116	0.4604309	52	0.2614031
	1	77	0.0803669	137	0.6145887	41	0.0959871	169	0.5765002	128	0.5605409	59	0.2462826
	2	135	0.0945604	145	0.6040702	53	0.0909970	94	0.5608380	159	0.6228988	23	0.2617285
24	3	65	0.0957706	136	0.6173430	33	0.0960672	102	0.5701540	78	0.6006057	39	0.2665687
	4	105	0.0892520	105	0.5927818	51	0.0954394	92	0.5686386	132	0.5602443	45	0.2553465
	5	113	0.0896075	90	0.5766120	38	0.0963628	139	0.5742068	94	0.6153796	31	0.2952910
	1	66	0.1067441	96	0.7707765	46	0.1065700	109	0.7439399	69	0.7319100	39	0.3155532
	2	77	0.0983267	64	0.7752740	40	0.1063283	87	0.7136793	97	0.7285006	31	0.3208118
26	3	88	0.1091614	107	0.7605805	44	0.1111016	112	0.7399192	61	0.7150211	44	0.2938275
	4	100	0.0911787	85	0.7670391	49	0.1214638	158	0.7498312	59	0.7476666	48	0.3117094
	5	63	0.1003244	84	0.7637739	37	0.1106565	120	0.7235053	50	0.7177529	43	0.3406253

Analysis of both tables 3.4 and 3.5 in Figure 3.2 shows that GSP's distribution of best-found solutions across tie-case types is nearly balanced, with a slight preference for random tie-cases. In contrast, DM exhibits a clear preference for random tie-cases. When partitioning the best-found solutions per type of order of cost matrices, the FIFO rule is notably prevalent for GSP, while DM shows some disparity between the ascending and descending orders.

This examination reveals two key insights; first, the efficacy of random tie-cases is notable and warrants consideration. Secondly, future implementations will prioritise the FIFO rule in the GSP heuristic for unsorted cost matrices, henceforth referred to simply as GSP.

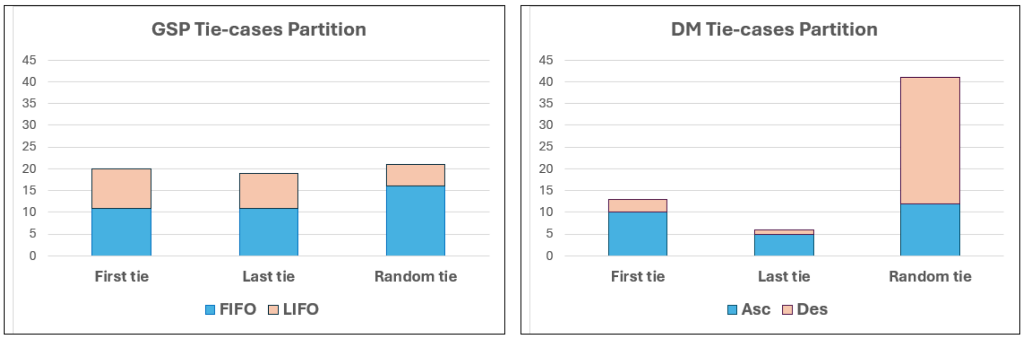

Figure 3.2: The partition of the best solutions found per variant.

The heuristics introduced in this study yield multiple solutions for the 3IAP, whereas the original version of DM generates merely one or two, focusing solely on the first tie-cases. Our examination indicates that the larger the set of solutions, the better the quality of the obtained solution. A comparative analysis of the DM results for the first tie-cases in Table 3.5 against the combined GSP and DM variants outcomes in the last column of Table 3.6 confirms the primacy of our heuristics for addressing the 3IAP, underscoring their efficacy.

Computational tests suggest that developing new variants of GSP and DM enables achieving feasible solutions, which often outperform those derived from the

Table 3.6: Summary of the results of GSP and DM methods.

		GSP Heuristic				**DM Method**				**Best solution**	
Size	Instance	Cost	Runtime	Tie	Rule	Cost	Runtime	Tie	Order	Cost	Runtime
4	1	46	0.00210214	First	Fifo	101	0.01127505	First	Desc	46	0.00210214
	2	73	0.00338864	Last	Lifo	51	0.01592710	Rand	Desc	51	0.01592710
	3	54	0.00194740	Last	Fifo	59	0.01972008	Last	Asc	54	0.00194740
	4	75	0.00362301	First	Lifo	75	0.01196718	Rand	Desc	75	0.00362301
	5	50	0.00099754	First	Fifo	77	0.01216674	Rand	Desc	50	0.00099754
6	1	73	0.00567222	First	Fifo	65	0.0139606	Rand	Desc	65	0.0139606
	2	51	0.00423217	First	Fifo	63	0.01791430	Rand	Desc	51	0.00423217
	3	101	0.00940967	First	Fifo	54	0.01150788	First	Asc	54	0.01150788
	4	113	0.00099707	Last	Fifo	80	0.01509666	Rand	Desc	80	0.01509666
	5	73	0.00000000	Last	Fifo	78	0.00000000	First	Asc	73	0.00000000
8	1	45	0.00316525	First	Fifo	57	0.01616502	First	Asc	45	0.00316525
	2	66	0.00710893	Last	Lifo	49	0.02497721	Rand	Desc	49	0.02497712
	3	46	0.04327226	Rand	Lifo	50	0.04393005	Last	Asc	46	0.04327226
	4	110	0.01602077	First	Lifo	68	0.01891255	Rand	Desc	68	0.01891255
	5	34	0.00715566	Rand	Fifo	8	0.01555824	First	Asc	8	0.01555824
10	1	75	0.00769567	Rand	Fifo	48	0.06215525	Last	Asc	48	0.06215525
	2	38	0.00756812	First	Lifo	52	0.01855993	Rand	Desc	38	0.00756812
	3	36	0.01908064	Last	Fifo	64	0.02492809	Rand	Desc	36	0.01908064
	4	55	0.01899695	Last	Fifo	64	0.02888250	Rand	Desc	55	0.01899695
	5	49	0.02084494	Last	Fifo	104	0.05873942	Last	Asc	49	0.02084494
12	1	47	0.02751851	Last	Fifo	57	0.03917313	Rand	Desc	47	0.02751851
	2	49	0.02922964	Rand	Fifo	65	0.03390956	Rand	Desc	49	0.02922964
	3	58	0.02030206	Rand	Fifo	79	0.08439660	Rand	Asc	58	0.02030206
	4	69	0.02265501	Last	Lifo	69	0.0156663	First	Asc	69	0.01566625
	5	24	0.02119994	First	Fifo	76	0.08339572	Rand	Asc	24	0.02119994
14	1	81	0.02422452	First	Fifo	35	0.10522699	Rand	Asc	35	0.10522699
	2	57	0.03359890	First	Lifo	24	0.02757072	First	Asc	24	0.02757072
	3	62	0.03149319	Last	Lifo	53	0.11740518	Last	Desc	53	0.11740518
	4	53	0.03472590	Last	Fifo	40	0.09359908	Rand	Desc	40	0.09359908
	5	79	0.03310132	Rand	Fifo	38	0.09622169	Rand	Desc	38	0.09622169
16	1	22	0.02643538	Last	Lifo	40	0.12687683	Rand	Desc	22	0.02643538
	2	39	0.03421617	Last	Lifo	46	0.14223480	Rand	Asc	39	0.03421617
	3	45	0.08908629	First	Lifo	23	0.17094541	First	Desc	23	0.17094541
	4	77	0.04227448	Rand	Fifo	63	0.12744117	Rand	Desc	63	0.12744117
	5	73	0.03916359	Rand	Fifo	34	0.17759848	First	Desc	34	0.17759848
18	1	68	0.05614901	Last	Fifo	43	0.15156841	Rand	Asc	43	0.15156841
	2	46	0.05033851	First	Fifo	31	0.16183233	Rand	Desc	31	0.16183233
	3	63	0.05366135	Last	Fifo	33	0.16207600	Rand	Desc	33	0.16207600
	4	64	0.14927626	First	lifo	55	0.15720010	Rand	Desc	55	0.15720010
	5	57	0.05199432	Rand	Fifo	47	0.05267501	First	Asc	47	0.05267501
20	1	19	0.06407595	Rand	Fifo	26	0.19521284	Rand	Desc	19	0.06407595
	2	47	0.07279825	First	Fifo	30	0.68915153	First	Asc	30	0.68915153
	3	45	0.05286050	First	Lifo	65	0.07082152	Rand	Asc	45	0.05286050
	4	51	0.06233215	Rand	Fifo	53	0.65729141	First	Asc	51	0.06233215
	5	57	0.18279243	First	Lifo	30	0.07081246	Rand	Asc	30	0.07081246
22	1	70	0.09581637	Rand	Fifo	47	0.07821298	First	Asc	47	0.07821298
	2	45	0.20022154	First	Lifo	33	0.07545853	Rand	Asc	33	0.07545853
	3	74	0.07457423	Rand	Fifo	37	0.07916522	Rand	Asc	37	0.07916522
	4	47	0.09236598	Last	Fifo	47	0.20891881	Rand	Desc	47	0.09236598
	5	67	0.08724236	Rand	Lifo	46	0.46239901	Last	Asc	46	0.46239901
24	1	38	0.09325886	Rand	Fifo	41	0.09598708	Rand	Asc	38	0.09325886
	2	44	0.08496857	Rand	Lifo	23	0.26172853	Rand	Desc	23	0.26172853
	3	11	0.23239470	Last	Lifo	33	0.09606719	Rand	Asc	11	0.23239470
	4	52	0.08409667	Rand	Fifo	45	0.25534654	Rand	Desc	45	0.25534654
	5	34	0.09209132	Rand	Fifo	31	0.29529095	Rand	Desc	31	0.29529095
26	1	44	0.14825583	First	Fifo	39	0.31555319	Rand	Desc	39	0.31555319
	2	45	0.09843779	Rand	Lifo	31	0.32081175	Rand	Desc	31	0.32081175
	3	64	0.11158848	Rand	Fifo	44	0.11110163	Rand	Desc	44	0.11110163
	4	51	0.11103368	Last	Lifo	48	0.31170940	Rand	Desc	48	0.31170940
	5	66	0.09597945	Rand	Lifo	37	0.11065650	Rand	Asc	37	0.11065650

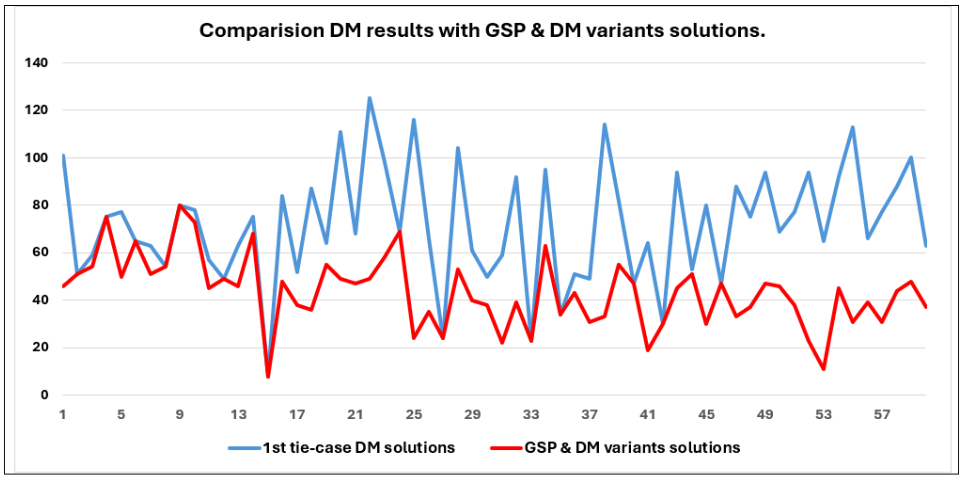

Figure 3.3: Comparison of DM results with the results of GSP and DM variants.

DM simple version, as displayed in Figure 3.3.

DM operates through two phases, the first orders the cost matrices, while the second, GSP, allows us to attain a feasible solution for 3IAP. The results in Table 3.6 indicate that the GSP should be considered because it could lead to better solutions.

The following chapter explores the development of two heuristic classes designed for swift approximation of high-quality solutions for 3IAP, substantiated by extensive numerical testing. The first class relies on statistical measures, while the second on matrix norms.

Chapter 4

Adavanced Heuristics for 3IAP

The rationale for sorting the cost matrices according to their diagonal factors is omitted in (Kadhem, 2017), a method not previously used or mentioned. This chapter not only questions the singular use of the diagonals method but also broadens the scope of strategies for matrices arrangement. The current chapter investigates alternate matrices sorting methodologies, introducing new heuristics aligned with the DM framework. Exploring the impact of cost matrices arrangements order on solutions is crucial. Applying the GSP heuristic with various matrices arrangements may lead to interesting results, highlighting how matrices sorting order significantly influences heuristic outcomes.

4.1 Heuristics of Category 1

The inquiry critiques the exclusive reliance on the diagonals method and explores to different strategies for sorting cost matrices in assignment problems. By analysing matrix attributes, two novel categories of expedient heuristics have been found to consistently generate quality feasible solutions. The first category of heuristics employs descriptive statistical measures, while the second refers to matrix norms.

The objective function of 3IAP is defined by n positive matrices of $\mathcal{M}^{n^2}$, each associated with a factory. Each cost matrix is treated as a one-dimensional vector of $\mathbb{R}^{n^2}$. Consequently, vector norms serve as criteria for sorting the cost matrices, facilitating a systematic method for solving the assignment problem.

Among vector norms defined over the vector space $\mathbb{R}^n$, the 1-norm and the infinity-norm are considered in the following. Let C be a cost matrix associated with an arbitrary factory whose coefficients are all nonnegative, and C is considered here as a one-dimensional vector of $\mathbb{R}^{n^2}$,

The 1-norm of C is $\| C \|_1 = \sum_{i=1}^{n^2} c_i$ and divided by n^2, which equates to $mean(C)$. This implies that ranking cost matrices by their mean values is equivalent to sorting them by their 1-norms. Therefore, 3IAP can be effectively addressed by reordering its matrices based on their mean values in ascending or descending order, supporting a streamlined solution approach.

However, if the costs take extreme values, to prevent the matrices order from being affected by the outliers, it is advisable to sort the matrices based on their median rather than their mean. Rearranging the matrices according to their median in either increasing or decreasing order, then applying GSP to obtain a feasible solution to 3IAP may yield a different solution.

The infinity-norm, $\| C \|_\infty$ is equal to the maximum of the absolute value of the coefficients of C. The matrices can be rearranged according to their maximum; subsequently, they can be sorted according to their coefficients range. Consequently, 3IAP will have two solutions, one from sorting its cost matrices according to their maxima and the second achieved after rearranging the matrices according to their range.

Similarly, the standard deviation, a common measure of dispersion in descriptive statistics, can be considered. Reorganising the matrices based on their standard deviation, in ascending or descending order, and then applying GSP leads to a feasible solution for 3IAP.

In summary, the development of five novel heuristics, in addition to GSP and DM, through sorting cost matrices by their mean, median, maximum, range, or standard deviation, forms a new category designated as Category 1. These heuristics, predicated on statistical sorting criteria, account for ten methodologies in both ascending and descending orders. Alongside a solution derived from GSP applied to unsorted matrices, this approach rapidly yields eleven additional feasible solutions for each instance of 3IAP. The best found solution is then selected from these alternatives.

To apply Category 1 heuristics to 3IAP, let us choose from Sample 1, for example, instance 3 of size 10. The problem handles 1,000 variables, with $1,3 \times 10^{13}$ potential feasible solutions. The heuristics operate concurrently across the first, last, and random tie cases, resulting in 13 outcomes for each case, the yielded solutions are presented in Tables 4.1, 4.2, and 4.3, respectively. The best solution found in each table is underlined. Table 4.4 summarises the experimental results obtained from the three previous tables.

Table 4.1: Solutions for a 3IAP instance of size 10, selecting the first tie-case.

Index	**Sort Type**	**Order**	**Cost**	**Runtime**
1	GSP	—	133	0.01562381
2	Diagonals	Ascending	107	0.01562095
3	Diagonals	Descending	87	0.01701713
4	Maximum	Ascending	160	0.01566124
5	Maximum	Descending	84	0.03124285
6	Mean	Ascending	71	0.01836586
7	Mean	Descending	$\underline{58}$	$\underline{0.01920557}$
8	Median	Ascending	165	0.02127552
9	Median	Descending	123	0.02434134
10	Range	Ascending	72	0.02418661
11	Range	Descending	86	0.02154136
12	Std. Deviation	Ascending	88	0.01696014
13	Std. Deviation	Descending	92	0.01572490

Next, for each instance, these heuristics are run six times, one for each variant of the tie-cases, multiplied by two for the ascending and descending sorting orders.

Table 4.2: Solutions for a 3IAP instance of size 10, selecting the last tie-case .

Index	**Sort Type**	**Order**	**Cost**	**Runtime**
1	GSP	—	<u>36</u>	<u>0.03793907</u>
2	Diagonals	Ascending	107	0.03272176
3	Diagonals	Descending	121	0.02891898
4	Maximum	Ascending	160	0.02564788
5	Maximum	Descending	75	0.02691746
6	Mean	Ascending	104	0.02027655
7	Mean	Descending	58	0.02844763
8	Median	Ascending	165	0.02450919
9	Median	Descending	123	0.02194047
10	Range	Ascending	72	0.00607562
11	Range	Descending	69	0.02133369
12	Std. Deviation	Ascending	108	0.02435017
13	Std. Deviation	Descending	130	0.03312850

Table 4.3: Solutions for a 3IAP instance of size 10, selecting a tie-case at random.

Index	**Sort Type**	**Order**	**Cost**	**Runtime**
1	GSP	—	133	0.02292633
2	Diagonals	Ascending	107	0.00855851
3	Diagonals	Descending	<u>27</u>	<u>0.03613329</u>
4	Maximum	Ascending	160	0.00891209
5	Maximum	Descending	75	0.03389192
6	Mean	Ascending	104	0.01489067
7	Mean	Descending	58	0.01526475
8	Median	Ascending	165	0.03229070
9	Median	Descending	123	0.02351594
10	Range	Ascending	72	0.02376366
11	Range	Descending	72	0.02641940
12	Std. Deviation	Ascending	79	0.02238274
13	Std. Deviation	Descending	126	0.02317810

Table 4.4: Summary of best solutions for a 3IAP instance of size 10.

Case	**Sort Type**	**Order**	**Cost**	**Runtime**
First tie-case	Mean	Descending	58	0.01920557
Last tie-case	GSP	—	36	0.03793907
Random tie	Diagonals	Descending	<u>27</u>	<u>0.03613329</u>

The program will retain only the best solution obtained for each instance.

To evaluate the efficiency of Category 1 heuristics, additional tests are conducted on Sample 1 data, with the findings summarised in Table 4.5. Each row represents an instance, highlighting the best solution achieved. It is observed that of the 60 best solutions identified, only six are yielded by the original DM version *i.e.*, 10%, while the remaining are generated by Category 1 heuristics.

To summarise, Category 1 heuristics not only provide feasible solutions for 3IAP but they also outperform the DM algorithm. Some solutions are optimal, and others can serve as effective starting points for the B&B method. Typically, the hybrid heuristics of Category 1 are associated with producing high-quality solutions.

The following section examines the second category of heuristics for 3IAP.

4.2 Heuristics of Category 2

The study proposes a new criterion for reorganising cost matrices using matrix norms instead of statistical measures. This approach leads to the design of four novel heuristics, termed Category 2, analogous to the heuristics of Category 1. The methodology promises enhanced efficacy in 3IAP solution approaches.

The 3IAP objective function is defined by cost matrices, which are two-dimensional arrays where each coefficient represents the cost associated with a specific row-column assignment. These matrices are elements of $\mathcal{M}^{n^2}$; thus, to rearrange them, it is helpful to refer to matrix norms. There are four common norms defined over the space vector $\mathcal{M}^{n^2}$.

Table 4.5: Outcomes of Category 1 heuristics implemented on Sample 1.

		First tie-case				**Last tie-case**				**Random tie-case**			
Size	Instance	Type	Order	Cost	Runtime	Type	Order	Cost	Runtime	Type	Order	Cost	Runtime
4	1	Std. Dev.	Asc	46	0.0153406	Unsorted	—	46	0.0159895	Maximum	Desc	46	0.0079737
	2	Diagonals	Desc	51	0.0197680	Diagonals	Desc	51	0.0256345	Diagonals	Desc	51	0.0079789
	3	Std. Dev.	Desc	54	0.0172117	Unsorted	Desc	54	0.0160148	Unsorted	—	54	0.0080109
	4	Mean	Asc	68	0.0279086	Mean	Asc	71	0.0256448	Mean	Asc	68	0.0089753
	5	Mean	Desc	42	0.0209127	Std. Dev.	Desc	22	0.0269482	Std. Dev.	Desc	22	0.0089760
6	1	Std. Dev.	Asc	54	0.0382040	Maximum	Asc	60	0.0436385	Median	Desc	42	0.0119770
	2	Unsorted	—	51	0.0320008	Unsorted	—	51	0.0334609	Unsorted	—	51	0.0119681
	3	Median	Desc	42	0.0349155	Median	Desc	42	0.0320060	Median	Desc	42	0.0119572
	4	Mean	Desc	50	0.0255313	Std. Dev.	Desc	50	0.0348604	Median	Desc	50	0.0119636
	5	Range	Desc	28	0.0404816	Range	Desc	28	0.0426095	Range	Desc	28	0.0119267
8	1	Range	Desc	37	0.0416465	Median	Desc	36	0.0465174	Median	Desc	36	0.0510972
	2	Maximum	Desc	44	0.0426154	Range	Asc	55	0.0400841	Maximum	Desc	44	0.0510507
	3	Std. Dev.	Asc	35	0.0463522	Std. Dev.	Asc	35	0.0385490	Std. Dev.	Asc	35	0.0169549
	4	Median	Desc	57	0.0424366	Median	Asc	56	0.0568943	Median	Asc	56	0.0534122
	5	Diagonals	Asc	8	0.0399439	Range	Desc	8	0.0400512	Range	Desc	8	0.0486159
10	1	Maximum	Desc	37	0.0156262	Maximum	Desc	37	0.0561564	Maximum	Desc	37	0.0630534
	2	Diagonals	Desc	52	0.0545759	Range	Asc	43	0.0735247	Range	Asc	43	0.0711098
	3	Mean	Des	58	0.01920557	Unsorted	—	36	0.03793907	Diagonals	Desc	27	0.03613329
	4	Median	Desc	32	0.0554729	Median	Desc	32	0.0626526	Range	Desc	32	0.0647616
	5	Maximum	Desc	43	0.0587294	Median	Asc	35	0.0667386	Mean	Asc	40	0.0721371
12	1	Median	Asc	62	0.0872922	Range	Asc	39	0.0839060	Range	Asc	75	0.0875769
	2	Range	Asc	65	0.0816712	Std. Dev.	Asc	53	0.0730062	Std. Dev.	Asc	53	0.0845430
	3	Mean	Desc	36	0.0703671	Range	Desc	61	0.0726814	Mean	Desc	36	0.0831275
	4	Range	Desc	64	0.0884013	Std Dev.	Asc	63	0.0789812	Range	Desc	64	0.0755398
	5	Unsorted	—	24	0.0794580	Unsorted	—	42	0.0701621	Mean	Desc	37	0.0860555
14	1	Mean	Asc	33	0.1059499	Median	Desc	49	0.1009805	Maximum	Desc	49	0.1198707
	2	Diagonals	Asc	24	0.1002367	Range	Asc	41	0.1020365	Maximum	Asc	55	0.1281445
	3	Range	Asc	28	0.0981150	Mean	Desc	31	0.1226015	Median	Desc	36	0.1004131
	4	Std. Dev.	Desc	44	0.0881739	Diagonals	Desc	40	0.1018407	Diagonals	Desc	40	0.1043491
	5	Diagonals	Asc	43	0.0979013	Mean	Asc	68	0.1243742	Median	Desc	62	0.1281118
16	1	Median	Desc	47	0.1142566	Maximum	Asc	60	0.1084011	Median	Desc	47	0.1223409
	2	Range	Asc	64	0.0952747	Std. Dev.	Desc	52	0.1191566	Range	Desc	28	0.1472983
	3	Diagonals	Desc	23	0.1209276	Mean	Desc	43	0.1139445	Median	Desc	42	0.1242037
	4	Mean	Asc	52	0.1106885	Maximum	Desc	70	0.1087453	Median	Desc	32	0.1276939
	5	Diagonals	Desc	34	0.0470531	Median	Desc	41	0.1305757	Mean	Asc	43	0.1306794
18	1	Maximum	Asc	25	0.0548527	Std. Dev.	Asc	40	0.1483743	Range	Desc	36	0.1832347
	2	Std. Dev.	Asc	37	0.0568481	Median	Desc	52	0.1427739	Median	Desc	32	0.1759551
	3	Range	Desc	55	0.0588415	Diagonals	Asc	37	0.1669588	Std. Dev.	Desc	69	0.1584294
	4	Mean	Asc	43	0.0628331	Std Dev.	Asc	51	0.1522281	Range	Desc	46	0.1755247
	5	Range	Desc	31	0.0528653	Mean	Asc	54	0.1569455	Range	Desc	31	0.1566219
20	1	Std. Dev.	Desc	53	0.0698125	Diagonals	Desc	40	0.1883280	Maximum	Desc	52	0.2294993
	2	Maximum	Desc	29	0.0802653	Diagonals	Desc	49	0.1882982	Std. Dev.	Asc	64	0.2192030
	3	Median	Desc	58	0.0658586	Median	Desc	77	0.1917918	Mean	Asc	58	0.1976802
	4	Diagonals	Asc	53	0.0703182	Range	Desc	66	0.1934402	Diagonals	Asc	47	0.2138100
	5	Range	Desc	25	0.0678186	Maximum	Desc	58	0.1991711	Range	Asc	39	0.2129843
22	1	Mean	Asc	42	0.0928795	Maximum	Asc	30	0.2470129	Maximum	Asc	44	0.0967760
	2	Maximum	Asc	53	0.0837317	Mean	Desc	41	0.2319818	Std. Dev.	Asc	65	0.2580817
	3	Median	Desc	39	0.0877655	Range	Asc	54	0.2341025	Range	Desc	55	0.2452874
	4	Mean	Desc	43	0.0837760	Mean	Desc	26	0.2202597	Mean	Asc	53	0.0917525
	5	Mean	Desc	47	0.0867689	Std. Dev.	Asc	39	0.2596839	Std. Dev.	Asc	57	0.0907536
24	1	Median	Asc	26	0.1042540	Mean	Asc	49	0.2847967	Std. Dev.	Asc	27	0.1067152
	2	Median	Desc	55	0.0987363	Median	Asc	40	0.2738364	Range	Asc	58	0.1057088
	3	Range	Desc	52	0.0987296	Std. Dev.	Asc	34	0.2768462	Std. Dev.	Desc	57	0.1056824
	4	Median	Desc	72	0.1017270	Mean	Asc	62	0.2795932	Std. Dev.	Desc	27	0.1047204
	5	Std Dev.	Desc	45	0.1062431	Mean	Desc	41	0.2797968	Unsorted	Desc	47	0.1082633
26	1	Unsorted	—	44	0.1361423	Unsorted	—	60	0.3399053	Mean	Asc	39	0.1266959
	2	Median	Desc	32	0.1186829	Maximum	Asc	54	0.3262534	Median	Asc	64	0.1286516
	3	Std. Dev.	Asc	62	0.1196902	Maximum	Asc	30	0.3403540	Median	Asc	52	0.1311564
	4	Diagonals	Desc	43	0.1216748	Diagonals	Desc	59	0.3425541	Mean	Asc	68	0.1276686
	5	Diagonals	Asc	63	0.1216748	Diagonals	Desc	50	0.3457615	Std. Dev.	Desc	33	0.1206770

Definition 17. Let A be an element of $\mathcal{M}^{n^2}$.

1. **Frobenius norm**: $\| A \|_F$ is equal to the square root of the sum of the squares of the coefficients of A. $\| A \|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2}$
2. **1-norm**: $\| A \|_1$ equals the maximum of the sum of the absolute values of the coefficients of the columns of A. $\| A \|_1 = \max\limits_{\|x\|=1} \| Ax \|_1 = \max\limits_{1 \leq j \leq n} \sum_{i=1}^{n} | a_{ij} |$
3. **Infinity norm**: $\| A \|_\infty$ is the maximum of the sum of the absolute values of the coefficients of the rows of A. $\| A \|_\infty = \max\limits_{\|x\|=1} \| Ax \|_\infty = \max\limits_{1 \leq i \leq n} \sum_{j=1}^{n} | a_{ij} |$
4. **2-norm**: $\| A \|_2$ is the square root of the dominant eigenvalue of the inner product of $A^T A$. $\| A \|_2 = \max\limits_{\|x\|=1} \| Ax \|_2 = \max\limits_{1 \leq i \leq n} \sqrt{\lambda_i (A^T A)}$

For comparative analysis, Category 2 heuristics are implemented on the same 3IAP instance of size 10, operating across the first-tie, last-tie, and random-tie cases. The implementation results in 11 solutions per case, documented in Tables 4.6, 4.7, and 4.8, respectively. In each row, the best solution found is highlighted by underlining, showcasing the efficiency of these heuristics.

Table 4.6: Solutions for a 3IAP instance of size 10, selecting the first tie-case.

Index	**Sort Type**	**Order**	**Cost**	**Runtime**
1	Unsorted	—	133	0.00423265
2	Diagonals	Ascending	107	0.01365447
3	Diagonals	Descending	87	0.00911498
4	Frobenius	Ascending	71	0.01427603
5	Frobenius	Descending	58	0.01166511
6	1-norm	Ascending	83	0.01676869
7	1-norm	Descending	56	0.00988197
8	Infinity	Ascending	83	0.02835202
9	Infinity	Descending	69	0.02092838
10	2-norm	Ascending	122	0.01850247
11	2-norm	Descending	$\underline{55}$	$\underline{0.00955343}$

Tables 4.6, 4.7, and 4.8 display eight additional solutions from Category 2 heuristics for an instance of size 10, compared with GSP and DM results where the best solutions are highlighted. Category 2 heuristics outperform the basic DM

Table 4.7: Solutions for a 3IAP instance of size 10, selecting the last tie-case.

Index	**Sort Type**	**Order**	**Cost**	**Runtime**
1	Unsorted	—	36	0.01769304
2	Diagonals	Ascending	107	0.02649331
3	Diagonals	Descending	121	0.03093147
4	Frobenius	Ascending	104	0.02595663
5	Frobenius	Descending	58	0.01140642
6	1-norm	Ascending	90	0.02021837
7	1-norm	Descending	56	0.01440573
8	Infinity	Ascending	$\underline{27}$	$\underline{0.02520680}$
9	Infinity	Descending	69	0.01849794
10	2-norm	Ascending	92	0.01979470
11	2-norm	Descending	54	0.01054478

Table 4.8: Solutions for a 3IAP instance of size 10, selecting a random tie.

Index	**Sort Type**	**Order**	**Cost**	**Runtime**
1	Unsorted	—	133	0.01906562
2	Diagonals	Ascending	107	0.02649331
3	Diagonals	Descending	121	0.03093147
4	Frobenius	Ascending	104	0.01951051
5	Frobenius	Descending	58	0.01783252
6	1-norm	Ascending	166	0.02023554
7	1-norm	Descending	56	0.00896358
8	Infinity	Ascending	$\underline{27}$	$\underline{0.02271700}$
9	Infinity	Descending	69	0.02086163
10	2-norm	Ascending	122	0.02661896
11	2-norm	Descending	54	0.02083349

version, showing improvements in objective function values and computational efficiency, illustrating the effectiveness of these heuristics. Table 4.9 summarises the best solutions found within the three previous tables.

Table 4.9: Summary of best solutions for a 3IAP instance of size 10.

Case	**Sort Type**	**Order**	**Cost**	**Runtime**
First tie-case	2-Norm	Descending	55	0.00955343
Last tie-case	Infinity	Ascending	27	0.02520680
Random tie	Infinity	Ascending	<u>27</u>	<u>0.02271700</u>

Table 4.10 presents the results from applying Category 2 heuristics to Sample 1, with computational tests mirroring those in Table 4.5. The best-found solution for each 3IAP instance is underlined. Out of the 60 premier solutions identified, only eleven are the outcome of DM, with the remainder arising from Category 2 heuristics. Combining data from Tables 4.5 and 4.10 yields solutions superior to those achieved through DM alone. Furthermore, even when cost values are identical for specific instances, solutions obtained by these heuristics feature reduced computational time.

The approach generates two sets of feasible solutions for 3IAP, including those derived from the DM and GSP. Among 21 feasible solutions, Table 4.11 displays the best result obtained from both heuristic categories applied to Sample 1. Thus, for every 3IAP instance within this dataset, a quality solution is achieved within a competitive timeframe, offering a dependable approach to solving the assignment problem.

Table 4.12 delineates the newly proposed heuristics for 3IAP, and the rearranged cost matrices according to two criteria: statistical measures and matrix norms.

Table 4.10: Outcomes of Category 2 Heuristics (best solutions found are underlined).

		First tie-case				**Last tie-case**				**Random tie-case**			
Size	Instance	Type	Order	Cost	Runtime	Type	Order	Cost	Runtime	Type	Order	Cost	Runtime
4	1	Unsorted	—	46	0.0169406	Unsorted	—	46	0.0010352	Unsorted	—	46	0.0010238
	2	Diagonals	Desc	51	0.0202684	Diagonals	Desc	51	0.0043216	Diagonals	Desc	51	0.0033302
	3	Unsorted	—	54	0.0219405	2-Norm	Desc	54	0.0010223	Unsorted	—	54	0.0069017
	4	Frobenius	Asc	68	0.0237219	Frobenius	Asc	71	0.0075302	2-Norm	Asc	71	0.0081911
	5	1-Norm	Asc	40	0.0222788	Infinity	Asc	22	0.0080113	Frobenius	Desc	22	0.0070922
6	1	Diagonals	Desc	65	0.0166636	Diagonals	Desc	65	0.0103643	Diagonals	Desc	65	0.0160758
	2	Frobenius	Asc	42	0.0190020	Frobenius	Asc	42	0.0117474	Frobenius	Asc	42	0.0150635
	3	Frobenius	Desc	48	0.0179522	Frobenius	Desc	48	0.0123291	Frobenius	Desc	48	0.0146244
	4	2-Norm	Desc	50	0.0378447	2-Norm	Desc	50	0.0109694	2-Norm	Desc	50	0.0146244
	5	Infinity	Asc	45	0.0270686	Infinity	Asc	45	0.0120251	Infinity	Asc	45	0.0159576
8	1	1-Norm	Desc	37	0.0246043	1-Norm	Desc	37	0.0139608	1-Norm	Desc	37	0.0010242
	2	1-Norm	Desc	49	0.0176947	1-Norm	Desc	49	0.0136608	1-Norm	Desc	49	0.0036578
	3	Infinity	Asc	35	0.02398442	1-Norm	Desc	47	0.0139973	Unsorted	—	46	0.0143979
	4	Frobenius	Desc	57	0.0392866	2-Norm	Asc	56	0.0042777	2-Norm	Asc	56	0.0127425
	5	1-Norm	Desc	8	0.0116396	1-Norm	Desc	8	0.0035248	1-Norm	Desc	8	0.0146234
10	1	2-Norm	Asc	71	0.0622561	Diagonals	Asc	48	0.0169549	2-Norm	Asc	71	0.0219409
	2	Infinity	Desc	40	0.0678885	2-Norm	Desc	47	0.0166063	Diagonals	Desc	53	0.0188389
	3	2-Norm	Desc	55	0.00955343	Infinity	Asc	27	0.02520680	Infinity	Asc	27	0.02271700
	4	2-Norm	Desc	44	0.0675690	2-Norm	Desc	44	0.0169876	2-Norm	Desc	44	0.0165770
	5	Infinity	Desc	79	0.0411978	Unsorted	—	49	0.0179517	2-Norm	Asc	43	0.0197444
12	1	1-Norm	Desc	59	0.0701144	Unsorted	—	47	0.0200324	1-Norm	Desc	59	0.0356193
	2	Unsorted	—	77	0.0310483	Infinity	Desc	29	0.0189824	Unsorted	—	59	0.0249324
	3	Infinity	Desc	60	0.0712464	Frobenius	Asc	28	0.0194366	Infinity	Desc	51	0.0110812
	4	1-Norm	Desc	40	0.0324903	Infinity	Asc	56	0.0233817	Frobenius	Asc	48	0.0222557
	5	Unsorted	—	24	0.0301309	Unsorted	—	42	0.0203908	2-Norm	Desc	37	0.0255072
14	1	Diagonals	Asc	66	0.0860665	Frobenius	Desc	46	0.0239670	Diagonals	Asc	35	0.0315444
	2	Diagonals	Asc	24	0.0957935	1-Norm	Desc	40	0.0209870	2-Norm	Desc	39	0.0297997
	3	1-Norm	Asc	90	0.0504882	2-Norm	Desc	42	0.0094311	1-Norm	Desc	38	0.0313828
	4	Diagonals	Desc	61	0.0547142	Diagonals	Desc	40	0.0110421	Unsorted	—	56	0.0350776
	5	Infinity	Asc	29	0.0520158	2-Norm	Desc	48	0.0216281	Diagonals	Desc	45	0.0413561
16	1	Infinity	Desc	34	0.064518	Diagonals	Desc	60	0.0292914	Diagonals	Desc	40	0.0228353
	2	Infinity	Desc	42	0.0678213	Infinity	Desc	34	0.0294390	Infinity	Desc	42	0.0267251
	3	Diagonals	Desc	23	0.0832505	1-Norm	Desc	42	0.0248818	Frobenius	Asc	68	0.0322201
	4	1-Norm	Desc	69	0.0899043	Frobenius	Desc	66	0.0230651	Infinity	Desc	48	0.0317247
	5	Diagonals	Desc	34	0.0840192	Diagonals	Desc	61	0.0298672	Infinity	Asc	43	0.0348649
18	1	Diagonals	Asc	51	0.0941238	Unsorted	—	68	0.0190866	2-Norm	Asc	47	0.0378969
	2	2-Norm	Desc	30	0.0986674	Frobenius	Desc	35	0.0347402	Unsorted	—	46	0.0349045
	3	Frobenius	Desc	46	0.1041887	Diagonals	Asc	37	0.0251563	Diagonals	Desc	33	0.0418868
	4	1-Norm	Desc	56	0.0944135	1-Norm	Desc	62	0.0309236	2-Norm	Asc	47	0.0388122
	5	Diagonals	Asc	47	0.0936954	Infinity	Desc	42	0.0311666	Diagonals	Desc	53	0.0366027
20	1	Frobenius	Desc	18	0.0948539	Infinity	Desc	36	0.0369012	2-Norm	Desc	26	0.0404551
	2	Diagonals	Asc	30	0.0942195	Frobenius	Asc	33	0.0341244	Diagonals	Asc	45	0.0524790
	3	2-Norm	Asc	57	0.1218150	2-Norm	Desc	62	0.0340006	2-Norm	Asc	58	0.0468812
	4	Infinity	Asc	38	0.1433766	2-Norm	Desc	47	0.0348995	Unsorted	—	51	0.0419254
	5	2-Norm	Asc	33	0.1541805	Unsorted	—	67	0.0359197	Diagonals	Desc	64	0.0388210
22	1	Diagonals	Asc	47	0.1413312	2-Norm	Asc	43	0.0442212	1-Norm	Asc	32	0.0522907
	2	1-Norm	Desc	73	0.1015856	1-Norm	Asc	57	0.0418875	Diagonals	Desc	42	0.0478554
	3	1-Norm	Asc	63	0.1750429	Infinity	Desc	65	0.0422199	2-Norm	Desc	68	0.0506885
	4	Frobenius	Desc	35	0.1676817	Unsorted	—	47	0.04038048	2-Norm	Asc	32	0.0519724
	5	1-Norm	Desc	62	0.1233559	Frobenius	Desc	43	0.0414922	1-Norm	Desc	44	0.0513620
24	1	2-Norm	Desc	54	0.1429453	2-Norm	Asc	45	0.0431852	Infinity	Asc	36	0.0596867
	2	Infinity	Asc	41	0.1293378	Unsorted	—	59	0.0385926	2-Norm	Desc	57	0.0737224
	3	Infinity	Asc	40	0.1985698	1-Norm	Desc	69	0.0513072	Diagonals	Asc	57	0.0619988
	4	1-Norm	Desc	51	0.2304580	2-Norm	Desc	67	0.0448926	Infinity	Asc	36	0.0613723
	5	1-Norm	Asc	25	0.2360208	Unsorted	—	46	0.0456920	Diagonals	Desc	48	0.0615945
26	1	Unsorted	—	44	0.1410162	Frobenius	Desc	40	0.0498521	Infinity	Desc	43	0.0683939
	2	Unsorted	—	66	0.1382613	2-Norm	Desc	60	0.0530870	Frobenius	Desc	59	0.0679481
	3	1-Norm	Desc	60	0.1403570	Frobenius	Desc	46	0.0462112	Diagonals	Asc	67	0.0696464
	4	Diagonals	Desc	43	0.1330237	Diagonals	Desc	59	0.0493228	Unsorted	—	51	0.0611022
	5	Frobenius	Desc	26	0.1785924	Diagonals	Desc	50	0.0518575	Frobenius	Desc	37	0.0625734

Table 4.11: Best solutions from both category heuristics implemented on Sample 1.

		Category 1					**Category 2**				
Size	Instance	Type	Order	Cost	Runtime	Tie-case	Type	Order	Cost	Runtime	Tie-case
4	1	Maximum	Desc	46	0.0079737	Rand	Unsorted	—	46	0.0010238	**Rand**
	2	Diagonals	Desc	51	0.0079789	Rand	Diagonals	Desc	51	0.0079789	Rand
	3	Unsorted	—	54	0.0080109	Rand	2-Norm	Desc	54	0.0010223	Rand
	4	Mean	Asc	68	0.0089753	Rand	Frobenius	Asc	68	0.0237219	Rand
	5	Std. Dev.	Desc	22	0.0089760	Rand	Frobenius	Desc	22	0.0070922	Rand
6	1	Median	Desc	42	0.0119770	Rand	Diagonals	Desc	65	0.0103643	Last
	2	Unsorted	—	51	0.0119681	Rand	Frobenius	Asc	42	0.0117474	**Last**
	3	Median	Desc	42	0.0119572	**Rand**	Frobenius	Desc	48	0.0123291	Last
	4	Median	Desc	50	0.0119636	Rand	2-Norm	Desc	50	0.0109694	Last
	5	Range	Desc	28	0.0119267	**Rand**	Infinity	Asc	45	0.0120251	Last
8	1	Median	Desc	36	0.0465174	Last	1-Norm	Desc	37	0.0010242	Rand
	2	Maximum	Desc	44	0.0426154	First	1-Norm	Desc	49	0.0036578	Rand
	3	Std Dev.	Asc	35	0.0169549	Rand	Infinity	Asc	35	0.0239842	First
	4	Median	Asc	56	0.0534122	Rand	2-Norm	Asc	56	0.0042777	Last
	5	Diagonals	Asc	8	0.0399439	First	1-Norm	Desc	8	0.0035248	**Last**
10	1	Maximum	Desc	37	0.0479422	First	Diagonals	Asc	48	0.0169549	Last
	2	Range	Asc	43	0.0711098	Rand	Infinity	Desc	40	0.0678885	First
	3	Maximum	Desc	27	0.0699704	Rand	Infinity	Asc	27	0.0092754	Rand
	4	Median	Desc	32	0.0554729	First	2-Norm	Desc	44	0.0165770	Rand
	5	Median	Asc	35	0.0667386	Last	2-Norm	Asc	43	0.0197444	Rand
12	1	Range	Asc	39	0.0839060	Last	Unsorted	—	47	0.0200324	Last
	2	Std Dev.	Asc	53	0.0730062	Last	Infinity	Desc	29	0.0189824	Last
	3	Mean	Desc	36	0.0703671	First	Frobenius	Asc	28	0.0194366	Last
	4	Std Dev.	Asc	63	0.0789812	Last	1-Norm	Desc	40	0.0324903	First
	5	Unsorted	—	24	0.0794580	First	Unsorted	—	24	0.0301309	First
14	1	Mean	Asc	33	0.1059499	First	Diagonals	Asc	35	0.0315444	Rand
	2	Diagonals	Asc	24	0.1002367	First	Diagonals	Asc	24	0.0957935	First
	3	Range	Asc	28	0.0981150	First	1-Norm	Desc	38	0.0313828	Rand
	4	Diagonals	Desc	40	0.1018407	Last	Diagonals	Desc	40	0.0110421	Last
	5	Diagonals	Asc	43	0.0979013	First	Infinity	Asc	29	0.0520158	First
16	1	Median	Desc	47	0.1142566	First	Infinity	Desc	34	0.0645180	First
	2	Range	Desc	28	0.1472983	Rand	Infinity	Desc	34	0.0294390	Last
	3	Diagonals	Desc	23	0.1209276	First	Diagonals	Desc	23	0.0832505	First
	4	Median	Desc	32	0.1276939	Rand	Infinity	Desc	48	0.0317247	Last
	5	Diagonals	Desc	34	0.0470531	First	Diagonals	Desc	34	0.0840192	First
18	1	Maximum	Asc	25	0.0548527	First	2-Norm	Asc	47	0.0378969	Rand
	2	Median	Desc	32	0.1759551	Rand	2-Norm	Desc	30	0.0986674	First
	3	Diagonals	Asc	37	0.1669588	Last	Diagonals	Desc	33	0.0418868	Rand
	4	Mean	Asc	43	0.0628331	First	2-Norm	Asc	47	0.0388122	Rand
	5	Range	Desc	31	0.0528653	First	Infinity	Desc	42	0.0311666	Last
20	1	Diagonals	Desc	40	0.1883280	Last	Frobenius	Desc	18	0.0948539	First
	2	Maximum	Desc	29	0.0802653	First	Diagonals	Asc	30	0.0942195	First
	3	Median	Desc	58	0.0658586	First	2-Norm	Asc	57	0.1218150	First
	4	Diagonals	Asc	47	0.2138100	Rand	Infinity	Asc	38	0.1433766	First
	5	Range	Desc	25	0.0678186	First	2-Norm	Asc	33	0.1541805	First
22	1	Maximum	Asc	30	0.2470129	Last	1-Norm	Asc	32	0.0522907	Rand
	2	Mean	Desc	41	0.2319818	Last	Diagonals	Desc	42	0.0478554	Rand
	3	Median	Desc	39	0.0877655	First	1-Norm	Asc	63	0.1750429	First
	4	Mean	Desc	26	0.2202597	Last	2-Norm	Asc	32	0.0519724	Rand
	5	Std Dev.	Asc	39	0.2596839	Last	Frobenius	Desc	43	0.0414922	Last
24	1	Median	Asc	26	0.1042540	First	Infinity	Asc	36	0.0596867	Rand
	2	Median	Asc	40	0.2738364	Last	Infinity	Asc	41	0.1293378	First
	3	Std Dev.	Asc	34	0.2768462	Last	Infinity	Asc	40	0.1985698	First
	4	Std Dev.	Desc	27	0.1047204	Rand	Infinity	Asc	36	0.0613723	Rand
	5	Mean	Desc	41	0.2797968	Last	1-Norm	Asc	25	0.2360208	First
26	1	Mean	Asc	39	0.1266959	Rand	Frobenius	Desc	40	0.0498521	Last
	2	Median	Desc	32	0.1186829	First	Frobenius	Desc	59	0.0679481	Rand
	3	Maximum	Asc	30	0.3403540	Last	Frobenius	Desc	46	0.0462112	Last
	4	Diagonals	Desc	43	0.1216748	First	Diagonals	Desc	43	0.1330237	First
	5	Std Dev.	Desc	33	0.1206770	Rand	Frobenius	Desc	26	0.17859244	First

The extensive computational tests indicate that the best solutions found are of good quality compared to DM results. Again, if the found solution is optimal or near-optimal, it will be accepted; otherwise, it will be used as a warm start for the B&B to produce a better solution.

Table 4.12: List of proposed heuristics for 3IAP.

	DM	**GSP**	**Category 1**	**Category 2**
Sorting criteria	Diagonals	—	Statistical	Matrix norms
	Diagonal factor		Mean	Frobenius
			Median	1-norm
			Maximum	Infinity
			Range	2-norm
			Std. Deviation	

4.3 Exact Solution Approaches

4.3.1 Branch & Bound and Hybrid DM

Heuristics from both Category 1 and Category 2 consistently provide feasible solutions for 3IAP, with a possibility of optimality. When the discrepancy between the linear program lower bound and its dual upper bound is zero or negligible, the identified solution is deemed optimal or near-optimal, respectively. In cases where this gap is wider, the derived solution serves as a warm start for the B&B method. The integration of DM with B&B leads to optimal or nearly optimal outcomes, outperforming the standard B&B approach as Table 4.13 demonstrates.

Upon conducting numerical experiments on Sample 1 dataset, Kadhem (2017) observes that a Hybridised DM reaches optimal costs akin to the B&B method in significantly reduced computation time, provided the problem size does not exceed 26. The author reported that any attempt to implement B&B on larger instances has consistently failed, highlighting a limitation in scalability for larger problems.

Table 4.13: Computational results of Hybridised DM on Sample 1

Instances	**B&B**		**DM**		**Hybrid DM**	
size n	Cost	Time	Cost	Time	Cost	Time
4	49.42	0.2530	94.28	0.0164	49.42	0.0490
6	36.64	0.3290	90.76	0.0128	36.64	0.1893
8	24.42	0.9430	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

After developing and implementing heuristic methods for 3IAP that share the DM structure, it is essential to assess their performance. Their evaluation should include comparing the heuristic solutions with those generated by exact algorithms to ascertain their effectiveness.

4.3.2 Gurobi Optimizer

For the optimal resolution of 3IAP, a computer program integrating the *Gurobi Optimization* solver (Gurobi, 2024) is designed and implemented. The Gurobi Optimizer, renowned for its cutting-edge capabilities in mathematical programming, employs the latest in algorithms and architectures. The program relies on the third 3IAP formulation LP2, implemented through the Python Notebook interface for Gurobi Optimizer.

The program performance is assessed using Sample 1 as the benchmark dataset. Table 4.14 presents the outcomes of the Gurobi Optimizer compared to the results of (Balas and Saltzman 1991). Each table row outlines the average cost and the average execution time of five instances of the same size. The findings indicate that Gurobi achieves optimal solutions comparable to those of Balas and Saltzman but with reduced CPU time.

Table 4.14: Comparison of the Gurobi outcomes with Balas and Saltzman's results.

	B & S		**Gurobi**	
Size	Cost	CPU	Cost	CPU
4	42.20	0.03	42.20	0.03671927
6	40.20	0.16	40.20	0.06198630
8	23.80	0.84	23.80	0.16521101
10	19.00	1.36	19.00	0.41342755
12	15.60	2.19	15.60	1.07067790
14	10.00	11.95	10.00	2.65978527
16	10.00	39.90	10.00	5.65702610
18	6.40	55.30	6.40	11.27514129
20	4.80	169.29	4.80	22.13064094
22	4.00	371.52	4.00	38.56125860
24	1.80	514.52	1.80	64.59890308
26	1.30	624.00	1.00	107.41192837

Let Sample 2 be a second random sample generated by the uniform probability from 0 to 100, of size $n = 5, 10, 15, \ldots, 40$; five instances per size, except the fifth instance from 100 to 999. Numerical experiments are conducted on Sample 2; the results are presented in Table 4.15.

The performance of the proposed program is evaluated through computational tests on larger instances, using a random sample of sizes up to 35. The program attained optimal solutions for these instances, demonstrating efficiency with a satisfactory runtime. Similar computational testing could be performed on instances of larger sizes but obviously for a longer execution time.

4.3.3 Python-MIP solver

After implementing and testing a first program incorporating the Gurobi Optimizer, a second was developed for solving optimally 3IAP, using a specific package, Python-MIP, an advanced library for Mixed-Integer Linear Programming problems, facilitating their modelling and resolution.

Table 4.15: Gurobi results using Sample 2

Size	Instance	Cost	Runtime
5	1	53	0.04751706
	2	25	0.03941488
	3	44	0.06288433
	4	22	0.04488349
	5	495	0.05727077
10	1	29	0.46303821
	2	37	0.41437745
	3	29	0.43761158
	4	33	0.40377784
	5	321	0.45508575
15	1	37	4.10386038
	2	23	4.00404882
	3	26	4.07738948
	4	24	4.12160254
	5	276	4.16600680
20	1	23	20.98679924
	2	24	21.75483418
	3	24	23.07856250
	4	23	23.40851784
	5	343	23.75501657
25	1	251	81.31217790
	2	251	83.30688334
	3	250	82.94617462
	4	250	93.56074834
	5	363	83.23414493
30	1	300	248.29142761
	2	300	261.12135029
	3	300	256.22693014
	4	300	257.87097788
	5	362	256.32356429
35	1	350	684.11812282
	2	350	878.05114627
	3	350	700.88513374
	4	350	677.22385907
	5	395	699.50055337
40	1	400	1,467.45966268
	2	400	1,523.32924628
	3	400	1,506.79422569
	4	400	1,470.06956792
	5	463	1,460.10495377

Two Brazilian computing researchers (Toffolo, T.A.M. and Santos, H.G., 2019) designed MIP as a fast open source which works with the Gurobi Optimization solver. Another computing program was proposed to solve 3IAP optimally by adapting the MIP solver through the Python programming language and using the first mathematical formulation LP1 of 3IAP.

Table 4.16 presents a comparative analysis between the Gurobi Optimizer and the Python-MIP program, using the same dataset, namely Sample 1. Each row in the table summarises the average results of five instances of the same size. The last column shows the percentage reduction in CPU time. The MIP program performance surpasses that of the Gurobi solver, as evidenced by the savings in computational time.

The MIP program implementation demonstrates excellent outcomes, producing optimal solutions identical to those generated by Gurobi but with significantly reduced runtime. Computational tests conducted on Sample 2 are detailed in Table 4.17, which compares the results from Gurobi with the MIP program. Figure 4.1 illustrates the average computation times for both Gurobi and MIP across five instances of the same size, ranging from $n = 5$ to 35, indicating an enhanced efficiency of the MIP program.

More numerical tests were conducted on random data of Sample 3, which is Sample 2 extended to the size $n = 80$. Sample 3 is composed of five instances of size $n = 40, 45, \ldots, 80$, generated by uniform probability between 10 and 999, and the test results are displayed in Table 4.18. Table 4.19 summarises the algorithms execution time for different random samples. The MIP performance exceeds that of the proposed heuristics and Gurobi program, as its execution time is significantly reduced from 10% to 90%. For example, when running 3IAP instances of size $n = 26$ and $n = 40$, Gurobi takes an average of fewer than 1.5 minutes and 24 minutes, respectively while MIP requires only 0.15 minutes and less than 1 minute,

Table 4.16: Comparison of Gurobi and MIP performances on Sample 1.

	Gurobi		MIP		
Size	Cost	Time	Cost	Time	% Saved time
4	42.20	0.03671927	42.20	0.06625228	–
6	40.20	0.06198630	40.20	0.09180932	–
8	23.80	0.16521101	23.80	0.14813294	10%
10	19.00	0.41342755	19.00	0.20423903	51%
12	15.60	1.07067790	15.60	0.29895229	72%
14	10.00	2.65978527	10.00	0.54588880	79%
16	10.00	5.65702610	10.00	0.93958402	83%
18	6.40	11.27514129	6.40	1.48142543	87%
20	4.80	22.13064094	4.80	2.61848969	88%
22	4.00	38.56125860	4.00	4.26741615	89%
24	1.80	64.59890308	1.80	6.88015480	89%
26	1.00	107.41192837	1.00	10.96012244	90%

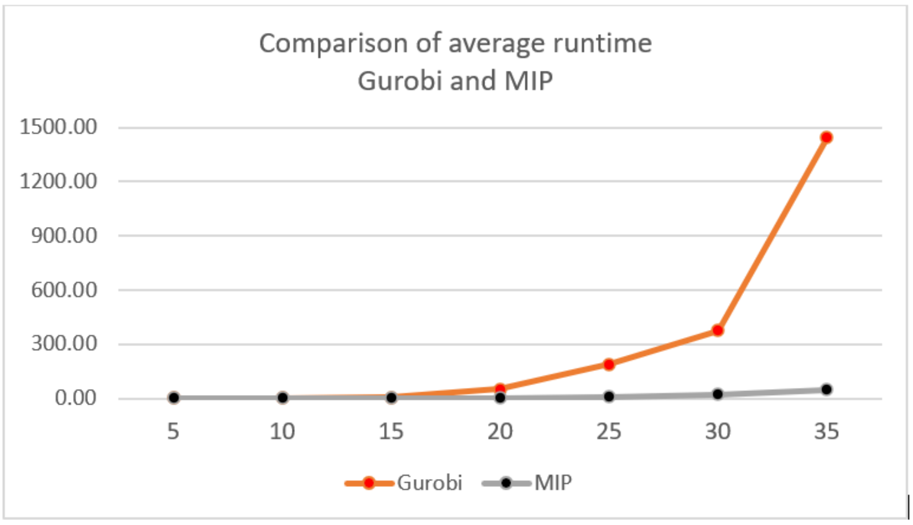

Figure 4.1: Average runtime of Gurobi and MIP.

equivalent to 90% and 97% reduction in running time, respectively. Hence, when executing 3IAP instances of size $n = 60$, the MIP program running time is less than 28 minutes. Further tests are planned for larger instances.

By exploring specific features of cost matrices, two new heuristic classes for solving 3IAP are proposed. The first class relies on statistical metrics, and the second is based on matrix norms. Extensive computational experiments support their efficiency, and the heuristics produce, within polynomial time, 21 outcomes that are optimal or of superior quality.

Table 4.17: Comparing Gurobi and MIP results using Sample 2 data.

		Gurobi		**MIP**	
Size	Instance	Cost	Runtime	Cost	Runtime
5	1	53	0.04751706	53	0.05464196
	2	25	0.03941488	25	0.05940747
	3	44	0.06288433	44	0.06146312
	4	22	0.04488349	22	0.06511664
	5	495	0.05727077	495	0.07313919
10	1	29	0.46303821	29	0.16452599
	2	37	0.41437745	37	0.11249900
	3	29	0.43761158	29	0.10823774
	4	33	0.40377784	33	0.11964583
	5	321	0.45508575	321	0.10815573
15	1	37	4.10386038	37	0.54287481
	2	23	4.00404882	23	0.59983540
	3	26	4.07738948	26	0.58981586
	4	24	4.12160254	24	0.58864856
	5	276	4.16600680	276	0.51120210
20	1	23	20.98679924	23	2.12738681
	2	24	21.75483418	24	2.32940912
	3	24	23.07856250	24	2.24340868
	4	23	23.40851784	23	2.21933031
	5	343	23.75501657	343	2.47173333
25	1	251	81.31217790	251	7.36894393
	2	251	83.30688334	251	7.20444775
	3	250	82.94617462	250	7.28113747
	4	250	93.56074834	250	7.40094900
	5	363	83.23414493	363	7.16370320
30	1	300	248.29142761	300	19.96155977
	2	300	261.12135029	300	19.91077828
	3	300	256.22693014	300	20.17518854
	4	300	257.87097788	300	20.64884663
	5	362	256.32356429	362	19.94497132
35	1	350	684.11812282	350	48.55294108
	2	350	878.05114627	350	47.78489661
	3	350	700.88513374	350	48.76661682
	4	350	677.22385907	350	48.96539617
	5	395	699.50055337	395	50.61369872
40	1	400	1,467.45966268	400	104.30422497
	2	400	1,523.32924628	400	103.74304414
	3	400	1,506.79422569	400	104.40737677
	4	400	1,470.06956792	400	106.03453565
	5	463	1,460.10495377	463	111.46146727

Table 4.18: MIP outcomes using random data of Sample 3.

MIP results using Sample 3.			
Size	Instance	Cost	Runtime
40	1	400	104.30422497
	2	400	103.74304414
	3	400	104.40737677
	4	400	106.03453565
	5	463	111.46146727
45	1	450	211.84999800
	2	450	210.61439395
	3	450	210.89135790
	4	450	218.12571907
	5	489	288.36444283
50	1	500	417.25897503
	2	500	445.65421462
	3	500	1,141.88306355
	4	500	1,121.17579913
	5	532	1,474.76695253
55	1	550	930.13321471
	2	550	754.00689173
	3	550	715.45358729
	4	550	708.25755429
	5	583	1,013.19422102
60	1	600	1,223.09543967
	2	600	1,225.40235543
	3	600	1,211.75682592
	4	600	1,221.94683933
	5	619	3,466.80567312

Table 4.19: Runtime in minutes.

		Gurobi	**MIP**	
	Size	Average runtine (in min.)		Time saved (%)
Sample 1	26	1.79	0.183	90
Sample 2	40	24.238	1.766	93
Sample 3	60	—	27.830	—

Finally, two computer programs, Gurobi and Python–MIP, are implemented for comparison purposes, achieving optimal solutions for large instances in competitive computational time. Both programs produced optimal solutions, compared with benchmark data and those randomly generated, but the MIP performance far outstrips that of Gurobi.

The literature indicates that the largest 3IAP instance solved to optimality is of size $n = 26$. By implementing Gurobi and MIP programs, we have optimally solved 3IAP instances of sizes $n = 40$, $n = 60$, and potentially higher sizes.

The next chapter investigates evolutionary algorithms, showing the genetic algorithm capability to solve *NP*-hard optimisation problems. The study utilises the problem permutation-based formulation to propose a new hybrid genetic algorithm tailored to 3IAP.

Chapter 5

Genetic Algorithm

This chapter of the thesis introduces the Genetic Algorithm (GA) which provides an approximate solution approach deeply rooted in biological evolution and natural selection principles. GA is an evolutionary algorithm-based strategy for solving complex optimisation problems. GA is applied to 3IAP to find optimal or near-optimal solutions by evolving a population of candidate solutions over multiple generations. While the basic GA produces results, this study seeks to enhance their quality by integrating a local search method.

The work presents the genetic algorithm followed by two local search approaches for 3IAP. The possibility of incorporating a local search method within the standard GA is discussed in this chapter. Any solution to 3IAP of size n can be defined by three permutations of n elements applied to cost matrices. Their order's insignificance permits their rearrangement, aligning one with the identity permutation. Consequently, a 3IAP solution can be formulated with only two permutations, simplifying the representation without compromising solution integrity.

$$\text{Minimize} \qquad Z = \sum_{i=1}^{n} c_{ip(i)q(i)}$$

$$\text{subject to} \qquad p, q \quad \text{two permutations of} \quad \pi_n,$$

where π_n is the set of all permutations of n elements.

5.1 Evolutionary Algorithm

Evolutionary algorithms, founded on heuristics, represent an effective strategy for solving NP-Hard problems characterised by their intractable complexity. Genetic algorithms serve as a preliminary heuristic step when applied independently to challenging combinatorial problems. Their role is to effectively identify a reasonable starting point, facilitating subsequent algorithms for refined exploration. This strategic combination harnesses the heuristic exploration of genetic algorithms, synergising with subsequent algorithms to navigate intricate problem spaces efficiently.

An evolutionary algorithm, rooted in the principles of biological evolution and natural selection, comprises four pivotal stages: initialisation, selection, genetic operators, and termination. The algorithm allows only fitter individuals to endure and propagate, while unfit members are discarded, akin to natural selection. The emulation of evolutionary dynamics signifies the evolutionary algorithm efficiency in navigating complex problem spaces.

5.2 Genetic Algorithm Description

GA is an approximation method based on the principles of biological evolution and natural selection. Initially introduced by (Holland, 1975), GA has emerged as a prominent metaheuristic for tackling intricate combinatorial optimisation challenges, (Cvetkovic and Parmee, 2002; Kim *et al.*, 2003). GA operates as a metaheuristic by incorporating evolutionary concepts of natural selection and genetics, simulating the *'survival of the fittest'* principle laid out by Darwin (1859). This approach has demonstrated remarkable efficacy across diverse combinatorial optimisation problems.

Since GAs are optimisation techniques inspired by natural selection, they maintain a population of potential solutions and evolve them through genetic operators. GAs find applications in diverse domains, including engineering, operations research, business, science, learning machine and artificial intelligence. The approach emulates natural selection, and GAs are particularly valuable for complex problems where finding optimal or near-optimal solutions is otherwise computationally infeasible. The interested reader is referred to (Sivanandam and Deepa, 2008).

GA acts as a heuristic search method which does not guarantee a global optimal solution. Nevertheless, the algorithm efficiently attains an optimal or a high-quality approximate solution within a reasonable CPU time. GA operates iteratively, generating a new solution set with improved solutions from an initial set randomly generated. The algorithm relies on three genetic operators — selection, crossover, and mutation. Following each iteration, a new generation of candidate solutions is reproduced, and the fitness value of each individual is computed.

Despite multiple advantages of GA, like the ability to explore a wider search space, and to deal with complex fitness functions and multiple local optima, GA encounters several barriers, such as the possible loss of genetic properties and premature or slow convergence (Drezner and Misevičius, 2013). To mitigate these drawbacks, researchers have explored strategies such as fine-tuning the parameters of GA (Schaffer *et al.*, 1989), altering genetic operators (Drezner and Marcoulides, 2003), (Fox and MacMahon, 1991), (Wu, 2007) and introducing new characteristics (Misevičius, 2008). Others have opted for a hybrid approach by combining GA with a local search method.

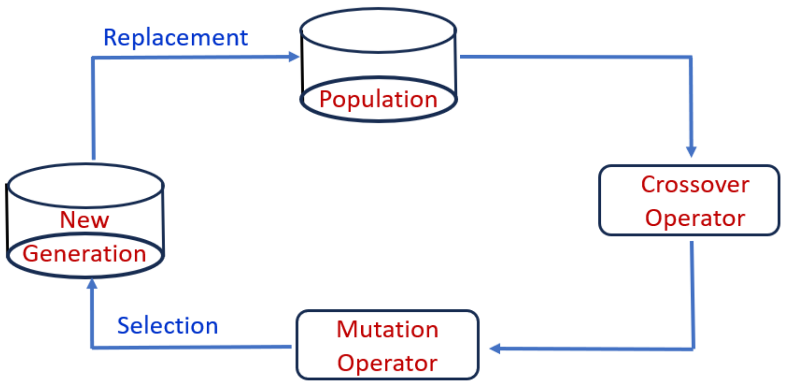

Figure 5.1: The structure of the Genetic Algorithm.

5.2.1 Genetic Algorithm Overview

GA commences with an initial population of randomly generated feasible solutions, called '*chromosomes*', which are subsequently encoded. A fitness function is defined to assess the fitness value of each chromosome, leading to population sorting based on fitness. The fitness function can be proportional or reciprocal to the problem objective function (Wang, 2014).

GA reproduces a new generation of chromosomes from the initial population following the next steps. Evolution starts by selecting a pair of chromosomes having the best fitness function value as parents. Selected parents undergo mating to generate one or more offspring. Each chromosome comprises usually several genes, and each child shares genes from both parents through the crossover operation.

The exclusive use of the crossover operator is insufficient to generate high-quality chromosomes in many combinatorial optimisation problems, including 3IAP. For increasing offspring quality, the mutation operation introduces stochastic changes to a few genes, enhancing population diversity and promoting escape from local optima. This random perturbation ensures the exploration of new regions within the search space, augmenting the algorithm capability to discover optimal solutions. Thus, the mutation operator ensures the exploration of novel regions within the search space. The selection operator is set to produce a new generation. Subsequent generations are derived from the offspring, in combination with parental chromosomes. The process iterates over a limited number of generations or until an acceptable solution is attained, negating further optimisation requirements, as shown in Algorithm 2.

Algorithm 2: Genetic Algorithm Model.

```
   Input: A sequence of cost matrices $\langle C_1, C_2, \ldots, C_n \rangle$.
1  - Initialise a random population of size $n$.
2  - Evaluate fitness of every individual of the population.
   Output: A feasible solution for 3IAP.
3  while (termination criteria is not met) do
4      - Select two parents from the population.
5      - Apply crossover operator to two selected parents with probability $Pc$.
6      - Apply mutation operator to random individuals with probability $Pm$.
7      - Compute fitness of the new generation.
8      - Replace the population by the new generation.
9      - Select survivors.
10 end while
11 return Best solution found.
```

5.2.2 Hybrid Genetic Algorithm Review

The simple GA generates offspring through selection, crossover, and mutation operators, aiming to enhance the population by replacing some existing members with improved individuals. In contrast, a Hybrid Genetic Algorithm (HGA) incorporates a local search into offspring generated by GA before their insertion into the population. Global search algorithms including GA explore new and promising regions in the solution space, while local search methods focus on refining solutions in local regions. HGAs have demonstrated significant effectiveness in the last decades.

A common strategy involves integrating additional local optimisation techniques to improve GA by refining the individuals' fitness. In particular, the resemblance between the role of local optimisation in GA and the role of knowledge in the evolutionary process suggest that local optimisation can function as a learning process. The primary goal is to enhance the simple GA performance by combining the exploration and exploitation capabilities with embedded algorithms. This objective is achieved by striking a balance between global exploration (discovering new, and more promising regions of the solution space) and local exploitation (focusing on high-quality solutions in favourable localised regions).

HGAs constitute a comprehensive framework for collaborative optimisation, where integrated algorithms cooperate with genetic operators. The diverse design options for specific components of the hybrid genetic algorithm offer numerous opportunities for innovation in optimisation strategies, (Drezner and Misevičius, 2013). HGAs are chosen as metaheuristic method for three reasons: (i) they are evolutionary, which is mandatory for the algorithm implementation; (ii) they are capable to tackle complex problems of large size; and (iii) they can reach a solution within an acceptable computational time (Jih and Hsu, 2004).

5.2.3 HGA Literature Review

In the last two decades, extensive studies have focused on solving challenging combinatorial optimisation problems through genetic algorithms. Tailoring hybrid genetic algorithms to such problems has yielded compelling outcomes, affirming the efficacy of these metaheuristics.

A typical example is the Travelling Salesman Problem (TSP). Given a set of cities, TSP searches for the shortest route, visiting each city exactly once, and returning to the starting point. Ahmed (2010, 2014) and Hussain *et al.*, (2017) developed new crossover operators for a genetic algorithm that generate quality solutions to solve a variation of the TSP. Buriol and França (2004) used a hybrid approach by combining GA with a local search procedure for solving the asymmetric TSP. Yousefikhoshbakht *et al.*, (2016) proposed a hybrid metaheuristic algorithm for solving the TSP. Mungwattana *et al.*, (2019) hybridised a genetic algorithm with three search operators for solving the TSP. Ha *et al.*, (2020) investigated the TSP with drones and proposed a hybrid genetic algorithm for its solution. Multiple-TSP was also investigated (Groba *et al.*, 2018). Mahmoudinazlou and Kwon (2023) recently designed a hybrid genetic algorithm to minimise the length of the longest tour.

Another example is the Vehicle Routing Problem (VRP) defined as an extension of TSP (Mehbali, 1990). However, the application of genetic algorithms to its resolution remains an area deserving more consideration. Baker and Ayechew (2003) applied genetic algorithms to tackle VRP. Tasan and Gen (2012) proposed another genetic algorithm addressing VRP with simultaneous pick-up and deliveries, and Mohammed *et al.*, (2017) sought to optimise VRP routes using genetic algorithms.

For specific assignments, Younas *et al.*, (2018) explored collaborative task assignment optimisation through a genetic algorithm. Toroslu and Arslanoğlu (2007) studied the personnel assignment problem with multiple objectives, considering hierarchical and team constraints. A multi-objective evolutionary algorithm was developed to solve this problem. Liu and Wang (2015) examined the Generalized Assignment Problem (GAP), proposing a scalable parallel genetic algorithm for its resolution. Dörterler *et al.*, (2017) introduced a tailored genetic algorithm designed for a particular case of the GAP.

GA was also proposed to address the Quadratic Assignment Problem (QAP) (Azarbonyada and Babazadeh, 2014). Misevičius and Verené (2021) introduced a novel hybrid genetic-hierarchical algorithm specifically designed for solving the QAP, leveraging the inherent hierarchical structure of the problem.

Huang and Lim (2006) pioneered resolving the three-index assignment problem using a genetic algorithm. They presented a new local search heuristic, simplifying the problem to the classical two-dimensional assignment problem. Furthermore, they integrated this heuristic with the genetic algorithm, resulting in a hybrid approach for the 3IAP solution (Huang and Lim, 2006).

Drawing on this study, Pérez (2017) devised a similar hybrid heuristic to solve the multi-dimensional assignment problem. He applied his hybrid algorithm to solve a case study, the school timetabling problem. Recently, Badoni *et al.*, (2023) examined the university course timetabling problem, employing a genetic algorithm (GA) to explore the search space and determine a solution within the global optimum region. They then applied an iterated local search procedure to further refine and enhance the obtained solution.

Based on prior research, this work aims to enhance existing methodologies by developing a more efficient hybrid genetic algorithm for solving 3IAP.

5.3 Local Search Methods for 3IAP

Local search methods play an important role in combinatorial optimisation. Several heuristics are designed to solve *NP*-hard discrete optimisation problems. From a feasible solution, a local search method moves to another neighbouring solution, through small local changes. If a potential improvement is achievable, the procedure moves to a new neighbouring solution. The local search method is an iterative procedure that continues running until an optimal solution is achieved or up to a predefined number of iterations.

The so-called neighbourhood-centred methods operate iteratively by exploring the neighbourhoods of the current solution. The literature includes various local search methods that consider diverse classes and sizes of neighbourhoods. The principle of these methods lies in the evaluation of the neighbourhoods in a predefined way. Karapetyan and Gutin (2011) highlighted the difference between construction heuristics and local search, which can sometimes be critical.

Among the early local search methods developed for 3IAP is the so-called variable depth interchange heuristic described in (Balas and Saltzman, 1991). The method starts with a feasible solution, then evaluates the objective function for all possible interchanges. An interchange will be performed only if it entails a reduction in the cost. The method repeats the process until an optimal solution is found, or no improvement is possible. Two iterative local search procedures for 3IAP are presented below.

5.3.1 Random Permuted Numbers method

Several local search methods exist in the literature. A first local search heuristic, the '*Random Permuted Numbers*' (RPN) method, is proposed in this section. The RPN method represents an efficient heuristic for solving 3IAP. Indeed, from an initial random permutation, the method generates a feasible solution to the problem within polynomial time.

For solving a 3IAP of size n, the RPN method initiates by selecting a random permutation p of order n. The heuristic then rearranges the cost matrices based on p. Next, the method chooses one row from each matrix to construct a new composite matrix. The Hungarian method is then applied to this composite matrix to address the resultant two-dimensional assignment problem, efficiently deriving a feasible solution for the original 3IAP.

Algorithm 3: Random Permuted Numbers Method.

```
Input: A sequence of cost matrices $\langle C_1, C_2, \dots, C_n \rangle$, and $p$ a random
       permutation encoding a chromosome.
Output: A feasible solution of 3IAP.
1 for $i \leftarrow 1$ to $n$ do
2     Select row $i$ from $C_i$ to form a new matrix $M$.
3 end for
4 - Apply Hungarian method to $M$.
5 - Find $q$ the permutation associated with the Hungarian method solution.
6 return Solution for 3IAP.
```

To illustrate Algorithm 3, let us solve an example of 3IAP of size $n = 4$, using the data of bs_4_3 instance.dat from (Balas and Salzman Dataset, 1991). Computational experiments were performed using Python, applying a numbering system for lists or arrays starting with 0 instead of 1. Therefore, this system is followed throughout this chapter. Given a random permutation p = [3,1,2,0].

$$C_0 = \begin{bmatrix} 24 & 45 & 78 & 07 \\ 47 & 15 & 24 & 94 \\ 61 & 98 & 47 & 79 \\ 74 & 38 & 47 & 52 \end{bmatrix} C_1 = \begin{bmatrix} 09 & 59 & 34 & 14 \\ 77 & 71 & 44 & 70 \\ 09 & 96 & 55 & 74 \\ 57 & 63 & 78 & 18 \end{bmatrix} C_2 = \begin{bmatrix} 30 & 28 & 68 & 29 \\ 56 & 41 & 30 & 44 \\ 56 & 48 & 60 & 41 \\ 13 & 25 & 03 & 97 \end{bmatrix} C_3 = \begin{bmatrix} 11 & 37 & 64 & 35 \\ 55 & 35 & 56 & 47 \\ 16 & 61 & 17 & 55 \\ 29 & 87 & 83 & 84 \end{bmatrix}$$

Each matrix is associated with a factory. Rows correspond to jobs and columns to machines.

In this case, job indices 0, 1, 2, and 3 are allocated to factories 3, 1, 2, and 0, respectively. The algorithm subsequently selects the first row from Factory 3, the second from Factory 1, the third from Factory 2, and the last from Factory 0. The selected rows collectively constitute a new matrix M, representing LAP.

$$M = \begin{bmatrix} 11 & 37 & 64 & 35 \\ 77 & 71 & 44 & 70 \\ 56 & 48 & 60 & 41 \\ 74 & 38 & 47 & 52 \end{bmatrix}$$

The Hungarian method is then applied to M to solve the associated LAP.

$$M = \begin{bmatrix} (\mathbf{11}) & 37 & 64 & 35 \\ 77 & 71 & (\mathbf{44}) & 70 \\ 56 & 48 & 60 & (\mathbf{41}) \\ 74 & (\mathbf{38}) & 47 & 52 \end{bmatrix}$$

The Hungarian method solution is expressed by coefficients represented in bold.

$$C_0 = \begin{bmatrix} 24 & 45 & 78 & 07 \\ 47 & 15 & 24 & 94 \\ 61 & 98 & 47 & 79 \\ 74 & \mathbf{38} & 47 & 52 \end{bmatrix} C_1 = \begin{bmatrix} 09 & 59 & 34 & 14 \\ 77 & 71 & \mathbf{44} & 70 \\ 09 & 96 & 55 & 74 \\ 57 & 63 & 78 & 18 \end{bmatrix} C_2 = \begin{bmatrix} 30 & 28 & 68 & 29 \\ 56 & 41 & 30 & 44 \\ 56 & 48 & 60 & \mathbf{41} \\ 13 & 25 & 03 & 97 \end{bmatrix} C_3 = \begin{bmatrix} \mathbf{11} & 37 & 64 & 35 \\ 55 & 35 & 56 & 47 \\ 16 & 61 & 17 & 55 \\ 29 & 87 & 83 & 84 \end{bmatrix}$$

The Hungarian method solution optimises the permutation q=[3,0,1,2] and allocates machines to factories respectively, leading to a feasible solution to 3IAP at total cost of 134.

5.3.2 Triple Local Search method

Huang and Lim (2006) proposed the Triple Local Search (TLS) method to solve the three-index assignment problem (3IAP) by reducing it to the classic assignment problem. TLS procedure has been adapted to the problem. Recall that this method utilises permutations and applies the Hungarian method iteratively to solve the LAPs. Computational tests run in Python validated its efficacy in producing rapid feasible solutions.

The TLS method iterates through a cyclic process involving three distinct steps rep-

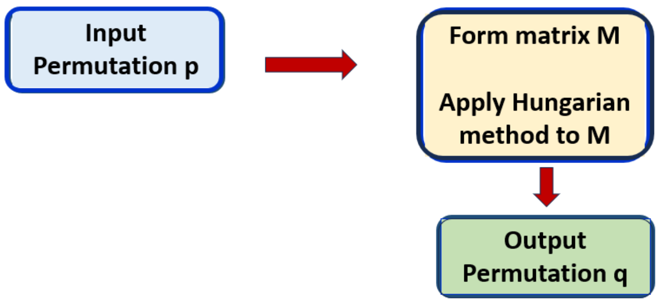

Figure 5.2: One step of TLS.

resenting three parts of one 3IAP solution aligned with the three permutations p, q and the identity I, each of which can be optimised separately and efficiently. The procedure keeps running until it attains an optimal solution or no further improvement can be achieved. Each step initiates with an input permutation used to construct a matrix. The Hungarian method is then applied to that matrix, resulting in a feasible solution for 3IAP, and a new permutation q, necessary for the next step. Figure 5.2 illustrates the execution of one such step.

The first step of the TLS algorithm executes the RPN method. Subsequently, in the second step matrices are reordered based on the permutation q, followed by the selection of a column from each matrix to form a new matrix, denoted M_1. The Hungarian method is then applied to solve the associated LAP, yielding another feasible solution for 3IAP and a new permutation q'. In the third step, matrices are rearranged according to q', and specific coefficients are chosen from each matrix to compose another matrix M_2. The Hungarian method is then applied to M_2. The iterative process continues within the TLS method until an optimal solution is attained or further improvement becomes unfeasible. Figure 5.3 illustrates the cyclic nature of the triple local search method.

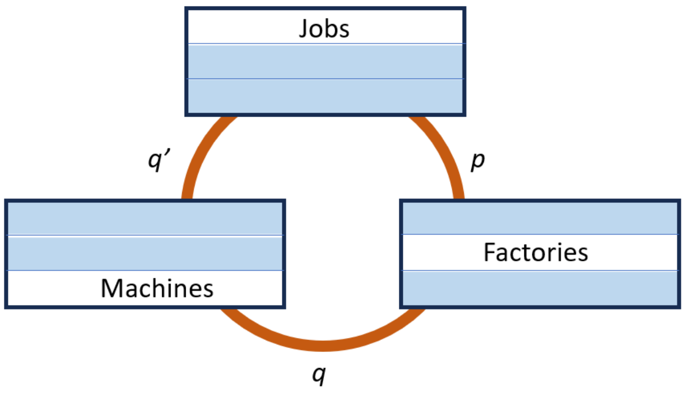

Figure 5.3: Iterative cycle of the TLS method.

Algorithm 4: Triple Local Search Method.

```
Input: A sequence of cost matrices $\langle C_1, C_2, \ldots, C_n \rangle$, and $p$ a random
       permutation encoding a chromosome.
Output: A feasible solution of 3IAP and a permutation $p'$.
1  - Initialise a random population $p$ of size $n$.
2  - Set $p_1 = p$ .
3  for $j \leftarrow 1$ to 3 do
4      - Sort cost matrices $C_i$ $(i = 1, 2, \ldots, n)$ according to $p_j$.
5      - Select specific coefficients of $C_i$ to form a matrix $M_j$.
6      - Apply Hungarian method to $M_j$.
7      - $p_{j+1}$ the permutation associated with the Hungarian method solution of $M_j$.
8  end for
9  - $p_1 \leftarrow p_4$
10 - Record $c$ cost of the final solution and the permutation $p_1$.
11 return Solution for 3IAP.
```

Given a permutation q, the second iteration of For Loop of Algorithm 4, assigns machines to factories based on q. From the cost matrices sorted according to q, for each index $i = 0, 1, \ldots, n-1$, a column is selected from matrix C_i, forming matrix M_1 representing LAP. The Hungarian method is then applied to optimise the permutation q', which allocates machines to jobs, achieving a feasible solution for 3IAP. Illustrating TLS approach, the 3IAP example of size four previously examined using the RPN method is revisited below.

At the end of the first step, a feasible solution is reached at a total cost of 134 and the resulting permutation q = [3,0,1,2]. To apply the second step of TLS, matrices are rearranged according to q, and the columns are selected to constitute another matrix M_1 shown below.

$$M_1 = \begin{bmatrix} 11 & 45 & 34 & 29 \\ 55 & 15 & 44 & 44 \\ 16 & 98 & 55 & 41 \\ 29 & 38 & 78 & 97 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 11 & 45 & (\mathbf{34}) & 29 \\ 55 & (\mathbf{15}) & 44 & 44 \\ 16 & 98 & 55 & (\mathbf{41}) \\ (\mathbf{29}) & 38 & 78 & 97 \end{bmatrix}$$

The highlighted coefficients represent the Hungarian method solution of LAP associated to M_1.

The Hungarian method optimises the permutation q' = [2,1,3,0], allocating machines to jobs and yielding a better feasible solution for 3IAP with a total cost of 119. In the third step, matrices are permuted by q' to construct a third matrix, M_2. The coefficients in M_2 occupy the same positions as in the above Hungarian method solution.

$$M_2 = \begin{bmatrix} 78 & 34 & 68 & 64 \\ 15 & 71 & 41 & 35 \\ 79 & 74 & 41 & 55 \\ 74 & 57 & 13 & 29 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 78 & (\mathbf{34}) & 68 & 64 \\ (\mathbf{15}) & 71 & 41 & 35 \\ 79 & 74 & 41 & (\mathbf{55}) \\ 74 & 57 & (\mathbf{13}) & 29 \end{bmatrix}$$

Table 5.1: Outcomes of two iterations of TLS method.

	Input permutation		Solution	Cost	Output permutation	
Iteration 1	p	[3, 1, 2, 0]	[11, 44, 41, 38]	134	q	[3, 0, 1, 2]
	q	[3, 0, 1, 2]	[29, 15, 34, 41]	119	q'	[2, 1, 3, 0]
	q'	[2, 1, 3, 0]	[34, 15, 55, 13]	117	p	[1, 0, 3, 2]
Iteration 2	p	[1, 0, 3, 2]	[14, 15, 16, 3]	48	q	[3, 0, 2, 1]
	q	[3, 0, 2, 1]	[16, 15, 3, 14]	48	q'	[3, 1, 0, 2]
	q'	[3, 1, 0, 2]	[14, 15, 16, 3]	48	p	[1, 0, 3, 2]

The coefficients in bold represent the Hungarian method solution of LAP associated with M_2.

The Hungarian method optimises the permutation p = [1,0,3,2], allocating jobs to factories and generating another feasible solution for 3IAP with a total cost of 117. The second iteration of the algorithm ends with an optimal solution, achieving a total cost of 48, as displayed in Table 5.1.

5.3.3 TLS Numerical Experiments

Computational experiments were conducted on five 3IAP instances of size 10, selected from a random dataset previously defined as Sample 2. Each instance undergoes five runs, wherein each run initiates with a random input permutation and proceeds through nine consecutive iterations. Table 5.2 presents the numerical results. Each table row represents a run, displaying input permutation, the best solution achieved, and the number of iterations required to attain that solution.

Each row summarises the execution of the TLS method across multiple iterations. For example, during the first run of Instance 1, the results of nine consecutive iterations show that the best solution found has a cost of 37, reached at the end the third iteration, as displayed in Table 5.3

Table 5.2: Outcomes of TLS application on five instances of size 10.

	Run	Input permutation	Iteration	Solution	Cost
Instance 1	1	[8,0,1,3,7,5,6,2,4,9]	3	[4,6,8,9,2,1,3,1,2,1]	37
	2	[8,5,3,9,6,1,2,0,4,7]	6	[7,6,5,3,2,7,2,6,1,1]	40
	3	[5,8,4,2,3,0,6,1,7,9]	4	[2,5,2,4,2,1,1,5,4,6]	32
	4	[6,3,2,5,8,0,7,9,1,4]	9	[2,6,2,3,3,1,1,5,5,7]	35
	5	[1,4,7,5,6,0,9,8,3,2]	5	[2,5,1,17,2,1,7,4,1,1]	41
Instance 2	1	[1,4,5,7,8,2,3,6,0,9]	6	[1,2,1,5,4,16,9,1,6,7]	52
	2	[2,5,9,8,1,6,0,4,3,7]	9	[15,2,1,5,3,5,1,7,6,8]	53
	3	[7,6,1,8,9,0,5,2,4,3]	4	[1,6,4,12,10,5,1,4,1,9]	53
	4	[4,0,5,6,3,8,2,1,9,7]	4	[2,1,8,2,5,8,3,7,11,7]	54
	5	[4,2,3,0,5,8,6,1,9,7]	7	[6,1,1,10,5,20,1,3,6,1]	54
Instance 3	1	[9,6,7,1,2,0,4,3,8,5]	6	[4,5,4,6,9,1,2,2,2,5]	40
	2	[1,5,0,3,7,6,8,9,4,2]	4	[6,1,9,6,7,11,2,5,4,1]	52
	3	[2,7,8,3,9,0,4,1,6,5]	7	[4,5,4,7,6,7,10,3,4,3]	53
	4	[4,0,8,2,1,9,5,7,6,3]	8	[4,1,4,8,2,7,8,2,2,1]	39
	5	[5,3,2,8,7,9,6,1,4,0]	1	[4,9,3,10,2,15,2,3,6,1]	55
Instance 4	1	[3,8,2,4,5,6,1,0,9,7]	5	[3,1,1,3,2,16,1,3,15,5]	50
	2	[1,9,6,3,7,4,2,0,5,8]	6	[3,16,2,6,1,11,1,2,6,6]	54
	3	[4,0,3,9,6,7,2,8,1,5]	6	[4,0,3,9,6,7,2,8,1,5]	47
	4	[2,3,8,4,1,9,7,6,0,5]	1	[3,1,2,16,2,6,1,2,3,17]	53
	5	[9,2,4,5,6,3,8,0,7,1]	5	[3,1,2,11,7,3,5,3,1,10]	46
Instance 5	1	[9,0,8,1,4,6,2,3,7,5]	5	[48,12,37,75,26,63,37,12,31,13]	354
	2	[5,8,4,1,9,3,0,7,2,6]	3	[18,12,36,159,34,143,15,23,31,13]	484
	3	[0,4,7,3,6,9,5,8,1,2]	9	[56,50,36,82,26,63,15,46,31,13]	418
	4	[9,1,3,4,5,6,8,2,0,7]	8	[153,12,36,42,26,63,15,54,31,13]	445
	5	[1,2,6,9,5,8,4,7,3,0]	9	[112,12,36,43,26,63,15,88,31,13]	439

Table 5.3: Running three iterations for the first run.

	Input permutation		Solution	Cost	Output permutation	
Iteration 1	p	[8,0,1,3,7,5,6,2,4,9]	[9,6,19,47,26,9,21,27,13,1]	178	q	[3,4,9,5,1,2,7,6,8,0]
	q	[3,4,9,5,1,2,7,6,8,0]	[6,13,1,9,19,41,2,22,9,6]	128	q'	[8,9,4,5,7,3,6,0,1,2]
	q'	[8,9,4,5,7,3,6,0,1,2]	[2,5,7,6,2,7,2,6,2,13]	52	p	[3,9,1,7,0,2,5,6,4,8]
Iteration 2	p	[3,9,1,7,0,2,5,6,4,8]	[7,8,8,11,28,9,6,4,13,7]	101	q	[3,4,1,8,2,0,5,6,9,7]
	q	[3,4,1,8,2,0,5,6,9,7]	[7,13,8,7,9,28,6,4,8,11]	101	q'	[0,8,2,9,5,4,6,7,1,3]
	q'	[0,8,2,9,5,4,6,7,1,3]	[3,25,2,4,2,9,2,8,2,7]	64	p	[1,0,5,4,7,2,3,6,9,8]
Iteration 3	p	[1,0,5,4,7,2,3,6,9,8]	[4,6,20,35,14,9,7,1,2,7]	105	q	[5,9,6,2,4,7,3,8,1,0]
	q	[5,9,6,2,4,7,3,8,1,0]	[9,2,1,3,1,8,1,2,4,6]	37	q'	[8,9,5,0,7,6,3,2,1,4]
	q'	[8,9,5,0,7,6,3,2,1,4]	[4,6,8,9,2,1,3,1,2,1]	37	p	[1,0,6,5,9,3,7,2,4,8]

5.4 Hybrid Algorithm Description

Genetic Algorithms show mixed success in addressing optimisation problems. In general, standard GAs may falter in some cases. Better outcomes are observed through integrating local improvement heuristics, exemplified by the local search methods. In this section, a new hybrid genetic algorithm is introduced for solving 3IAP. The approach integrates the adapted TLS method with the genetic algorithm, yielding promising outcomes, as observed in its application to diverse optimisation challenges.

5.4.1 Encoding Solution

Any feasible solution to 3IAP is defined by a pair of permutations; once the first permutation is set, the Hungarian method determines the second. The chosen encoding method is the permutation representation, where each feasible solution is associated with a chromosome represented by a permutation of n nonnegative integers. Each element of a chromosome is called a '*gene*'. Every member of the initial population is generated by a random permutation.

Example 1: Let p be a chromosome representing a feasible solution of 3IAP of size 9, as shown in Table 5.4.

Table 5.4: A permutation representing a chromosome.

3	2	6	4	9	1	5	7	8

5.4.2 Initial Population

Various methods, including the Diagonals Method, Greedy-Style Procedure, Random Permuted Number technique, or the local search method can be employed to generate the initial population. The TLS method is preferentially applied to enhance chromosome quality; hence, every member of the initial population indicates a local minimum within the solution space. Starting with a good initial population may accelerate GA convergence.

An iteration resulting in a new population is called '*generation*'. Across generations, the iterative evolutionary process preserves specific hereditary features, and enhances the overall fitness function. A better fitness value offers an individual a better chance to be selected for the next generation. Methods exist to define an appropriate fitness function. Given that the 3IAP objective is to minimise the total assignment cost, the objective function can be considered as a fitness function.

Algorithm 5: Initial Random Population.

```
Input: A nonnegative integer number $N$.
Output: A random population of chromosomes.
1 $P := \varnothing$.
2 for $i \leftarrow 1$ to $N$ do
3     Generate a random permutation $p_i$.
4     Apply TLS method starting with $p_i$ .
5     Generate permutations $q_i$ and $q'_i$.
6     Update the population $P := P \cup \{ p_i,\ q_i,\ q'_i\}$.
7 end for
8 return An initial random population.
```

5.4.3 Stopping Criteria

In theory, the search process can continue indefinitely unless the optimal solution of the problem is known beforehand. In practice, algorithms necessitate termination. Common criteria include a fixed number of iterations or CPU time. Termination also occurs when neither a local nor a global optimum are achieved, or convergence is unattainable. Another criterion involves termination when the solution found shows no improvement over a specified iteration count, indicating a local or global optimum has been reached. Additionally, termination may be prompted by a generation average fitness closely aligning with the fitness value of the solution found. Two termination criteria will be used in this project: convergence detection and no improvement detection.

5.4.4 Crossover

From an initial population of a predetermined size, two random chromosomes are selected as parents. The principal genetic operator, the crossover, involves swapping a segment of one parent chromosome with the corresponding segment of the other chromosome at a random position. The single-point crossover is a fundamental operator. Alternatively, multipoint crossover allows the exchange of multiple segments between chromosomes, contributing to increased evolutionary efficiency of genetic algorithms and partially mapped crossover, whose principle is to exchange a partial segment between two parents. Utilising these crossover operators enhances the adaptability of genetic algorithms by enabling exploration across diverse solution spaces, thereby increasing the algorithm efficacy in solution refinement.

Example 2: Let p and q be two chromosome parents, as shown in Table 5.5 below.

Table 5.5: Two chromosomes chosen from the population.

3	2	6	4	9	1	5	7	8
4	1	9	2	6	5	7	8	3

Algorithm 6: The single-point crossover.

```
Input: Two permutations $p$ and $q$ of size $n$ chosen as parents.
Output: Two offspring $p'$ and $q'$ resulting from the single-point crossover.
1 Select a random position $i$ from $[1,2,\ldots,n]$.
2 Split both parents at position $i$, such that $p = p_1 \cup p_2$ and $q = q_1 \cup q_2$.
3 Set $p' = p_1 \cup q_2$ and $q' = q_1 \cup p_2$.
4 return Two offspring $p'$ and $q'$ resulting from the crossover.
```

Table 5.6 illustrates an example of two offspring resulting from a single-point crossover.

Example 3: Two offspring resulting after the single-point crossover of p and q.

Table 5.6: The single-point crossover.

3	2	6	4	9	5	7	8	3
4	1	9	2	6	1	5	7	8

Partially Mapped Crossover (PMX) is a genetic operator commonly used in genetic algorithms to generate new offspring by combining two parent chromosomes randomly selected from the population. Algorithm 7 outlines the steps of PMX.

Algorithm 7: PMX Crossover.

```
Input: Two permutations $p$ and $q$ of size $n$ chosen as parents.
Output: Two offspring $p'$ and $q'$ resulting from the PMX crossover.
1 Select two random positions $i$ and $j$ from $[1,2,\dots,n]$.
2 Split both parents at these positions.
3 $p = p_l \cup p_m \cup p_r$ and $q = q_l \cup q_m \cup q_r$.
4 Set $p' = p_l \cup q_m \cup p_r$ and $q' = q_l \cup p_m \cup q_r$.
5 if $p'$ and $q'$ contain duplicated genes then
6     Repair $p'$ and $q'$ to preserve their feasibility.
7     Update $p'$ and $q'$.
8 end if
9 return Two offspring $p'$ and $q'$ resulting from the PMX crossover.
```

Let p and q be two random chromosomes selected as parents. Select in both parents, two random positions, i and j, then split p and q into three segments. The PMX algorithm is used to reproduce new chromosomes p' and q' by exchanging the middle segment of p with the corresponding segment of q. Genes outside these segments remain unchanged. However, the occurrence of duplicated values in the resulting sequences renders them invalid, necessitating repair to ensure solution feasibility. Tables 5.7 and 5.8 illustrate the two phases of the PMX algorithm.

The following example refers to the same permutations p and q to illustrates the PMX crossover.

Example 4: Two sequences resulting from the PMX crossover of p and q.

Table 5.7: The PMX crossover result before the repair.

3	2	6	2	6	5	5	7	8
4	3	9	4	9	1	7	8	1

These two sequences are deemed unfeasible solutions due to duplicated genes, needing repair for feasibility. The repair involves substituting the duplicated genes with the omitted ones. The updated chromosomes are presented in Table 5.8.

Table 5.8: The PMX crossover result after the repair.

3	4	9	2	6	5	1	7	8
2	3	6	4	9	1	7	8	5

5.4.5 Mutation Operator

The mutation operator follows the crossover operator with a low probability p_m facilitating a good diversification of the solution space exploration. The mutation modifies one or more genes within a chosen chromosome, reintroducing novel genetic material and widening population variability. The stochastic alteration of solution elements increases the population diversity, offering a mechanism to escape a local minimum. In particular, the mutation operator frequently yields solutions beyond the explored subspace.

The mutation causes small random adjustments in the chromosome structure to obtain new solutions. The objective is to maintain population diversity, which is often applied with a low p_m probability; otherwise, it will reduce the effectiveness of GA. In summary, the mutation operator is responsible for diversifying the search process. Literature contains a variety of mutation operators.

5.4.5.1 Swap Mutation

In the swap mutation of genetic algorithms, two genes within a chromosome are randomly chosen, and their values are exchanged. This operator has the potential to generate a better solution, as described through Algorithm 8, below.

Algorithm 8: Swap Mutation.

```
Input: A chromosome $p$ of size $n$ and its fitness $c(p)$.
Output: A fitter chromosome $p'$ of size $n$ resulting from the swap mutation.
1 Select two random positions $i$ and $j$ from [1,2,...,$n$].
2 if $p = ( p_1, p_2, \ldots, p_i, p_{i+1}, \ldots, p_j, p_{j+1}, \ldots, p_n)$ then
3     Set $p' = ( p_1, p_2, \ldots, p_j, p_{i+1}, \ldots, p_i, p_{j+1}, \ldots, p_n)$.
4     Compute $c(p')$ fitness of chromosome $p'$.
5 end if
6 if $c(p') < c(p)$ then
7     Substitute $p$ with $p'$.
8 end if
9 return A new chromosome $p'$ can result from the swap mutation.
```

Let $p = (p_1, p_2, \ldots, p_i, p_{i+1}, \ldots, p_j, p_{j+1}, \ldots, p_n)$ be a chromosome. Randomly select two distinct genes at positions i and j, interchanging their values, to generate a new chromosome denoted $p' = (p_1, p_2, \ldots, p_j, p_{i+1}, \ldots, p_i, p_{j+1}, \ldots, p_n)$. The chromosome p is substituted with p' only if its cost $c(p')$ is less than $c(p)$. This process fosters diversity and exploration in the population, as illustrated in the example below.

Example 5: Let p be a chromosome similar to that in Table 5.4.

Table 5.9: The permutation p representing a chromosome.

3	2	6	(4)	9	1	(5)	7	8

Table 5.10: A chromosome p' can replace p by the swap mutation.

3	2	6	(5)	9	1	(4)	7	8

5.4.5.2 Scramble Mutation

Let p represent a chromosome. Randomly select two positions, i and j and scramble all genes between i and j, yielding a new chromosome denoted p'. This process is described in Algorithm 9 below.

Algorithm 9: Scramble Mutation.

```
Input: A chromosome $p$ of size $n$ and its fitness $c(p)$.
Output: Another chromosome $p'$ of size $n$ resulting from the scramble mutation.
1 Select two random positions $i$ and $j$ from [1,2,...,$n$].
2 Set $q = (p_i, p_{i+1}, \ldots, p_j)$ a segment of $p$.
3 Shuffle $q$ into $q'$.
4 Set $p' = (p_1, p_2, \ldots, p_i) \cup q' \cup (p_{i+1}, \ldots, p_j, p_{j+1}, \ldots, p_n)$.
5 Compute $c(p')$ fitness of chromosome $p'$.
6 if $c(p') < c(p)$ then
7     Substitute $p$ with $p'$.
8 end if
9 return A new chromosome $p'$ can result from the scramble mutation.
```

Consider a chromosome p. Randomly select a segment q of p, and scramble all genes of q, giving rise to a new chromosome, p'. The scramble mutation is displayed in the example below.

Example 6: Let p be a chromosome similar to that in Table 5.4.

Table 5.11: The permutation p representing a chromosome.

3	2	6	(4)	(9)	(1)	(5)	7	8

Let $q = [4, 9, 1, 5]$ be a segment of p, and q and is shuffled into $q' = [5, 4, 1, 9]$.

Table 5.12: A chromosome p' resulting from p by the scramble mutation.

3	2	6	(5)	(4)	(9)	(1)	7	8

5.4.5.3 Inversion Mutation

The inversion mutation is similar to a scramble mutation. Let p be a chromosome. By randomly selecting two positions i and j, reverse the order of the genes between i and j to generate a new chromosome, p'. This operator, promoting genetic diversity and exploration within the population, is described in Algorithm 10 below.

Algorithm 10: Inversion Mutation.

```
Input: A chromosome $p$ of size $n$ and its fitness $c(p)$.
Output: Another chromosome $p'$ of size $n$ resulting from the inversion mutation.
1  Let $p = (\ p_1,\ p_2, \ldots,\ p_n)$ be a permutation of size $n$.
2  Select two random positions $i$ and $j$ from [1,2,...,$n$].
3  Set $q = (\ p_i,\ p_{i+1}, \ldots,\ p_j)$ a segment of $p$.
4  Set $q' = (\ p_j,\ p_{j-1}, \ldots,\ p_i)$ inverse of segment $q$.
5  Set $p' = (\ p_1,\ p_2, \ldots,\ p_i) \cup q' \cup (\ p_{i+1}, \ldots,\ p_j,\ p_{j+1}, \ldots,\ p_n)$.
6  Compute $c(p')$ fitness of chromosome $p'$.
7  if $c(p') < c(p)$ then
8      Substitute $p$ with $p'$.
9  end if
10 return A new chromosome $p'$ can result from the inversion mutation.
```

Let $p = (\ p_1,\ p_2, \ldots,\ p_i,\ p_{i+1}, \ldots,\ p_j,\ p_{j+1}, \ldots,\ p_n)$ be a chromosome. Randomly select two genes at positions i and j, reverse the order between them, to generate a new chromosome $p' = (\ p_1,\ p_2, \ldots,\ p_{i-1},\ p_j,\ p_{j-1}, \ldots,\ p_i,\ p_{j+1}, \ldots,\ p_n)$. This process is illustrated in the example below.

Example 7: Let p be a chromosome similar to that in Table 5.4.

Table 5.13: The permutation p representing a chromosome.

3	2	6	(4)	(9)	(1)	(5)	7	8

Let $q = [4, 9, 1, 5]$ be a segment of p, and q and is inversed into $q' = [5, 1, 9, 4]$.

Table 5.14: A chromosome p' resulting from p by the inversion mutation.

3	2	6	(5)	(1)	(9)	(4)	7	8

5.5 Hybrid Algorithm Implementation

After introducing the local search method TLS, renowned for its efficacy in solving 3IAP, this technique will be hybridised with the genetic algorithm. The implementation of the

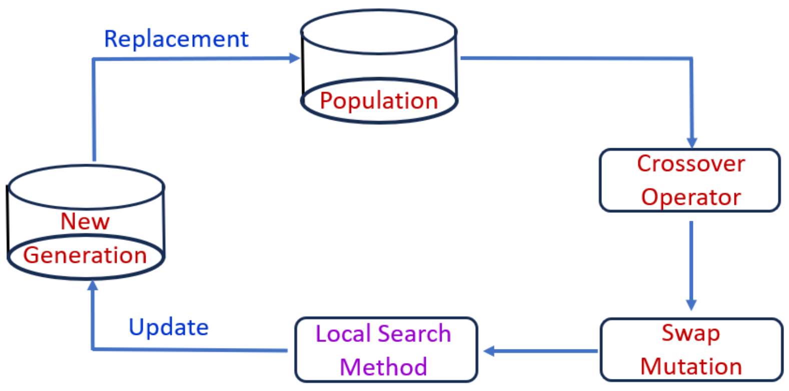

Figure 5.4: The Hybrid Genetic Algorithm structure.

hybrid genetic algorithm follows the ensuing steps. Figure 5.4 presents the main components of the hybrid genetic algorithm, exhibiting a structure nearly identical to that of the simple genetic algorithm. The sole difference lies in integrating the local search method just after the mutation operator.

The genetic algorithm commences by randomly reproducing an initial population of a predetermined size. Subsequently, the crossover operator selects two parent chromosomes based on their fitness function values, producing two offspring. The mutation operator is then applied to the newly generated chromosomes. The algorithm computes the fitness values of newly generated individuals and sorts the whole in ascending order. Following the principle of survival of the fittest, the survivors for the next generation are identified. Then, the algorithm finds the best solution from this generation. This process iterates until one of the two termination criteria is satisfied.

Figure 5.5 represents the Hybrid Genetic Algorithm flowchart. The initial population is randomly generated, every feasible solution is produced by the adapted TLS method which enhances its quality. As a result, each member of the initial population is already a local minimum within the solution space. The initial population is of size $N = 100$ which remains invariable through the whole process. Sort the population in increasing order then select the candidate with minimum cost as the best solution found for the current generation. The While loop attempts to improve the solution found.

The implementation features two stopping criteria, the first ends the algorithm if the solution quality does not improve over a fixed predefined number of consecutive generations.

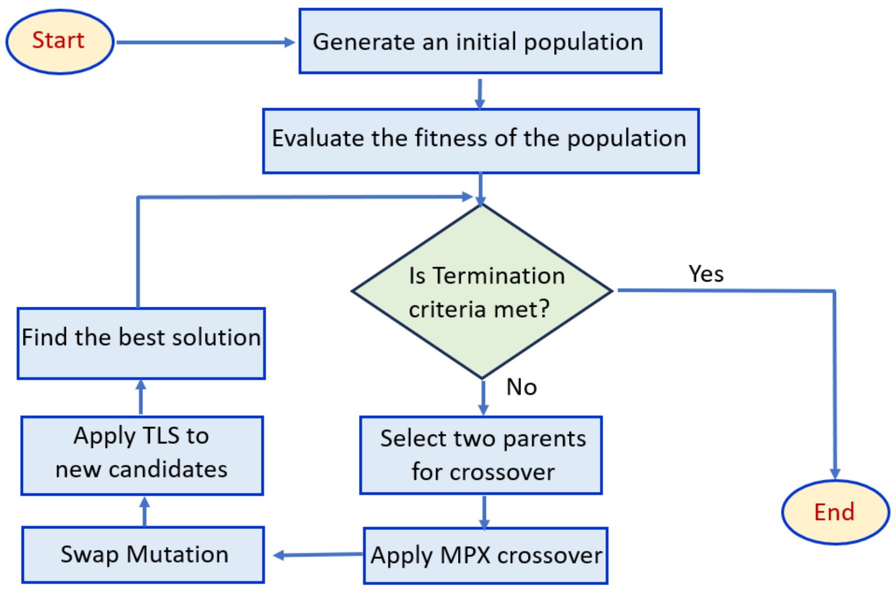
Start
Generate an initial population
Evaluate the fitness of the population
Is Termination criteria met?
Yes
No
Find the best solution
Apply TLS to new candidates
Select two parents for crossover
End
Swap Mutation
Apply MPX crossover

Figure 5.5: The Hybrid Genetic Algorithm flowchart.

The second criterion terminates the algorithm after a maximum number of iterations.

Various crossover operators are documented in the literature, and there is no evidence of superiority of any one among them. The proposed algorithm in this project employs the partially mapped crossover (PMX) operator introduced by (Goldberg and Lingle, 1985) to address the travelling salesman problem (TSP). Our hybrid algorithm realises the exchange of a random partial segment between two parents. Subsequently, an adjustment step may be necessary to preserve the feasibility of the new solutions.

The mutation operator within the algorithm initiates three successive swaps on random chromosomes, followed by the application of the TLS method to enhance their quality. A subsequent fitness assessment determines whether the modified chromosomes surpass their predecessors in fitness value. Only the best chromosomes are incorporated into the population, maintaining a constant size through equivalent removals. The current generation is updated each time to achieve a better solution. The hybrid approach integrates a local search with the genetic algorithm as illustrated in Figure 5.4.

Based on the principle of survival of the fittest, the algorithm only incorporates newly generated individuals if their fitness values show improvement. Then, the current generation is sorted in ascending order according to the fitness function values. When a newly generated solution outperforms the existing one, it is added to the candidate pool, prompting the algorithm to repeat the process. The iterative loop continues until there is no improvement in the fitness value across a predetermined number of iterations. Thus, the algorithm terminates after multiple generations.

Algorithm 11: Hybrid Genetic Algorithm - HGA

```
Input: pop_size=100, num_generations=1000, num_crossovers=75,
       num_mutations=25, median OR minimum, tol_size=10, cost_dataset.
Output: Best solution possible for 3IAP.
1  Apply TLS to generate initial population of size pop_size.
2  Compute the cost of each individual in the population.
3  Sort the population by cost in ascending order.
4  if median then
5      Compute pop_cost as the median cost of the population.
6  end if
7  else
8      Compute pop_cost as the minimum cost of the population.
9  end if
10 for niter $\leftarrow$ 1 to num_generations do
11     for nx $\leftarrow$ 1 to num_crossovers do
12         Apply PMX crossover a few times and update the population.
13     end for
14     for nm $\leftarrow$ 1 to num_mutations do
15         Select a random element from the population
16         Apply swap mutation to the element.
17         Apply TLS to that element.
18         Compute the Cost of the resulting element as e_cost.
19         Compute the Median Cost of the current population as temp_cost
20         if $e\_cost < temp\_cost$ then
21             Drop the element with the highest cost from the population.
22             Add the resulting element to the population.
23         end if
24     end for
25     Recompute pop_cost for the population as done above.
26     if pop_cost did not change for the last tol_size iterations then
27         Exit
28     end if
29 end for
30 return Solution for 3IAP with the minimum cost possible.
```

5.5.1 Numerical Experiments

Implementing the hybrid genetic algorithm, Algorithm 11, in Python via a Jupyter Notebook begins with loading the necessary data file. Subsequently, the program generates an initial population comprising 100 individuals, using a random function for initial seeding and the TLS method, which computes the fitness value of each individual within this population.

In this algorithm, each iteration produces a new generation. Users have two metrics for comparing generations: minimum or median cost. The program execution is capped at 1,000 iterations, with an adaptive termination criterion allowing cessation past 10, 15, or 20 consecutive iterations without fitness function improvement. Such a mechanism allows for the balancing of exploration and exploitation, ensuring the algorithm remains efficient.

The program core loop encompasses two For loops dedicated to crossover and mutation operators. The PMX crossover involves selecting two random parents to produce offspring and calculating their fitness values. Subsequently, the algorithm enforces a selection criterion, retaining only the two fittest individuals out of the four individuals (parents and offspring), thereby discarding the lesser-fit individuals to optimise the population genetic quality.

The mutation operator executes three swaps on randomly chosen individuals, with the stipulation that modified elements supplant their predecessors merely if an enhanced fitness value is observed.

The program ceases execution upon fulfilment of either of two predefined criteria. Upon termination, it prints the initial and final populations, each comprising 100 feasible solutions, arranged in ascending order by cost. The program also reports the total iterations completed before its end, accompanied by a graphical representation of the decreasing fitness function. The graph below shows the variation of the median of the fitness function of the resulting generations according to the number of iterations. Moreover, it assesses population diversity by enumerating the distinct individuals, offering insights into the genetic variability and evolutionary dynamics experienced throughout its execution.

Table 5.15 illustrates the 3IAP example of instance 3 of size 10 from Sample 1.

Table 5.15: Outcomes of HGA for one 3IAP instance of size 10.

Index	**Initial Population**	Cost	|	**Final Population**	Cost
1	[6, 1, 7, 8, 0, 9, 3, 5, 2, 4]	24	||	[6, 1, 9, 8, 3, 2, 0, 7, 4, 5]	21
2	[3, 1, 8, 0, 2, 4, 6, 7, 9, 5]	31	||	[6, 1, 7, 8, 0, 9, 3, 5, 2, 4]	24
3	[9, 3, 2, 1, 6, 8, 4, 7, 5, 0]	36	||	[3, 6, 1, 9, 7, 8, 5, 4, 0, 2]	25
4	[8, 6, 5, 3, 4, 7, 1, 9, 0, 2]	37	||	[2, 6, 0, 3, 5, 7, 8, 1, 4, 9]	25
5	[6, 0, 3, 5, 1, 4, 9, 2, 8, 7]	37	||	[4, 0, 3, 7, 2, 1, 9, 5, 8, 6]	27
...	...	...	||	...	...
96	[2, 4, 9, 5, 1, 0, 7, 8, 3, 6]	68	||	[4, 1, 0, 3, 8, 2, 5, 6, 9, 7]	41
97	[1, 9, 3, 5, 6, 0, 8, 2, 4, 7]	70	||	[0, 9, 2, 6, 4, 7, 8, 5, 1, 3]	41
98	[7, 1, 2, 5, 0, 8, 4, 9, 3, 6]	71	||	[8, 7, 2, 6, 5, 4, 9, 1, 0, 3]	41
99	[2, 3, 4, 6, 1, 0, 5, 8, 9, 7]	71	||	[2, 1, 9, 5, 0, 6, 8, 7, 4, 3]	41
100	[2, 7, 4, 5, 8, 0, 3, 9, 6, 1]	86	||	[4, 0, 7, 8, 6, 2, 5, 9, 3, 1]	41

The same table displays an extract of the initial and final populations, arranged in ascending order by cost. The initial best solution of cost 24 improved to an optimal final solution of cost 21, achieved over 106 iterations after the stagnation of the algorithm during the last 20 consecutive iterations.

The results in Table 5.15 show a significant reduction in the cost range, from 62 in the initial population to 20 in the last population, evidencing the algorithm effectiveness in optimisation. The best result achieved for the chosen instance of size 10 equals 21, which is an optimal solution. Furthermore, 85 distinct individuals in each population undergo substantial genetic diversity during the evolutionary process.

In this study, I introduced a hybrid genetic algorithm (HGA) called TLS, which incorporates an effective local search technique into the conventional GA structure. This enhancement significantly improves solution quality for 3IAP. Using Python Jupyter Notebook for HGA deployment, the algorithm starts with a random initial population and iteratively achieves optimal or near-optimal solutions for 3IAP over successive generations. Numerical experiments illustrate the HGA efficiency in handling large instances of 3IAP.

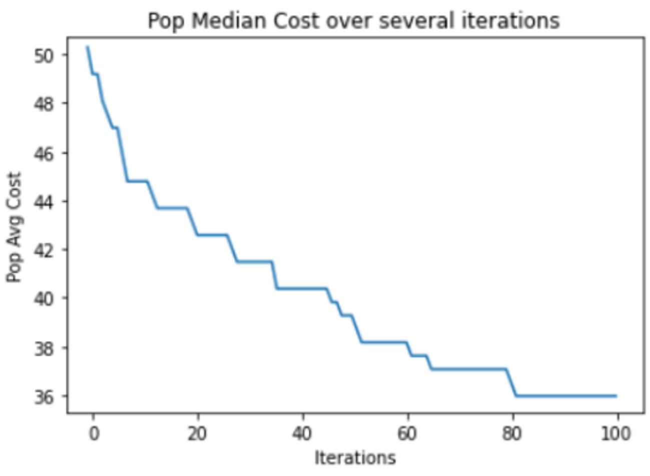

Figure 5.6: The graph of the HGA results.

Chapter 6

Future Developments

Two potential future developments and expected applications are proposed at the end of this project. The first extension suggests a the correlation between cost matrices and the optimal total cost of the assignment. The second extension involves integrating variable costs defined by specific probability distributions

6.1 First Extension

As part of future developments of the current project, an investigation into possible extensions is planned, including the exploration of the potential correlation between the structure of cost matrices and the total assignment cost and the dependence of the matrices structure and their connection to optimal solutions. The focus will be on analysing the stochastic dependence of a square matrix, let us say of size four, and exploring its multivariate properties. The aim will be to examine whether a correlation exists between the matrices' configuration, such as individual columns, and their impact on the optimal solution utilising copula theory and Sklar's theorem (1959) as analytical tools.

A copula is a probability model representing a multivariate uniform distribution, facilitating variable association analysis. This powerful statistical tool helps to discern the dependency structure in multivariate distributions and understand joint probabilities.

Copulas are functions that separate the marginal distributions from the dependency structure of a given multivariate distribution, as they can identify shady correlations observed in data.

Copula modelling has emerged as a pivotal tool in financial analysis, sparking debate since the 2008/2009 financial crisis. Despite challenges associated with its application, the comprehension of copulas retains significance in risk management. Currently, copulas find application in advanced financial analysis to understand better outcomes featuring fat tails and skewness.

The investigation will explore the possibility of identifying rules or patterns of structure of cost matrices having a high chance of yielding the optimal solution in a reasonable time.

6.2 Second Extension

A potential project application in the economic and commercial context is worth considering. The Three-index Assignment Problem pertains to the optimal allocation of tasks to machines within factories. This assignment problem can be applied to manufacturing TV solar panels in the UK, with the primary objective of optimising profit. Over the past decade, the United Kingdom has witnessed significant growth in solar PV installations, driven by government incentives and increasing demand for clean electricity. Escalating domestic energy prices, up by 221% for gas and 193% for electricity as of 2020, have spurred the expansion of the residential solar PV market, with increased solar module orders in 2020 and 2021. Anticipated technological advancements are expected to further boost residential solar PV demand in the future.

UKSOL, a solar PV module producer in Buckinghamshire, aims to manufacture a predetermined quantity of solar panels weekly or monthly for global distribution. Production involves fixed and variable costs for raw materials, equipment, maintenance, and labour. All incurred costs are assumed to be normally distributed. Imagine that UKSOL plans to expand its production throughout the UK, and the cost distribution varies across regions but remains follows a Normal probability distribution. A comparative analysis between PV solar panel manufacturing in China and the UK is worth conducting, assuming identical product specifications. A business person owning production units in both countries needs to satisfy global demands effectively. The project will aim to optimise task allocation to machines in different factories to achieve an optimal outcome that meets specific demand requirements. If a particular type of PV solar panels is exclusively produced in the UK and China, with costs following a normal distribution, the company manager will strategically plan the production quantities in the UK and China to serve a particular demand let us say a third country Italy.

The variation of the cost distributions affects the allocation of tasks to machines; economists often model cost distributions based on sectors. The next step will identify suitable references where economists have successfully estimated cost distributions for adoption in the Three-index Assignment project. Some authors have remarkably, formulated energy cost estimation models for various countries, offering valuable insights for the study.

Focusing solely on energy costs simplifies the model application. The availability of monthly time-series data for energy costs in each country facilitates the establishment of empirically derived cost distributions. This data comprises the "Index for Energy," providing daily energy cost prices. Leveraging energy cost estimation enables proactive planning of PV solar panel production, alternating between one week in the UK and the subsequent three weeks in China, optimising production decisions based on cost fluctuations. Energy cost estimation aids proactive scheduling of PV solar panel production for optimal efficiency and cost-effectiveness.

During the sixties, seventies and eighties, economists conducted extensive studies on cost distributions, particularly in manufacturing, as evidenced in prominent economic journals like the American Quarterly Review. Some investigations focused on cost distributions related to capital production. Concurrently researchers also explored wage distribution, concluding that wages are log-normal distributed. Analyses of wage distribution were consistently initiated by considering the log-normal model. In financial markets, stock returns are commonly assumed to be normally distributed, leading to the popularity of the normal probability distribution model supported by several authors who proved it to fit 95% of their cases.

Future research will focus on the costs structure by choosing either rows or columns of the matrices, situating the study in an economic framework. Labour distribution becomes variable as costs fluctuate due to varying job expenses across countries. Thus, this variability is critical for a comprehensive analysis of economic contexts.

In conclusion, it is imperative to explore empirical studies about the costs distribution related to the project.

Chapter 7

Conclusion

This thesis investigated the axial Three-Index Assignment Problem (3IAP), an extended version of the classical two-dimensional assignment problem. The study adopted an algorithmic approach to solving the problem and introduced two efficient heuristics, the Greedy-style Procedure (GSP) and the Diagonals Method (DM). These methods were further refined to manage tie-cases, resulting in three distinct variants. Furthermore, by exploring specific features of the cost matrices, two novel categories designed to solve 3IAP were proposed to address 3IAP.

Comprehensive computational tests demonstrated that the proposed heuristics efficiently generate feasible solutions, usually of high quality within polynomial time. This approach yielded 21 distinct solutions, from which the best result from the two heuristic classes was selected. Therefore, for every 3IAP case, a quality solution was achieved within a competitive time frame.

Two computer programs utilising Gurobi and Python-MIP solvers were developed and implemented for comparative evaluation. Each program reached optimal solutions within a competitive timeframe, benchmarked against existing data and randomly generated instances. However, the Python-MIP performance significantly exceeded Gurobi's, demonstrating superior efficiency in solving 3IAP.

The literature review indicates that the largest 3IAP instance previously solved to optimality was size n=26. The programs developed in this study have notably exceeded this benchmark, achieving optimal solutions for 3IAP instances of sizes n=35, 60, and 70, with the potential for more. For larger sizes, the proposed heuristics produced well-approximated solutions within a reasonable CPU time frame.

The research project then introduced the Genetic Algorithm (GA) as an approximation method, drawing inspiration from biological evolution and natural selection to solve *NP*-hard optimisation problems. The study specifically applied GA to 3IAP, endeavouring to achieve optimal or near-optimal solutions through the iterative evolution of a pool of candidate solutions across multiple generations.

Although traditional GAs achieve satisfactory results, for this study I introduced a hybrid genetic algorithm (HGA) that incorporates an effective local search technique, TLS, into the conventional GA structure, significantly enhancing solution quality for 3IAP. Using Python Jupyter Notebook for HGA deployment, starting with a random initial population, the algorithm iteratively attained optimal or near-optimal solutions for 3IAP over successive generations. The numerical experiments demonstrate HGA efficacy in efficiently handling large instances of 3IAP.

Chapters 3, 4 and 5 constitute the core contribution of this study. Summaries of Chapters 3 and 4 were submitted as an article to a specialised journal, RAIRO - Operations Research (Cairo-ro.org), which was recently accepted on July 4, 2024, and the manuscript is now under print. Following this thesis completion, Chapter 5 will be similarly prepared and submitted for publication, continuing the trend of disseminating significant research findings to the academic community.

My contribution presents a new algorithmic methodology for addressing 3IAP, highlighted by the development of efficient heuristics, exact algorithms for finding optimal solutions, and a hybrid genetic algorithm designed for large-scale instances, all poised for potential publication. This project has allowed me to acquire novel strategies for tackling *NP*-hard optimisation problems, enhance my computer programming proficiency in Python, and familiarise myself with the LaTeX word processing software.

References

[1] L. G. Afraimovich and M. D. Emelin, Combining Solutions of the Axial Assignment Problem. *Automation and Remote Control, 2021, 82*(8) (2021) 1418–1425.

[2] L. G. Afraimovich and M. D. Emelin, Complexity of Solutions Combination for the Three-Index Axial Assignment Problem. *Mathematics, 10*(7) (2022) 1062.

[3] Z. H. Ahmed, The ordered Cluster Travelling Salesman Problem: A Hybrid Genetic Algorithm. Research Article. Hindawi Publishing Corporation. *The Scientific World Journal* (2014), Article ID 258207, 13 pages. https://dx.doi.org/10.1155/2014/258207

[4] Z. H. Ahmed, Genetic Algorithm for the Traveling Salesman Problem using Sequential Constructive Crossover Operator. *International Journal of Biometrics and Bioinformatics (IJBB), 3*. (6) 96 –105.

[5] G. Appa, D. Magos and I. Mourtos, On Multi-index Assignment Polytopes. *Linear Algebra and its Applications, 416*(2-3) (2006) 224–241. ISSN 0024-3795, https://doi.org/10.1016/j.laa.2005.11.009.

[6] A. Arora, *Multiple Alignment of Protein Interaction Networks by Three-Index Assignment Algorithm*, MSc. thesis in Computer Science, University of Windsor, Ontario, Canada (2013).

[7] H. Azarbonyad and R. Babazadeh, A Genetic Algorithm for solving Quadratic Assignment Problem(QAP). In *Proc. of 5th International Conference of Iranian Operations Research Society*, ICIORS Tabriz, Iran, 2012, https://doi.org/10.48550/arXiv.1405.5050

[8] R. P., Badoni, J., Sahoo, M., Srivastava, Mann, D. K., Gupta, P. S., Verma, Stanimirović, L. A., Kazakovtsev, and D. Karabašević, An Exploration and Exploitation-

Based Metaheuristic Approach for University Course Timetabling Problems. *Axioms 2023, 12*(8) https://doi.org/10.3390/axioms12080720

[9] B. M. Baker and M.A. Ayechew, A genetic algorithm for the vehicle routing problem. *Computers and Operations Research 30* (2003), 787–800.

[10] E. Balas and P. R. Landweer, Traffic assignment in communication satellites. *Operations Research Letters, 2*(4) (1983) 141–147.

[11] E. Balas and L. Qi, Linear-time separation algorithms for the three-index assignment polytope, *Discrete Applied Mathematics, 43*(1) (1993) 1–12.

[12] E. Balas and M. J. Saltzman, An algorithm for the three-index assignment problem. *Operations Research, 39*(1) (1991) 150-161. Available from: https://doi.org/10.1287/opre.39.1.150.

[13] E. Balas and M. J. Saltzman, Facets of the three-index assignment polytope. *Discrete Applied Mathematics, 23*(3) (1989) 201–229.

[14] L. Boschetti, C. O. Justice, J. Ju and D. P. Roy, The collection 5 MODIS burned area product — Global evaluation by comparison with the MODIS active fire product. *Remote Sensing of Environment, 112*(9) (2008) 3690–3707.

[15] L. Buriol and P. M. França, A New Memetic Algorithm for the Asymmetric Traveling Salesman Problem. *Journal of Heuristics, 10*. (2004) Kluwer Academic Publishers. 483–506.

[16] R. E. Burkard, R. Rudolf and G. J. Woeginger, Three-dimensional axial assignment problems with decomposable cost coefficients,. *Discrete Applied Mathematics, 65*(1) (1996) 123–139. doi:10.1016/0166-218X(95)00031-L.

[17] P. Camion, Characterization of Totally Unimodular Matrices. *Proceedings of the American Mathematical Society, 16*(5) (1965) 1068–1073. https://doi.org/10.2307/2035618

[18] Y. Crama and F. C. R. Spieksma, Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research, 60*(3) (1992) 273–279. doi:10.1016/0377-2217(92)90078-N.

[19] . D. Cvetkovic and I. Parmee, Agent-based support within an interactive evolutionary design system. Published online by *Cambridge University Press* on 30 June 2003.

[20] C. Darwin, *On the Origin of Species, by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, 2nd ed., London (1859).

[21] T. Dokka, A. N. Letchford and M. H. Mansoor, Using surrogate relaxation as a matheuristic. *Internal report, Department of Management Science, Lancaster University*, Lancaster. UK (2021).

[22] T. Dokka and F. C. R. Spieksma, Facets of the axial three-index assignment polytope. *Discrete Applied Mathematics 201* (2016) 86–104. doi.org/10.1016/j.dam.2015.07.018.

[23] T. Dokka, Y. Crama and F. C. R. Spieksma, Multi-dimensional vector assignment problems. *Discrete Optimization, 14* (2014) 111–125. doi:10.1016/j.disopt.2014.08.005.

[24] M. Dörterler, O. F. Bay, and M. A. Akcayol, A modified genetic algorithm for a special case of the generalized assignment problem. *Turkish Journal of Electrical Engineering and Computer Sciences (2017), 25* 794–805. doi:10.3906/elk-1504-250.

[25] Z. Drezner and A. Misevičius, A. Enhancing the performance of hybrid genetic algorithms by differential improvement. *Computers and Operations Research 40* (2013) 1038–1046

[26] Z. Drezner and G. A. Marcoulides, A distance-based selection of parents in genetic algorithms. In: *Resende MGC, de Sousa JP, editors. Metaheuristics: computer decision-making*. Kluwer Academic Publishers; (2003) 257–78.

[27] R. Euler, Odd cycles and a class of facets of the axial 3-index assignment polytope. *Applicationes Mathematicae, 19*(3-4) (1987) 375–386.

[28] S. Faudzi, S. Abdul-Rahman and R. Abd Rahman, An assignment problem and its application in education domain: A review and potential path, Hindawi (2018). DOI:10.1155/2018/8958393.

[29] B. R. Fox and M. B. McMahon, Genetic Operators for Sequencing Problems. In: *Rawlins G, editor. Foundations of genetic algorithms*. San Mateo: Morgan Kaufmann; (1991). 284-–300.

[30] A. M. Frieze and J. Yadegar, An algorithm for solving 3-dimensional assignment problems with application to scheduling a teaching practice. *Journal of the Operational Research Society, 32*(11) (1981) 989–995.

[31] B. Gabrovšek, T. Novak, J. Povh, D.R. Poklukar, and J. Žerovnik, Multiple Hungarian method for k-assignment problem. *Mathematics, 8*(11) (2020) 2050. doi:10.3390/math8112050.

[32] M. R. Garey and D. S. Johnson, Computers and intractability. *Freeman*, San Francisco (1979).

[33] H. Gauvrit, J. Le Cadre and C. Jauffret, formulation of multitarget tracking as an incomplete data problem. *IEEE Transactions on Aerospace and Electronic Systems, 33*(4) (1997) 1242–1257..

[34] D. Goldberg and R. Lingle, Alleles, loci, and the traveling salesman problem. In *Proc. of the 1st International Conference on Genetic Algorithms and their Applications*, Lawrence Erlbaum, Hillsdale, NJ, (1985), 154–159.

[35] C. Groba, A. Sartal, and X. H. Vázquez, Integrating forecasting in metaheuristic methods to solve dynamic routing problems: Evidence from the logistic processes of tuna vessels. *Engineering Applications of Artificial Intelligence 76* (2018), 55–66.

[36] D. Grundel, R. Murphey, P.M. and Pardalos, Theory and algorithms for cooperative systems. *World Scientific* (2004).

[37] Gurobi (2024). https://www.gurobi.com/

[38] Q. M. Ha, Y. Deville, Q. D., Pham, and M. H. Hà, A hybrid genetic algorithm for the traveling salesman problem with drone. *Journal of Heuristics, 26* (2020), 219–247 https://doi.org/10.1007/s10732-019-09431-y

[39] J. He, S. Zhang, Z. Ren, X. Lai and Y. Piao, Approximate muscle guided beam search for three-index assignment problem. In *Proceedings of the 5th International Conference on Swarm Intelligence, 2014*. https://doi.org/10.48550/arXiv.1703.01893

[40] A. J. Hoffman and J. B. Kruskal, Integral boundary points of convex polyhedra, in *Linear Inequalities and Related Systems, Ann. Math. Stud.* 38, Princeton (1956).

[41] J. H. Holland, Adaptation in Natural and Artificial Systems, *University of Michigan Press*, Ann Arbor (1975).

[42] G. Huang and A. Lim, A hybrid genetic algorithm for three-index assignment problem. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation, CEC 2003* (2003) 2762–2768.

[43] A. Hussain, Y. S. Muhammad, M. N. Sajid, I. Hussain, A. M. Shoukry, and S. Gani, Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. *Hindawi Computational Intelligence and Neuroscience.* (2017), Article ID 7430125, 7 pages. https://doi.org/10.1155/2017/7430125

[44] W. R. Jih and Y. Hsu, A family competition genetic algorithm for the pick-up and delivery problems with time windows. In *Bulletin of the College of Engineering* (90). National Tawan University, (2004),121–130.

[45] M. Johnsson, G. Magyar and O. Nevalainen, On the Euclidean 3-Matching Problem. *Nordic Journal of Computing, 5* (1998).

[46] D. Kadhem, *Heuristic solution approaches to the solid assignment problem*, Ph.D. thesis, University of Essex, UK (2017).

[47] A. R. Kammerdiner and C. F.Vaughan, Very large-scale neighborhood search for the multidimensional assignment problem. *Optimization Methods and Applications.* Springer (2017) 251–262.

[48] A. Kammerdiner, A. Semenov and E. Pasiliao, Landscape properties of the very large-scale and the variable neighborhood search metaheuristics for the multidimensional assignment problem (2021). *arXiv Preprint*, arXiv:2112.11849.

[49] A. Kammerdiner, A. Semenov and E.L. Pasiliao, Multidimensional assignment problem for multipartite entity resolution. *Journal of Global Optimization* (2022) 1–33.

[50] D. Karapetyan and G. Gutin, Local search heuristics for the multidimensional assignment problem. *Journal of Heuristics, 17*(3) (2011) 201–249.

[51] D. Karapetyan and G. Gutin, A new approach to population sizing for memetic algorithms: A case study for the multidimensional assignment problem. *Evolutionary Computation, 19*(3) (2011) 345–371.

[52] R. M. Karp, (ed.) Reducibility among Combinatorial Problems. Complexity of Computer Computations. *The IBM Research Symposia Series*. ed. Boston, MA: Springer (1972) 85–103.

[53] V. M. Kravtsov, Combinatorial properties of non-integer vertices of a polytope in a three-index axial assignment problem. *Cybernetics and Systems Analysis, 43*(1) (2007) 25–33.

[54] V. M. Kravtsov, Description of the types of maximum non-integer vertices of the polyhedron in the three-index axial assignment problem. *Computational Mathematics and Mathematical Physics, 46*(10) (2006) 1821–1825.

[55] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society* (7) (1956) 48–50.

[56] H. W. Kuhn, The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly, 2*(1-2) (1955) 83–97.

[57] Y. Kuroki and T. Matsui,An approximation algorithm for multidimensional assignment problems minimizing the sum of squared errors. *Discrete Applied Mathematics, 157*(9) (2009) 2124–2135.

[58] J. Li, R. Tharmarasa, D. Brown, T., Kirubarajan, and K.R. Pattipati, A novel convex dual approach to three-dimensional assignment problem: Theoretical analysis. *Computational Optimization and Applications, 74*(2) (2019) 481–516.

[59] B. Lin, X. Sun, and S. Salous, Solving Travelling Salesman Problem with an Improved Hybrid Genetic Algorithm, *Journal of Computer and Communications, 4* (2016), 98–106. http://dx.doi.org/10.4236/jcc.2016.415009

[60] Y. Y. Liu and S. Wang, A scalable parallel genetic algorithm for the Generalized Assignment Problem, *Parallel Computing, 46* (2017), 98–119

[61] Y. K. Kim, K. Park, and J. Ko, A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling, *Computers and Operations Research, 30*(8) (2003), 1151–1171. DOI: 10.1016/S0305-0548(02)00063-1

[62] G. Magyar, M., Johnsson and O. Nevalainen, An adaptive hybrid genetic algorithm for the three-matching problem. *IEEE Transactions on Evolutionary Computation, 4*(2) (2000) 135–146.

[63] A.R. Mahjoub, Polyhedral Approaches. *Concepts of Combinatorial Optimization*, John Wiley & Sons, Inc. (2014) 261–324.

[64] A.R. Mahjoub, Two-edge connected spanning subgraphs polyhedra. *Mathematical Programming, 64* (1-3), Springer-Verlag (1994) 199–208.

[65] S. Mahmoudinazlou and C. Kwon, A Hybrid Genetic Algorithm for the min-max Multiple Traveling Salesman Problem. *arXiv:2307.07120v3*, July 2023, https://doi.org/10.48550/arXiv.2307.07120.

[66] S. N. Medvedev and O. A. Medvedeva, An Adaptive Algorithm for Solving the Axial Thre-Index Assignment Problem. *Automation and Remote Control, 2021, 80*(4) (2019) 718–732.

[67] M. Mehbali, (2024). https://github.com/mmehbali/Three-IAP

[68] M. Mehbali, The Vehicle Routing Problem, *Ars Combinatoria, 29 A* (1990), 129–140.

[69] A. D. Misevičius, (2008) Restart-based genetic algorithm for the quadratic assignment problem. In: Bramer M, Coenen F, Petridis M, editors. Research and development in intelligent systems. *Proceedings of AI-2008, the Twenty-eighth SGAI international conference on innovative techniques and applications of artificial intelligence*. London. Springer; (2008) 91–104.

[70] A. D. Misevičius and D. Verenė, A Hybrid Genetic-Hierarchical Algorithm for the Quadratic Assignment Problem. *Entropy 2021, 23*(108), (2021). https://doi.org/10.3390/e23010108.

[71] A. M. Mohammed, M. K. Abd Gani, R. I. Hameed, S. A. Mostafa, M. S. Ahmad, and D. A. Ibrahim, Solving vehicle routing problem by using improved genetic algorithm for optimal solution. *Journal of Computational Science* (21), July 2017, 255–262.

[72] G. Monge, Mémoire sur la théorie des déblais et des remblais, *Histoire de l'Académie Royale des Sciences de Paris, avec les Mémoires de Mathématique et de Physique pour la même année.* Paris (1781), 666–704.

[73] A. Mungwattana, K., Soonpracha, and G. K. Janssens, A Real-World, Case Study of a Vehicle Routing Problem under uncertain demand. *International Journal for Traffic and Transport Engineering, 2019, 9*(1), 101–117.

[74] J. Munkres, Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics, 5*(1) (1957) 32–38.

[75] S. Pal, I. Khan and M. K Maiti, A heuristic approach to solve multidimensional assignment problem. In *Proceedings of the 2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, IEEE (2018) 1–7.

[76] E.L. Pasiliao, P.M. Pardalos, and L.S. Pitsoulis, Branch and bound algorithms for the multidimensional assignment problem. *Optimization Methods and Software, 20*(1) (2005) 127–143.

[77] S. L. P. Pérez, *Personnel assignment problems through the multidimensional assignment problem*, Ph.D. thesis on Optimization Universidad Autónoma Metropolitana, Mexico City, Mexico (2017).

[78] W.P. Pierskalla, Letters to the editor - The multidimensional assignment problem. *Operations Research, 16*(2)(1968) 422–454. doi:10.1287/opre.16.2.422.

[79] W.P. Pierskalla, The tri-substitution method for the multidimensional assignment problem. *Canadian Operations Research Society, 5*(2) (1967) 71–81.

[80] A. B. Poore, S. Lu and B. J. Suchomel, Data association using multiple-frame assignments. In, *Handbook of multisensor data fusion*, CRC Press (2017) 319–338.

[81] A. B. Poore and S. Gadaleta, S. (2006). Some assignment problems arising from multiple target tracking. *Mathematical and Computer Modelling, 43*(9 - 10), 1074–1091.

[82] A. B. Poore, Multidimensional assignment problems arising in multitarget and multisensor tracking. In *Nonlinear Assignment Problems*, Springer (2000) 13–38.

[83] A. B. Poore and A. J. Robertson, A new Lagrangian relaxation based algorithm for a class of multidimensional assignment problem. *Computational Optimization and Applications, 8*(2) (1997) 129–150.

[84] L. Qi, L. and D. Sun, Polyhedral methods for solving three index assignment problems. In *Nonlinear Assignment Problems*, Springer (2000) 91–107.

[85] L. E. U. Rivero, M. R. B. Escarcega, and J. Velasco, An Integer Linear Programming Model for a Case Study in Classroom Assignment Problem. *Computacion y Sistemas, 24*(1) (2020) 97–104. ISSN 2007-9737. doi: 10.13053/CyS-24-1-3191

[86] J. D. Schaffer, R. A. Caruana and L. J. Eshelman, A study of control parameters affecting online performance of genetic algorithms. In: Schaffer JD, editor. *Proceedings of the 3rd international conference on genetic algorithms*. San Mateo Morgan Kaufmann (1989), 51–60.

[87] S. N. Sivanandam and S. N. Deepa, *Introduction to genetic algorithms*. Berlin, Heidelberg, New York, Springer; (2008).

[88] A. Sklar, Fonctions de Rpartition n Dimensions et Leurs Marges, France, Paris. *Publ. Inst. Statist. Univ. Paris*, Springer (1959) 229–231.

[89] F. C. R. Spieksma, Multi-index assignment problems: complexity, approximation, applications. *Nonlinear Assignment Problems*, Springer (2000) 1–12.

[90] F. C. R. Spieksma and G. J. Woeginger, Geometric three-dimensional assignment problems. *European Journal of Operational Research, 91*, (3) (1996) 611–618.

[91] A. S. Tasan and M. Gen, A genetic algorithm-based approach to vehicle routing problem with simultaneous pick-up and deliveries. *Computers and Industrial Engineering 62* (2012), 755–761.

[92] T. A. M. Toffolo and H. G. Santos, Introduction - Python-MIP documentation (2019). https://docs.python-mip.com/en/latest/intro.html

[93] I. H. Toroslu and Y. Arslanoğlu, Genetic algorithm for the personnel assignment problem with multiple objectives. *Information Sciences, 177.* (3) 787 –803. https://doi.org/10.1016/j.ins.2006.07.032.

[94] S. Vadrevu and R. Nagi, A dual-ascent algorithm for the multi-dimensional assignment problem: application to multi-Target tracking. In *Proc. of the 2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, IEEE (2020) 1–8.

[95] C.E. Valencia, C.A. Alfaro, F.J.Z. Martinez, M.C.V Magana, . and S.L.P. Pérez, Outperforming Several Heuristics for the Multidimensional Assignment Problem. In *Proc. of the 2018 15th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, IEEE (2018) DOI: 10.1109/ICEEE.2018.8533978.

[96] C.E. Valencia, F.J.Z. Martinez, S.L.P. and Pérez, A simple but effective memetic algorithm for the multidimensional assignment problem. In *Proc. of the 2017 14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, IEEE (2017) DOI: 10.1109/ICEEE.2017.8108889.

[97] C.J. Veenman, M.J.T. Reinders and E. Backer, Establishing motion correspondence using extended temporal scope. *Artificial Intelligence, 145*(1-2) (2003) 227–243.

[98] X. S. Wang, Chap. 5 Genetic Algorithm, in *Nature-Inspired Optimization Algorithms* (2014) Elsevier 77–87. https://doi.org/10.1016/B978-0-12-416743-8.00005-1.

[99] Y. Wu and P. Ji, Solving the quadratic assignment problems by a genetic algorithm with a new replacement strategy. *International Journal of Mathematical and Computational Sciences, 1* (6) (2007), pp. 269–273.

[100] E.R. Wulan, D. Jamaluddin, and W.N. Ramadhan, Determining optimal solutions in learning outcome using one to one fixed method, In *Proc. of the International Conference on Science and Engineering (ICSE-UIN-SUKA 2021)*, Atlantis Press (2021) 7–11.

[101] I. Younas, F. Kamrani, C. Schulte and R. Ayani, Optimization of task assignment to collaborating agents, *2011 IEEE Symposium on Computational Intelligence in Scheduling* (SCIS), Paris, France, 2011, 17–24. DOI: 10.1109/SCIS.2011.5976547.

[102] M. Yousefikhoshbakht, N. Malekzadehb, and M. Sedighpourc, Solving the Traveling Salesman Problem Based on The Genetic Reactive Bone Route Algorithm whit Ant Colony System. *International Journal of Production Management and Engineering 4*(2), (2016) 65–73.

[103] L. Zhang, D. Sidoti, S. Vallabhaneni, K.R. Pattipati and D.A. Castnon, Approaches to obtain a large number of ranked solutions to 3-dimensional assignment problems. *Journal of Advances in Information Fusion, 13*(1) (2017) 50–65.

[104] Y. Zhang, Y. Li, S. Li, J. Zeng, Y. Wang, and S. Yan, Multi-Target Tracking in Multi-Static Networks with Autonomous Underwater Vehicles Using a Robust Multi-Sensor Labeled Bernoulli Filter. *Journal of Marine Science and Engineering, 11*(4) (2023). 875. https://doi.org/10.3390/jmse11040875