Postgraduate in Optimization

Personnel assignment problems through the multidimensional assignment problem

Thesis author

Sergio Luis Pérez Pérez

for the grade

PhD in Optimization

Thesis advisors

PhD Carlos E. Valencia
PhD Francisco Javier Zaragoza Martínez

Mexico City, Mexico **July 2017**

Abstract

Personnel assignment problems appear at several industries. The efficient assignment of personnel to jobs, projects, tools, time slots, etcetera, has a direct impact in cost terms for the business.

The Multidimensional Assignment Problem (MAP) is a natural extension of the well-known assignment problem and can be used for applications where the assignment of personnel is required. The most studied case of the MAP is the three dimensional assignment problem, though in recent years have been proposed some local search heuristics and memetic algorithms for the general case. Let $X_1, \ldots, X_s$ be a collection of $s \geq 3$ disjoint sets, consider all the combinations that belong to the Cartesian product $X = X_1 \times \cdots \times X_s$ such that each vector $x \in X$, where $x = (x_1, \ldots, x_s)$ with $x_i \in X_i \; \forall \; 1 \leq i \leq s$, has associated a weight $w(x)$. A MAP in s dimensions is denoted as sAP. A feasible assignment is a collection $A = (x^1, \ldots, x^n)$ of n vectors if $x^i_k \neq x^j_k$ for each $i \neq j$ and $1 \leq k \leq s$. The weight of an assignment A is given by $w(A) = \sum_{i=1}^{n} w(x_i)$. The objective of sAP is to find an assignment of minimal weight.

We focused our attention in a particular case of personal assignment problem: the School Timetabling Problem (STP). The STP consider the assignment of personnel to other two or more sets, for example can be required the assignment of professors to courses to time slots to rooms, and sometimes also to students. The hardness of this problem is not only the effectiveness at solving the corresponding sAP but also at its modeling, since setting the weight for each possible combination is difficult by itself.

In this thesis we make a deep study of MAP by starting with the state-of-the-art of algorithms, heuristics and metaheuristics for its solving. We describe some algorithms and we propose a new one for solving optimally medium problem size instances of MAP. We propose the generalization of the known dimensionwise variation heuristics as well as a new generalized local search heuristic that provide a new state-of-the-art of local searches for MAP. We also propose a new simple memetic algorithm that is competitive against the state-of-the-art memetic algorithm for MAP.

In the last part of this thesis, we provide a brief description of the state-of-the-art for STP. Then, we introduce a new way for modeling this problem as a MAP and present the results of such modeling by applying it to a real life case of study: STP at UAM-A. We provide the particular aspects for STP at UAM-A and we propose a new solution for this problem. Our solution is based on the solving of several 3AP considering the introduced modeling and our proposed techniques.

Key words: Personnel assignment problem, multidimensional assignment problem, school timetabling problem, local search, dimensionwise variation algorithm, memetic algorithm.

Resumen

El problema de asignación de personal aparece en dicersas industrias. La asignación eficiente de personal a trabajos, proyectos, herramientas, horarios, entre otros, tiene un impacto directo en términos monetarios para el negocio.

El problema de asignación multidimensional (PAM) es la extensión natural del problema de asignación y puede ser utilizado en aplicaciones donde se requiere la asignación de personal. El caso más estudiado de PAM es el problema de asignación en tres dimensiones, sin embargo en años recientes han sido propuestas algunas heurísticas de búsqueda local y algoritmos meméticos para el caso general. Sean $X_1, \ldots, X_s$ una colección de $s \geq 3$ conjuntos disjuntos, considere todas las combinaciones que pertenecen al producto Cartesino $X = X_1 \times \cdots \times X_s$ tal que cada vector $x \in X$, donde $x = (x_1, \ldots, x_s)$ con $x_i \in X_i \ \forall \ 1 \leq i \leq s$, tiene asociado un peso $w(x)$. Un MAP en s dimensiones se denota como PAs. Una asignación factible es una colección $A = (x^1, \ldots, x^n)$ de n vectores si $x^i_k \neq x^j_k$ para cada $i \neq j$ y $1 \leq k \leq s$. El peso total de una asignación A está dado por $w(A) = \sum_{i=1}^{n} w(x_i)$. El objetivo de PA$s$ consiste en encontrar una asignación de peso mínimo.

Enfocamos nuestro estudio en un caso particular de problema de asignación de personal: el Problema de Asignación de Horarios (PAH). El PAH considera la asignadción de personal a otros dos o más conjuntos, por ejemplo puede ser es requerida la asignación de profesores a cursos a periodos de tiempo a salones, y algunas veces a estudiantes. La dificultad de este problema no sólo consiste en resolver eficientemente la correspondiente instancia sAP sino también en cómo modelarlo, esto debido a que el establecimiento de los costos es difícil en sí mismo.

En este trabajo de tesis se realiza un estudio profundo de PAM comenzando con un resumen del estado del arte de algoritmos, heurísticas y metaheurísticas para su resolución. Luego, se describen algunos algoritmos y se propone uno nuevo que resulve instancias de tamaño medio para PAM. También se propone la generalización de las conocidas heurísticas de varación de dimensión así como una nueva heurística generalizada de búsqueda local que proporciona un nuevo estado del arte de búsquedas locales para PAM. Adicionalmente, se propone un algoritmo memético simple que es competitivo en comparación con el algoritmo memético del estado del arte para PAM.

Hacia el final de la tesis, se presenta el estado del arte para PAH. Luego, se describe una nueva forma de modelar este problema como un PAM y presentamos los resultados de dicho modelado aplicándolo a un case real: el PAH en la UAM-A. Se mencionan los aspectos particulares de PAH en la UAM-A y proponemos una nueva solución la cual esta basada en la resolución de múltiples PA3, considera el modelado mencionado y aplica nuestros algoritmos y heurísticas propuestos.

Palabras clave: Problema de asignación de personal, problema de asignación multidimensional, problema de asignación de horarios, búsqueda local, heurísticas de variación de dimensión, algoritmo memético.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The fast growth of some companies results in the loss of visibility of most of their employees around its different departments. Personnel assignment is a difficult task and, most of the time, it is easier to hire a new person instead of using an existing resource. The efficient assignment of human resources is of great importance because it has a huge monetary impact.

Some examples where the assignment of personnel is relevant are: crew assignment in the navy; pilots and flight attendants to flights; nurses, surgeons and medical assistants to surgeries; professors, schedules, and rooms to courses; software engineers to technological projects; among many others.

The assignment of personnel can be modeled as an assignment problem. An assignment problem deals with the question of how to assign persons to jobs where each person is rated for a job through considering some aspects, for example: his skills for performing a job, his availability, his experience, etc.

Even when the assignment of personnel is a very common problem it is usually solved by hand. In general, the personnel is assigned to projects by their managers whom not always have the visibility of all the people working at the company. On the cases where the managers have the visibility of all their personnel, another problem consists in determining a rating for the relation person-job, which is even a more difficult task. Once the values for the relations are somehow calculated, the corresponding problem can be solved as an assignment problem.

However, there are some problems in which an assignment is required between persons and other two or more sets of objects, for example as in the school timetabling problem. In the school timetabling problem an assignment is required between professors, courses, rooms and time slots. This is an example of a multidimensional assignment problem which is a generalization of the classical assignment problem.

The assignment problem has been widely studied and several algorithms have been proposed for its resolution. The most popular algorithm was proposed by [Kuhn, 1955] and it is known as the Hungarian method which is a polynomial time algorithm. Faster algorithms are currently known [Ahuja et al., 1994], [Bertsekas, 2009],

[Goldberg and Kennedy, 1997], [Ramshaw and Tarjan, 2012a].

The multidimensional assignment problem has been studied mainly in the case with three dimensions. Only in the last ten years have been proposed some heuristics for the case with an arbitrary number of dimensions. The most popular algorithm was proposed by [Balas and Saltzman, 1991] for the case with 3 dimensions and is a branch and bound based technique which is an exponential time algorithm. It is known that, unless P = NP, there is no polynomial time algorithm to solve this problem: in 1972 Karp proved that 3 dimensional matchings is a NP-hard problem.

The multidimensional assignment problem is not only related to problems where the assignment of personnel is required, it has many other applications, e.g. for the multisensor data association problem where the objective is to determine which measurements from one or more sensors are related to the same object; for the problem of selecting roots of a system of polynomial equations; for the geometric three-dimensional assignment problem; among many others.

We consider the school timetabling problem as a case of study of assignment of personnel because it has been approached by many authors, although most of them do not deal with the multidimensional assignment problem involved, they solve a simplified version. On the other side, it is easy to get real data for this problem.

We claim that the same methodology developed for our case of study could be applied to other cases of assignment of personnel.

1.1 Motivation

The assignment problem is one of the most important problems in operations research, and the multidimensional assignment problem arises naturally in many problems in industry. However, there is no general purpose software that helps to deal with the problem of how to assign persons to jobs or resources where more than two sets are involved in the assignment.

Recently were published some heuristics and a memetic algorithm for the multidimensional assignment problem for the cases with an arbitrary number of dimensions ([Karapetyan and Gutin, 2011a] and [Karapetyan and Gutin, 2011b]). One of the main purposes of this thesis is to develop better heuristics to solve larger problem size instances.

Another motivation for this thesis was to solve a problem that involved the assignment of personnel, the school timetabling problem is a good option because still is a very difficult problem to solve.

We promote the development of a generic tool to solve instances of the school timetabling problem by considering some basic restrictions. Similar tools also could be used to solve other assignment problems where two or more sets are involved in the assignment.

1.2 Problem description

The school timetabling problem is taken as a case of study for the assignment of personnel.

The school timetabling problem could be modeled as a multidimensional assignment problem. It is stated as follows: let p be a set of n professors and let c be a set of n courses and let r be a set of n rooms and let t be a set of n time slots. A weight for the relation professor-course-room-time slot is somehow calculated. An assignment is a combination of the permutations of the elements from each set such that the elements from each set are present in only one relation and all the elements are present among all the relations.

The timetabling school problem can be formulated as a multidimensional assignment problem. In this case it is required an assignment of n professors to n curses to n rooms to n time slots.

The easiest way of assigning the costs of the relations in the timetabling school problem is to set a 0 value in the case of valid relations, that is, if a professor p is able to teach the course d in the room r at the time slot t, otherwise it should be 1.

Several options for the selection of the costs could be explored, the main disadvantage is that different ways of setting the costs may give a very different set of solutions. The only way to evaluate the results of this part is to compare them against the previous history and compare them to hand generated solutions.

The timetabling problem may vary widely from one educational institution to another. This thesis is focused on the generality of this problem more than in specific restrictions from particular scenarios.

1.3 Methodology

This thesis was developed by going through several stages. In summary, we started with the study of the state of the art of the assignment problem and, mainly, of the multidimensional assignment problem, then we proposed some new heuristics for the multidimensional assignment problem and, finally, we took a real instance of the school timetabling problem in order to test our heuristics.

In this way, the methodology adopted was as follows:

- The study of some state-of-the-art algorithms for the assignment problem and comparing them.
- The study of several algorithms and heuristics for the multidimensional assignment problem, focusing on those that were proven to be the best ones (in terms of quality solution and time complexity).
- Development of some algorithms for the multidimensional assignment problem.

- Development of several local search heuristics for the multidimensional assignment problem and its experimental evaluation.
- The study of the school timetabling problem.
- The procurement of a real instance of the school timetabling problem considering some real data.
- Modeling and solving this real timetabling problem as a multidimensional assignment problem.
- Analysis of results, conclusions and proposal of future work.

Some of these stages were accomplished before the start of my doctoral studies.

1.4 Thesis structure

This document is structured as follows: In Chapter 2 we state the assignment problem and we compare some of the best algorithms known for this problem. In Chapter 3 we stated the multidimensional assignment problem and we summarize its state-of-the-art. In Chapter 4 we propose several algorithms and heuristics for the multidimensional assignment problem, then we compare such procedures against state-of-the-art heuristics and meta-heuristics. In Chapter 5 we study the timetabling problem as a case of study for the multidimensional assignment problem. In Chapter 6 we give the conclusions and we propose some future work.

Chapter 2

The assignment problem

The assignment problem (AP) is introduced before the multidimensional assignment problem (MAP) because it is easier to start with the problem in its two dimensions version and then extended it to many dimensions. On the other side, it is necessary because some of the later presented heuristics consider a simplification of a multidimensional assignment problem to an assignment problem as part of its machinery.

The *assignment problem* deals with the question of how to assign a set X of n items to a set Y of n items such that $X \cap Y = \emptyset$. An assignment could be stated as a bijection φ of n items between X and Y. By considering such sets, the representation of an assignment is given by a permutation φ such that:

$$\begin{pmatrix} 1 & 2 & \dots & n-1 & n \\ \varphi(1) & \varphi(2) & \dots & \varphi(n-1) & \varphi(n) \end{pmatrix}$$

where 1 is mapped to $\varphi(1)$, ..., and n is mapped to $\varphi(n)$. For example, let $X = \{x_1, x_2, x_3, x_4\}$, $Y = \{y_1, y_2, y_3, y_4\}$ and the permutation $\varphi = \{3, 1, 4, 2\}$, then a possible assignment is:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ \varphi(3) & \varphi(1) & \varphi(2) & \varphi(4) \end{pmatrix} \Leftrightarrow \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_3 & y_1 & y_4 & y_2 \end{pmatrix}$$

Each permutation φ of the set with n items has a unique correspondence with the permutation matrix $P_\varphi = (p_{ij})$ of size $n \times n$ where:

$$p_{ij} = \begin{cases} 1 & \text{if } j = \varphi(i) \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, a permutation matrix is constrained to the system of linear equations:

$$\begin{aligned} \sum_{j=1}^{n} p_{ij} = 1 \text{ for } i \text{ with } 1 \leq i \leq n \\ \sum_{i=1}^{n} p_{ij} = 1 \text{ for } j \text{ with } 1 \leq j \leq n \end{aligned} \tag{2.1}$$

where $p_{ij} \in \{0, 1\}$ for all i, j with $1 \leq i, j \leq n$.

In the case of the previous example where $X = \{x_1, x_2, x_3, x_4\}$, $Y = \{y_1, y_2, y_3, y_4\}$ and $\varphi = \{3, 1, 4, 2\}$, the matrix corresponding to the system of linear equations is:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

An assignment can also be described through bipartite graphs. Let $G = (X, Y; E)$ be a bipartite graph with disjoin vertex sets X and Y and with edges $E \subseteq X \times Y$. A matching M in G is a subset of edges of E such that every vertex of G meet at exactly one edge of the matching. In this way an assignment could be represented as a matching M of G. The representation for the last example of an assignment as a matching in a bipartite graph is shown in the Figure 2.1.

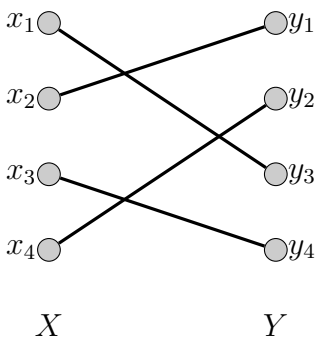

Figure 2.1: Representation of an assignment as a matching in a bipartite graph.

The assignment problem becomes an optimization problem when we consider the cost of assigning some $x \in X$ to some $y \in Y$. Let $C = (c_{ij})$ be a matrix of size $n \times n$, where c_{ij} is the cost of assigning x_i to y_j for all i, j with $1 \leq i, j, \leq n$.

Given the assignment problem as a permutation φ an let S_n be the set of all the permutations with n items, the objective is defined as:

$$\min_{\varphi \in S_n} \sum_{i=1}^{n} c_{i\varphi(i)} \tag{2.2}$$

Similarly, if the permutation matrix is related to the cost matrix C then the corresponding 0-1 integer linear programming formulation is:

$$\begin{aligned} \min & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}p_{ij} \\ \text{subject to} : & \sum_{j=1}^{n} p_{ij} = 1 \text{ for } i \text{ with } 1 \leq i \leq n \\ & \sum_{i=1}^{n} p_{ij} = 1 \text{ for } j \text{ with } 1 \leq j \leq n \end{aligned} \tag{2.3}$$

where $p_{ij} \in \{0, 1\}$ for all i, j with $1 \leq i, j \leq n$.

For the case of an assignment formulated as a matching in a bipartite graph, consider weighted edges E, then the cost of a matching is:

$$c(M) = \sum_{(x,y)\in M} c(x, y) \tag{2.4}$$

where the objective is to minimize $c(M)$.

There are some variants of the problem for the cases when the sizes of the sets are different or not all the relations are present. Such variants are described by considering an assignment as a matching in a bipartite graph.

Let $G = (X \cup Y; E)$ be a bipartite graph that admits either $|X| = |Y|$ or $|X| \neq |Y|$, a bipartite graph is said to be balanced when $|X| = |Y|$, otherwise, the bipartite graph is said to be unbalanced. Moreover, it could be that M in G does not include all the vertices of G even when $|X| = |Y|$. A perfect matching is said to be a matching in which $|M| = |X| = |Y|$. In 1935, Hall provided the necessary and sufficient conditions for the existence of a perfect assignment [Hall, 1935].

Let $\nu(G)$ be the maximum size of any matching in G an let τ be a target value, we require a matching M in G whose cost is minimum compared to all the possible matchings of size τ. According to [Ramshaw and Tarjan, 2012b], three variants of an assignment problem can be stated:

Perfect assignments: Let G be a balanced bipartite graph with weighted edges. If $\nu(G) = n$, then calculate the perfect matching of minimum cost in G; otherwise the assignment is not feasible.

Imperfect assignments: Let G be a bipartite graph, either balanced or unbalanced, with weighted edges and let $t \geq 1$ be a target size. Calculate the matching of minimum cost in G of size $\tau = \min(t, \nu(G))$.

Incremental assignments: Let G be a bipartite graph, either balanced or unbalanced, with weighted edges. Calculate the minimum cost matchings in G of sizes $1, 2, \ldots, \nu(G)$ presenting the result for each size. The size τ is selected from the closed interval $[1, \nu(G)]$, the process ends when the desired value of τ is reached.

Perfect assignments are easier than the other two variants and the most difficult are the incremental assignments. The most studied case has been perfect assignment due to imperfect assignments can be reduced to perfect assignment as is described in **??**. For the problem of incremental assignments the best algorithms known are just variants of the Hungarian method.

2.1 State of the art

The first algorithm for the assignment problem was proposed by [Easterfield, 1946] and its temporal complexity was $O(2^n n^2)$, however the problem was described in terms of a combinatorial problem more than as an assignment problem.

The assignment problem was formally described in a paper entitled "The personnel assignment problem" [Votaw and Orden, 1953].

The Hungarian method was proposed by [Kuhn, 1955]. One year later Kuhn proposed more variants of the assignment problem [Kuhn, 1956]. The name of "Hungarian method" was because the algorithm was largely based on the earlier work of two Hungarian mathematicians: König and Egervary. The Hungarian method was reviewed in [Munkres, 1957], where was determined that its time complexity was $O(n^4)$. The Hungarian method is also known as the Kuhn-Munkres algorithm.

Since then, a lot of algorithms have been proposed for the assignment problem. In general, there are some general methodologies to solve it, here we describe the most relevant among them.

2.1.1 Auction algorithms

Auction algorithms for the assignment problem were introduced by [Bertsekas, 1981] early in the eighties. This type of algorithms has been used extensively in business environments to determine the best prices between a set of offered products.

An auction algorithm is an iterative procedure where we compare a set of offers and then a sell is performed to the best bidder, the goal of the algorithm is to select optimal prices and an assignment that maximizes the benefit. The classical methods for the assignment problem are based on iterative improvements of some cost function, which may be a primal cost (similar to primal simplex methods) or a dual cost (as in the Hungarian method), but auction algorithms perform local updates which may deteriorate both the primal and dual cost, although in the end it finds an optimal assignment, which is due to the principle of approximate optimality. This is detailed explained in [Bertsekas, 2009].

The auction algorithms are excellent at solving perfect assignments. Even when some auction algorithms cannot deal directly with problems in unbalanced graphs, there are some techniques that allow us to transform an assignment problem over an unbalanced graph to an equivalent problem in a balanced graph by adding some dummy vertices and edges so that this type of algorithm is able to solve the corresponding problem. Theoretical time complexity of faster auction algorithms is approximately $O(nm \log(nC))$ where n is the number of vertices, m is the number of edges and C is the maximum weight among all the edges.

One of the main advantages of auction algorithms for the assignment problem is that they are highly parallelizable, as shown in [Bertsekas, 1988] and, for the matching problem, as shown in [Naparstek and Leshem, 2016]. This is due to the fact that auction algorithms perform local improvements which can occur at the same time. Theoretical time complexity of parallel auction algorithms is approximately $O(n^2 \log n)$ if implemented on n parallel machines.

There are several works that deal with different variants of the assignment problem and its applications, e. g. for the classical linear network flow problem and some of its special cases as max-flow and shortest path [Bertsekas, 1992], for the asymmetric assignment problem [Bertsekas and Castanon, 1993], for multi-assignment problems where persons may be assigned to several objects and conversely [Bertsekas et al., 1993].

2.1.2 Weight scaling algorithms

Weight scaling algorithms were introduced by [Gabow and Tarjan, 1989] later in the eighties. This is one of the most common techniques to solve the assignment problem.

A weight scaling algorithm creates a flow network based on the bipartite graph of the corresponding assignment problem. By using a parameter called ϵ, it performs some scaling phases aimed to reduce the flow error obtained at each of the previous scaling phases. Each scaled phase gives an approximate optimal solution. The minimum cost is obtained at the last scaling phase. Theoretical time complexity of the first weight scaling algorithm was $O(\sqrt{n}m \log nC)$ where n is the number of vertices, m is the number of edges and C is the maximum weight among all the edges.

In contrast with auction algorithms, this type of technique is not so good at solving perfect assignments but it is able to solve imperfect assignments. Another difference is that, whereas auction algorithms perform local improvements, weight scaling algorithms perform global updates. Weight scaling algorithms could also be parallelizable as is shown in [Gabow and Tarjan, 1989], which allows to obtain a time complexity of $O(n^2 \log n)$ using n processors. Even when auction algorithms and weight scaling algorithms present some differences, [Orlin and Ahuja, 1992] was able to combine both ideas for creating a hybrid version to solve assignment problems.

This type of technique could also be used to solve other type of problems, e. g. for the shortest path problem [Goldberg, 1995] and for network problems that work by scaling the numeric parameters [Garbow, 1985].

Weight scaling algorithms were later improved by [Ramshaw and Tarjan, 2012a], achieving a complexity of $O(m\sqrt{s}\log sC)$ and by [Duan and Su, 2012] reducing the complexity to $O(m\sqrt{n}\log n)$.

2.1.3 Push relabel algorithms

Push-relabel algorithms for the assignment problem were introduced by [Goldberg and Kennedy, 19
in the middle nineties.

A push-relabel algorithm, also known as preflow-push algorithm, is an algorithm for computing maximum flows. This type of technique consists in converting a preflow into a maximum flow by moving flow between neighboring nodes using push operations under the guidance of relabel operations. A push relabel algorithm is able to solve an assignment problem by converting the bipartite graphs into a flow network .

This type of technique can be combined with a weight scaling technique in order to obtain better bounds for the assignment problem as in [Goldberg and Kennedy, 1997] where they achieve a reduction to $O(m\sqrt{n}\log(nC))$.

2.2 Selection of an efficient algorithm

Some of the heuristics that will be described in the next chapter require the use of a solver for instances of the assignment problem. The type of instances that will be solved correspond to the variant of perfect assignments. In order to choose a fast algorithm for this purpose three implementations of state-of-the-art algorithms were experimentally evaluated.

2.2.1 The Hungarian method

The selected version of the Hungarian method consists in a improvement of its original version and it was proposed by [Ramshaw and Tarjan, 2012a].

As the classical Hungarian method, this version works on the bipartite graph of the assignment problem. Algorithm 1 shows the general structure of the Hungarian method.

Let $G = (X \cup Y; E)$ be the bipartite graph of the assignment problem with vertices in $X \cup Y$ and weights in E, the Hungarian method works as follows: the algorithm starts with an empty matching, then builds up its matching by augmenting along tight augmenting paths. By using a variant of Dijkstra's algorithm a shortest path forest is build aimed to reach all the remaining $x \notin A$, if some y is reachable from some x through an alternating path then a new augmenting path of minimum cost is obtained. At each step of the algorithm a matching of size s and minimum cost is obtained. The algorithm ends when the maximum size $\nu(G)$ of any matching at G is reached. This algorithm has a overall time complexity of $O(ms + s^2 \log r)$ where m

Algorithm 1: Improved version of the Hungarian method (by Ramshaw and Tarjan)

Input: $G = (X \cup Y; E)$: Bipartite graph of an assignment problem.
Result: A: The min-cost assignment of size s.

```
Set A to the empty matching;
Set prices at X to 0, at Y to C;
Let ν(G) be the max size of any matching at G. for s in 0 : ν(G) do
    use Dijkstra to build a shortest path forest with roots at all x ∉ A;
    if some y ∉ A was reached then
        Raise prices to tighten the tree path to y;
        Augment A along that tight path;
    else
        return A of size ν(G);
```

is the number of edges, $s = \nu(G)$ and $r = |Y|$. This algorithm is explained in detail in[Ramshaw and Tarjan, 2012a].

The Hungarian method solves the problem of incremental assignments which is more difficult than perfect assignments, however it is the obligated reference in order to have a clear idea about how good are other algorithms against this technique.

2.2.2 The flow assign algorithm

In order to evaluate a weight scaling technique the FlowAssign algorithm proposed by [Ramshaw and Tarjan, 2012b] was implemented.

In contrast with the Hungarian algorithm, this technique works on a derived flow network from the original bipartite graph. Algorithm 2 shows the general structure of the FlowAssign algorithm.

Let $G = (X \cup Y; E)$ be the bipartite graph of the assignment problem with vertices in $X \cup Y$ and weights in E, let t be a target value of the desired size of the assignment such that $t \leq \min(|X|, |Y|)$ and let N_G be a derived flow network from G, the FlowAssign algorithm works as follows: first, the Hopcroft-Karp algorithm is applied in order to obtain a matching M of size s such that $s = t$ if assignment of size t exists or $s = \nu(G) < t$, otherwise. Then, the matching M is converted into an integral flow f on N_G, this transforms the problem of finding minimum cost matchings in G to the equivalent problem of finding minimum cost integral flows in N_G. A set of scalings is performed by starting from some predefined ϵ value that denotes the precision of the solution reached at each scaling-phase. The Refine function builds a shortest-path forest, finds a maximal set P of augmenting paths that are compatible and augment along them. It starts by considering prices for the vertices that are $\bar{\epsilon}$ optimum and it is improving the current solution at each iteration. Once a required

Algorithm 2: FlowAssign algorithm (by Ramshaw and Tarjan)

Input: $G = (X \cup Y; E)$: Bipartite graph of an assignment problem;
t: The required size of the assignment.
Result: A: The min-cost assignment of size s;
s: The maximum size reached of an assignment such that $s \leq t$.

```
(M, s) = HopcroftKarp(G, t);
Convert M into an integral flow f on N_G with |f| = s;
Set ε = ε̄;
∀v ∈ N_G set the prices p_d(v) = 0;
while ε > ε̲ do
    ε = ε/q;
    Refine(f, p, ε) builds a shortest path forest, finds a maximal set of
    augmenting paths and augments along them;
Round prices to integers that make all arcs proper;
```

precision $\underline{\epsilon}$ is reached, the main loop ends and the last step round prices to integers that make all arcs proper and gives the required solution. This algorithm has a overall time complexity of $O(m\sqrt{s}\log(sC))$ where m is the number of edges, s is the size of the assignment A and C is the maximum weight in the bipartite graph. This algorithm is explained in detail in [Ramshaw and Tarjan, 2012a].

The FlowAssign algorithm solves the problem of imperfect assignments which is more difficult that perfect assignments but easier than incremental assignments, this is why time complexity is better than the one of the Hungarian method.

2.2.3 The $\epsilon-$scaling auction algorithm

In order to evaluate a faster algorithm, we consider the $\epsilon-$scaling auction algorithm proposed by [Bertsekas, 2009]. In particular, an specific implementation provided by [Vargas, 2017] was evaluated.

This technique differs from the previous ones because this performs local improvements in order to reach a global optimum whereas the other ones perform global updates for the same goal. The $\epsilon-$scaling Auction algorithm operates like a real auction. The core of this algorithm is that, at each step of the process, a condition called *complementary slackness* should be kept. This condition provides the conditions for the feasible primal and dual solutions of an assignment problem to be optimal. This algorithm only works on feasible instances of the problem of the perfect assignment problems category. The Algorithm 3 shows the general structure of this auction algorithm.

Let $G = (X \cup Y; E)$ be the bipartite graph of the assignment problem with vertices in $X \cup Y$ and weights in E and let ϵ be a required parameter, the $\epsilon-$scaling auction

Algorithm 3: The $\epsilon-$scaling Auction algorithm (by Bertsekas)

Input: $G = (X \cup Y; E)$: Bipartite graph of an assignment problem;
$\epsilon > 0$: The precision parameter to satisfy the Complementary Slackness condition.
Result: A: The min-cost assignment of size n.

```
A = ∅;
Let p the prices that will satisfy the ε− Complementary Slackness condition
(such that w(x1y1) − p(y1) ≤_{y2∈N(x1)} min{w(x1y2) − p(y2)} + ε);
while is some x ∉ A do
    Consider some x1 ∉ A;
    Find the edges x1y1, x1y2 with the two minimum costs;
    if If u has only 1 neighbor then
        Set γ = ∞;
    else
        Set γ = (w(x1y1) − p(y2)) − (w(x1y2) − p(y1));
    Set p(y1) = p(y1) − γ − ε;
return A
```

algorithm works as follows: first, a matching A is set up to the empty set and any arbitrary initial prices p are considered. At each step of the main loop an unassigned vertex of the set X is considered as well as its two neighbors with the two minimum costs. Then, depending on the number of neighbors of the selected vertex its price is updated. If the procedure is applied to an instance that has perfect matchings then the procedure always terminates with an optimal assignment, otherwise, the main loop will never end. This algorithm has an overall time complexity of $O(mn \log(nC))$ where m is the number of edges, n is the number of vertices (recall here $n = |X| = |Y|$) and C is the maximum weight in the bipartite graph. This algorithm is explained in detailed in [Vargas, 2017].

2.3 Families of instances for the AP

One way to distinguish families of instances for the assignment problem is by the number of vertices in X and Y. In this way, there are two general types of families of instances: the family of instances with an equal number of vertices ($|X| = |Y|$) and the family of instances with a different number of vertices $|X| \neq |Y|$, abbreviated ENV and DNV respectively. We are interested on the family of instances ENV.

The easiest way to generate a testbed for the family of instances ENV is to consider a complete bipartite graph with uniformly random generated weights in the closed interval $[a, b]$. To consider a complete bipartite graph is not the only option however, for the purposes of this thesis, the complete bipartite graph is the only type

of instances we are going to analyze. On the other side, all the instances that consider non-complete bipartite graphs can be transformed into instances with complete bipartite graphs where the missing edges can be added with a very high value (or very low depending on the optimization function) such that those edges can be discarded from the optimal solution. This is why, in order to provide a fairest comparison between each solver will be considered other distributions.

The next subsections describe the used types of distributions to set the weights of the edges among some instances of the type ENV that consider complete bipartite graphs. Some distributions are continuous and are described by a probability density function which is used to specify the probability of the random variable for falling within a range of values. The other distributions are discrete and are described by a probability mass function which gives the probability for a discrete random variable to be an exact value.

2.3.1 Uniform distribution

The uniform distribution is a family of symmetric distributions such that each member of the family have the same probability to occur. This distribution is described by the next probability density function:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{for } x < a \text{ or } x > b \end{cases} \tag{2.5}$$

This family of instances will be generated by considering x under the closed integer interval $[1, 100]$.

2.3.2 Normal distribution

The normal distribution is a family of continuous distributions such that each member x is associated to the normal random variable with a cumulative probability. This distribution is described by the following function:

$$f(x) = [1/\sigma\sqrt{2\pi}]e^{-(x-\mu)^2/2\sigma^2}. \tag{2.6}$$

This family of instances have the particularity that the sum of the members generated have a bell distribution.

This family of instances will be generated by considering the parameters $\mu = 50$ and $\sigma = 10$ such that the generated values tend to 50 and the general values belong to an interval near to $[1, 100]$. Since this is a continuous distribution the generated values will be truncated in order to get only the corresponding integer values.

2.3.3 Poisson distribution

The Poisson distribution is a discrete probability distribution that models the probability of the number of events occurring in a given interval of time or space. The events occur within a known average rate and independently of the time since the last event. This distribution is described by the following probability mass function:

$$Pr(k \text{ events in the interval}) = e^{-\lambda}\frac{\lambda^k}{k!} \tag{2.7}$$

This family of instances will be generated by considering a $\lambda = 50$ such that the generated values tend to something similar like in the normal distribution. For this distribution will be used a formulation proposed by [Ahrens and Dieter, 1982] who provided samples from Poisson distributions of mean $\mu \geq 10$ by truncating suitable normal deviates and applying a correction with low probability. The advantage of such technique is that provides a competitive method for random generating values.

2.3.4 Binomial distribution

The binomial distribution is a discrete probability distribution that considers the probability of the number success events in a sequence of n independent experiments with a success probability of p. This distribution is described by the following probability mass function:

$$Pr(x = k) = \binom{n}{k} p^k (1-p)^{n-k} \text{ for } k = 0, 1, \ldots n. \tag{2.8}$$

This family of instances will be generated by considering the parameters $n = 100$ and $p = 0.5$ such that the generated values tend also to 50, within the closed interval of $[0, 100]$.

2.3.5 Hypergeometric distribution

The hypergeometric distribution is a discrete probability distribution that describes the probability of k successes in n draws, without replacement, from a population of size N which contains exactly K successes. This distribution is described by the following probability mass function:

$$Pr(x = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}. \tag{2.9}$$

This family of instances will be generated by considering the parameters $N = 200$, $K = 100$, $n = 100$ such that the generated values tend also to 50, within the closed interval of $[0, 100]$. For this distribution will be used a formulation proposed by [Kachitvichyanukul and Schmeiser, 1985].

2.4 Performance results

All the instances were generated in the programming language R version 3.1.3 because it provides an easy way for number generating under the previously described distributions.

The Algorithm 4 shows the used code to generate the five families of instances for the AP. Each generated instance file consists on a header value n that indicates the number of vertices on the bipartite graph (with $n = |X| = |Y|$); then are following n^2 integer values corresponding with the n^2 edge costs given in the order $w(x_1, y_1)$, ..., $w(x_1, y_n)$, $w(x_2, y_1)$, ..., $w(x_2, y_1)$, ..., $w(x_n, y_1)$, ..., $w(x_n, y_n)$. This format is commonly used when are considered complete bipartite graphs. There were generated five instances for each combination of *distribution* and N for a total of 125 instances.

Algorithm 4: Instances generator for a variety of families of instances for the AP

Input:

Result: A total of (number_of_instances * length(n_values) * 5) instances for the AP under the distributions uniform, normal, Poisson, binomial, and hypergeometric.

```
1  set.seed(0);
2  Set n_values < − c(1024);
3  Set number_of_instances < − 5;
4  for iteration in 1:number_of_instances do
5      for n in n_values do
6          edges < − n * n;
7          values < − round(runif(edges, min = 1, max = 100));
8          write(c(n, values), file = sprintf("ap_unif_n%d_k%d.in", n, iteration),
           ncolumns = 1);
9          values < − abs(round(rnorm(edges, 50, 10)));
10         write(c(n, values), file = sprintf("ap_norm_n%d_k%d.in", n, iteration),
           ncolumns = 1);
11         values < − rpois(edges, 50);
12         write(c(n, values), file = sprintf("ap_pois_n%d_k%d.in", n, iteration),
           ncolumns = 1);
13         values < − rbinom(edges, 100, 0.5);
14         write(c(n, values), file = sprintf("ap_binom_n%d_k%d.in", n, iteration),
           ncolumns = 1);
15         values < − rhyper(edges, 100, 100, 100);
16         write(c(n, values), file = sprintf("ap_hyper_n%d_k%d.in", n, iteration),
           ncolumns = 1);
```

The format name for the instance files is ap_*distribution*_nN_k*iteration*.in where

distribution corresponds with one of unif (uniform), norm (normal), pois (Poisson), binom (binomial), and hyper (hypergeometric); N corresponds with the number of vertices; and *iteration* corresponds with the consecutive number for the generated instance.

All the instances were solved through a improved version of the Hungarian method, the FlowAssign algorithm, and the $\epsilon-$scaling Auction algorithm. These algorithms were implemented in C++ and its performance was evaluated on a platform with an Intel Core i5-3210M 2.5 GHz processor with 4 GB of RAM under Windows 8.

The Table 2.1 shows the running times for the described families of instances.

Table 2.1: Averaged running times for the five families of instances for the AP solved through Auction, FlowAssign and Hungarian algorithms.

Family of instances	N	Averaged seconds for five instances Auction	FlowAssign	Hungarian
Binomial	64	0.005	0.017	0.018
	128	0.007	0.053	0.073
	256	0.034	0.386	0.451
	512	0.118	1.431	3.178
	1024	0.427	6.441	17.372
Hypergeometric	64	0.005	0.017	0.018
	128	0.009	0.062	0.081
	256	0.029	0.276	0.414
	512	0.107	1.196	2.623
	1024	0.471	6.692	18.222
Normal	64	0.004	0.018	0.02
	128	0.008	0.069	0.099
	256	0.026	0.291	0.41
	512	0.083	1.157	2.571
	1024	0.441	7.023	19.491
Poisson	64	0.006	0.018	0.017
	128	0.008	0.055	0.077
	256	0.023	0.243	0.403
	512	0.085	1.112	2.486
	1024	0.374	6.255	16.882
Uniform	64	0.005	0.022	0.021
	128	0.009	0.061	0.081
	256	0.032	0.261	0.442
	512	0.171	1.296	3.153
	1024	0.864	5.147	23.712
Ave. time		0.134	1.583	4.492

In the Table 2.1, each value represents the averaged running times for the five instances for each combination of family of instances and problem size N. It can be observed that the fastest algorithm was the $\epsilon-$scaling Auction algorithm, followed by the FlowAssign algorithm and then by the Hungarian algorithm. In fact, this is the expected behavior since each algorithm is more able to solve a different version of the assignment problem: Hungarian method for incremental assignments (which is the most difficult version of the AP), the FlowAssign algorithm for imperfect assignments, and the $\epsilon-$scaling Auction algorithm for perfect assignments (which corresponds to the easiest version of the AP). All the generated instances corresponds with the problem of perfect assignments. This is why, the $\epsilon-$scaling Auction algorithm outperformed the other two algorithms.

By considering the averaged total time results reported in Table 2.1, it can be observed that the ϵ-scaling Auction algorithm is 11.8 times faster than the FlowAssign algorithm whereas it is 2.8 times faster than the Hungarian method. The highlight result is that the ϵ-scaling Auction can be 33 times faster than an improved version of the Hungarian method. This comparison is relevant because a solver for the version of perfect assignment problem is required as part of some heuristics that will be described in the next chapter. Those heuristics are able to get better solutions if are executed many times in certain period of time, so the possibility to have a better algorithm to solve instances of perfect assignment allows to increase the quality solution in a shorter period of time.

Finally, the Table 2.2 shows the sum of the running times showed in the Table 2.1 for the families of instances solved by each algorithm.

Table 2.2: Comparative results for the sum of running times of Auction, FlowAssign and Hungarian over the families of instances for the AP.

	Auction	FlowAssign	Hungarian
Binomial	0.591	8.328	21.092
Hypergeometric	0.621	8.243	21.358
Normal	0.562	8.558	22.591
Poisson	0.496	7.683	19.865
Uniform	1.081	6.787	27.409
Average	0.670	7.920	22.463
Standard dev.	0.234	0.710	2.930

In the Table 2.2, the row Average shows the averaged running times for each family of instances and the row Sstandard dev. shows the standard deviation σ among the running times for each family. It can be observed that the families of instances binomial, hypergeometric and normal seems to have the same degree of difficulty whereas the behavior over the Poisson family and the uniform family is a bit different. The Poisson family of instances seems to be the easiest family of

instances to solve under these algorithms, except for the FlowAssign algorithm. In the opposite, the uniform family of instances seems to be the most difficult, except for the FlowAssign algorithm, in whose case resulted in the easiest family.

Based on the experimental results of Table 2.2 we can conclude that the weights distribution have an impact on the algorithmic performance. In the case of the $\epsilon-$scaling Auction algorithm and the FlowAssign algorithm the time complexity have an explicit relation with the weights but, in the case of the Hungarian method, such relation is not part of the theoretical time complexity. In this case, the experimental evaluation showed that some relation should exist. The correct path of such analysis is out of the scope of this thesis work however it is proposed as future work.

2.5 Conclusions

There is a wide variety of algorithms that solves different versions of the assignment problem and is recommendable to use the better algorithm according to the version to solve.

Here were implemented some of the best algorithms to solve each version of the problem in order to show the expected performance in practice for the version of the problem that we care, which is the perfect assignment version.

There were proposed five families of instances based on five types of distributions for the weights generation of each family of instances. This families were specifically generated for the perfect assignment problem under complete bipartite graphs.

The $\epsilon-$scaling Auction algorithm obtained the fastest running times for solving all the families of instances proposed and the proportional relation between its running times and those obtained by the Hungarian algorithm is of approximately $30x$ times faster.

The weights distributions are important because they have an impact on the implemented algorithms, however it is unclear the type of structure that it is easier or more difficult to solve at each case.

This study was very important to do because some of the heuristics that are used as part of this thesis work were considering a classical version of the Hungarian method, however by considering the $\epsilon-$scaling Auction algorithm, which is a more adequate algorithm for the required tasks, the performance of such heuristics is considerable better. Such results are presented in the next chapter.

Chapter 3

The multidimensional assignment problem

The multidimensional assignment problem (MAP), also known as sAP in the case of s dimensions, is a natural extension of the well-known assignment problem (AP).

The *multidimensional assignment problem*, also called the axial multi-index assignment problem, deals with the question of how to perform an assignment between the elements of s disjoint sets with n items at each.

Let $s \geq 2$ be a fixed number of dimensions and let $X_1, X_2, \ldots, X_s$ be a collection of s disjoint sets, without loss of generality we assume that $n = |X_1| = |X_2| = \cdots = |X_s|$ (otherwise we add some dummy elements to equilibrate them), a sAP could be equivalently stated in one of several ways.

A s-dimensional assignment can be stated as $s-1$ bijections φ_i (for $1 \leq i < s$) of n items where φ_1 maps X_1 and X_2, ..., φ_{s-1} maps X_1 and X_s. Then, the representation of an assignment is given by a set of $s-1$ permutations φ_i such that:

$$\begin{pmatrix} 1 & 2 & \ldots & n-1 & n \\ \varphi_1(1) & \varphi(2) & \ldots & \varphi(n-1) & \varphi(n) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \varphi_{s-1}(1) & \varphi_{s-1}(2) & \ldots & \varphi_{s-1}(n-1) & \varphi_{s-1}(n) \end{pmatrix}$$

where 1 is mapped to $\varphi_1(1)$, ..., and n is mapped to $\varphi_1(n)$, and $\varphi_1(1)$ is mapped to $\varphi_2(1)$, ..., and $\varphi_1(n)$ is mapped to $\varphi_2(n)$, and so on. For example, let $s = 3$ and $X_1 = \{x_1^1, x_2^1, x_3^1, x_4^1\}$, $X_2 = \{x_1^2, x_2^2, x_3^2, x_4^2\}$ and $X_3 = \{x_1^3, x_2^3, x_3^3, x_4^3\}$ and the permutations $\varphi_1 = (3, 1, 4, 2)$ and $\varphi_2 = (2, 3, 1, 4)$, then a possible assignment is:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ \varphi_1(3) & \varphi_1(1) & \varphi_1(4) & \varphi_1(2) \\ \varphi_2(2) & \varphi_2(3) & \varphi_2(1) & \varphi_2(4) \end{pmatrix} \Leftrightarrow \begin{pmatrix} x_1^1 & x_2^1 & x_3^1 & x_4^1 \\ x_3^2 & x_1^2 & x_4^2 & x_2^2 \\ x_2^3 & x_3^3 & x_1^3 & x_4^3 \end{pmatrix}.$$

Each combination of the permutations $\varphi_1, \ldots, \varphi_{s-1}$ of the sets with n items has a

unique correspondence with the permutation matrix $P_{\varphi_1,\ldots,\varphi_{s-1}} = (p_{i^1 i^2 \ldots i^s})$ of size n^s where:

$$p_{i^1 i^2 \ldots i^s} = \begin{cases} 1 & \text{if } i^2 = \varphi_1(i^1) \text{ and } \ldots \text{ and } i^s = \varphi_{s-1}(i^1) \\ 0 & \text{otherwise} \end{cases}.$$

Furthermore, a permutation matrix is constrained by the following system of linear equations:

$$\begin{aligned} \sum_{i^2=1}^{n} \sum_{i^3=1}^{n} \cdots \sum_{i^s=1}^{n} p_{i^1 i^2 \ldots i^s} &= 1 \text{ for } i \text{ with } 1 \leq i^1 \leq n \\ &\ldots \\ \sum_{i^1=1}^{n} \sum_{i^2=1}^{n} \cdots \sum_{i^{s-1}=1}^{n} p_{i^1 i^2 \ldots i^s} &= 1 \text{ for } i \text{ with } 1 \leq i^s \leq n \end{aligned} \tag{3.1}$$

where $p_{i^1 i^2 \ldots i^s} \in \{0, 1\}$ for all $i^1, i^2, \ldots, i^s$ with $1 \leq i^1, i^2, \ldots, i^s \leq n$.

The previous example corresponds with the permutation matrices:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

An assignment can also be described through multipartite hypergraphs. Let $G = (X_1, X_2, \ldots, X_s; E)$ be a multipartite hypergraph with disjoin vertex sets X_1, X_2, $\ldots$, X_s and with hyperedges $E \subseteq X_1 \times X_2 \times \cdots \times X_s$. A matching M in G is a subset of hyperedges of E such that every vertex of G meets at most one edge of the matching. In this way an assignment could be represented as a matching M of G. The representation for our example as a matching in a bipartite graph is shown in Figure 3.1. Each hyperedge of the assignment is represented with a different color, for example, vertices x_1^1, x_3^2 and x_2^3 are related through a hyperedge colored with black.

The multidimensional assignment problem becomes an optimization problem when we consider the cost of assigning some $x^1 \in X_1$ to some $x^2 \in X_2, \ldots,$ to some $x^s \in X_s$. Let $C = (c_{i^1 i^2 \ldots i^s})$ be a matrix of size n^s, where $c_{i^1 i^2 \ldots i^s}$ is the cost of assigning $x_{i^1}^1$ to $x_{i^2}^2$ to $\ldots$ to $x_{i^s}^s$ for all $i^1, i^2, \ldots, i^s$ with $1 \leq i^1, i^2, \ldots, i^s \leq n$.

Given the assignment problem as a combination of permutations $\varphi_1, \varphi_2, \ldots, \varphi_{s-1}$, let CS_n be the set of all the combinations of $s-1$ sets of permutations with n items at each, the objective is defined as:

$$\min_{\varphi_1 \varphi_2 \ldots \varphi_{s-1} \in CS_n} \sum_{i=1}^{n} c_{i \varphi_1(i) \varphi_2(i) \ldots \varphi_{s-1}(i)} \tag{3.2}$$

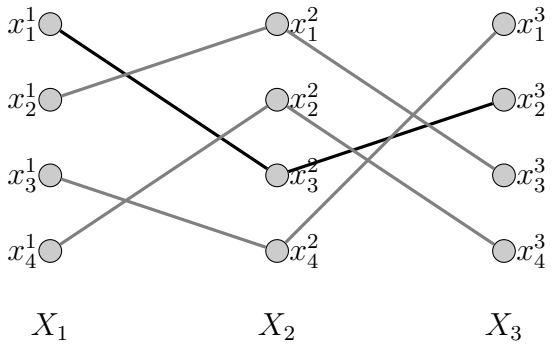

Figure 3.1: Representation of a 3-dimensional assignment as a matching in a multipartite hypergraph.

Similarly, if the permutation matrix is related to the cost matrix C then the corresponding 0-1 integer linear programming formulation is:

$$\begin{aligned}
\min & \sum_{i^1=1}^{n}\sum_{i^2=1}^{n}\cdots\sum_{i^s=1}^{n} c_{i^1 i^2 \ldots i^s} p_{i^1 i^2 \ldots i^s} \\
\text{subject to}: & \sum_{i^2=1}^{n}\sum_{i^3=1}^{n}\cdots\sum_{i^s=1}^{n} p_{i^1 i^2 \ldots i^s} = 1 \text{ for } i^1 \text{ with } 1 \leq i^1 \leq n \\
& \sum_{i^1=1}^{n}\sum_{i^3=1}^{n}\cdots\sum_{i^s=1}^{n} p_{i^1 i^2 \ldots i^s} = 1 \text{ for } i^2 \text{ with } 1 \leq i^2 \leq n \\
& \cdots \\
& \sum_{i^1=1}^{n}\sum_{i^2=1}^{n}\cdots\sum_{i^{s-1}=1}^{n} p_{i^1 i^2 \ldots i^s} = 1 \text{ for } i^s \text{ with } 1 \leq i^s \leq n
\end{aligned} \tag{3.3}$$

where $p_{i^1 i^2 \ldots i^s} \in \{0, 1\}$ for all $i^1, i^2, \ldots, i^s$ with $1 \leq i^1, i^2, \ldots, i^s \leq n$.

For the case of an assignment formulated as a matching in a multipartite hypergraph, consider weighted hyperedges E Then the cost of a matching is:

$$c(M) = \sum_{(x^1, x^2, \ldots, x^s) \in M} c(x^1, x^2, \ldots, x^s) \tag{3.4}$$

where the objective is to minimize $c(M)$.

3.1 State of the art

The multidimensional assignment problem has been studied since the fifties by [Schell, 1955] and [Koopmans and Beckmann, 1957] and was formally described by [Pierskalla, 1968].

[Karp, 1972] showed that the problem of deciding whether there exists a 3-dimensional matching of size at least k is NP-complete and, consequently, the optimization problem of finding the largest 3-dimensional matching is NP-hard. [Frieze, 1974] proposed for the first time an integer programming formulation for the 3AP. In general, the Multidimensional Matching Problem (*MMP*), abbreviated sMP, is a particular case of the MAP in which we assign a value of 0 to the present relations and a value of 1 to those non present, the objective is to find a multidimensional assignment of minimum cost. The optimal assignment will contain the relations of the largest s-dimensional matching plus some non present relations which we need to discard. The MMP is NP-hard and, indeed, the MAP is NP-hard for every $s \geq 3$ as shown in [Garey and Johnson, 1979].

It has been proven that unless P = NP, there is no ϵ-approximate polynomial time algorithm for the multidimensional assignment problem [Crama and Spieksma, 1992]. However, the special case of 3AP where a distance, verifying the triangle inequalities, is defined on the elements from each dimension, and the cost of the weight is either the sum of the lengths of its distances or the sum of the lengths of its two shortest distances was proven to be ϵ-approximable [Crama and Spieksma, 1992].

In the middle nineties [Spieksma and Woeginger, 1996] studied two more geometric special cases of the 3AP: given are three sets X_1, X_2, X_3 of n points in the Euclidean plane one possible goal is to find a partition of $X_1 \cup X_2 \cup X_3$ into n three-colored triangles such that (a) the total circumference of all triangles is minimum or (b) the total area of all triangles is minimum. The special cases were proven to be NP-hard.

Even when the MAP is NP-hard, [Grundel et al., 2004] studied weights coefficients from three different random distribution: uniform, exponential and standard normal. They showed that in the cases of uniform and exponential distributions, experimental data indicates that the mean optimal value converges to zero when the problem size increases. Such results allow to have an estimation about the optimal value of instances generated under some random distributions.

About the same time, [Grundel and Pardalos, 2005] proposed a test problem generator for the MAP. The advantage of this generator is that it guarantees the existence of a unique solution. Its main disadvantage falls in its time complexity which is exponential.

The most studied case of MAP is the 3AP, though in recent years several algorithms and heuristics were proposed for the sAP.

In order to provide a better summary and analysis about the state-of-the-art for the multidimensional assignment problem a general classification of the developed techniques was created. The next subsections describe such categories.

3.1.1 Exact algorithms

There are just a few papers that describe algorithms to solve exactly the multidimensional assignment problem. Even when several techniques can be applied aimed to find optimal solutions in a shorter time, as soon as the size of the instance starts to grow, the exponential time complexity of any exact technique is unavoidable.

One of the first algorithms was developed by [Balas and Saltzman, 1991]. They presented a branch and bound algorithm for 3AP. In their work, they apply a Lagrangian relaxation which incorporates a class of facet inequalities in order to find lower bounds. The apply a primal heuristic based on the principle of minimizing maximum regret and a variable depth interchange phase for finding upper bounds. The results are reported for instances with $n \in \{4, 6, \ldots, 24, 26\}$ vertices and uniformly random generated weights $w(x) \in \{0, \ldots, 100\}$, obtaining running times from some seconds to some minutes. The time complexity of this algorithm is not presented but, based on the time results, it seems to be approximately $O(2^n)$.

A more recent technique was presented by [Magos and Mourtos, 2009]. They studied the classes of Clique facets for the axial and planar assignment polytopes, then they developed a polynomial-time separation procedure which allows to incorporate such facet classes within an Integer Programming solver. This reduced the solving time of instances of the multidimensional assignment problem. Some of the studied facets that they considered were taken from the work of [Balas and Saltzman, 1991], in this way, this work represents an improvement of the older Branch and Bound algorithm of [Balas and Saltzman, 1991].

3.1.2 Approximate algorithms

The approximate algorithms are valid only for some particular cases of the geometric version of the 3AP, however, they have been widely studied and many authors have developed several heuristics.

Early in the nineties, [Crama and Spieksma, 1992] described the first approximation algorithms for the special case of the 3AP when the costs are associated with the triangle inequalities. In their work it was showed that the geometric special cases of the 3AP are ϵ-approximate and proposed 1/2 and 1/3 approximate algorithms, i. e. heuristics which always deliver a feasible solution whose cost is at most 3/2 and 4/3, respectively, the optimal cost. The time complexity of such heuristics is $O(n^3)$.

Some years later, [Bandelt et al., 1994] extended the ideas previously provided by [Crama and Spieksma, 1992] to more than three dimensions. [Bandelt et al., 1994] defined four cost functions that allow to deal with the problem similarly to the geometric version in three dimensions. Several approximation algorithms and heuristics for each particular case were proposed.

Later in the nineties, [Johnsson et al., 1998] described a slightly different version of the geometric special case of the 3AP in which it is required to find a partition

set of $n = 3p$ points into p disjoint subsets, each consisting of three points; the objective is to minimize the total cost of the triplets. They provide some of the first ideas of the possible approximate algorithms for this problem but could not provide a proper upper bound. Instead, they developed other heuristics based on tabu-search, simulated annealing and genetic algorithms.

In recent years, [Kuroki and Matsui, 2009] also extended the geometric special case of 3AP to the d-dimensional space and the weight of an edge is defined by the square of the Euclidean distance between its two endpoints. Their work reduced the previously known bound from $(4 - 6/s)$ to $(5/2 - 3/s)$ times the optimal value. The time complexity of this approximation algorithm is $O(s \cdot n^4)$

3.1.3 Local search heuristics

In this field there is a wide variety of methods that consider different classes and sizes of neighborhoods. The main characteristic of this type of technique is that it evaluates some selected neighborhood and if some improvement could be obtained then the procedure moves to the corresponding new solution. Most of the developed techniques evaluate several neighborhoods in some predefined way and end when no improvement is obtained.

Some of the first local search heuristics were described by [Balas and Saltzman, 1991] for the 3AP. They described a local search heuristic called *variable depth interchange heuristic* which consists on considering two triplets of vertices of a current feasible solution and look for a combination that improves the cost of both triplets. Other heuristics called *greedy*, *reduced cost*, and *minimax-regret* were proposed and are detailed explained in the same work. All the heuristics were tested on instances with $n \in \{20, 25, \ldots, 65, 70\}$ vertices and uniformly random generated weights $w(x) \in \{0, \ldots, 1000\}$, obtaining running times of less than a minute. The time complexity for the greedy and reduced cost heuristics is $O(n^2)$, for the minimax-regret heuristic is $O(n^3 \log n)$ and for the variable depth interchange heuristic is $O(\binom{n}{2}) \sim O(n^2)$.

Early in the 2000s, [Robertson, 2001] presented a greedy randomized adaptive search procedure (*GRASP*) for the MAP which considered as part of its machinery the four heuristics proposed by [Balas and Saltzman, 1991]. A GRASP is a multistart metaheuristic for combinatorial optimization problems. It consists of a construction procedure based on a greedy randomized algorithm of a local search. The four variants of the GRASP were tested on instances with $s = 5$ and $n = 25$. The experimental evaluation was focused on instances of the data association problem that appear in the centralized multisensor multitarget tracking systems. Even when all the heuristics are described in detail the time complexity and the effectiveness of such heuristics is not reported.

About the same time, [Huang and Lim, 2003] developed a heuristic framework called Fragmental Optimization for the MAP. This heuristic consists in an iterative

improvement algorithm that follows the principle of easy things first. The goal of the heuristic is to optimize a portion or fragment of the entire problem iteratively. The experimental evaluation was performed on the data set instances provided by [Balas and Saltzman, 1991] and by [Crama and Spieksma, 1992], obtaining the optimal values in all data sets. The running times showed an improvement in comparison with the previous works.

Some years later, [Aiex et al., 2005] presented another GRASP with path relinking for the 3AP. Path relinking is an intensification strategy that explores trajectories that connect high-quality solutions while the iterations of the GRASP occurs. Seven variants of such heuristics were developed and evaluated on the instances provided by [Balas and Saltzman, 1991] and by [Crama and Spieksma, 1992]. The time complexity of all the heuristics is $O(n^3)$ for each iteration. The reported results consider executions that go from 100 until 10,000 iterations.

Later in the 2000s, [Gutin et al., 2007] carried out the worst-case analysis of the greedy and max-regret heuristics proposed by [Balas and Saltzman, 1991]. They showed that max-regret may find the unique worst possible solution for some instances of the 3AP. Finally, two new heuristics based on max-regret are proposed but its experimental evaluation was not performed.

Early in the 2010s, [Karapetyan and Gutin, 2011a] proposed several local search heuristics and generalized some others for the MAP. The most representative heuristics were called dimensionwise variation heuristics, k-opt, and variable depth interchange. A dimensionwise variation heuristic consists in a simplification of a sAP to a 2AP. This allows to explore neighborhoods of size $O(n!)$ and to find a local optimum on it. The k-opt heuristics considers a feasible solution and takes a set of k vectors from the feasible assignment and optimizes them such that a local optimum over such vectors is achieved. The used values for k were 2 and 3. The variable depth interchange heuristics is a variation of the one proposed by [Balas and Saltzman, 1991]. In contrast with many of the previous authors, [Karapetyan and Gutin, 2011a] proposed several new families of instances and evaluated their heuristics on each. The families of instances consider problem sizes with $s = \{3, 4, 5, 6, 7, 8\}$ and $n = \{150, 50, 30, 18, 12, 8\}$, respectively. The reported results showed a relative solution error of approximately 5% in most of the instances with the best heuristic. The dimensionwise variation heuristic was proven to be the best local search heuristic for the MAP in the last years.

In recent years, [Nguyen et al., 2014] developed a new approach based on cross-entropy methods for the MAP. Cross-entropy methods can be applied to combinatorial optimization problems. These methods consist on the construction of a random sequence of solutions which converges probabilistically to the optimal or near-optimal solution. This methods have two main steps: in the first one are generated random data according to some pre-established mechanism; in the second step the parameters of the random mechanism are updated to produce a better sample in the next iteration. The experimental evaluation was performed on the data set provided by

[Grundel and Pardalos, 2005] and the reported results showed a relative solution error of approximately 5% which is similar to the obtained by the dimensionwise variation heuristics, however the test bed used by [Karapetyan and Gutin, 2011a] contains larger problem sizes.

3.1.4 Evolutionary algorithms

An evolutionary algorithm is a generic population based metaheuristic optimization technique. This type of algorithms uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. The procedure starts with a set of candidate solutions having the role of population and, through a fitness functions, the quality of the solutions is measured. This type of technique has proven to perform well approximating solutions to many optimization problems.

One of the first genetic algorithms in this line was proposed by [Magyar et al., 2000] for the 3MP. They proposed a genetic algorithm hybridized with some local search heuristics, which also contains an adaptive control parameters that tunes the parameters at the running time. Even though they used their own random generated data set, they claimed to obtain a relative solution error that is approximately 3.7% away from the optimal solutions, which improved the earlier results obtained by [Johnsson et al., 1998].

In the middle 2000s, [Huang and Lim, 2006] designed a new genetic algorithm hybridized with a local search heuristic for the 3AP, which was the base for the designing of more powerful heuristics for the MAP. A generic genetic algorithm was considered but a local search heuristic replaced the mutation operator. The local search heuristic they used consists on a simplification of a 3AP to a 2AP and this idea was the base for the creation of the dimensionwise variation heuristics later proposed by [Karapetyan and Gutin, 2011a]. Until that moment, this heuristic outperformed all the previous heuristics for the 3AP. The experimental evaluation was performed by considering the classical data set provided by [Balas and Saltzman, 1991].

Even when the concept of memetic algorithm (which is a genetic algorithm combined with a local search heuristic) is older [Moscato, 1989], the term was popularized later in the 2000s and some authors preferred to use the term genetic algorithm hybridized with a local search heuristic.

Some years later, [Bozdogan and Efe, 2008] designed an ant colony optimization heuristic for the MAP. This type of heuristic consists on iteratively constructing random candidate solutions which are biased to be in good regions of the problem space under the influence of two forces: information about the specific problem and pheromone trails. The information about the specific problem is gathered from a fitness function to be optimized whereas the pheromone trail is a specialty of the ant colony optimization achieved by the positive feedback from ant paths constructed throughout the algorithm. This method did not represent a significant improvement however the authors let some open research lines in order to build better heuristics

with this ideas.

Late in the 2000s, [Gutin and Karapetyan, 2009] proposed the first ideas for a new memetic algorithm for the MAP which was pretended to be combined with the dimensionwise variation heuristic. At the same time, such work described the preliminary ideas for the development of a general purpose memetic algorithm aimed to be able to vary the population size at the running time of the algorithm in order to start with a lot of individuals and to have the best ones to the end of the execution.

Early in the 2010s, [Karapetyan and Gutin, 2011b] finally described a new general purpose memetic algorithm which was able to vary the population size during the running time. The first sections of the work describe how to build such a memetic algorithm and how its parameters should be tuned in order to vary the population size. By the end of such work, some families of instances of the MAP are solved by using this technique and the results are compared against the best known heuristics for the MAP. The obtained results through its experimental evaluation showed a relative solution error of approximately 1% which outperformed all the previous known heuristics for the MAP and represents the state of the art for solving the MAP.

3.2 Families of instances for the MAP

Several families of instances have been proposed to evaluate the effectiveness of algorithms and heuristics for the MAP. Here we summarize the most relevant families, which are used in this thesis. The main difference between families lies in the way we set the weight of the tuples of the corresponding instance.

The most common family of instances is the Random family that considers an independent weight distribution. Other families were introduced in the works of [Karapetyan and Gutin, 2011a] and [Karapetyan and Gutin, 2011b] with so-called decomposable weights such as Clique, Square Root, Geometric and Product.

3.2.1 The Random family

This family has the property that the weight of each tuple is assigned with a value generated uniformly at random over some closed interval $[a, b]$.

The best known family of instances of this type was provided by [Balas and Saltzman, 1991] for the 3AP. It includes 60 test instances with problem size $s = 3$ and $n \in \{4, 6, \ldots, 24, 26\}$. For each n, five instances were created with the integer weight coefficients $w(x)$ generated uniformly at random in the interval $[0, 100]$. The names of the instances are referred in this work as s_bs_n where $s = 3$, bs comes from Balas and Saltzman and n is the number of vertices.

A second family of instances of this type was provided by [Magos and Mourtos, 2009] for the 4AP, 5AP, and 6AP. It includes 200 test instances with three different dimension sizes: $s = 4$ and $n \in \{10, 11, \ldots, 19, 20\}$; $s = 5$ and $n \in \{7, 8, \ldots, 13, 14\}$; $s = 6$

and $n = 8$. For each combination of s and n, ten instances were created with the integer weight coefficients $w(x)$ generated uniformly at random in the interval $[1, n^s]$. The names of the instances are referred in this work as s_axial_n where s is the number of dimensions and n is the number of vertices.

A third family of instances was provided by [Karapetyan and Gutin, 2011a] and [Karapetyan and Gutin, 2011b]. It includes 120 test instances with four different dimension sizes: $s = 3$ and $n \in \{40, 70, 100\}$; $s = 4$ and $n \in \{20, 30, 40\}$; $s = 5$ and $n \in \{15, 18, 25\}$; $s = 6$ and $n \in \{12, 15, 18\}$. For each combination of s and n, ten instances were randomly generated. The names of the instances are referred in this work as srn where s is the number of dimensions, r comes from the word Random and n is the number of vertices.

3.2.2 The Clique family

This family of instances has weights defined through s-partite graphs $G = (X_1 \cup \cdots \cup X_s, E)$. The weight $w(e)$ of every edge $e \in E$ was generated uniformly at random in the interval $[1, 100]$. Let C be a clique in G and let E_C be the set of edges induced by this clique, then the weight of a vector, corresponding to the clique C, is given by:

$$w_C(E_C) = \sum_{e \in E_C} w(e). \tag{3.5}$$

A specific set of instances for this family was provided by [Karapetyan and Gutin, 2011b], which includes 120 test instances with four different dimension sizes: $s = 3$ and $n \in \{40, 70, 100\}$; $s = 4$ and $n \in \{20, 30, 40\}$; $s = 5$ and $n \in \{15, 18, 25\}$; $s = 6$ and $n \in \{12, 15, 18\}$. For each combination of s and n, ten instances were randomly generated. The names of the instances are referred in this work as $scqn$ where s is the number of dimensions, cq comes from the word Clique and n is the number of vertices.

3.2.3 The Square Root family

This family of instances is similar to the Clique family. The main difference is that it considers the square root of a sum of squares of the involved weights, as in Equation 3.6.

$$w_{SR}(E_C) = \sqrt{\sum_{e \in E_C} w(e)^2}. \tag{3.6}$$

A specific set of instances for this family was provided by [Karapetyan and Gutin, 2011b] under the same conditions as for the Clique family. The names of the instances are referred in this work as $ssqn$ where s is the number of dimensions, sq comes from the word square root and n is the number of vertices.

3.2.4 The Geometric family

This family of instances is a special case of the Clique family. In this case, the sets $X_1, \ldots, X_s$ corresponds to s sets of points in a s-dimensional Euclidean space, and the distance between two points $p \in X_i$ and $q \in X_j$ is defined by the Euclidean distance.

$$d_G(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}. \tag{3.7}$$

It has been proven by [Spieksma and Woeginger, 1996] that the Geometric family of instances for $s \geq 3$ is NP-hard but they can be ϵ-approximated.

A specific set of instances for this family was provided by [Karapetyan and Gutin, 2011a], which includes 60 test instances with six different dimension sizes: $s = 3$ and $n = 150$; $s = 4$ and $n = 50$; $s = 5$ and $n = 30$; $s = 6$ and $n = 18$; $s = 7$ and $n = 12$; $s = 8$ and $n = 8$. For each combination of s and n, ten instances were randomly generated. The names of the instances are referred in this work as sgn where s is the number of dimensions, g comes from the word Geometric and n is the number of vertices.

3.2.5 The Product family

This family of instances was originally proposed by [Burkard et al., 1996]. In the Product family, the weight of a vector x is defined as follows:

$$w_P(x) = \prod_{j=1}^{s} a^j_{x_j} \tag{3.8}$$

where a^j is an array of n values, selected uniformly at random from $\{1, \ldots, 10\}$.

A specific set of instances for this family was provided by [Karapetyan and Gutin, 2011a] under the same conditions as for the Geometric family. The names of the instances are referred in this work as spn where s is the number of dimensions, p comes from the word Product and n is the number of vertices.

3.3 Exact algorithms

In this section we describe some basic algorithms and a state-of-the-art implementation to solve the MAP.

There are many ways to represent a solution of a MAP but the most representative is a matrix of size $n \times s$ where each of the n rows is a vector of size s and represents a selected tuple from the set of hyperedges E. Such representation allows easily to validate the feasibility of an instance because it reduces to verifying that all the elements from each column are different between them and belong to the set $\{1, \ldots, n\}$.

3.3.1 Brute force

A natural solution consists in evaluating all the possible combinations from Equation 3.2 to find the optimum. This gives a time complexity of $O(n!^{s-1})$ and a space complexity of $O(n \cdot s)$.

Algorithm 5 shows a generic implementation through recursion of the Brute Force algorithm for the MAP which generates all the possible assignments of a sAP through the generation of all the possible combinations of permutations of sets with n elements. At each recursive call, the i-th column of the matrix A is assigned with a permutation of the corresponding vector x_i. At each call of the base case a new possible assignment is evaluated and the best one is updated according to the best (minimum) cost.

The relevance of this technique is due to some local search heuristics, as 2-opt and 3-opt [Karapetyan and Gutin, 2011a], using this technique as part of their machinery and, it offers a better option for solving very small instances of sAP (at least in comparison with more complex techniques).

Algorithm 5: The brute force algorithm for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n.
Result: A: The min cost assignment as a matrix of size $n \times s$.

```
1  Set A_{n×s} := 0;
2  Set A^1 := {1, ..., n};
3  Set bestCost := ∞;
4  Set bestAssignment := A;
5  Call BruteForce(s, A);
6  BruteForce ← function(currentDimension, A){
7  if currentDimension = 1 then
8      if bestCost > calculateCost(A) then
9          bestCost := calculateCost(A);
10         bestAssignment := A;
11     return;
12 Set x_{currentDimension} := {1, ..., n};
13 Set origin := x;
14 repeat
15     Set A^{currentDimension} = x_{currentDimension};
       // A^i refers to the column i of the matrix A
16     Call BruteForce(currentDimension - 1, A);
17     Set y := next_premutation(x);
18 until x = origin;
19 };
20 return bestAssignment;
```

3.3.2 Wise brute force

This approach consists in generating all the first $O(n!^{s-2})$ possible Cartesian products of permutations for the first $s-1$ sets and then applying a 2AP solution for each combination against the last set. This allows a complexity time reduction from $O(n!^{s-1})$ to $O(n!^{s-2}n^3)$, and the space complexity changes from $O(n \cdot s)$ to $O(n \cdot s + n^2)$.

Algorithm 6 shows the improved version of the brute force algorithm for the MAP. The main difference against the brute force algorithm is that in this case there is one recursion level less by stopping at $currentDimension = 2$ instead of at $currentDimension = 1$ and in the base case the optimal permutation of the column 2, for the current state of A, is found through the solving of a 2AP. This allows to reduce the complexity of this level from $O(n!)$ to $O(n^3)$ which is the general complexity of solving a 2AP.

This algorithm is relevant because it provided the first ideas to the creation of the dimensionwise variation heuristics.

3.3.3 Dynamic programming reduction

This approach consists in reducing the set of all possible candidates from $O(n!^{s-1})$ to $O(2^{(s-1)\cdot n})$ by memorizing the optimal solution for some sub-problems of the original problem in a similar way that it is performed in the dynamic programming solution for the Traveling Salesman Problem proposed simultaneously by [Bellman, 1962] and [Held and Karp, 1962].

Let $S_1, \ldots, S_s$ be sets of n vertices each. Lets denote by S_i^k the set of any k vertices of S_i, for $1 \leq i \leq n$, and let $X^k = \{k\} \times S_2 \times \cdots \times S_s$ be all members of the Cartesian product between the elements of all the sets. Recall that the set S_1 can be fixed. An appropriate subproblem should be defined with an optimal partial solution for the MAP. Suppose we take one vector $x \in X$. The corresponding vertices $x_i \in x$ should be deleted from each set S_i, then an optimal partial solution is calculated for the rest of the vertices. The optimal solution consists in choosing the vector x whose sum with its corresponding optimal partial solution is minimum. Then the appropriate subproblem is established.

For a set of subsets of vertices $S_2 \subseteq \{1, \ldots, n\}, \ldots, S_s \subseteq \{1, \ldots, n\}$ with $|S_2| = \cdots = |S_s| = k$ which includes a vector $x^k \in X^k$ (here $X^k = \{k\} \times S_2 \times \cdots \times S_s$) where $x^k = (k, x_2^k, \ldots, x_s^k)$, let $C(S_2, \ldots, S_s)$ be the cost of the minimum assignment that considers $|S_2| = \cdots = |S_s| = k$ vectors with vertices in the corresponding sets $S_2, \ldots, S_s$.

When $k = 0$, it is established $C(S_2, \ldots, S_s) = 0$ since the assignment has no vectors.

In order to define $C(S_2, \ldots, S_s)$ in terms of smaller subproblems, one vector $x^k = (k, x_2^k, \ldots, x_s^k)$ should be selected such that $x_i^k \in S_i$ for all $2 \leq i \leq s$. All the vectors $x^k \in X^k$ should be evaluated in order to get the best x^k such that $C(S_2, \ldots, S_s) =$

Algorithm 6: The wise brute force algorithm for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n.
Result: A: The min cost assignment as a matrix of size $n \times s$.

```
1  Set A_{n×s} := 0;
2  Set A^1 := {1, ..., n};
3  Set bestCost := ∞;
4  Set bestAssignment := A;
5  Call WiseBruteForce(s, A);
6  WiseBruteForce ← function(currentDimension, A){
7  if currentDimension = 2 then
8      Set A^2 = {1, ..., n};
9      Fix the columns 1, 3, ..., s against the elements from the column 2;
10     Generate the matrix M_{n×n} of the corresponding 2AP and solve it;
11     Let A := 2AP(M_{n×n}) be the solution of the corresponding 2AP;
12     if bestCost > calculateCost(A) then
13         bestCost := calculateCost(A);
14         bestAssignment := A;
15     return;
16 Set x_{currentDimension} := {1, ..., n};
17 Set origin := x;
18 repeat
19     Set A^{currentDimension} = x_{currentDimension};
       // A^i refers to the column i of the matrix A
20     Call WiseBruteForce(currentDimension - 1, A);
21     Set y := next_premutation(x);
22 until x = origin;
23 };
24 return bestAssignment;
```

$\min_{x^k \in X^k} C(S_2 - x_2^k, \ldots, S_s - x_s^k) + w(x^k)$

A recursive function could be stated in the following way:

$$C(S_2, \ldots, S_s) = \begin{cases} 0 & \text{if } |S_2| = \cdots = |S_s| = 0 \\ \min_{x^k \in X^k} \left(C(S_2 - x_2^k, \ldots, S_s - x_s^k) + w(x^k)\right) & \text{if } |S_2| = \cdots = |S_s| \geq 1 \end{cases} \tag{3.9}$$

As we can see in Equation 3.9, at each recursive call one vertex is subtracted from each set so that the size of all the sets always remains equilibrated. The process of selecting the vector with the minimum weight among all the available vectors from X^k takes $O(k^{s-1})$ time. The possible state of a vertex $x_i \in S_i$ at the step

k is to be present or not, therefore the total number of possible states is $2^{(s-1)\cdot n}$. By applying a memoization technique we can avoid repeated recursive calls. The total time complexity for this algorithm is $O(2^{(s-1)\cdot n}n^{s-1})$ which is better than the previously described algorithms but, still exponential. A disadvantage of this solution is the requirement of exponential space, which is of $O(2^{(s-1)\cdot n})$, since it needs to memorize the answers for the derived substates.

Algorithm 7 shows the general structure of the procedure for the dynamic programming reduction. The interesting part of this algorithm lies in the AvailableTuples function because it consist in obtaining the possible tuples x based on the turned on bits from the bit set bitSetS. This function should carefully select the x vectors such that prevent repeated vectors for the next iterations. The algorithm returns the best cost of a partial assignment and, by the end of the algorithm, returns the optimal assignment.

This algorithm is relevant because it provides a better upper bound on the complexity to optimally solve a particular instance of a sAP of size n.

Algorithm 7: The dynamic programming reduction for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n.
Result: A: The min cost assignment as a matrix of size $n \times s$.

```
1  Let bitSetS be a bitset of size n × s;
2  Set bitSetS := (1 << (n × s)) − 1;
   // This turned on all the bits of bitSetS
3  Set result := DynamicProgrammingReduction(bitSetS);
4  DynamicProgrammingReduction ← function(bitSetS){
5  if bitSetS = 0 then
6      return {0, ∅};
7  bestCost := ∞;
8  bestPartialA := ∅;
   // The function AvailableTuples extract all the available tuples
      considering the turned on bits from the given bitset
9  for x ∈ AvailableTuples(bitSetS) do
10     Turn off the corresponding bits of bitSetS according to the elemnents in x;
11     Set result := DynamicProgrammingReduction(bitSetS);
12     if result.cost + cost(x) < bestCost then
13         bestCost := result.cost;
14         bestPartialA := result.assignment ∪ x;
15 return {bestCost, bestPartialA};
16 };
17 return result.assignment;
```

3.3.4 MAP-Gurobi

The 0-1 integer linear programming formulation, provided in Equation 4.1, was implemented by using the libraries from Gurobi Optimizer 6.0[1]. This implementation was called MAP-Gurobi. Gurobi Optimizer is designed from the ground up to exploit modern architectures and multi-core processors, using the most advanced implementations of the latest algorithms. Even when Gurobi Optimizer is a commercial software, it is easy to get a free academic version for purposes of research [Gurobi Optimization, 2016].

This implementation provides us with a very strong machinery to solve small instances of sAP and here we used it as part of the proposed heuristics. In order to provide an approach on the time and space complexity for this algorithm, two different sets of small and medium size families of instances were solved. We also tried another family of larger instances. However, our implementation only had success with the smaller instances of that set, the rest of the instances were not solved due to the limitation of computer power. When this solver has to tackle problems with a very large search space, the program crashes due to the large amount of memory required to solve this type of instances.

The results of this experiments are shown in the next subsection.

3.3.5 Results

The three first algorithms were implemented specifically for the 3AP, however the brute force algorithm becomes intractable for $n \geq 8$, the wise brute force algorithm for $n \geq 10$ and the dynamic programming reduction for $n \geq 13$. Even when the three algorithms are exponential, there is a considerable difference between the complexity of each algorithm.

Figures 3.2, 3.3, and 3.4 show the difference between the growth of each curve as soon as the number of vertices n is increased. The scale of the complexity function is logarithmic in order to show the growth of all curves simultaneously. The number of dimensions have a significant impact on the curves growth. In particular, for 5AP it can be observed that the behavior of the brute force algorithm and the wise brute force algorithm tends to be similar, whereas the dynamic programming reduction grows slowly. Do not forget that all three curves have at least exponential growth, however, the first two are factorials which is worse than exponential.

In order to evaluate the performance of the MAP-Gurobi implementation several the families of instances were considered.

MAP-Gurobi was implemented in C++ and its performance was evaluated on a platform with an Intel Core i5-3210M 2.5 GHz processor with 4 GB of RAM under Windows 8.

The smallest problem size family of instances is $s$bs$n$ ([Balas and Saltzman, 1991]).

[1]The Gurobi Optimizer is the state-of-the-art solver for mathematical programming.

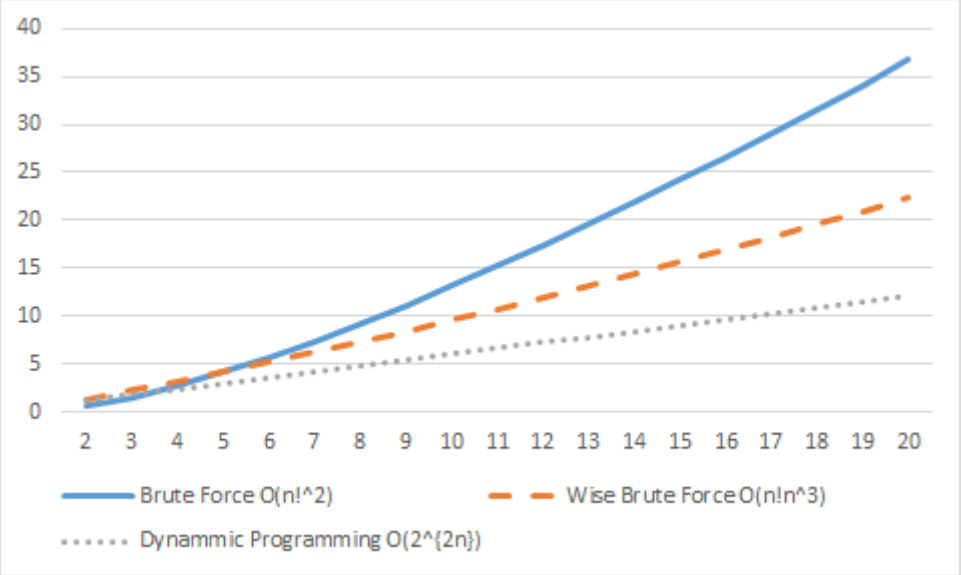

Figure 3.2: Complexity curves (in logarithmic scale) of exact algorithms for 3AP.

Table 3.1 shows the running times for the instances *sbsn* solved by MAP-Gurobi. Since Gurobi is a state of the art optimization solver for integer programming, it can be observed how this family of instances was solved by MAP-Gurobi without any problem in approximately one second. The disadvantage is that this set of instances do not allow us to have any guess about the complexity of MAP-Gurobi to solve 3AP.

Table 3.1: Computational results for the family of instances *sbsn* (MAP-Gurobi)

Instance	Optimum	Variables	Seconds
3_bs_4	42.2	64	0.1
3_bs_6	40.2	216	0.1
3_bs_8	23.8	512	0.1
3_bs_10	19.0	1000	0.1
3_bs_12	15.6	1728	0.1
3_bs_14	10.0	2744	0.2
3_bs_16	10.0	4096	0.3
3_bs_18	6.4	5832	0.4
3_bs_20	4.8	8000	0.9
3_bs_22	4.0	10648	0.8
3_bs_24	1.8	13824	0.9
3_bs_26	1.0	17576	1.3

A bigger problem size family of instances is *saxialn* ([Magos and Mourtos, 2009]). Table 3.2 shows the corresponding running times for this family of instances. The results obtained by Magos' algorithm are shown just for an illustrative reference be-

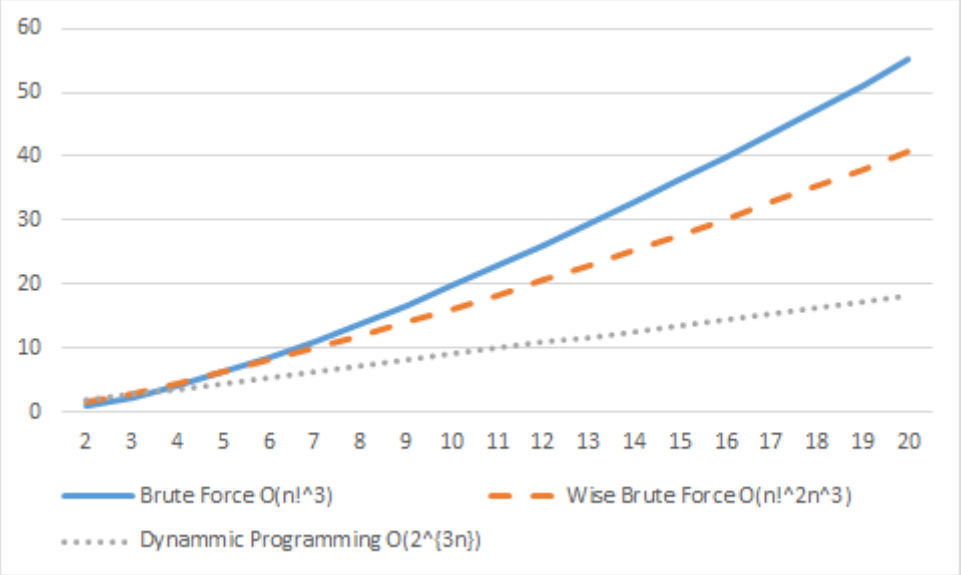

Figure 3.3: Complexity curves (in logarithmic scale) of exact algorithms for 4AP.

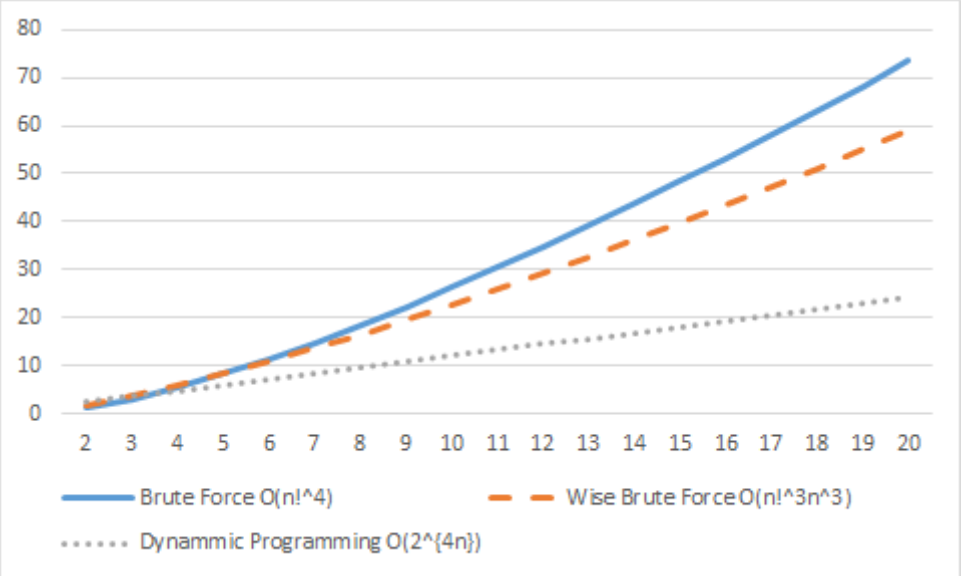

Figure 3.4: Complexity curves (in logarithmic scale) of exact algorithms for 5AP.

cause we did not implement their algorithm. The time complexity for this algorithm was also not provided. In order to provide a guess on the exponential function that is related to the complexity of the MAP-Gurobi algorithm, we considered a factor called relative increment. This factor allows us to observe the proportional increment between solving two instances with the same number of dimensions but whose difference in the number of vertices is one. For the family of instances 4axial$n$ the average relative increment of the running times for the Magos' algorithm was about

1.8 as well as for MAP-Gurobi, hence the experimental evaluation shows that the complexity function is approximately $O(2^n)$. In contrast, for the family of instances 5axial$n$ the average relative increment was 3.7 for the Magos' algorithm and 2.7 for MAP-Gurobi, hence the experimental evaluation shows that the complexity functions are approximately $O(4^n)$ and $O(3^n)$, respectively. Since there is only one instance for the family of instances 6axial$n$ it is not possible to provide any guess. By considering these results, a good guess for the complexity of sAP by using MAP-Gurobi could be $O((s-2)^n)$ for $s \in \{4, 5\}$.

Table 3.2: Computational results for the family of instances $s$axial$n$ (Magos and Mourtos, MAP-Gurobi)

			Magos' algorithm		MAP Gurobi	
Instance	Optimum	Variables	Seconds	Relative Inc.	Seconds	Relative Inc.
4_axial_10	480.3	10000	28	-	1.3	-
4_axial_11	603.0	14641	45	1.6	2.8	2.1
4_axial_12	711.3	20736	80	1.7	3.2	1.1
4_axial_13	839.7	28561	144	1.8	7.5	2.3
4_axial_14	831.3	38416	232	1.6	10.6	1.4
4_axial_15	822.1	50625	384	1.6	24.9	2.3
4_axial_16	736.9	65536	686	1.7	45.0	1.8
4_axial_17	643.5	83521	794	1.1	91.7	2.0
4_axial_18	608.7	104976	2031	2.5	132.7	1.4
4_axial_19	533.5	130321	5142	2.5	147.4	1.1
4_axial_20	503.8	160000	8714	1.6	336.3	2.2
5_axial_7	407.8	16807	8	-	1.4	-
5_axial_8	587.3	32768	47	5.8	4.8	3.4
5_axial_9	471.9	59049	188	4.0	14.0	2.9
5_axial_10	341.3	100000	882	4.6	30.4	2.1
5_axial_11	274.4	161051	4811	5.4	84.0	2.7
5_axial_12	219.7	248832	10328	2.1	300.4	3.5
5_axial_13	177.5	371293	19104	1.8	704.3	2.3
5_axial_14	141.1	537824	42141	2.2	1311.8	1.8
6_axial_8	156.9	262144	19812	-	107.9	-

The largest problem size instances were provided by [Karapetyan and Gutin, 2011b]. Tables 3.3, 3.4, and 3.5 show the corresponding running times for three families of instances. It can be observed that all the instances with 3 dimensions were solved to optimality but there is a big difference in terms of the solving time between each family of instances. Some of the instances in 4, 5, and 6 dimensions could not be solved. It is important to highlight that MAP-Gurobi found optimal values for some instances for which state-of-the-art heuristics could not find the optimal value. This is highlighted in bold in the Optimum columns found on each table. In this case,

it was not possible to provide a guess on the time complexity function because the corresponding relative increment factor can not be calculated. Further experiments could be performed by considering instances of consecutive problem sizes for each family, however the estimation provided thanks to the family of instances $s$axial$n$ was good enough to obtain the required time complexity approach.

In conclusion, considering the running times for the family of instances $s$axial$n$ a fair guess to the time complexity function for MAP-Gurobi is $O((s-1)^n)$, however this seems to be a big upper bound since in the experimental evaluation MAP-Gurobi performed better than this. On the other side, according to the running times for the families of instances provided by [Karapetyan and Gutin, 2011b] was determined that, in addition to the size of the input, weight distribution has an impact on the running time, in particular, the experimental evaluation shows that MAP-Gurobi performs better when the weights of the vectors of an instance are distributed uniformly at random within a range among all the vectors of X.

Finally, it can be observed that more than the number of variables of an instance, what matters is the size of the search of an instance, e. g. whereas instances with $s = 5$ dimensions and $n = 18$ vertices were solved, instances with $s = 4$ dimensions and $n = 30$ vertices were not solved to optimality due to the fact that $18!^4 < 30!^3$.

Table 3.3: Computational results for the family of instances random (MAP-Gurobi)

s	n	Variables	Previous best known	Optimum found	Seconds
3	40	64000	40.0	40.0	4.1
3	70	343000	70.0	70.0	11.9
3	100	1000000	100.0	100.0	50.9
4	20	160000	20.0	20.0	10.3
4	30	810000	30.0	30.0	56.5
4	40	2560000	40.0	40.0	173.0
5	15	759375	15.0	15.0	68.0
5	18	1889568	18.0	18.0	186.5
5	25	9765625	25.0	-	-
6	12	2985984	12.0	12.0	493.1
6	15	11390625	15.0	-	-
6	18	34012224	18.0	-	-

3.4 Local search heuristics

Local search is a heuristic method for solving optimization problems. This heuristic can be used on problems that can be formulated through minimization or maximization functions that allow to evaluate the optimality of several candidate solutions. Local search heuristics move from one solution to another by applying some local

Table 3.4: Computational results for the family of instances clique (MAP-Gurobi)

s	n	Variables	Previous best known	Optimum found	Second
3	40	64000	939.9	939.9	3.3
3	70	343000	1158.4	**1157.1**	50.7
3	100	1000000	1368.1	**1345.9**	771.7
4	20	160000	1901.8	1901.8	8.4
4	30	810000	2281.9	-	-
4	40	2560000	2606.3	-	-
5	15	759375	3110.7	3110.7	43.0
5	18	1889568	3458.6	3458.6	196.5
5	25	9765625	4192.7	-	-
6	12	2985984	4505.6	4505.6	689.1
6	15	11390625	5133.4	-	-
6	18	34012224	5765.5	-	-

Table 3.5: Computational results for the family of instances square root (MAP-Gurobi)

s	n	Variables	Previous best known	Optimum found	Seconds
3	40	64000	610.6	**606.9**	2.6
3	70	343000	737.1	**733.6**	62.0
3	100	1000000	866.3	**838.1**	1062.1
4	20	160000	929.3	929.3	9.4
4	30	810000	535.1	-	-
4	40	2560000	1271.4	-	-
5	15	759375	1203.9	1203.9	79.4
5	18	1889568	-	**1343.8**	1029.0
5	25	9765625	1627.5	-	-
6	12	2985984	-	**1436.8**	427.3
6	15	11390625	1654.6	-	-
6	18	34012224	1856.3	-	-

changes over a candidate solution until a better solution is found or a time bound is elapsed.

These techniques have been proven to be some of the most successful to deal with MAP and this is why we study them extensively in this work.

Each of the next heuristics requires an initial feasible solution $A = (x^1, x^2, \ldots, x^n)$ somehow generated. A feasible solution can easily be generated just by choosing a random permutation for each set of vertices from each dimension except the first (recall that the first dimension can be fixed), and by creating the corresponding vectors x^i from combining the i-th elements from each dimension as a valid vector

$x^i \in A$.

Lets consider a toy instance with $s = 4$ dimensions and $n = 5$ vertices. Let x_i^j be the i-th vertex of dimension j, weights are set up as the product of the corresponding indices of the related vertex from each dimension, so that the weights of all the vectors are $w(x_1^1, x_1^2, x_1^3, x_1^4) = 1$, $w(x_1^1, x_1^2, x_1^3, x_2^4) = 2$, $\ldots$, $w(x_5^1, x_5^2, x_5^3, x_5^4) = 625$. For simplicity the vectors $(x_a^1, x_b^2, x_c^3, x_d^4)$ will be written as (a, b, c, d) since the dimension to which the vertex belongs can be deducted from the position of the index in the vector.

A possible feasible solution for an instance is:

$$A = \begin{cases} x^1 & : & (1,2,3,4) \\ x^2 & : & (2,3,1,3) \\ x^3 & : & (3,5,4,2) \\ x^4 & : & (4,1,2,1) \\ x^5 & : & (5,4,5,5) \end{cases}$$

The cost of this solution is $w(A) = w(x^1) + w(x^2) + w(x^3) + w(x^4) + w(x^5) = 24 + 18 + 120 + 8 + 500 = 670$.

The structure of a feasible solution is illustrated because it helps to exemplify the description of the next heuristics. Note that this structure allows us to have a solution as the desired matrix with n rows and s columns where the i-th row is related to the vector $x^i \in A$ and the j-th column is related to the set of vertices X_j.

3.4.1 Basic local search heuristics

Our first approach was to develop some simple local search heuristics based on known techniques in order to have a reference about the effectiveness of simple methods against more robust and complex techniques.

3.4.1.1 Simple 2-opt

The simple 2-opt heuristic is a local search heuristic proposed by [Croes, 1958] for solving the traveling salesman problem. In the traveling salesman problem (TSP), a list of cities and the distances between each pair of cities is given. The goal is to find the shortest route that visits each city exactly once and returns to the original city. Given an initial route, the 2-opt heuristic consider two cities and swaps them and if the new route is shorter than the initial route, then the feasible solution is updated.

Given a feasible solution for a sAP, this heuristic consists on choosing a dimension d and swapping two of its vertices indexed by $i, j \in \{1, \ldots, n\}$ with $i < j$. If the new solution is better than the previous one then the feasible solution is updated with the new vectors.

Algorithm 8 shows the pseudo-code of the simple 2-opt heuristic MAP. A complete

iteration of this heuristic considers all the dimensions and all the possible combination of pairs from each dimension. There are $\binom{n}{2} = O(n^2)$ possible combinations of vertices by each of the s dimensions, then one iteration takes $O(sn^2)$ time. The variable *iter* helps to stop the heuristic after some maximum number of desired iterations. In our experimental evaluation no more than 10 iterations were performed.

Algorithm 8: The simple 2−opt heuristic for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n.
iter. Maximum number of iterations of the heuristic.
Result: A: The min cost assignment as a matrix of size $n \times s$.

```
1  Set A := FeasibleRandomGeneratedSolutionForMAP(G);
2  Set improved := true;
3  Set iterations := 1;
4  while improved = true and iterations ≤ iter do
5      Set improved := false;
6      Set iterations := iterations +1;
7      foreach d in {1, ..., s} do
8          foreach i = 1 to n do
9              foreach j = i + 1 to n do
10                 A' := A;
11                 swap(x_i^d, x_j^d) from A';
12                 if assignmentCost(A') < assignmentCost(A) then
13                     Set improved := true;
14                     Set A := A';
15 return {A};
```

For example, consider the feasible solution A for our toy instance and choose the dimension $d = 2$ and the vertices x_1^2 and x_3^2. The swapping is performed as follows:

$$\begin{bmatrix} (1,2,3,4) \to (1,\mathbf{5},3,4) \\ (3,5,4,2) \to (3,\mathbf{2},4,2) \end{bmatrix}.$$

The corresponding weights are:

$$\begin{bmatrix} 24 \to 60 \\ 120 \to 48 \end{bmatrix}.$$

The new two vectors have a lower cost ($144 = 24 + 120 > 60 + 48 = 108$) so the feasible solution A is updated.

$$A = \begin{cases} x^1 & : & (\mathbf{1}, \mathbf{5}, \mathbf{3}, \mathbf{4}) \\ x^2 & : & (2, 3, 1, 3) \\ x^3 & : & (\mathbf{3}, \mathbf{2}, \mathbf{4}, \mathbf{2}) \\ x^4 & : & (4, 1, 2, 1) \\ x^5 & : & (5, 4, 5, 5) \end{cases}$$

The advantage of this heuristic is that it is very easy to implement. The main disadvantage is that the search space is of size $O(sn^2)$, which is too small compared to the whole search space.

3.4.1.2 Inversion

The inversion heuristic is a generalization of the simple 2-opt, which was originally proposed for the TSP. Given an initial route, two cities are selected and its corresponding tour is reversed, if the new route is shorter than initial route, then the feasible solution is updated.

Given a feasible solution for a sAP, this heuristic consists on choosing a dimension d and reverse the sub-array with two end point vertices indexed by $i, j \in \{1, \ldots, n\}$ with $i < j$. The sub-array of vertices $x_i^d, x_{i+1}^d, \ldots, x_{j-1}^d, x_j^d$ is reversed as $x_j^d, x_{j-1}^d, \ldots, x_{i+1}^d, x_i^d$ from the feasible solution. If the new solution is better than the previous one then the feasible solution is updated with the new vectors.

Algorithm 9 shows the pseudo-code of the inversion heuristic for MAP. A complete iteration of this heuristic considers all the dimensions and all the possible combinations of pairs from each dimension. An inversion takes $O(j - i) = O(n)$ time. There are $\binom{n}{2} = O(n^2)$ possible combinations of vertices by each of the s dimensions, then one iteration takes $O(sn^3)$ time. The variable *iter* has the same objective as in the 2-opt heuristic.

For example, consider the feasible solution A for our toy instance and choose the dimension $d = 2$ and the vertices x_1^2 and x_4^2. The swapping is performed as follows:

$$\begin{bmatrix} (1, 2, 3, 4) \to (1, \mathbf{1}, 3, 4) \\ (2, 3, 1, 3) \to (2, \mathbf{5}, 1, 3) \\ (3, 5, 4, 2) \to (3, \mathbf{3}, 4, 2) \\ (4, 1, 2, 1) \to (4, \mathbf{2}, 2, 1) \end{bmatrix}.$$

The corresponding weights are:

$$\begin{bmatrix} 24 \to 12 \\ 18 \to 30 \\ 120 \to 72 \\ 8 \to 16 \end{bmatrix}.$$

The new two vectors have a lower cost ($170 = 24+18+120+8 > 12+30+72+16 = 130$) so the feasible solution A is updated.

Algorithm 9: The inversion heuristic for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n.
iter. Maximum number of iterations of the heuristic.
Result: A: The min cost assignment as a matrix of size $n \times s$.

```
1  Set A := FeasibleRandomGeneratedSolutionForMAP(G);
2  Set improved := true;
3  Set iterations := 1;
4  while improved = true and iterations ≤ iter do
5      Set improved := false;
6      Set iterations := iterations +1;
7      foreach d in {1, ..., s} do
8          foreach i = 1 to n do
9              foreach j = i + 1 to n do
10                 A' := A;
11                 reversingVertices(x_i^d, x_j^d) from A';
12                 if assignmentCost(A') < assignmentCost(A) then
13                     Set improved := true;
14                     Set A := A';
15 return {A};
```

$$A = \begin{cases} x^1 : (\mathbf{1}, \mathbf{1}, \mathbf{3}, \mathbf{4}) \\ x^2 : (\mathbf{2}, \mathbf{5}, \mathbf{1}, \mathbf{3}) \\ x^3 : (\mathbf{3}, \mathbf{3}, \mathbf{4}, \mathbf{2}) \\ x^4 : (\mathbf{4}, \mathbf{2}, \mathbf{2}, \mathbf{1}) \\ x^5 : (5, 4, 5, 5) \end{cases}$$

The advantage of this heuristic is that its search space, which is of size $O(sn^3)$, is larger than that of the 2-opt, however it is still smaller than the whole search space of the instance.

3.4.1.3 Circular rotation

We proposed the circular rotation heuristic as an option that considers a bigger search space than the inversion heuristic. Furthermore the search spaces of the circular rotation and the inversion heuristics are different. The ideas for this heuristic come from its applying to the TSP. Given an initial route, two cities are selected and its corresponding tour is circularly rotated one by one either left to right or right to left, but in only one direction. Each time that a new route is shorter than the initial route, the feasible solution is updated.

Given a feasible solution for a sAP, this heuristic consists on choosing a dimension

d and selecting two end point vertices indexed by $i, j \in \{1, \ldots, n\}$ with $i < j$ and rotate the sub-array of vertices $x_i^d, x_{i+1}^d, \ldots, x_{j-1}^d, x_j^d$, either left to right or right to left, from the feasible solution. For example, in one single circular left rotation the vertices are rotated as $x_{i+1}^d, x_{i+2}^d, \ldots, x_j^d, x_i^d$. Each time that the new solution is better than the previous one, the feasible solution is updated with the new vectors.

Algorithm 10 shows the pseudo-code of the circular rotation heuristic for MAP. A complete iteration of this heuristic considers all the dimensions and all the possible combinations of pairs from each dimension. A single circular rotation process takes $O(|j - i|) = O(n)$ time and the total number of rotations is $O(|j - i|) = O(n)$, so the total complexity time by a circular rotation indexed by i and j is $O(n^2)$. There are $\binom{n}{2} = O(n^2)$ possible combinations of vertices by each of the s dimensions, then one iteration takes $O(sn^4)$ time. The variable *iter* has the same objective as for the previous heuristics.

Algorithm 10: The circular rotation heuristic for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n.
iter. Maximum number of iterations of the heuristic.
Result: A: The min cost assignment as a matrix of size $n \times s$.

```
1  Set A := FeasibleRandomGeneratedSolutionForMAP(G);
2  Set improved := true;
3  Set iterations := 1;
4  while improved = true and iterations ≤ iter do
5      Set improved := false;
6      Set iterations := iterations +1;
7      foreach d in {1, ..., s} do
8          foreach i = 1 to n do
9              foreach j = i + 1 to n do
10                 A' := A;
11                 x_orig := x_j^d;
12                 repeat
13                     rotateVertices(x_i^d, x_j^d) from A';
                       // either left rotation or right rotation
14                     if assignmentCost(A') < assignmentCost(A) then
15                         Set improved := true;
16                         Set A := A';
17                 until x_orig ≠ x_i^d;
18 return {A};
```

For example, consider the feasible solution A for our toy instance and choose the dimension $d = 2$ and the vertices x_1^2 and x_4^2. A single circular left rotation is as follows:

$$\begin{bmatrix} (1,2,3,4) \rightarrow (1,\mathbf{3},3,4) \\ (2,3,1,3) \rightarrow (2,\mathbf{5},1,3) \\ (3,5,4,2) \rightarrow (3,\mathbf{1},4,2) \\ (4,1,2,1) \rightarrow (4,\mathbf{2},2,1) \end{bmatrix}.$$

The corresponding weights are:

$$\begin{bmatrix} 24 \rightarrow 36 \\ 18 \rightarrow 30 \\ 120 \rightarrow 24 \\ 80 \rightarrow 16 \end{bmatrix}.$$

The new two vectors have a lower cost ($242 = 24+18+120+80 > 36+30+24+16 = 106$) so the feasible solution A is updated.

$$A = \begin{cases} x^1 & : & (\mathbf{1},\mathbf{3},\mathbf{3},\mathbf{4}) \\ x^2 & : & (\mathbf{2},\mathbf{5},\mathbf{1},\mathbf{3}) \\ x^3 & : & (\mathbf{3},\mathbf{1},\mathbf{4},\mathbf{2}) \\ x^4 & : & (\mathbf{4},\mathbf{2},\mathbf{2},\mathbf{1}) \\ x^5 & : & (5,4,5,5) \end{cases}$$

It is easy to verify that the sense of the rotations does not matter since always the best option is chosen among all the rotations. In a circular left to right rotation the first single rotation corresponds to the same configuration as in the last rotation of a circular right to left rotation. At the end of a circular rotation, without matter the direction, what survives is the best configurations among all the rotations.

3.4.1.4 $k-$vertex permutation

We proposed the $k-$vertex permutation heuristic as a more exhaustive option that considers even a larger search space than the previous heuristics. The ideas for this heuristic come similarly from its applying to the TSP. Given an initial route, two cities are selected and its corresponding tour starts to be exhaustively permuted such that the best permutation is selected. This heuristic is a generalization of the three previous heuristics. However, it is not a good idea to apply it over an arbitrary length path.

Given a feasible solution for a sAP, this heuristic consists on choosing a dimension d and a positive integer $0 < k < n$. Perform all the $k!$ permutations of the vertices indexed by $i, i+k-1 \in 1,\ldots,n$, that is $x_i^d, x_{i+1}^d, \ldots, x_{i+k-1}^d, x_{i+k}^d$, from the feasible solution. At the end of a step the new solution is at least as good as the initial permutation because the optimal permutation over such search space is found exhaustively.

Algorithm 11 shows the pseudo-code of the $k-$vertex permutation heuristic for MAP. A complete iteration of this heuristic considers all the dimensions and all the

possible valid indexes for i and $i+k$. A single permutation process over k vertices takes $O(k!)$ time and the total number of contiguous subsets of vertices is $O(s(n-k+1))$, then one iteration takes $O(s(n-k+1)k!)$ time. In this case, larger values of k will increase considerably the complexity of the heuristic. In fact, selecting $k > 10$ will definitively be intractable in a conventional computer. Again the variable *iter* should be given.

Algorithm 11: The contiguous $k-$vertex permutation heuristic for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n.
iter. Maximum number of iterations of the heuristic.
Result: A: The min cost assignment as a matrix of size $n \times s$.

```
1  Set A := FeasibleRandomGeneratedSolutionForMAP(G);
2  Set improved := true;
3  Set iterations := 1;
4  while improved = true and iterations ≤ iter do
5      Set improved := false;
6      Set iterations := iterations +1;
7      foreach d in {1, ..., s} do
8          foreach i = 1 to n do
9              foreach j = i + 1 to n do
10                 A' := A;
11                 x_orig := x_j^d;
12                 foreach Permutation P of the vertices x_i^d, ..., x_j^d do
13                     Replace the vertices x_i^d, ..., x_j^d from A' with the
                       corresponding permutation P;
14                     if assignmentCost(A') < assignmentCost(A) then
15                         Set improved := true;
16                         Set A := A';
17 return {A};
```

For example, consider the feasible solution A for our toy instance and choose the dimension $d = 2$, $k = 3$ and the index $i = 1$. A 3-vertex permutation will have to select between the next options:

$$\begin{bmatrix} (1,2,3,4) \to (1,\mathbf{2},3,4), (1,\mathbf{2},3,4), (1,\mathbf{3},3,4), (1,\mathbf{3},3,4), (1,\mathbf{5},3,4), (1,\mathbf{5},3,4) \\ (2,3,1,3) \to (2,\mathbf{3},1,3), (2,\mathbf{5},1,3), (2,\mathbf{2},1,3), (2,\mathbf{5},1,3), (2,\mathbf{2},1,3), (2,\mathbf{3},1,3) \\ (3,5,4,2) \to (3,\mathbf{5},4,2), (3,\mathbf{3},4,2), (3,\mathbf{5},4,2), (3,\mathbf{2},4,2), (3,\mathbf{3},4,2), (3,\mathbf{2},4,2) \end{bmatrix}.$$

The corresponding weights are:

$$\begin{bmatrix} 24 \to 24, & 24, & 36, & \mathbf{36}, & 60, & 60 \\ 18 \to 18, & 30, & 12, & \mathbf{30}, & 12, & 18 \\ 120 \to 120, & 72, & 120, & \mathbf{48}, & 72, & 48 \end{bmatrix}.$$

The new two vectors have a lower cost ($242 = 24+18+120 > 36+30+48 = 114$) so the feasible solution A is updated.

$$A = \begin{cases} x^1 & : & (\mathbf{1}, \mathbf{3}, \mathbf{3}, \mathbf{4}) \\ x^2 & : & (\mathbf{2}, \mathbf{5}, \mathbf{1}, \mathbf{3}) \\ x^3 & : & (\mathbf{3}, \mathbf{2}, \mathbf{4}, \mathbf{2}) \\ x^4 & : & (4, 1, 2, 1) \\ x^5 & : & (5, 4, 5, 5) \end{cases}$$

It can be observed that the optimal permutation can be found by solving a 2AP between the permuted k vertices and the vertices in the same vectors from the other dimensions. In this way, the time complexity of this heuristic can be reduced from $O(s(n-k+1)k!)$ to $O(s(n-k+1)^4)$. This idea can be considered as the predecessor of the dimensionwise variation heuristics.

3.4.2 Dimensionwise variation heuristics

The dimensionwise variation heuristics (DVH) are a set of heuristics that perform local improvements over a feasible solution by applying an exact technique. Originally, [Huang and Lim, 2006] applied this technique for 3AP and [Karapetyan and Gutin, 2011a] extended it for more than 3 dimensions, however this technique can be more general.

In general, a DVH works as follows: at one step of the heuristic all the dimensions but a proper subset $F \subset \{1, \ldots, s\}$ are fixed and a matrix M of size $n \times n$ with entries $M_{i,j} = w(v_{i,j})$ is generated. Let $v_{i,j}^d$ denote the d-th element of the vector $v_{i,j}$, all the vectors are built according to the next function:

$$v_{i,j}^d = \begin{cases} x_i^d \text{ if } d \in F \\ x_j^d \text{ if } d \notin F \end{cases} \quad \text{for } 1 \leq d \leq s \ . \tag{3.10}$$

The corresponding 2AP can be solved in $O(n^3)$ time and gives a local optimum for the original instance.

The simplest version of DVH allows to have only one dimension in F. However, allowing to have bigger subsets can be explored more neighborhoods.

Algorithm 12 shows the pseudo-code of a dimensionwise variation heuristic. A complete iteration of this heuristic consists in trying every possible non empty subset $F \in \wp(\{1, \ldots, s\})$. There are $O(2^s)$ possible combinations, then one iteration takes $O(2^s n^3)$ time. This heuristic performs iterations until no improvement is obtained or the required number of iterations *iter* are reached. Even when the theoretical time complexity could be high, [Karapetyan and Gutin, 2011a] reported that at most ten

iterations are performed before they converge to a local optimum. Such observation was also verified.

Algorithm 12: The dimensionwise variation heuristic for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n. iter. Maximum number of iterations of the heuristic.
Result: A: The min cost assignment as a matrix of size $n \times s$.

```
Set A := FeasibleRandomGeneratedSolutionForMAP(G);
Set improved := true;
Set bestCost := assignmentCost(A);
Set iterations := 1;
while improved = true and iterations ≤ iter do
    Set improved := false;
    Set iterations := iterations +1;
    foreach F ∈ {℘({1,...,s})\∅} do
        By considering the Equation 3.10, obtain the matrix M_{n×n} for the
        corresponding 2AP;
        Set A' := HungarianMethod(M_{n×n}) ;
        if assignmentCost(A') < bestCost then
            Set improved := true;
            Set bestCost := assignmentCost(A');
            Map the solution A' to the corresponding feasible solution onto A;
return {A};
```

In order to exemplify the reduction of a feasible solution of a sAP to a 2AP consider the previous randomly generated feasible solution A (for the toy instance) and pick the set $F = \{2, 4\}$, then M is obtained as:

$$\begin{bmatrix} (1,2,3,4) & (2,2,1,4) & (\mathbf{3},\mathbf{2},\mathbf{4},\mathbf{4}) & (4,2,2,4) & (5,2,5,4) \\ (1,3,3,3) & (2,3,1,3) & (3,3,4,3) & (\mathbf{4},\mathbf{3},\mathbf{2},\mathbf{3}) & (5,3,5,3) \\ (\mathbf{1},\mathbf{5},\mathbf{3},\mathbf{2}) & (2,5,1,2) & (3,5,4,2) & (4,5,2,2) & (5,5,5,2) \\ (1,1,3,1) & (2,1,1,1) & (3,1,4,1) & (4,1,2,1) & (\mathbf{5},\mathbf{1},\mathbf{5},\mathbf{1}) \\ (1,4,3,5) & (\mathbf{2},\mathbf{4},\mathbf{1},\mathbf{5}) & (3,4,4,5) & (4,4,2,5) & (5,4,5,5) \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 24 & 16 & \mathbf{96} & 64 & 200 \\ 27 & 18 & 108 & \mathbf{72} & 225 \\ \mathbf{30} & 20 & 120 & 80 & 250 \\ 3 & 2 & 12 & 8 & \mathbf{25} \\ 60 & \mathbf{40} & 240 & 160 & 500 \end{bmatrix}$$

The values marked in bold denote the new vectors that will be considered as the new feasible solution after solving the corresponding 2AP. Then, the new feasible solution is:

$$A' = \begin{cases} x^{1'} & : (1,5,3,2) \\ x^{2'} & : (2,4,1,5) \\ x^{3'} & : (3,2,4,4) \\ x^{4'} & : (4,3,2,3) \\ x^{5'} & : (5,1,5,1) \end{cases}$$

The cost of the new solution A' is $w(A') = w(x^{1'}) + w(x^{2'}) + w(x^{3'}) + w(x^{4'}) + w(x^{5'}) = 30 + 40 + 96 + 72 + 25 = 263$ which is equal to the optimal minimum cost of the solved 2AP.

This heuristic considers a search space of size $O(n!)$ at each step, which corresponds with the search space of a 2AP. It is easy to see that at the end of one step of this heuristic, the current feasible solution cannot be worse, due to the fact that we are optimizing the derived search space by solving the corresponding 2AP. Another way to verify this property is by observing that the new vectors selected, among the set of derived vectors, always consider the original vectors plus other combinations such that, if some vectors should be changed to get a better solution, then the 2AP solution obtains the best ones.

3.4.3 The generalized dimensionwise variation heuristics

The simplification of a sAP to a 2AP can be applied for any sAP with $s \geq 3$. Here we propose a generalization of this heuristic which consists in reducing a sAP to a tAP with $2 \leq t < s$.

The generalized dimensionwise variation heuristic (GDVH) works as follows: let t be an integer value such that $2 \leq t \leq s-1$ and, based on t, suppose to have $F_1, \ldots, F_{t-1}$ non empty proper subsets of $\{1, \ldots, s\}$ such that $F_1 \cap \cdots \cap F_{t-1} = \emptyset$ and $F_1 \cup \cdots \cup F_{t-1} \subset \{1, \ldots, s\}$. At one step of the heuristic all the dimensions but $F_1 \cup \cdots \cup F_{t-1}$ are fixed and a t-dimensional matrix M^t of size $n^1 \times \cdots \times n^t$ (recall the simplification $n^i = \text{n}$ for $1 \leq i \leq t$) with entries $M_{i^1,\ldots,i^t} = w(v_{i^1,\ldots,i^t})$ is generated. Let $v^d_{i^1,\ldots,i^t}$ denote the d-th element of the vector $v_{i^1,\ldots,i^t}$, all the vectors are built according to the next function:

$$v^d_{i^1,i^2,\ldots,i^{t-1},i^t} = \begin{cases} x^d_{i^1} \text{ if } d \in F_1 \\ x^d_{i^2} \text{ if } d \in F_2 \\ \quad \ldots \\ x^d_{i^{t-1}} \text{ if } d \in F_{t-1} \\ x^d_{i^t} \text{ otherwise} \end{cases} \quad \text{for } 1 \leq d \leq s \ . \tag{3.11}$$

The corresponding tAP instance can be solved by using some exact technique, in this case with the MAP-Gurobi. Algorithm 13 shows the pseudo-code of the generalized dimensionwise variation heuristic. A complete iteration tries every possible combination of t subsets of $\{1, \ldots, s\}$. There are about $O(t^s)$ combinations, therefore one iteration takes $O(t^s \cdot (t-1)^n)$ time where $O((t-1)^n)$ is the complexity of

solving an instance of size $O(n^t)$ with the MAP-Gurobi, except for the cases when $t = 2$ because those are the same as for the DVH and can be solved in $O(n^3)$. In the experimental evaluation we also found that, as for DVH, less than ten iterations are performed before they converge to a local optimum. In any case, the input parameter *iter* for the bound of the number of iterations is required.

Algorithm 13: The generalized dimensionwise variation heuristic for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n. t. The value of the desired problem size reduction such that $2 \leq t < s$. iter. Maximum number of iterations of the heuristic.

Result: A: The min cost assignment as a matrix of size $n \times s$.

```
1  Set A := FeasibleRandomGeneratedSolutionForMAP(G);
2  Set improved := true;
3  Set bestCost := assignmentCost(A);
4  Set iterations := 1;
5  while improved = true and iterations ≤ iter do
6      Set improved := false;
7      Set iterations := iterations +1;
8      foreach F_1, ..., F_{t-1} ∈ {CombinationSets({1, ..., s}) do
9          By considering the Equation 3.13, obtain the matrix M^t for the
           corresponding tAP;
10         Set A' := MAP-Gurobi(M^t) ;
11         if assignmentCost(A') < bestCost then
12             Set improved := true;
13             Set bestCost := assignmentCost(A');
14             Map the solution A' to the corresponding feasible solution onto A;
15 return {A};
```

This reduction is not so difficult to see. In order to exemplify a reduction of this type consider the same randomly generated feasible solution A for the toy instance and pick $t = 3$ and the sets $F_1 = \{2, 4\}, F_2 = \{3\}$ such that $F_1 \cup F_2 \subset \{1, \ldots, s\}$, then M^3 is obtained as:

$$M_1^3 \begin{bmatrix} (1,2,3,4) & (2,2,3,4) & (3,2,3,4) & (4,2,3,4) & (5,2,3,4) \\ (1,2,1,4) & (2,2,1,4) & (3,2,1,4) & (4,2,1,4) & (5,2,1,4) \\ (1,2,4,4) & (\mathbf{2},\mathbf{2},\mathbf{4},\mathbf{4}) & (3,2,4,4) & (4,2,4,4) & (5,2,4,4) \\ (1,2,2,4) & (2,2,2,4) & (3,2,2,4) & (4,2,2,4) & (5,2,2,4) \\ (1,2,5,4) & (2,2,5,4) & (3,2,5,4) & (4,2,5,4) & (5,2,5,4) \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 24 & 48 & 72 & 96 & 120 \\ 8 & 16 & 24 & 32 & 40 \\ 32 & \mathbf{64} & 96 & 128 & 160 \\ 16 & 32 & 48 & 64 & 80 \\ 40 & 80 & 120 & 160 & 200 \end{bmatrix}$$

$$M_2^3 \begin{bmatrix} (1,3,3,3) & (2,3,3,3) & (3,3,3,3) & (4,3,3,3) & (5,3,3,3) \\ (1,3,1,3) & (2,3,1,3) & (3,3,1,3) & (4,3,1,3) & (5,3,1,3) \\ (1,3,4,3) & (2,3,4,3) & (3,3,4,3) & (4,3,4,3) & (5,3,4,3) \\ (1,3,2,3) & (2,3,2,3) & (\mathbf{3},\mathbf{3},\mathbf{2},\mathbf{3}) & (4,3,2,3) & (5,3,2,3) \\ (1,3,5,3) & (2,3,5,3) & (3,3,5,3) & (4,3,5,3) & (5,3,5,3) \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 27 & 54 & 81 & 108 & 135 \\ 9 & 18 & 27 & 36 & 45 \\ 36 & 72 & 108 & 144 & 180 \\ 18 & 36 & \mathbf{54} & 72 & 90 \\ 45 & 90 & 135 & 180 & 225 \end{bmatrix}$$

$$M_3^3 \begin{bmatrix} (1,5,3,2) & (2,5,3,2) & (3,5,3,2) & (4,5,3,2) & (5,5,3,2) \\ (1,5,1,2) & (2,5,1,2) & (3,5,1,2) & (\mathbf{4},\mathbf{5},\mathbf{1},\mathbf{2}) & (5,5,1,2) \\ (1,5,4,2) & (2,5,4,2) & (3,5,4,2) & (4,5,4,2) & (5,5,4,2) \\ (1,5,2,2) & (2,5,2,2) & (3,5,2,2) & (4,5,2,2) & (5,5,2,2) \\ (1,5,5,2) & (2,5,5,2) & (3,5,5,2) & (4,5,5,2) & (5,5,5,2) \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 30 & 60 & 90 & 120 & 150 \\ 10 & 20 & 30 & \mathbf{40} & 50 \\ 40 & 80 & 120 & 160 & 200 \\ 20 & 40 & 60 & 80 & 100 \\ 50 & 100 & 150 & 200 & 250 \end{bmatrix}$$

$$M_4^3 \begin{bmatrix} (1,1,3,1) & (2,1,3,1) & (3,1,3,1) & (4,1,3,1) & (5,1,3,1) \\ (1,1,1,1) & (2,1,1,1) & (3,1,1,1) & (4,1,1,1) & (5,1,1,1) \\ (1,1,4,1) & (2,1,4,1) & (3,1,4,1) & (4,1,4,1) & (5,1,4,1) \\ (1,1,2,1) & (2,1,2,1) & (3,1,2,1) & (4,1,2,1) & (5,1,2,1) \\ (1,1,5,1) & (2,1,5,1) & (3,1,5,1) & (4,1,5,1) & (\mathbf{5},\mathbf{1},\mathbf{5},\mathbf{1}) \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 3 & 6 & 9 & 12 & 15 \\ 1 & 2 & 3 & 4 & 5 \\ 4 & 8 & 12 & 16 & 20 \\ 2 & 4 & 6 & 8 & 10 \\ 5 & 10 & 15 & 20 & \mathbf{25} \end{bmatrix}$$

$$M_5^3 \begin{bmatrix} (\mathbf{1},\mathbf{4},\mathbf{3},\mathbf{5}) & (2,4,3,5) & (3,4,3,5) & (4,4,3,5) & (5,4,3,5) \\ (1,4,1,5) & (2,4,1,5) & (3,4,1,5) & (4,4,1,5) & (5,4,1,5) \\ (1,4,4,5) & (2,4,4,5) & (3,4,4,5) & (4,4,4,5) & (5,4,4,5) \\ (1,4,2,5) & (2,4,2,5) & (3,4,2,5) & (4,4,2,5) & (5,4,2,5) \\ (1,4,5,5) & (2,4,5,5) & (3,4,5,5) & (4,4,5,5) & (5,4,5,5) \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} \mathbf{60} & 120 & 180 & 240 & 300 \\ 20 & 40 & 60 & 80 & 100 \\ 80 & 160 & 240 & 320 & 400 \\ 40 & 80 & 120 & 160 & 200 \\ 100 & 200 & 300 & 400 & 500 \end{bmatrix}$$

In this example, we can observe that the vertices of the dimensions in the set $\{1,\ldots,s\}\backslash\{F_1 \cup F_2\}$ are fixed in the corresponding vectors of M; the vertices of the dimensions in the set F_1 just vary between each matrix M_i and M_j for all $i \neq j$ and $1 \leq i,j \leq 5$; the vertices of the dimensions in the set F_2 vary at each matrix M_i for all $1 \leq i \leq 5$. The optimal solution of this 3AP provides us the vectors of the new feasible solution A' which are:

$$A' = \begin{cases} x^{1'} : (1,4,3,5) \\ x^{2'} : (2,2,4,4) \\ x^{3'} : (3,3,2,3) \\ x^{4'} : (4,5,1,2) \\ x^{5'} : (5,1,5,1) \end{cases}$$

The cost of the new solution A' is $w(A') = w(x^{1'}) + w(x^{2'}) + w(x^{3'}) + w(x^{4'}) + w(x^{5'}) = 60 + 64 + 54 + 40 + 25 = 243$ which is equal to the optimal minimum cost of the 3AP solved.

This heuristic considers a search space of size $O(n!^{t-1})$ at each step and, as for DVH, at the end of one step of this heuristic the current feasible solution cannot be worse. Since the search space is bigger than in the case of DVH this heuristics tend to provided better solutions at each simple reduction.

Keep in mind that if the reduction to some tAP with $3 \leq t \leq s-1$ still has a big search space then the resolution of the reduction could take a while or could not be solved due to the computer power. However, the same may occur in reductions to some 2AP with n equal to many thousands of vertices.

We called SDVt a dimensionwise variation heuristic that reduces a sAP to a tAP and considers $|F_i| = 1$ for all $1 \leq i < t$. We called DVt a dimensionwise variation heuristic that reduces a sAP to a tAP and considers $|F_i| \geq 1$ for all $1 \leq i < t$.

3.4.4 The k-opt heuristic

The k-opt heuristic for 3AP was proposed originally by [Balas and Saltzman, 1991] and was extended for sAP by [Karapetyan and Gutin, 2011a].

A k-opt heuristic works as follows: for every possible subset R of k vectors with $R \in A$, solve the corresponding sAP subproblem with k vertices on each dimension. The corresponding sAP subproblem with $k < n$ vertices is solved with some exact technique, which may result in the replacement of the selected vectors for better ones. In particular, the most common option is to implement the 2-opt and 3-opt heuristics by using the Brute Force algorithm because it is the fastest option for instances with 2 or 3 vertices and many dimensions.

Algorithm 14 shows the pseudo-code of the k-opt heuristic. A complete iteration considers each of the $\binom{n}{k}$ possible combinations of vectors. This heuristic repeats iterations until no improvement is performed. By applying the Brute Force algorithm an iteration has a time complexity of $O(\binom{n}{k}k!^{s-1})$. By applying Gurobi-MAP an iteration has a time complexity of $O(\binom{n}{k}(s-1)^k)$. As in the case for DVH a bound of ten iterations is fixed.

Algorithm 14: The k-opt heuristic for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n.
k. The value of the desired k-Opt heuristic to use.
iter. Maximum number of iterations of the heuristic.
Result: A: The min cost assignment as a matrix of size $n \times s$.

```
1  Set A := FeasibleRandomGeneratedSolutionForMAP(G);
2  Set improved := true;
3  Set iterations := 1;
4  while improved = true and iterations ≤ iter do
5      Set improved := false;
6      Set iterations := iterations +1;
7      foreach combination P of vectors in (n choose k) ∈ A do
8          P' = BruteForce(P);// MAP-Gurobi(P) is suggested for k ≥ 4
9          if assignmentCost(P') < assignmentCost(P) then
10             Set improved := true;
11             Replace the set of vectors in P with P' in A;
12 return {A};
```

In order to illustrate this heuristic consider the previous generated feasible solution A for the toy instance and pick $k = 2$ vectors, in particular the vectors x^4 and x^5. The optimal solution for this two vectors can be found from the next pairs:

$$\begin{bmatrix} (4,1,2,1) & (4,1,2,5) & (4,1,5,1) & (4,1,5,5) & (4,4,2,1) & (4,4,2,5) & (\mathbf{4},\mathbf{4},\mathbf{5},\mathbf{1}) & (4,4,5,5) \\ (5,4,5,5) & (5,4,5,1) & (5,4,2,5) & (5,4,2,1) & (5,1,5,5) & (5,1,5,1) & (\mathbf{5},\mathbf{1},\mathbf{2},\mathbf{5}) & (5,1,2,1) \end{bmatrix}.$$

The corresponding weights are $\begin{bmatrix} 8 & 40 & 20 & 100 & 32 & 160 & \mathbf{80} & 400 \\ 500 & 100 & 200 & 40 & 125 & 25 & \mathbf{50} & 10 \end{bmatrix}$.

The vectors marked with bold are the new vectors that will be considered as part of the new feasible solution after applying one step of this heuristic. The new feasible solution will be:

$$A = \begin{cases} x^1 : (1,2,3,4) \\ x^2 : (2,3,1,3) \\ x^3 : (3,5,4,2) \\ x^4 : (\mathbf{4},\mathbf{4},\mathbf{5},\mathbf{1}) \\ x^5 : (\mathbf{5},\mathbf{1},\mathbf{2},\mathbf{5}) \end{cases}$$

This heuristic considers a search space of size $O(k!^{s-1})$ at each step. As in the previously described heuristics, at the end of each iteration of this heuristic the current feasible solution cannot be worse.

3.4.5 Combined heuristics

The idea of this type of heuristics consists in running several types of heuristics one after the other. The main advantage of a combined heuristic is that it allows to explore in different types of neighborhoods which may result in a significant improvement.

A correct way to combine several heuristics is to run each heuristic until no improvement is obtained and then continue with the next one. The most common is to combine a DVH with a k-opt heuristic because they include all the basic heuristics plus several other neighborhoods.

We evaluated three heuristics of this type:

1. Basics combined. Is a combination of all the basic heuristics. The execution order is simple 2-opt, inversion, circular rotation and, $k-$vertex permutation.

2. DV2+3-opt. Is a combination of a DV2 and then a 3-opt heuristic. This heuristic was proposed by [Karapetyan and Gutin, 2011a].

3. DV3+3-opt. Is a combination of a DV3 and then a 3-opt heuristic. We combine our DV3 with the 3-opt proposed by [Karapetyan and Gutin, 2011a].

3.4.6 A generalized local search heuristic

We introduce a new heuristic called Generalized Local Search Heuristic (GLSH), which extends and combines the ideas from the GDVH and the k-opt heuristic.

A GLSH works as follows: let r be an integer value such that $2 \leq r \leq s$ and, suppose we have $F_1, \ldots, F_{r-1}$ non empty proper subsets such that $F_1 \cap \cdots \cap F_{r-1} = \emptyset$ and $F_1 \cup \cdots \cup F_{r-1} \subset \{1, \ldots, s\}$. Notice the difference of 1 between the considered range for r and the integer value t used in GDVH. Let k be an integer value with $2 \leq k \leq n$. The integer values r and k should be chosen such that a reduction of the original problem (this in terms of the searching space) is achieved, which means that the next restrictions should be hold:

$$\begin{array}{c} 2 \leq r \leq s \\ 2 \leq k \leq n \\ r + k < s + n \end{array} . \tag{3.12}$$

At one step of this heuristic all the dimensions but $F_1 \cup \cdots \cup F_{r-1}$ are fixed and a set Q of k vectors from the feasible solution A are chosen, then a r-dimensional matrix M^r of size $k^1 \times \cdots \times k^r$ (recall the simplification $k^i = k$ for $1 \leq i \leq r$) with entries $M_{i^1,\ldots,i^r} = w(v^{i^1,\ldots,i^r})$ is generated. Let $v^d_{i^1,\ldots,i^r}$ denote the d-th element of the vector $v_{i^1,\ldots,i^r}$, all the vectors are built according to the next function:

$$v^d_{i^1,i^2,\ldots,i^{r-1},i^r} = \begin{cases} x^d_{i^1} \text{ if } d \in F_1 \\ x^d_{i^2} \text{ if } d \in F_2 \\ \ldots \\ x^d_{i^{r-1}} \text{ if } d \in F_{r-1} \\ x^d_{i^r} \text{ otherwise} \end{cases} \quad \text{for } 1 \leq d \leq s \ . \tag{3.13}$$

The corresponding rAP instance with k vertices at each dimension can be solved by using some exact technique, again, we suggest the MAP-Gurobi implementation. The search space of this heuristic is $O(k^{r-1}!)$. One of the main differences with the GDVH is that GLSH only considers the vectors in Q instead of the complete list of vectors of A at each time.

This heuristic extends and generalizes GDVH and k-opt heuristics because by selecting the values r, k as $2 \leq r < s$ and $k = n$ we have the case of a GDVH and by selecting the values r, k as $r = s$ and $2 \leq k < n$ we have a k-opt heuristic. Finally, by selecting the values r, k as $2 \leq r < s$ and $2 \leq k < n$ we have a particular case of GLSH which is not considered neither in GDVH nor k-opt heuristics.

The GLSH can work as a GDVH or as a k-opt, however the parameters r and k could be tuned at each step of the heuristic, instead of being fixed.

Even when one of the main advantages of GLSH is its flexibility to move among different searching spaces due to the possibility of tunning of the parameters r and k, in the experimental evaluation experience showed in the next section it was determined that, for the particular case when the GLSH is equal to the GDVH, the quality

solution is comparable to the one of more complex meta-heuristics as the memetic algorithm proposed by [Karapetyan and Gutin, 2011b].

3.4.7 Experimental evaluation

Several local search heuristics were compared in our experiments:

1. Basic heuristics. We implemented the simple 2-opt, inversion, circular rotation and, the 6-vertex permutation heuristic. We decided to implement a version of the k-vertex permutation heuristics with $k = 6$ because such value even provide acceptable running times.

2. Dimensionwise variation heuristics. We implemented the SDV2 and the DV2 proposed by [Karapetyan and Gutin, 2011a]. Also, we implemented the SDV3 and the exhaustive DV3.

3. $k-$opt heuristics. We implemented the brute force 2-opt and 3-opt heuristics proposed by [Karapetyan and Gutin, 2011a], and the same techniques but implemented with MAP-Gurobi.

4. Combined heuristics. We implemented the basics combined heuristic, the DV2+3-opt heuristic and the DV3+3-opt heuristic.

We evaluated all these heuristics under the families Random, Clique, Square Root, Geometric and Product. The problem size instances are $s = 4, n \in \{20, 30, 40, 50\}$, $s = 5, n \in \{15, 18, 25, 30\}$ and $s = 6, n \in \{12, 15, 18\}$. For Random, Clique and Square Root we used the same instances used in [Karapetyan and Gutin, 2011b], excluding the instances with $s = 3$ dimensions since all of them were solved optimally by our MAP-Gurobi. The problem sizes $s = 4, n = 50$ and $s = 5, n = 30$ are not included in their set and it were generated for our own analysis. For the Geometric and Product families we generated our own set of instances because such families were not included theirs.

For each family of instances evaluated under some heuristic we calculate the relative solution error which was also the metric used by [Karapetyan and Gutin, 2011a] and [Karapetyan and Gutin, 2011b]. The relative solution error RSE is a metric to measure the size of an error with respect to the size of the solution. In this case, let *opt* be the optimal solution for a particular instance and let A be the feasible solution then the relative solution error of A is calculated as:

$$\frac{w(A) - w(opt)}{w(opt)} \times 100 \tag{3.14}$$

Each heuristic was included as part of a multi-start heuristic which consists on executing them a certain number of times z and returning the best solution found after all the executions. We decided to set $z = 30$ because in our computational experience

a higher number of executions did not provide significant higher quality solutions and it was computationally more expensive.

All the heuristics were also implemented in C++ and its performance was evaluated on a platform with an Intel Core i5-3210M 2.5 GHz processor with 4 GB of RAM under Windows 8.

All the tables of solutions and running times show the averaged results for ten instances of each problem size for the corresponding family.

3.4.7.1 Computational results for basic heuristics

Basic heuristics, excepting the $k-$vertex permutation, are very simple procedures that explore small neighborhoods, however, they provide high quality solutions for the family of instances Geometric.

Tables 3.6 and 3.7 show the experimental results for all the basic heuristics. At each of the results cells, we show the averaged results for ten instances of each type. The first column corresponds to the instance name, the second column is the best known value averaged for ten instances of the corresponding type and the rest of the columns show the best value found by each heuristic for the ten instances.

It can be observed that in all the cases the simple 2-opt is the most effective heuristic among the basic heuristics. However, all the results are pretty far away from the optimal value, except for the Geometric family under the simple 2-opt, where the results are near to the optimal solutions. The combination of all of the basic heuristics result in a slightly most effective heuristic but at a very high computational cost. We can conclude that there is not a big advantage about the combination of all the basic heuristics.

Table 3.8 shows a summary of the relative solution error for the five families of instances under the basic heuristics. The basic combined heuristic provides the lowest relative solution error for this techniques followed by the simple 2-opt.

Tables 3.9 and 3.10 show the execution time for all the basic heuristics. The k-vertex permutation heuristic and, in consequence, the basics combined heuristic are computationally the most expensive heuristics.

3.4.7.2 Computational results for k-opt heuristics

The k-opt heuristics explore higher neighborhoods than the basic heuristics and they offer better solutions than these heuristics.

Tables 3.11 and 3.12 show the experimental results for the k-opt heuristics. It can be observed that the results for the k-opt heuristics are similar without matter the implementation, either through brute force as in [Karapetyan and Gutin, 2011a] or through our MAP-Gurobi, however it is more expensive. The advantage of the brute force version is that we can solve the corresponding k-opt problem on s dimensions in place whereas for the MAP-Gurobi version we need to create the corresponding s-

Table 3.6: Random, Clique and SquareRoot under basic heuristics.

Instance name	Best known	Simple 2−opt	Inversion	Circular rotation	6− vertex permutation	Basic combined
4r20	20.0	66.8	146.5	153.3	94.9	62.8
4r30	30.0	92.4	232.1	236.4	147.7	85.6
4r40	40.0	110.4	317.5	340.0	196.0	106.4
4r50	50.0	132.2	409.8	425.0	254.7	127.7
5r15	15.0	50.8	93.6	102.2	58.9	44.7
5r18	18.0	55.4	114.6	127.5	73.0	52.3
5r25	25.0	72.3	169.1	184.1	101.3	68.7
5r30	30.0	81.4	212.7	224.9	125.8	76.1
6r12	12.0	39.8	66.3	72.9	41.2	34.5
6r15	15.0	47.0	84.2	95.1	53.1	42.7
6r18	18.0	52.8	105.0	116.4	64.7	48.7
Avg	24.8	72.8	177.4	188.9	110.1	68.2
RSE	-	193.5	615.3	661.7	344	175
4cq20	1901.8	2235.4	3011.3	3070.2	2688.2	2188.7
4cq30	2281.9	3012.3	4523.0	4561.8	3968.8	2923.2
4cq40	2606.3	3664.5	5929.5	6089	5256.9	3668.5
4cq50	3120.5	4441.9	7518.2	7559.1	6672.1	4439.7
5cq15	3110.7	3438.0	4206.1	4276.7	3825.5	3400.5
5cq18	3458.6	3962.4	5070.0	5136.4	4552.7	3907.9
5cq25	4192.7	5134.1	6978.8	7050.6	6250.3	5062.6
5cq30	4677.9	5948.4	8387.3	8457.2	7515.6	5858.7
6cq12	4505.6	4842.5	5546.9	5627.4	5110.6	4771.3
6cq15	5133.4	5702.1	6831.6	6917.2	6217.9	5595.7
6cq18	5765.5	6575.8	8085.8	8265.8	7412.6	6488.5
Avg	3705	4450.7	6008	6091.9	5406.5	4391.4
RSE	-	20.1	62.2	64.4	45.9	18.5
4sr20	929.3	1123	1503.1	1529.7	1344.7	1101.3
4sr30	1118.6	1524.2	2230.3	2265.0	1975.1	1499.1
4sr40	1271.4	1914.7	2999.6	3016.1	2616.2	1878.8
4sr50	1530.1	2320.9	3755.7	3811.7	3318.6	2294.8
5sr15	1203.9	1360.2	1644.9	1668.9	1501.3	1336.7
5sr18	1343.9	1582.6	1989.2	2004.0	1798.2	1558.8
5sr25	1627.5	2071.4	2738.3	2766.7	2470.4	2029.7
5sr30	1852.4	2414.8	3272.8	3291.9	2948.0	2373.8
6sr12	1436.8	1557.0	1761.1	1784.4	1630.7	1531.6
6sr15	1654.6	1862.3	2181.2	2216.8	2006.1	1833.6
6sr18	1856.3	2151.4	2600.3	2627.1	2382.4	2127.3
Avg	1438.6	1807.5	2425.1	2452.9	2181.1	1778.7
RSE	-	25.6	68.6	70.5	51.6	23.6

Table 3.7: Geometric and Product under basic heuristics.

Instance name	Best known	Simple 2−opt	Inversion	Circular rotation	6− vertex permutation	Basic combined
4g20	2380.5	2383.2	3079	3173	2917.2	2382.1
4g30	3015.2	3026.9	4569.6	4685.3	4286.9	3026.0
4g40	3523.8	3553.0	6072.8	6209.9	5728.4	3551.1
4g50	4102.3	4148.9	7631.9	7833.3	7184.5	4145.9
5g15	3423.7	3427.2	4102.5	4207.6	3890.8	3426.8
5g18	3799.3	3807.7	4815	4970.4	4578.9	3806.2
5g25	4594.9	4615.4	6620.4	6827.4	6286.7	4611.6
5g30	5036.8	5078.1	7813	8066.9	7443.1	5072.4
6g12	4483.7	4487.3	5148.2	5262.6	4920	4485.6
6g15	5242.4	5252.8	6352.9	6469.1	6088.9	5248.0
6g18	5767.4	5785.7	7480.3	7587.4	7115.0	5778.1
Avg	4124.5	4142.4	5789.6	5935.7	5494.6	4139.4
RSE	-	0.4	40.4	43.9	33.2	0.4
4p20	8397.3	8402.9	8443.2	8464.4	8437.6	8403.2
4p30	13154.1	13159	13237.2	13284.8	13241.1	13159.0
4p40	16810	16817.4	16944.2	17011.0	16959.2	16817.7
4p50	20705.6	20716.1	20891.2	20965.6	20905.2	20716.9
5p15	21422.8	27070.6	27678.6	28194.1	26494.5	27067.9
5p18	23371.5	29763.9	30580.8	30101.6	32239.7	29763.8
5p25	30150.4	43046.5	43517.1	43928.8	43390.9	43051.0
5p30	34818	50234.6	50993.1	51318.1	52256.7	50232.8
6p12	7421	30578.3	32048.7	33045.3	32400.1	30436.8
6p15	8888.9	39227.4	42392	42291.5	42485.3	39115.5
6p18	9610.1	47585.2	52434.2	51415.7	52785.9	47384.8
Avg	17704.5	29691.1	30832.8	30911	31054.2	29649.9
RSE	-	67.7	74.2	74.6	75.4	67.5

Table 3.8: Summary of relative solution error for basic heuristics.

Family of instances	Simple 2−opt	Inversion	Circular rotation	6− vertex permutation	Basic combined
Random	193.5	615.3	661.7	344	**175**
Clique	20.1	62.2	64.4	45.9	**18.5**
SquareRoot	25.6	68.6	70.5	51.6	**23.6**
Geometric	0.4	40.4	43.9	33.2	**0.4**
Product	67.7	74.2	74.6	75.4	**67.5**

Table 3.9: Running times for Random, Clique and SquareRoot under basic heuristics.

Instance name	Simple 2−opt	Inversion	Circular rotation	$k-$ vertex permutation	Basic combined
4r20	0.0	0.0	0.3	3.9	4.2
4r30	0.2	0.2	1.6	9.5	10.9
4r40	0.4	0.4	4.4	15.5	20.7
4r50	1.4	1.2	11.5	30.3	44.1
5r15	0.0	0.0	0.2	3.1	2.9
5r18	0.1	0.1	0.4	4.7	4.7
5r25	0.7	0.6	1.6	9.5	10.4
5r30	3.7	2.5	4.9	23.5	26.6
6r12	0.2	0.2	0.2	2.7	2.4
6r15	0.8	0.8	1.0	6.0	5.9
6r18	2.5	2.4	2.9	10.8	10.6
Average	0.9	0.8	2.6	10.9	13
4cq20	0.3	0.2	0.7	7.9	8.0
4cq30	0.6	0.3	2.6	17.1	19.4
4cq40	0.9	0.7	7.2	26.8	37.1
4cq50	1.8	1.5	17.3	46.3	70.9
5cq15	0.1	0.0	0.2	4.1	3.9
5cq18	0.1	0.1	0.5	6.2	6.3
5cq25	0.6	0.6	1.9	12.7	13.9
5cq30	2.9	2.2	5.2	28.2	32.2
6cq12	0.2	0.1	0.3	3.9	3.4
6cq15	0.7	0.6	0.8	7.2	6.7
6cq18	1.9	1.8	2.2	11.5	11.3
Average	0.9	0.7	3.5	15.6	19.3
4sr20	0.0	0.0	0.3	4.2	4.4
4sr30	0.1	0.1	1.6	9.4	11.4
4sr40	0.4	0.4	4.9	17	23.4
4sr50	1.9	1.4	17.7	46.4	70.8
5sr15	0.0	0.0	0.2	4	3.8
5sr18	0.1	0.1	0.5	6.1	6.1
5sr25	0.6	0.6	1.8	12.3	13.5
5sr30	3.1	2.2	5.2	28.2	31.7
6sr12	0.2	0.1	0.3	3.8	3.4
6sr15	0.6	0.6	0.8	7.1	6.5
6sr18	1.9	1.8	2.2	11.3	11.1
Average	0.8	0.6	3.2	13.6	16.9

Table 3.10: Running times for Geometric and Product under basic heuristics.

Instance name	Simple 2−opt	Inversion	Circular rotation	$k-$ vertex permutation	Basic combined
4g20	0.1	0.1	0.5	6.2	6.2
4g30	0.2	0.2	1.7	11.8	12.5
4g40	0.4	0.4	4.5	17.9	21.9
4g50	0.9	0.8	10.0	28.7	37.8
5g15	0.0	0.0	0.2	4.3	3.7
5g18	0.1	0.1	0.4	6.2	5.7
5g25	0.6	0.6	1.7	12.8	12.5
5g30	1.5	1.5	3.5	20.1	20.5
6g12	0.2	0.2	0.3	4.0	3.5
6g15	0.7	0.7	0.9	7.4	6.6
6g18	2.5	1.8	2.3	12.5	11.4
Average	0.6	0.5	2.3	11.9	12.9
4p20	0.1	0.1	0.4	5.2	4.4
4p30	0.2	0.2	1.7	12.4	10.6
4p40	0.5	0.5	5.1	20.6	18.6
4p50	2.4	1.1	10	40.8	30.1
5p15	0.3	0.2	0.4	6.0	5.0
5p18	0.4	0.3	0.8	9.2	7.4
5p25	1.3	1.2	2.6	19.3	15.2
5p30	2.7	2.6	5.5	28.8	23.7
6p12	0.4	0.3	0.4	5.2	4.2
6p15	1.3	1.0	1.3	9.3	7.5
6p18	4.6	3.0	3.4	14.9	12.7
Average	1.3	1.0	2.9	15.6	12.7

dimensional matrix and to build model for the MAP-Gurobi, which is a very expensive process. In summary, it is not convenient to use MAP-Gurobi for solving a lot of small instances of MAP.

It can be observed that k-opt heuristics provide better solutions than basic heuristics however are still far away from optimal solutions, but in the case of the Geometric family of instances, in which the results are very near to the optimal solutions. It can be observed that there is a significant difference between the 2-opt and the 3-opt heuristics in all the cases. We believe that the 4-opt heuristic can even be higher than the 3-opt heuristic, however its evaluation is computationally very expensive so that we need a higher computer power. It can be interesting to determine the value of k for which the k-opt Gurobi version is better than the brute force algorithm. We let such analysis as future work.

Table 3.13 shows a summary of the relative solution error for the five families of instances under the k-opt heuristics. The 3-opt heuristic provides the lowest relative solution error among this type of techniques and, in particular, the Gurobi version is slightly better than the brute force version.

Tables 3.14 and 3.15 show the corresponding running times for the k-opt heuristics. It can be observed that the computational cost of the MAP-Gurobi versions for the 2-opt and 3-opt heuristics is computationally high.

3.4.7.3 Computational results for dimensionwise variation heuristics

This family of techniques is one of the most competitive heuristics for MAP.

Tables 3.16 and 3.17 show the experimental results for some versions of the DVH. We can observe that by combining DV2 + 3-opt the quality of the obtained results is increased considerably, such that the relative solution error is lower than 10% for all the families of instances. By the other side, both of our proposed versions of DVH, the SDV3 and DV3, are superior to the combination of DV2 + 3-opt, obtaining the optimal solutions for all the instances of the Random and the Geometric families of instances. In our case, the combination of DV3 + 3-opt does not provide a significant advantage against the DV3 by itself. Even when we averaged the relative solution error for all the families of instances, we can calculate this metric for each family of instances and for each dimension and, in general, the lower values of relative solution error belongs to the instances with the dimension $s = 3$, followed by those for $s = 5$ and finally for $s = 6$. We believe that the use of reductions of a sAP to a $(s-1)$AP can even provide higher quality solutions, however we let such analysis as future work.

Table 3.18 shows a summary of the relative solution error for the five families of instances under DVH. The heuristics DV3 and DV3 + 3-opt provide the lowest relative solution error among this type of techniques. We consider that there is not a significant difference between the use of DV3 and its combination with the 3-opt. Probably the combination of DV3 with more powerful k-opt heuristics can provide higher results like in the case of the combination with DV2 with the 3-opt. We let

Table 3.11: Random, Clique and SquareRoot under k-opt heuristics.

Instance name	Best known	2-opt (GK)	3-opt (GK)	2-opt Gurobi	3-opt Gurobi
4r20	20.0	52.5	25.7	53.7	24.1
4r30	30.0	72.7	35.3	70.2	33.1
4r40	40.0	87.8	44.0	84.7	41.7
4r50	50.0	105.6	53.3	101.7	50.6
5r15	15.0	31.3	16.5	30.4	15.7
5r18	18.0	36.2	19.0	34.1	18.3
5r25	25.0	47.8	25.4	45.3	25.0
5r30	30.0	54.8	30.1	48.6	30.0
6r12	12.0	21.1	12.0	19.5	12.0
6r15	15.0	24.5	15.0	23.2	15.0
6r18	18.0	28.3	18.0	28.5	18.0
Avg	24.8	51.1	26.8	49.1	25.8
RSE	-	106	8.1	98.0	4.0
4cq20	1901.8	2179.8	1983.5	2189.2	1985.9
4cq30	2281.9	2913.0	2549.9	2917.3	2555.3
4cq40	2606.3	3627.7	3055.3	3666.5	3088.5
4cq50	3120.5	4413.9	3670.3	4383.7	3608.4
5cq15	3110.7	3365.5	3185.3	3358.3	3191.1
5cq18	3458.6	3879.3	3597.3	3845.8	3579.6
5cq25	4192.7	5061.8	4562.1	5025.1	4528.2
5cq30	4677.9	5817.3	5237.3	5790.1	5249.8
6cq12	4505.6	4744.0	4577.6	4737.3	4562.6
6cq15	5133.4	5552.8	5309.8	5583.1	5326.7
6cq18	5765.5	6511.4	6041.8	6529.1	6033.9
Avg	3705	4369.7	3979.1	4366	3973.6
RSE	-	17.9	7.4	17.8	7.2
4sr20	929.3	1094.2	979.5	1081.1	981.1
4sr30	1118.6	1492.3	1272.5	1492.1	1283.4
4sr40	1271.4	1878.5	1527.9	1861.3	1550.6
4sr50	1530.1	2265.2	1852.0	2260.6	1842.7
5sr15	1203.9	1330.6	1248.6	1322.7	1246.7
5sr18	1343.9	1533.8	1416.6	1537.0	1419.1
5sr25	1627.5	2020.7	1814.0	1991.0	1808.6
5sr30	1852.4	2323.6	2080.0	2338.7	2048.0
6sr12	1436.8	1521.3	1468.3	1530.1	1460.5
6sr15	1654.6	1830.4	1730.9	1828.3	1722.8
6sr18	1856.3	2115.5	1979.0	2105.8	1983.9
Avg	1438.6	1764.2	1579.0	1759.0	1577.0
RSE	-	22.6	9.8	22.3	9.6

Table 3.12: Geometric and Product under k-opt heuristics.

Instance name	Best known	2-opt (GK)	3-opt (GK)	2-opt Gurobi	3-opt Gurobi
4g20	2380.5	2381.7	2380.5	2384.0	2380.5
4g30	3015.2	3022.6	3017.3	3021.9	3015.9
4g40	3523.8	3551.9	3529.6	3546.1	3525.5
4g50	4102.3	4131.5	4112.9	4143.3	4110.5
5g15	3423.7	3424.8	3423.7	3424.7	3423.7
5g18	3799.3	3806.2	3799.3	3803.6	3799.3
5g25	4594.9	4608.5	4596.5	4607.1	4595.8
5g30	5036.8	5068.7	5040.2	5059.5	5041.3
6g12	4483.7	4488.7	4483.7	4487.1	4483.7
6g15	5242.4	5247.7	5243.1	5248.0	5242.5
6g18	5767.4	5780.8	5767.4	5780.9	5767.4
Avg	4124.5	4137.6	4126.7	4136.9	4126.0
RSE	-	0.3	0.1	0.3	0.0
4p20	8397.3	8399.1	8397.4	8398.4	8397.6
4p30	13154.1	13155.7	13154.2	13155.3	13154.1
4p40	16810.0	16812.3	16810.2	16812.3	16810.1
4p50	20705.6	20710.4	20706.3	20709.8	20705.7
5p15	21422.8	24213.4	21423.4	22861.5	21551.4
5p18	23371.5	26435.7	24012.4	24993.9	23440.6
5p25	30150.4	37232.6	30475.0	35341.0	30508.1
5p30	34818.0	43474.0	35811.6	41606.7	35727.9
6p12	7421.0	11882.0	8172.2	11366.7	7793.9
6p15	8888.9	15302.1	10356.6	14185.8	9514.6
6p18	9610.1	17181.3	11379.1	16506.6	10411.9
Avg	17704.5	21345.3	18245.3	20539.8	18001.4
RSE	-	20.6	3.1	16.0	1.7

Table 3.13: Summary of relative solution error for basic heuristics.

Family of instances	2-opt (GK)	3-opt (GK)	2-opt Gurobi	3-opt Gurobi
Random	106	8.1	98	**4.0**
Clique	17.9	7.4	17.8	**7.2**
SquareRoot	22.6	9.8	22.3	**9.6**
Geometric	0.3	0.1	0.3	**0.0**
Product	20.6	3.1	16	**1.7**

Table 3.14: Times for Random, Clique and SquareRoot under k-opt heuristics.

Instance name	2-opt (GK)	3-opt (GK)	2-opt Gurobi	3-opt Gurobi
4r20	0.4	2.2	21.3	614.8
4r30	0.7	7.9	53.1	2532.5
4r40	1.3	19.3	91.8	6261.3
4r50	1.5	35.1	114.1	10998.2
5r15	0.4	3.2	16.2	1135.5
5r18	0.4	6.9	27.2	2491.8
5r25	1.2	16.7	49.6	6533.4
5r30	2.9	36.3	98.6	18209.3
6r12	0.6	10.9	25.2	613.7
6r15	1.5	23.5	44.6	1201.3
6r18	3.9	42.9	69.5	2666.6
Average	1.4	18.7	55.6	4841.7
4cq20	0.3	2.3	26.3	287.6
4cq30	0.6	10.0	70.0	1130.1
4cq40	1.1	28.7	127.1	3486.4
4cq50	1.8	52.3	174.6	9938.3
5cq15	0.2	3.7	19.1	1126.3
5cq18	0.4	7.2	31.5	2493.5
5cq25	1.1	23.8	66.4	7007.8
5cq30	3.0	52.2	129.8	15830.2
6cq12	0.6	10.8	24.7	429.9
6cq15	1.1	25.8	43.0	973.3
6cq18	3.1	48.5	64.6	2161.5
Average	1.2	24.1	70.6	4078.6
4sr20	0.3	2.1	21.7	486.5
4sr30	0.5	8.4	55.6	2095.8
4sr40	0.9	24.5	112.7	5668.5
4sr50	1.7	52.7	187.7	11579.6
5sr15	0.3	4.8	27.8	1354.7
5sr18	0.4	8.4	39.1	3253.0
5sr25	1.4	28.0	88.6	8205.7
5sr30	2.9	53.2	126.1	17488.1
6sr12	0.5	12.9	30.1	622.3
6sr15	1.4	29.6	55.6	1244.0
6sr18	3.7	59.5	87.7	2505.2
Average	1.3	25.8	75.7	4954.9

Table 3.15: Times for Geometric and Product under k-opt heuristics.

Instance name	2-opt (GK)	3-opt (GK)	2-opt Gurobi	3-opt Gurobi
4g20	0.2	1.2	16.9	370.4
4g30	0.3	4.9	42.1	1402.2
4g40	0.7	13.7	85.0	4002.1
4g50	1.3	29.4	135.5	7860.4
5g15	0.4	2.4	15.0	1090.5
5g18	0.5	5.6	29.2	2318.3
5g25	1.1	16.8	61.9	5891.4
5g30	2.3	31.7	95.2	9285.7
6g12	0.4	8.9	22.3	350.7
6g15	1.3	19.6	38.5	739.2
6g18	3.1	37.6	60.5	1394.6
Average	1.1	15.6	54.7	3155.0
4p20	0.2	0.9	13.3	393.1
4p30	0.3	4.1	42.0	1145.2
4p40	0.7	10.7	79.3	3608.8
4p50	1.3	22.1	129.1	7085.8
5p15	0.3	3.1	26.4	671.4
5p18	0.4	6.1	46.6	1420.0
5p25	1.3	18.1	105.1	4477.8
5p30	4.8	396.7	158.0	6520.8
6p12	0.5	10.8	34.8	650.5
6p15	1.1	24.3	60.4	1404.8
6p18	3.0	45.7	98.2	3068.1
Average	1.3	49.4	72.1	2767.8

Table 3.16: Random, Clique and SquareRoot under DVH.

Instance Name	Best known	Simple DVH2	DVH2	DVH2 + 3-opt	Simple DVH3	DVH3	DVH3 + 3-opt
4r20	20.0	42.2	34.9	26.6	20.0	20.0	20.0
4r30	30.0	50.7	43.1	35.4	30.0	30.0	30.0
4r40	40.0	56.7	51.7	44.2	40.0	40.0	40.0
4r50	50.0	65.5	57.9	52.3	50.0	50.0	50.0
5r15	15.0	33.7	24.2	16.1	15.2	15.0	15.0
5r18	18.0	38.9	26.6	18.8	18.0	18.0	18.0
5r25	25.0	42.6	34.6	25.2	25.0	25.0	25.0
5r30	30.0	46.8	36.2	30.0	30.0	30.0	30.0
6r12	12.0	27.3	16.1	12.0	12.6	12.0	12.0
6r15	15.0	31.9	19.8	15.0	15.1	15.0	15.0
6r18	18.0	33.7	22.8	18.0	18.0	18.0	18.0
Avg	24.8	42.7	33.4	26.7	24.9	24.8	24.8
RSE	-	72.2	34.7	7.7	0.4	0.0	0.0
4cq20	1901.8	2009.0	2000.5	1964.4	1910.3	1909.0	1909.0
4cq30	2281.9	2530.0	2515.3	2474.8	2319.1	2322.6	2322.6
4cq40	2606.3	3018.6	2992.8	2956.5	2722.6	2714.2	2714.2
4cq50	3120.5	3504.9	3498.9	3467.9	3153.7	3144.8	3144.8
5cq15	3110.7	3229.4	3223.4	3190.9	3148.0	3128.0	3128.0
5cq18	3458.6	3695.2	3624.2	3564.8	3507.7	3507.9	3507.9
5cq25	4192.7	4597.6	4570.1	4509.4	4340.4	4337.2	4335.2
5cq30	4677.9	5205.7	5195.3	5119.6	4918.4	4886.1	4886.1
6cq12	4505.6	4651.5	4615.5	4550.2	4534.9	4532.6	4532.6
6cq15	5133.4	5375.9	5382.1	5303.6	5237.2	5216.7	5214.9
6cq18	5765.5	6131.0	6123.0	6018.5	5917.6	5895.5	5894.0
Avg	3705	3995.3	3976.5	3920.1	3791.8	3781.3	3780.8
RSE	-	7.8	7.3	5.8	2.3	2.1	2.0
4sr20	929.3	998.6	981.8	969.9	935.0	937.4	937.4
4sr30	1118.6	1267.3	1265	1244.5	1153.5	1153.2	1153.2
4sr40	1271.4	1496.0	1487.6	1478.4	1331.5	1337.0	1337.0
4sr50	1530.1	1765.4	1774.2	1743.3	1543.6	1551.6	1551.6
5sr15	1203.9	1276.6	1263.0	1243.3	1220.0	1215.4	1215.4
5sr18	1343.9	1460.2	1437.8	1416.5	1377.2	1369.3	1364.0
5sr25	1627.5	1826.0	1824.7	1779.2	1704.7	1703.9	1703.9
5sr30	1852.4	2082.1	2079.0	2058.8	1939.6	1935.3	1933.5
6sr12	1436.8	1502.4	1489.8	1467.7	1452.7	1447.6	1447.6
6sr15	1654.6	1757.5	1728.7	1706.2	1692.0	1689.1	1685.2
6sr18	1856.3	2008.5	1987.1	1970.6	1919.9	1911.9	1911.6
Avg	1438.6	1585.5	1574.4	1552.6	1479.1	1477.4	1476.4
RSE	-	10.2	9.4	7.9	2.8	2.7	2.6

Table 3.17: Geometric and Product under DVH.

Instance Name	Best known	Simple DVH2	DVH2	DVH2 + 3-opt	Simple DVH3	DVH3	DVH3 + 3-opt
4g20	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5
4g30	3015.2	3015.4	3015.2	3015.2	3015.2	3015.2	3015.2
4g40	3523.8	3524.4	3524.0	3523.8	3523.8	3523.8	3523.8
4g50	4102.3	4103.6	4102.4	4102.3	4102.3	4102.3	4102.3
5g15	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7
5g18	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3
5g25	4594.9	4595.7	4595.4	4594.9	4594.9	4594.9	4594.9
5g30	5036.8	5038.7	5037.1	5037.0	5036.8	5036.8	5036.8
6g12	4483.7	4485.6	4483.7	4483.7	4483.7	4483.7	4483.7
6g15	5242.4	5242.4	5242.6	5242.4	5242.4	5242.4	5242.4
6g18	5767.4	5767.4	5767.4	5767.4	5767.5	5767.4	5767.4
Avg	4124.5	4125.2	4124.7	4124.6	4124.6	4124.5	4124.6
RSE	-	0.0	0.0	0.0	0.0	0.0	0.0
4p20	8397.3	8401.1	8397.9	8397.4	8397.3	8397.3	8397.3
4p30	13154.1	13156.3	13154.5	13154.2	13154.1	13154.1	13154.1
4p40	16810.0	16814.5	16810.9	16810.1	16810.0	16810.0	16810.0
4p50	20705.6	20713.4	20708.1	20706.1	20705.6	20705.6	20705.6
5p15	21422.8	27688.9	22293.3	21423.0	21671.8	21422.8	21422.8
5p18	23371.5	29756.5	25520.0	23508.4	24635.0	23371.8	23371.7
5p25	30150.4	43044.7	34971.2	30448.2	34333.0	30258.4	30258.0
5p30	34818.0	50230.6	41908.2	36243.8	40137.2	35231.3	35231.2
6p12	7421.0	29396.4	10314.4	8300.7	9282.6	7664.0	7664.0
6p15	8888.9	37193.8	12245.8	9889.0	10768.7	8888.9	8888.9
6p18	9610.1	46417.1	13125.6	10842.4	12085.9	9610.1	9610.1
Avg	17704.5	29346.7	19950	18156.7	19271.0	17774.0	17774.0
RSE	-	65.8	12.7	2.6	8.8	0.4	0.4

Table 3.18: Summary of relative solution error for DVH.

Family of instances	Simple DVH2	DVH2	DVH2 + 3-opt	Simple DVH3	DVH3	DVH3 + 3-opt
Random	72.2	34.7	7.7	0.4	**0.0**	**0.0**
Clique	7.8	7.3	5.8	2.3	2.1	**2.0**
SquareRoot	10.2	9.4	7.9	2.8	2.7	**2.6**
Geometric	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
Product	65.8	12.7	2.6	8.8	**0.4**	**0.4**

such analysis as future work.

Tables 3.19 and 3.20 show the corresponding running times for the DVH and the combined heuristics. It can be observed that the computational cost of reductions of a sAP to a 3AP are more expensive than reductions to a 2AP, however we highly believe that the cost at the obtaining benefit is a good deal since for some families of instances our DV3 is able to find the optimal solutions and, for the rest of families, the relative solution error is approximately of 2% or lower which is competitive against more complex meta-heuristics such as the state of the art memetic algorithm proposed by [Karapetyan and Gutin, 2011a].

In summary, it can be observed that the DVH techniques hold the title as the better heuristics for the MAP and, our developed versions, the SDV3 and DV3, provide competitive results against more complex metaheuristics which use as part of its machinery the simplest versions of the DVH family of heuristics, that is the SDV2 and DV2.

3.5 A new simple memetic algorithm for the MAP

The concept of memetic algorithm comes from the idea of combining a genetic algorithm with a local search. A genetic algorithm is a metaheuristic inspired by the process of natural selection.

Genetic algorithms approach optimization problems by considering some bio-inspired operators such as mutation, crossover and selection. A genetic algorithm requires a genetic representation of a feasible solution as well as a fitness function to evaluate the quality of the solution. The advantage of this type of technique is that it works on a set of multiple feasible solutions and improve them by taking the best individuals among an evolutionary process. The diversification of individuals avoids to direct the set of feasible solutions to a local optimum since it allows to explore over several neighborhoods at the same time.

Algorithm 15 shows the general structure of a genetic algorithm. Let *generations* be the number of evolutionary steps, let *populationSize* be the required size for the population among the evolutionary process and, let *mutationProbability* the

Table 3.19: Running times for Random, Clique and SquareRoot under DVH.

Instance name	Simple DVH2 (GK)	DVH2 (GK)	DV2 + 3−opt	Simple DVH3	DVH3	DVH3+ 3−opt
4r20	0.2	0.3	2.5	177.8	178.1	179.4
4r30	0.3	0.5	8.6	875.4	1297.0	866.0
4r40	0.9	0.8	21.4	2474.2	2895.3	2112.8
4r50	1.3	1.4	43.7	3035.8	3541.7	4011.4
5r15	0.3	0.2	4.3	65.2	120.9	125.1
5r18	0.5	0.4	10.3	200.1	383	388.8
5r25	1.1	1.2	29.8	678.9	2013.5	1739.1
5r30	2.7	2.7	54.4	1514.6	4984.7	4813.6
6r12	0.4	0.3	9.6	38.9	159.9	183.1
6r15	1.0	0.9	21.5	77.0	269.6	303.7
6r18	2.5	103.1	36.7	183.9	676.6	645.4
Average	1	10.2	22.1	847.4	1501.8	1397.1
4cq20	0.5	0.2	2.0	120	122.2	124.1
4cq30	0.4	0.4	6.1	431.3	426.7	432.0
4cq40	0.7	0.8	17.5	2128.8	1550.9	1540.8
4cq50	1.3	1.6	38.4	2478.1	2505.5	2520.7
5cq15	0.3	0.3	4.2	52.5	128.4	132.3
5cq18	0.4	0.4	7.1	129.9	296.7	300.6
5cq25	1.6	1.7	21.8	527.5	832.8	861.8
5cq30	2.4	2.1	36.9	602.8	1334.2	1364.8
6cq12	0.4	0.5	11.6	41.7	231.2	240.1
6cq15	1.1	1.2	26.6	80.1	442.9	457.2
6cq18	3.1	3.0	54.5	295.3	1099.0	1193.4
Average	1.1	1.1	20.6	626.2	815.5	833.4
4sr20	0.3	0.2	1.6	110.1	108.3	162.5
4sr30	0.4	0.4	6.1	505.4	521.3	528.0
4sr40	0.7	0.8	15.8	1539.4	2033.3	1512.0
4sr50	1.2	1.2	28.6	2405.5	2496.3	2381.3
5sr15	0.2	0.3	4.5	56.8	131.0	130.9
5sr18	0.5	0.3	5.9	100.5	230.9	245.2
5sr25	1.1	0.9	18.6	306.3	884.9	727.4
5sr30	2.5	2.7	44.1	860.0	2011.4	2640.9
6sr12	0.5	0.4	9.1	33.1	194.1	199.2
6sr15	1.1	1.0	19.8	64.8	362.1	375.0
6sr18	3.1	2.8	47.1	195.5	1547.9	1060.3
Average	1.1	1	18.3	561.6	956.5	905.7

Table 3.20: Running times for Geometric and Product under DVH.

Instance name	Simple DVH2 (GK)	DVH2 (GK)	DV2 + 3−opt	Simple DVH3	DVH3	DVH3+ 3−opt
4g20	0.1	0.1	0.8	85.3	82.9	54.1
4g30	0.2	0.2	2.6	224.9	208.9	172.8
4g40	0.4	0.5	6.7	660.9	646.0	515.8
4g50	1.2	1.6	18.0	1224.9	1011.7	1028.4
5g15	0.5	0.1	3.2	28.7	73.5	123.9
5g18	1.4	0.2	5.5	68.5	173.1	304.6
5g25	0.9	0.9	19.6	233.7	926.3	797.8
5g30	2.6	2.9	33.1	563.0	1431.9	1312.2
6g12	0.4	0.4	8.4	35.9	215.7	255.7
6g15	1.1	1.1	18.7	65.3	406.5	486.0
6g18	3.2	3.2	40.9	159.9	1001.2	1133.5
Average	1.1	1.0	14.3	304.6	561.6	562.3
4p20	0.1	0.2	1.9	485.6	117.28	153.6
4p30	0.5	0.5	6.1	849.0	827.29	840.0
4p40	0.7	0.9	15.7	2010.8	1770.13	1961.5
4p50	1.4	1.9	30.9	7837.3	8506.31	9160.0
5p15	0.1	0.3	6.6	92.1	191.89	295.5
5p18	0.4	0.6	12.3	286.4	637.2	831.6
5p25	1.2	1.7	28.3	1015.9	2836.16	3108.4
5p30	2.8	3.9	57.8	3655.8	7110.86	7033.6
6p12	0.5	0.7	21.7	75.1	341.4	438.4
6p15	1.3	1.9	44.9	163.6	639.5	842.9
6p18	7.1	3.6	96.7	346.4	1544.7	2703.6
Average	1.5	1.5	29.4	1528.9	2229.3	2488.1

probability of mutating an individual, a genetic algorithm works as follows: at first, all the individuals are created according to the required size *populationSize*. Then are executed a total of *generations* iterations. At each iteration, first a selection of the best individuals is performed. Based on the selected individuals, a set new of individuals is created. Finally, with a low probability, some mutations are applied to the individuals. The expected is to have better individuals as soon as the generations are passed.

Algorithm 15: The general structure of a genetic algorithm.

Input: generations. The number of generations to iterate.
populationSize. The size of the population among the iterations.
mutationProbability. The probability to mutate an individual.
Result: best_individual. The best individual among all the generations.

```
1  Set individuals := Initialization(PopulationSize);
2  Set iterations := 1;
3  while iterations ≤ generations do
4  |  Set individuals := SelectionOfSurvivors(individuals);
5  |  Set offspring := Crossover(individuals);
6  |  foreach individual in offspring do
7  |  |  Set individual := MutateIndividual(individual, mutationProbability);
8  |  Set individuals := offspring;
9  |  Set iterations := iterations +1;
10 return {GetBestOfIndividuals(individuals)};
```

In a memetic algorithm the main idea is to mutate the individuals by improving them instead of by performing random changes over them. Such improving can be performed by a local search heuristic. The disadvantage is that the technique can direct the individuals to some local optimum because the diversification provided by random mutations is lost. In some cases, can even be applied some random changes over a low proportion of the individuals in order to avoid lose the diversification.

The first memetic algorithm for the MAP was proposed by [Huang and Lim, 2006]. It was called LSGA (Local Searching Genetic Algorithm) and consist on a basic structure of a genetic algorithm with an initial population of 100 individuals random generated and improved by a SDV2, the partially mapped crossover as crossover operator, the DV2 instead of a mutation operator, a basic selection similar to the elitist selection and, a stop criteria consisted on ending either after 10 generations of no improvement or when in the set of individuals there were many duplicates. The Figure 3.5 shows the general structure of an iteration of the LSGA.

We introduce a similar memetic algorithm to the one proposed by [Huang and Lim, 2006] with the difference that we evaluate several crossover operators, selection functions, local search heuristics and some mutations, in order to obtain a more robust memetic algorithm that provides higher quality solutions.

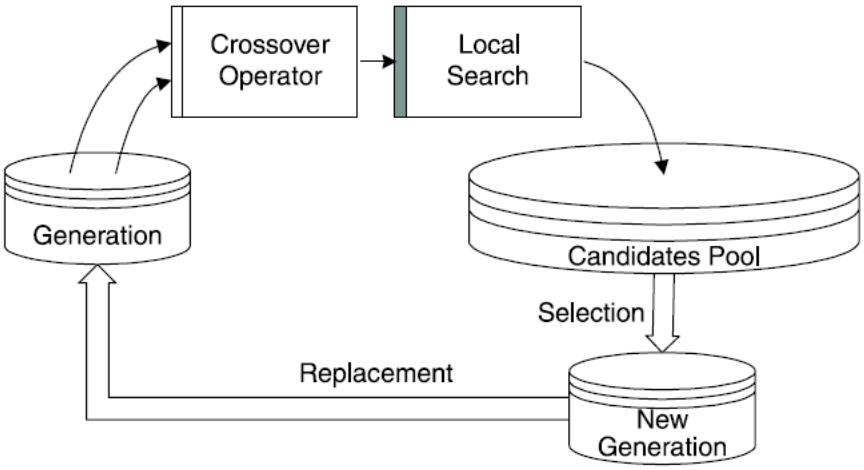

Figure 3.5: Structure of the memetic algorithm proposed by Huang and Lim.

Algorithm 16 shows the proposed simple memetic algorithm (SMA). Our SMA returns the best individual found among all the iterations. In the initialization, the population is randomly generated and improved by a local search heuristic, then the first best individual is obtained. At each iteration, first a crossover method is applied but, instead of obtaining a offspring equal to the required size of the population, just a proportion determined by the input parameter $survivorPerc$ is generated. These individuals will be the first part of the next generation. After the crossing, each individual from the offspring should be improved by the local search heuristic. The second part of the new generation of individuals is obtained from the selection method. Just the proportion of the current population indicated by the input parameter $survivorPerc$ is going to survive. At the end of each iteration, some individuals of the new population are mutated with a probability of $mutationProb$. Finally, we update the best individual by considering the individuals from the next generations.

One of the main differences of our memetic algorithm against others is that we select uniformly at random two individuals to be crossed instead of using a selection function. In our case, we decided to use the selection function to select a set of individuals from the current generation to be part of the next generation joined with the individuals resulted from the crossover function. By performing this process we allow a higher diversification as well as we increase the fitness of the population. Even when a good individual can be crossed with an individual of low quality, the local search heuristic makes to converge it to a local optimum solution.

3.5.1 Genetic representation and fitness function

The genetic representation is the way of representing solutions in an evolutionary method. The genetic representation must have all the characteristics that allow us to evaluate the quality of a feasible solution.

Algorithm 16: A new memetic algorithm for the MAP.

Input: $G = (X_1, \ldots, X_s; E)$. Weighted hypergraph of a sAP of size n.
generations. The number of generations to iterate.
crossoverMethod. The type of crossover method to be used.
LSHMethod. The type of local search heuristic to be used.
mutationMethod. The type of mutation method to be used.
mutationProb. The probability to mutate an individual.
N. The size fo the population among the iterations.
selectionMethod. The type of selection method to be used.
survivorPerc. The % of survivor individuals from the previous generation.
Result: best_individual: The best fitted individual.

```
1  Set individuals := ∅;
2  while |individuals| ≤ N do
3      new_individual := FeasibleRandomGeneratedSolutionForMAP(G);
4      new_individual := LSH(individual, LSHMethod);
5      individuals := individuals ∪ new_individual;
6  Set best_individual := GetBestOfIndividuals(individuals);
7  Set iterations := 1;
8  while iterations ≤ generations do
9      Set offspring := Crossover(individuals, crossoverMethod, 1.0 -
       survivorPerc);
10     foreach off in offspring do
11         Set off := LSH(off, LSHMethod);
12     Set new_individuals := Selection(individuals, selectionMethod,
       survivorPerc);
13     Set individuals := new_individuals ∪ offspring;
14     foreach individual in individuals do
15         Set individual := Mutate(individual, mutationMethod, mutationProb);
16         Set individual := LSH(individual, LSHMethod);
17     Set best_individual := GetBestOfIndividuals(individuals ∪ best_individual);
18     Set iterations := iterations+1;
19 return {best_individual};
```

The most common representation of a solution is as a binary array. In the case of our problem we decided to adopt the genetic representation as a set of s permutations, where each of the s dimensions of a feasible solution corresponds with a permutation of the vertices on that dimension. This representation is the same used for the feasible solutions for our developed local search heuristics and will allow us to incorporate them as part of our memetic algorithm. The disadvantage of this representation is that the crossing of individuals, in most of the cases, produce unfeasible solutions which should be repaired.

The fitness function is the way to evaluate how close to the optimum is a feasible solution. Our fitness function consists in summing the weights of the hyperedges of each feasible solution. A solution A is better than a solution B if the sum of the weights of A is lower than the sum of the weights of B. The process of evaluating each solution takes $O(sn)$ time.

3.5.2 The selection function

In our case, the selection function allow us to obtain the next generation of individuals from a current population. There are several methods to perform the selection for the next generation of individuals. We implemented three different selection functions: elitist, tournament and roulette wheel selection.

3.5.2.1 Elitist selection

This strategy consists in choosing a limited number of the best candidates of a population to survive for the next generation. This technique tends to avoid the crossover and mutation operators. The number of elite individuals should not be high, otherwise the population will tend to degenerate and we lose diversification.

Algorithm 17 shows this procedure. It is quite simple, the first step consists in ordering the individuals according to its fitness function. Then the best $survivorPerc$ percent of the individuals is selected to be returned. Since only a sorting is required, this process takes $O(N \log N)$ time where N is the number of individuals in the population.

Algorithm 17: The elitist selection function.

Input: *Individuals*. A set of assignments.
survivorPerc. The percentage of survivor individuals from the current set.
Result: survivors: The set of survivor individuals.

```
1 orderedIndividuals := Sort individuals descending by its fitness value;
2 survivors := Select the best survivorPerc individuals from orderedIndividuals;
3 return {survivors};
```

The main advantage of this strategy is that it tries to avoid previously discarded solutions that were not so good. The main disadvantage is that it may cause a fast convergence to local optimums.

3.5.2.2 Tournament selection

This is a rank-based strategy. It consists on randomly choosing a set of individuals and performing a set of tournaments between them. The winner of each tournament is selected to be part of the next generation. If the size of a tournament is short

then weak individuals have a bigger chance to be selected whereas if the size of a tournament is large then weak individuals have a lower chance.

Algorithm 18 shows this procedure. In the first step, a random permutation of the elements from $\{1, \ldots, |individuals|\}$ is performed. Such permutation will be used to match in a tournament the 1st and the 2nd individuals, the 3rd and the 4th individuals and so on until we have the required population size. The winner from each tournament is added to the survivors set. A random permutation of the individuals can be performed in $O(N)$ time and the tournaments are comparisons between a pair of fitness functions, therefore this procedure takes $O(N)$ time.

Algorithm 18: The tournament selection function.

Input: *Individuals*. A set of assignments.
survivorPerc. The percentage of survivor individuals from the current set.
Result: survivors: The set of survivor individuals.

```
1 Set N := |individuals|;
2 permutation := Make a random permutation of the elements from 1, ..., N;
3 survivors := ∅;
4 index := 1;
5 permutation := permutation || permutation while
  |survivors| < survivorPerc * |individuals| do
6 |   winner := MakeTorunament(individuals[ permutation[index] ], individuals[
  |   permutation[index + 1]]);
7 |   survivors := survivors ∪ winner;
8 |   index := index + 2;
9 return {survivors};
```

The main advantage of this strategy is that it promotes the diversification of individuals. The main disadvantage is that the best individuals from the previous generation can be discarded.

3.5.2.3 Roulette wheel selection

This is strategy that selects individuals according to some probability of selection determined by the fitness of each individual. This strategy is also known as fitness proportionate selection. The idea is that individuals with a better fitness have a higher chance to be selected whereas lower fitness individuals have a lower probability of being selected. Let f_i the fitness of the individual i of the current population then its probability of being selected is:

$$p_i = \frac{f_i}{\sum_{i=1}^{|individuals|} f_i} \tag{3.15}$$

Algorithm 19 shows this procedure. In the first step, the individuals are sorted descending order by its fitness value. The total fitness sum $fitSum$ is calculated and

an array *dp* stores the cumulative fitness sum until each individual. At each iteration, a uniformly random value α is generated from the interval $[0, fitSum)$. Then we perform a binary search over the array *dp* to find the first index j at which $\alpha \leq dp[j]$. Then, the j-th individual is added to the next generation.

Algorithm 19: The roulette wheel selection function.

Input: *Individuals*. A set of assignments.
survivorPerc. The percentage of survivor individuals from the current set.
Result: survivors: The set of survivor individuals.

```
1  Set N := |individuals|;
2  orderedIndividuals := Sort individuals descending by its fitness value;
3  Set fitnessSum := 0;
4  Set dp := An array of N elements;
5  foreach index in 1 : N do
6  |   fitnessSum := fitnessSum + fitness(orderedIndividuals[i]);
7  |   Set dp[index] := fitnessSum;
8  Set survivors := ∅;
9  while |survivors| < survivorPerc * |orderedIndividuals| do
10 |   Set alpha := random() % fitnessSum;
11 |   Set index := binarySearch(dp, dp + N, alpha);
   |   // returns the lowest index with alpha ≤ dp[index]
12 |   Set survivors := survivors ∪ orderedIndividuals[index];
13 return {survivors};
```

The main advantage of this function is that all individuals have a probability of being selected according to its fitness value so the expected behavior is to have a bigger proportion of the most apt individuals but, it is allowed to have some weak individuals, promoting a higher diversification. The main disadvantage is that the same individual can be selected to appear more than once for the next generation, especially the best fitted individuals. The binary search in the main cycle takes $O(\log N)$ time and there are required $survivorPerc \cdot N$ iterations so the time complexity is $O(survivorPerc \cdot N \cdot \log N)$.

3.5.3 The crossover operator

The crossover is a genetic operator used to generate new individuals (chromosomes) from one generation to the next. This is an analogous process to the biological reproduction. A crossover takes some characteristics from two or more parents and combine them to create a new individual.

Usually the selection function is used to select the individuals to cross, however we decided to uniformly at random choose two elements from the current generation

to cross them, even if they have a very different fitness value. The main reason to apply this criteria is that a new individual is alway improved through a local search, therefore if the new individual from the crossing is not so good, then it will be improved, resulting in an individual that, in fact, can be better than its parents. In addition, in our experimental evaluation we observe that this allow us to have a higher diversification among the generations.

We implemented three basic crossover techniques: partially mapped crossover, cycled crossover and order crossover. All this crossover techniques were implemented to work over permutations, recall, the vertices on each dimension are represented as a permutation.

3.5.3.1 Partially mapped crossover

The partially mapped crossover (PMX) was introduced by [Goldberg and Lingle, 1985] and was proposed for the TSP. The PMX consists on passing an ordered subsegment tour from the parents to the offspring. A string portion from one parent is mapped onto a string portion from the other parent and the remaining information is exchanged. If the resulting offspring is invalid it should be repaired.

Algorithm 20 shows the PMX procedure. Initially we select two points for a crossing segment and each individual of the offspring is copied from its corresponding parents. Then, the segment indicated by the crossing points is swapped between the two elements of the offspring. If required, the repairing process consists on changing the duplicated values (out of the exchanged segment) of each permutation by its corresponding mapped value from the other permutation. If the corresponding mapped value is already present in the current permutation then the mapped value should be evaluated with its corresponding mapped value from the other permutation and so on in a cyclic way until a non present value in the current permutation is reached.

Suppose we have the permutations $ind1 = (2, 3, 5, 1, 4)$ and $ind2 = (3, 5, 1, 4, 2)$ and the crossing points $p1 = 1$ and $p2 = 3$. Then the corresponding offspring are initialized as $o1 = (2, 3, 5, 1, 4)$ and $o2 = (3, 5, 1, 4, 2)$. The crossing step is as follows:

$$\left[\begin{array}{l} o1 : (2, 3, 5, 1, 4) \rightarrow (2, \mathbf{5}, \mathbf{1}, \mathbf{4}, 4) \\ o2 : (3, 5, 1, 4, 2) \rightarrow (3, \mathbf{3}, \mathbf{5}, \mathbf{1}, 2) \end{array} \right].$$

We can observe that $o1$ has the second 4 repeated, then the cyclic process is $4^1 \rightarrow 1^2$, $1^1 \rightarrow 5^2$, $5^1 \rightarrow 3^2$, then 3 is the replacing value for the repeated 4. In $o2$ the first 3 is repeated, then the cyclic process is $3^2 \rightarrow 5^1$, $5^2 \rightarrow 1^1$, $1^2 \rightarrow 4^1$, then 4 is the replacing value for the repeated 3.

$$\left[\begin{array}{l} o1 : (2, 5, 1, 4, 4) \rightarrow (2, 5, 1, 4, \mathbf{3}) \\ o2 : (3, 3, 5, 1, 2) \rightarrow (\mathbf{4}, 3, 5, 1, 2) \end{array} \right].$$

Let n be the size of each permutation (which is in fact the number of vertices by dimension) and s the number of permutations (dimensions) to repair, the swapping

Algorithm 20: The partially mapped crossover.

Input: $ind1, ind2$. The parents to cross.
Result: $o1 \cup o2$. The two new individuals after the crossing.

```
1  Set p1 := random() % father1.assignment.size;
2  Set p2 := random() % father2.assignment.size;
3  Set p1 := min(p1, p2), p2 := max(p1, p2);
4  Set o1 := ind1;
5  Set o2 := ind2;
6  foreach index in p1:p2 do
7      swap(o1.assignment[index], o2.assignment[index]);
   // Repair o1
8  foreach dim in 1:s do
9      Set already := An array of n elements initialized with zeros;
10     foreach index in p1:p2 do
11         Set already[ o1.matching[index][dim] ] := index;
12     foreach index in 0 : (n − 1) do
13         if index < p1 or index > p2 then
14             while (next = already[o1.matching[index][dim]]) ≠ 0 do
15                 Set o1.matching[index][dim] = o2.matching[next][dim];
16 Repair o2 analogous to o1;
17 return {o1 ∪ o2};
```

process can be performed in $O(n)$ time whereas each repairing process can take $O(n)$ time and can be required $O(n)$ repairing processes. Then, the time complexity of this operator is $O(sn^2)$.

3.5.3.2 Cycled crossover

The cycled crossover (CX) is a procedure that consists on swapping the elements of some cyclic permutation between the elements of two vectors. It consists on choosing a random index and, for each permutation, it swaps all the elements from the two parents that belong to the cyclic permutation that contains the element pointed by the chosen index.

Algorithm 21 shows the CX procedure. Initially, we choose a uniformly random index $p1$ in the interval $[0, n)$ and each individual of the offspring is copied from its corresponding parents. Then, the corresponding cyclic permutation at $p1$ is detected and we move among all its elements. The two elements from each permutation vector along the cyclic permutation are swapped.

Suppose we have the permutations $ind1 = (2, 3, 1, 5, 4)$ and $ind2 = (3, 1, 2, 4, 5)$

Algorithm 21: The cycled crossover.

Input: $ind1, ind2$. The parents to cross.
Result: $o1 \cup o2$. The two new individuals after the crossing.

```
1  Set p1 := random() % ind1.assignment.size;
2  Set o1 := ind1;
3  Set o2 := ind2;
4  Let s be the number of dimensions of the assignment;
5  Let n be the number of vertices by dimension;
6  foreach dim in 1:s do
7      Set indexes := An array of n elements;
8      foreach index in 1:n do
9          Set indexes[ o1.matching[i][dim] ] := index;
10     Set start_vertex := o1.matching[point][dim];
11     Set index := point;
12     while start_vertex ≠ o2.matching[index][dim] do
13         Set next_point := o2.matching[index][dim];
14         swap(o1.assignment[index][dim], o2.assignment[index][dim]);
15         index := indexes[ next_point ];
16     swap(o1.assignment[index][dim], o2.assignment[index][dim]);
17 return {o1 ∪ o2};
```

and the chosen index $p1 = 1$. The corresponding offspring are $o1 = (2, 3, 1, 5, 4)$ and $o2 = (3, 1, 2, 4, 5)$. The cyclic permutation is as follows:

$$\begin{bmatrix} o1 : (2,3,1,5,4) \to (2,\mathbf{1},1,5,4) \to (2,1,\mathbf{2},5,4) \to (\mathbf{3},1,2,5,4) \\ o2 : (3,1,2,4,5) \to (3,\mathbf{3},2,4,5) \to (3,3,\mathbf{1},4,5) \to (\mathbf{2},3,1,4,5) \end{bmatrix}.$$

Since there are s permutations of size n this process takes $O(sn)$ time. The advantage of this procedure is that no repairing is required. The disadvantage is that if the cyclic permutation includes all the elements of the permutation then no new individuals are generated.

3.5.3.3 Order crossover

The order crossover was also introduced by [Goldberg and Lingle, 1985] for the TSP. It is similar to the PMX crossover but a different repairing process. The repairing is performed by removing the duplicated values and replacing them by the missing values but in the same order as they appear in the opposite parent.

Algorithm 22 shows the OX procedure. After the swapping process, as in PMX, the repairing process is performed. All the duplicated elements out of the exchanged segment are replaced with the missing elements of the corresponding permutation according with their position in the opposite parent, going from the left to right.

Algorithm 22: The ordered crossover.

Input: $ind1, ind2$. The parents to cross.
Result: $o1 \cup o2$. The two new individuals after the crossing.

```
1  Set p1 := random() % father1.assignment.size;
2  Set p2 := random() % father2.assignment.size;
3  Set p1 := min(p1, p2), p2 := max(p1, p2);
4  Set o1 := ind1;
5  Set o2 := ind2;
6  foreach index in p1:p2 do
7      swap(o1.assignment[index], o2.assignment[index]);
   // Repair o1
8  foreach dim in 1:s do
9      Set flag := An array of n booleans;
10     Set vertices_order := An array of n integers;
11     foreach index in 0 : (n − 1) do
           // Set true if p1 ≤ i and i ≤ p2, otherwise set false
12         Set flag[ o1.matching[index][dim] ] := (p1 ≤ i and i ≤ p2);
13         Set vertices_order[index] := ind2.matching[index][dim];
14     Set next := 0;
15     foreach index in 0 : (n − 1) do
16         if index < p1 or index > p2 then
17             while next < n and flag[vertices_order[next]] = True do
18                 Set next := next + 1;
19             Set o1.matching[index][dim] = vertices_order[next];
20             Set next := next + 1;
21 Repair o2 analogous to o1;
22 return {o1 ∪ o2};
```

Suppose we have the permutations $ind1 = (5, 3, 1, 2, 4)$ and $ind2 = (3, 1, 2, 4, 5)$ and the crossing points $p1 = 2$ and $p2 = 4$. The corresponding offspring are $o1 = (5, 3, 1, 2, 4)$ and $o2 = (3, 1, 2, 4, 5)$. The crossing step is as follows:

$$\begin{bmatrix} o1 : (5, 3, 1, 2, 4) \to (2, \mathbf{1}, \mathbf{2}, \mathbf{4}, 4) \\ o2 : (3, 1, 2, 4, 5) \to (3, \mathbf{3}, \mathbf{1}, \mathbf{2}, 5) \end{bmatrix}.$$

Then the repairing process is as follows:

$$\begin{bmatrix} o1 : (2, 1, 2, 5, 4) \to (*, 1, 2, 4, *) \to (\mathbf{3}, 1, 2, 4, \mathbf{5}) \\ o2 : (3, 3, 1, 4, 5) \to (*, 3, 1, 4, 5) \to (\mathbf{2}, 3, 1, 4, 5) \end{bmatrix}.$$

Since there are s permutations of size n this process takes $O(sn)$ time. The advantage of this procedure is that the repairing process is faster than in PMX. The disadvantage is that the general structure of new individuals can be very different.

3.5.4 The mutation operator

Mutation is a genetic operator used to maintain the diversification from one generation to the next one. In a mutation process the original solution can be changed completely. The mutation process should occur with a low probability over the individuals of a population.

We implemented two different mutations that are followed by an improvement through a local search. In this way, even when the mutated individual can be very different to its original version, the local search will be able to, by at least, increase the quality of such individual. The implemented mutations are the analogous procedures to some of our basic local search heuristics.

3.5.4.1 The swapping mutation

The swapping mutation SM performs slight changes on a feasible solution. This mutation is similar to the simple 2-opt heuristic. This operator consists on choosing two arbitrary indexes from each permutation and swapping them.

Algorithm 23 shows the SM procedure. For each dimension, two random indices, $p1$ and $p2$ are swapped. We decided to swap different points from each dimension because if we swap the same vertices at all the dimensions it only will change the vectors of place without representing any change in the current solution. By swapping a different pair of vertices at each dimension we will make changes in up to $2s$ vectors.

Algorithm 23: The swapping mutation.

Input: *ind*. The individual to mutate.
Result: *ind*. The individual mutated.

```
1 foreach dim in 1 : s do
2     Set p1 := random() % ind.assignment.size;
3     Set p2 := random() % ind.assignment.size;
4     swap(ind.assignment[p1][dim], ind.assignment[p2][dim]);
5 return {ind};
```

There are s changes performed in $O(1)$ time each, so the time complexity of this procedure is $O(s)$.

3.5.4.2 The inversion mutation

The inversion mutation IM performs significant changes on a feasible solution. This mutation is similar to the inversion heuristic. This operator consists on choosing two arbitrary indexes from each permutation and reversing all the inclusive interval of vertices

Algorithm 24 shows the IM procedure. For each dimension, two random indices, $p1$ and $p2$, are selected and the vertices between the given closed interval given by $[p1, p2]$ are reversed.

Algorithm 24: The inversion mutation.

Input: *ind.* The individual to mutate.
Result: *ind.* The individual mutated.

```
1 foreach dim in 1:s do
2     Set p1 := random() % ind.assignment.size;
3     Set p2 := random() % ind.assignment.size;
4     while p1 < p2 do
5         swap(ind.assignment[p1][dim], ind.assignment[p2][dim]);
6         Set p1 := p1 + 1;
7         Set p2 := p2 − 1;
8 return {individual};
```

There are s inversions performed in $O(n)$ time each, so the time complexity of this procedure is $O(sn)$.

3.5.5 Experimental evaluation

For our experimental evaluation we considered the same five families of instances, which include the family of instances provided by [Karapetyan and Gutin, 2011b] to evaluate their state-of-the art memetic algorithm. In the case of our memetic algorithm we solved the instances by considering all the possible combinations of selections, crossovers and mutations (for a total of $3 \times 3 \times 2 = 18$ combinations). We considered a population size of $N = 100$, a mutation probability $mp = 10\%$, a survivors probability $sp = 50\%$ and a running time of 30 seconds for each instance. We evaluated our SMA under two variants of DVH: SDV2 and DV2.

Table 3.21 shows the best known values for the set of families of instances considered in our experimental evaluation. We show the averaged results for ten instances of each type. In some cases we already know the optimal solutions, thanks to our MAP-Gurobi, in those cases we highlight with bold the corresponding average. This table can be used as reference to verify how close are the obtained results of our SMA from the best known solutions.

Tables 3.22, 3.23 and 3.24 show the results for the families of instances provided by [Karapetyan and Gutin, 2011b] plus our additional set of ten instances for $s = 4$ with $n = 50$ and for $s = 5$ with $n = 30$ for each family of instances. In this case, the RSE is higher in the case of the Random family which is due to the fact that the SDV2 and DV2 provide a lower quality solution error for this family of instances. An alternative for the Random family of instances can be to use the local searches SDV3

Table 3.21: Averaged best known solutions.

s	n	Random (r)	Clique (cq)	SquareRoot (sq)	Geometric (g)	Product (p)
4	20	**20.0**	**1901.8**	**929.3**	**2380.5**	**8397.3**
4	30	**30.0**	2281.9	1118.6	**3015.2**	**13154.1**
4	40	**40.0**	2606.3	1271.4	3523.8	16810.0
4	50	**50.0**	3120.5	1530.1	4102.3	20705.6
5	15	**15.0**	**3110.7**	**1203.9**	**3423.7**	**21422.8**
5	18	**18.0**	**3458.6**	**1343.9**	**3799.3**	23371.5
5	25	**25.0**	4192.7	1627.5	4594.9	30150.4
5	30	**30.0**	4677.9	1852.4	5036.8	34818.0
6	12	**12.0**	**4505.6**	**1436.8**	**4483.7**	**7421.0**
6	15	**15.0**	5133.4	1654.6	5242.4	8888.9
6	18	**18.0**	5765.5	1856.3	5767.4	9610.1
Avg	-	**24.8**	3705.0	1438.6	4124.5	17704.5

or DV3 as part of our SMA, however, these heuristics are able to find the optimal values for this family of instances, then we consider unnecessary to solve this set of instances with such combination. It is an example in which our heuristics SDV3 and DV3 are stronger than a meta-heuristic. In the case of the families of instances Clique and Square Root our SMA provides similar quality results to those reported by the state-of-the-art memetic algorithm. For the family of instances Clique the RSE of SMA combined with the SDV2, considering the elitist selection, the CX and the IM operators, is approximately 0.3% whereas for the state-of-the-art memetic algorithm is 0.1%. For the family of instances Clique the RSE of SMA combined with the SDV2, considering the elitist selection, the CX and the IM operators, is approximately 0.7% whereas for the state-of-the-art memetic algorithm is 0.2%. In summary, the obtained results of the SMA combined with the SDV3 are superior to those obtained with the only use of SDV3 and DV3 and are competitive against the state-of-the-art memetic algorithm. It is important to mention that the RSE of the state-of-the-art memetic algorithm were obtained after running times of 300 seconds whereas ours were obtained in 30 seconds.

Finally, we evaluated our proposed set of instances of the families Geometric and Product. Tables 3.25 and 3.26 show the obtained results. In the case of the Geometric family the SMA combined with either SDV2 or DV2 is able to find the optimal solutions in all the evaluated instances. In the case of the Product family the SMA combined with the DV2 got the best solutions, overall considering the roulette wheel selection, the PMX and the IM operators, obtaining a RSE of approximately 2.1%. For the Product family can be considered to combine the SMA with the SDV3 or the DV3, however the only use of such local searches provides a RSE of approximately 0.4%.

Table 3.22: Random under SMA combined with SDV2 and DV2.

Inst.	SMA combined with SDV2						SMA combined with DV2					
	PMX		CX		OX		PMX		CX		OX	
	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM
	Elitist						Elitist					
4r20	28.2	27.9	29.3	28.4	28.9	28.0	25.7	25.6	26.2	25.6	25.7	25.9
4r30	39.7	38.9	39.7	40.6	40.2	40.2	36.1	36.4	37.1	36.0	35.6	36.5
4r40	48.5	48.0	49.0	49.0	48.9	48.6	45.2	44.9	45.9	45.5	45.1	45.5
4r50	57.0	57.6	58.0	58.2	57.6	58.5	53.7	54.0	54.4	53.5	54.4	53.8
5r15	21.2	20.6	21.3	20.8	20.8	21.3	17.8	17.7	17.7	18.0	17.6	18.0
5r18	25.5	25.0	25.9	25.7	24.9	25.1	20.7	21.0	21.0	21.6	20.9	21.1
5r25	33.4	33.5	33.8	32.7	33.4	33.0	28.3	28.3	28.2	27.9	28.0	28.3
5r30	38.7	37.9	38.5	38.6	38.5	37.7	32.9	32.9	33.6	33.3	33.1	32.7
6r12	16.5	16.6	16.9	17.1	17.1	16.5	13.3	13.0	13.2	13.3	12.8	12.9
6r15	21.2	20.0	21.7	20.9	20.4	21.2	16.2	15.8	16.3	16.6	16.2	15.8
6r18	24.4	24.3	24.4	24.8	24.4	24.9	19.3	19.3	19.8	19.1	19.1	19.5
Avg	32.2	31.8	32.6	32.4	32.3	32.3	28.1	28.1	28.5	28.2	28.0	28.2
RSE	29.8	28.2	31.5	30.6	30.2	30.2	13.3	13.3	14.9	13.7	12.9	13.7
	Tournament						Tournament					
4r20	28.5	28.7	29.3	28.6	29.3	29.8	25.4	24.9	26.1	25.6	26.0	25.9
4r30	40.1	39.2	40.9	39.8	40.2	40.3	36.2	36.3	36.3	36.4	36.3	36.3
4r40	49.5	48.8	49.4	49.6	49.3	49.1	45.4	44.7	45.3	45.4	45.2	45.6
4r50	57.3	57.6	57.9	56.9	58.0	57.7	53.9	53.8	54.4	54.6	53.8	54.4
5r15	20.8	21.4	21.7	21.2	21.3	20.6	17.7	17.6	17.8	17.7	17.6	17.2
5r18	25.3	25.0	25.8	26.0	25.6	25.7	20.9	20.9	20.7	20.8	21.1	21.2
5r25	33.4	33.4	32.8	33.5	32.9	33.1	28.5	28.5	28.4	28.5	27.6	28.1
5r30	37.9	38.0	38.0	37.7	38.4	37.7	32.8	32.8	33.5	33.0	33.1	33.3
6r12	16.8	16.0	16.8	16.8	17.0	16.5	13.0	13.0	12.6	12.9	12.9	13.1
6r15	20.8	20.8	20.9	20.8	21.2	20.5	16.3	16.2	16.1	16.0	16.7	16.4
6r18	23.9	24.1	24.3	24.8	24.4	25.1	19.4	19.0	19.1	19.5	19.3	19.4
Avg	32.2	32.1	32.5	32.3	32.5	32.4	28.1	28.0	28.2	28.2	28.1	28.3
RSE	29.8	29.4	31.0	30.2	31.0	30.6	13.3	12.9	13.7	13.7	13.3	14.1
	Roulette wheel						Roulette wheel					
4r20	29.1	28.4	29.4	28.5	29.2	29.5	26.1	25.9	26.7	26.4	25.4	25.6
4r30	39.8	39.9	40.0	40.6	40.3	39.9	35.9	35.8	36.6	36.6	36.7	36.9
4r40	49.2	49.4	50.0	49.4	49.6	49.7	45.3	45.3	45.8	45.6	44.8	45.2
4r50	57.5	57.2	58.6	58.3	58.0	57.4	54.0	53.9	54.3	54.5	54.3	54.0
5r15	21.7	21.3	20.6	21.5	21.8	22.1	17.7	17.5	18.0	18.1	17.8	17.5
5r18	25.1	25.3	25.4	25.8	25.5	25.4	21.2	21.0	21.0	21.1	21.1	21.5
5r25	33.6	32.9	33.3	34.0	34.1	33.5	28.0	28.1	28.4	28.5	28.3	28.6
5r30	38.5	38.7	39.7	38.5	38.4	38.7	33.0	33.0	33.4	33.1	33.2	32.8
6r12	16.2	16.4	16.7	17.4	16.3	16.9	13.6	12.7	13.1	12.8	13.1	13.1
6r15	19.9	21.0	21.2	20.9	20.9	20.9	16.3	16.7	16.3	16.2	16.5	16.4
6r18	25.1	24.1	25.2	24.7	23.5	23.9	19.5	19.0	19.6	19.5	19.5	19.6
Avg	32.3	32.2	32.7	32.7	32.5	32.5	28.2	28.1	28.5	28.4	28.2	28.3
RSE	30.2	29.8	31.9	31.9	31.0	31.0	13.7	13.3	14.9	14.5	13.7	14.1

Table 3.23: Clique under SMA combined with SDV2 and DV2.

Inst.	SMA combined with SDV2						SMA combined with DV2					
	PMX		CX		OX		PMX		CX		OX	
	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM
	Elitist						Elitist					
4cq20	1901.9	1901.9	1901.9	1901.8	1901.8	1902.2	1901.9	1901.8	1902.2	1901.8	1901.8	1902.2
4cq30	2294.2	2293.6	2291.3	2293.1	2315.6	2312.4	2291.7	2288.3	2291.1	2293.1	2317.3	2312.7
4cq40	2679.0	2674.6	2647.0	2639.3	2752.4	2752.0	2692.1	2684.7	2657.0	2654.3	2761.8	2766.1
4cq50	3196.7	3178.4	3178.4	3159.0	3283.9	3270.5	3259.1	3235.2	3233.1	3230.3	3294.9	3323.6
5cq15	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7
5cq18	3459.4	3458.6	3459.4	3458.6	3458.6	3460.1	3459.4	3459.4	3458.6	3459.3	3459.4	3459.4
5cq25	4204.3	4204.8	4205.2	4201.0	4247.8	4226.4	4216.5	4207.3	4206.2	4205.1	4259.5	4260.6
5cq30	4712.3	4713.5	4701.8	4699.3	4839.7	4803.7	4744.0	4725.4	4729.2	4704.8	4827.8	4816.0
6cq12	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4507.1	4506.8	4505.6	4505.6	4505.6
6cq15	5134.4	5134.4	5134.4	5133.4	5133.4	5133.4	5136.9	5133.7	5136.4	5139.3	5139.3	5145.3
6cq18	5775.4	5775.7	5785.2	5777.2	5779.3	5772.8	5794.1	5795.4	5795.1	5809.4	5822.6	5832.0
Avg	3724.9	3722.9	3720.1	3716.3	3757.2	3750.0	3737.5	3731.7	3729.7	3728.5	3763.7	3766.7
RSE	0.5	0.5	0.4	0.3	1.4	1.2	0.9	0.7	0.7	0.6	1.6	1.7
	Tournament						Tournament					
4cq20	1901.8	1901.8	1901.9	1901.9	1902.1	1902.1	1901.9	1902.2	1902.7	1902.2	1902.4	1902.5
4cq30	2297.8	2292.3	2295.2	2294.3	2335.7	2320.2	2291.9	2293.7	2298.3	2299.5	2340.0	2328.8
4cq40	2695.7	2696.4	2675.7	2652.8	2757.9	2768.8	2701.3	2701.5	2678.0	2661.1	2776.3	2767.4
4cq50	3216.4	3219.2	3193.5	3175.7	3305.2	3290.8	3249.3	3254.9	3234.5	3228.3	3305.1	3327.7
5cq15	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7
5cq18	3460.1	3460.1	3460.1	3460.1	3460.1	3462.2	3460.1	3459.4	3460.1	3459.3	3462.6	3459.4
5cq25	4212.1	4203.9	4205.5	4205.5	4267.1	4250.7	4219.6	4218.1	4213.9	4212.6	4263.5	4282.5
5cq30	4767.0	4732.3	4712.0	4700.0	4839.2	4848.3	4782.5	4770.5	4717.1	4726.4	4853.3	4894.0
6cq12	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6
6cq15	5133.4	5134.4	5134.4	5134.4	5134.4	5135.6	5138.3	5137.4	5134.6	5139.0	5147.0	5139.9
6cq18	5776.4	5775.3	5779.3	5779.9	5789.1	5789.1	5808.4	5811.1	5799.5	5800.4	5841.4	5828.5
Avg	3734.3	3730.2	3724.9	3720.1	3764.3	3762.2	3742.7	3742.3	3732.3	3731.4	3773.4	3777.0
RSE	0.8	0.7	0.5	0.4	1.6	1.5	1.0	1.0	0.7	0.7	1.8	1.9
	Roulette wheel						Roulette wheel					
4cq20	1901.8	1903.4	1903.1	1902.3	1903.8	1902.4	1902.7	1901.8	1903.1	1901.8	1902.4	1903.0
4cq30	2301.5	2296.4	2293.0	2295.6	2315.1	2320.6	2301.6	2300.0	2298.3	2291.8	2310.9	2323.9
4cq40	2701.8	2678.6	2666.0	2673.4	2751.5	2754.3	2680.6	2683.4	2675.9	2672.8	2760.2	2749.4
4cq50	3208.6	3194.2	3166.0	3186.5	3277.7	3273.7	3225.1	3258.8	3220.7	3207.9	3275.6	3277.8
5cq15	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7	3110.7
5cq18	3460.6	3458.6	3459.3	3459.4	3459.4	3459.4	3460.1	3460.6	3460.1	3460.1	3459.4	3458.9
5cq25	4223.8	4218.0	4218.3	4208.4	4242.8	4227.2	4229.5	4229.6	4216.2	4217.3	4274.6	4271.6
5cq30	4749.0	4714.1	4726.9	4713.1	4779.6	4788.8	4770.9	4762.4	4744.2	4724.1	4801.1	4855.4
6cq12	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4505.6	4506.8	4505.6	4505.6	4505.6
6cq15	5133.4	5135.6	5133.4	5134.4	5133.6	5135.1	5144.8	5139.1	5138.5	5136.6	5145.9	5148.1
6cq18	5780.3	5783.5	5782.4	5776.0	5782.3	5797.5	5811.3	5816.0	5798.8	5804.7	5823.9	5826.2
Avg	3734.3	3727.2	3724.1	3724.1	3751.1	3752.3	3740.3	3742.5	3733.9	3730.3	3760.9	3766.4
RSE	0.8	0.6	0.5	0.5	1.2	1.3	1.0	1.0	0.8	0.7	1.5	1.7

Table 3.24: Squareroot under SMA combined with SDV2 and DV2.

Inst.	SMA combined with SDV2						SMA combined with DV2					
	PMX		CX		OX		PMX		CX		OX	
	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM
	Elitist						Elitist					
4sq20	929.4	931.1	929.7	929.7	929.4	929.3	930.2	930.5	929.7	929.4	930.1	931.0
4sq30	1122.3	1122.3	1119.7	1126.1	1146.7	1138.0	1123.1	1121.6	1126.9	1122.3	1148.3	1144.5
4sq40	1335.4	1313.0	1311.9	1295.9	1362.1	1366.3	1336.0	1322.1	1306.0	1304.8	1366.8	1366.0
4sq50	1597.4	1592.2	1577.1	1567.4	1637.8	1636.0	1630.8	1621.9	1611.7	1607.2	1645.7	1648.3
5sq15	1203.9	1205.0	1206.8	1205.3	1206.5	1204.9	1205.0	1204.7	1206.8	1205.3	1204.9	1204.8
5sq18	1345.2	1346.9	1346.7	1349.3	1345.5	1345.1	1348.0	1345.4	1347.3	1347.8	1345.9	1349.1
5sq25	1642.0	1645.2	1637.0	1647.1	1670.0	1669.9	1672.5	1667.8	1658.2	1643.6	1694.0	1699.8
5sq30	1896.3	1881.8	1871.2	1863.0	1941.3	1934.3	1934.5	1939.2	1927.2	1928.4	1962.3	1956.7
6sq12	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8
6sq15	1657.3	1656.2	1658.3	1658.0	1658.8	1657.8	1658.9	1658.1	1658.7	1660.0	1660.7	1661.9
6sq18	1859.1	1858.4	1858.7	1860.4	1861.4	1858.6	1867.5	1864.5	1867.2	1866.5	1888.2	1879.6
Avg	1456.8	1453.5	1450.4	1449.0	1472.4	1470.6	1467.6	1464.8	1461.5	1459.3	1480.3	1479.9
RSE	1.3	1.0	0.8	0.7	2.3	2.2	2.0	1.8	1.6	1.4	2.9	2.9
	Tournament						Tournament					
4sq20	930.6	930.8	930.1	930.6	931.2	931.3	929.6	929.3	930.0	929.7	931.8	931.8
4sq30	1125.6	1123.8	1124.5	1122.7	1157.5	1149.2	1132.2	1129.3	1129.3	1125.1	1151.8	1151.1
4sq40	1344.6	1332.6	1324.9	1303.3	1381.5	1385.0	1336.7	1333.7	1328.5	1322.1	1376.9	1380.7
4sq50	1611.3	1606.3	1592.6	1570.2	1652.3	1664.8	1622.6	1629.5	1610.4	1605.6	1657.0	1652.2
5sq15	1205.5	1205.3	1205.1	1205.5	1205.1	1205.5	1205.7	1203.9	1204.4	1205.5	1207.3	1205.1
5sq18	1344.8	1347.0	1347.8	1348.5	1347.9	1346.4	1347.1	1344.7	1351.7	1348.5	1346.5	1349.1
5sq25	1644.5	1644.3	1639.4	1648.8	1685.7	1672.7	1672.1	1667.8	1666.7	1658.8	1711.9	1697.6
5sq30	1897.4	1896.3	1881.5	1867.2	1951.2	1941.9	1946.3	1934.8	1944.6	1934.2	1960.8	1961.4
6sq12	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1437.7	1436.8	1436.8
6sq15	1656.4	1657.5	1657.8	1656.9	1659.0	1658.7	1658.9	1657.9	1660.8	1660.9	1666.8	1661.3
6sq18	1859.7	1860.3	1860.7	1858.1	1867.3	1865.9	1871.7	1873.9	1865.9	1866.2	1888.1	1886.2
Avg	1459.7	1458.3	1454.7	1449.9	1479.6	1478.0	1469.1	1467.4	1466.3	1463.1	1485.1	1483.0
RSE	1.5	1.4	1.1	0.8	2.8	2.7	2.1	2.0	1.9	1.7	3.2	3.1
	Roulette wheel						Roulette wheel					
4sq20	929.8	930.7	930.1	930.5	933.3	931.2	931	931.1	930.8	931.5	930.9	932.1
4sq30	1133.6	1127.2	1130.7	1129.4	1160.9	1144.5	1138.0	1133.4	1134.3	1126.7	1159.3	1152.2
4sq40	1344.5	1339.3	1332.9	1320.3	1373.8	1374.0	1337.8	1334.6	1332.2	1328.4	1373.1	1371.3
4sq50	1616.9	1606.9	1589.8	1579.7	1639.7	1632.3	1635.2	1630.1	1606.7	1605.1	1647.6	1647.4
5sq15	1204.2	1204.8	1204.7	1205.2	1204.7	1205.1	1205.9	1205.5	1204.7	1204.4	1205.6	1204.7
5sq18	1346.4	1347.4	1346.6	1350.5	1348.2	1347.3	1348.7	1346.8	1347.0	1348.8	1351.8	1350.0
5sq25	1644.4	1644.7	1647.4	1654.5	1660.8	1680.8	1683.2	1667.6	1672.8	1669.1	1699.6	1687.2
5sq30	1914.3	1917.4	1894.9	1888.0	1947.1	1947.4	1942.5	1930.1	1927.7	1928.5	1966.4	1972.8
6sq12	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1436.8	1437.5	1436.8	1436.9	1436.8
6sq15	1658.6	1658.2	1657.9	1656.5	1659.4	1656.3	1664.1	1658.4	1663.4	1661.7	1665.1	1663.4
6sq18	1859.3	1859.9	1860.7	1859.3	1870.0	1864.1	1872.6	1870.8	1872.2	1868.4	1889.3	1890.4
Avg	1462.6	1461.2	1457.5	1455.5	1475.9	1474.5	1472.3	1467.7	1466.3	1464.5	1484.1	1482.6
RSE	1.7	1.6	1.3	1.2	2.6	2.5	2.3	2.0	1.9	1.8	3.2	3.1

Table 3.25: Geometric under SMA combined with SDV2 and DV2.

Inst.	SMA combined with SDV2						SMA combined with DV2					
	PMX		CX		OX		PMX		CX		OX	
	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM
	Elitist						Elitist					
4g20	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5
4g30	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2
4g40	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8
4g50	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3
5g15	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7
5g18	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3
5g25	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9
5g30	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8
6g12	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7
6g15	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4
6g18	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4
Avg	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5
RSE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Tournament						Tournament					
4g20	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5
4g30	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2
4g40	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8
4g50	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3
5g15	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7
5g18	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3
5g25	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9
5g30	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8
6g12	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7
6g15	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4
6g18	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4
Avg	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5
RSE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Roulette wheel						Roulette wheel					
4g20	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5
4g30	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2	3015.2
4g40	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.8	3523.9
4g50	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3	4102.3
5g15	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7	3423.7
5g18	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3	3799.3
5g25	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9	4594.9
5g30	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8	5036.8
6g12	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7	4483.7
6g15	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4	5242.4
6g18	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4	5767.4
Avg	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.5	4124.6
RSE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 3.26: Product under SMA combined with SDV2 and DV2.

Inst.	SMA combined with SDV2						SMA combined with DV2					
	PMX		CX		OX		PMX		CX		OX	
	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM	SM	IM
	Elitist						Elitist					
4p20	8397.5	8397.5	8397.8	8397.7	8397.7	8397.6	8397.4	8397.4	8397.4	8397.4	8397.4	8397.3
4p30	13154.3	13154.3	13154.6	13154.6	13154.5	13154.5	13154.2	13154.2	13154.2	13154.2	13154.2	13154.2
4p40	16811.0	16811.1	16811.1	16811.1	16811.1	16811.1	16810.2	16810.2	16810.3	16810.3	16810.3	16810.2
4p50	20709.5	20709.4	20709.8	20710.4	20709.7	20709.5	20706.6	20706.6	20706.7	20706.5	20706.6	20706.6
5p15	21592.3	21427.3	21744.5	21580.4	21429	21710.9	21423.2	21423	21423.5	21423.3	21423.1	21423.1
5p18	23589.2	23493.7	24500.7	24274.8	23640.5	23658.9	23371.7	23371.5	23513.9	23372.2	23531.6	23513.9
5p25	30543.6	30916.7	33519.6	32275.5	31990.2	31456.2	31001.6	31367.2	31576.8	31017.6	31016.6	30859.8
5p30	38024.6	37337.5	42653.9	41361.3	40490.0	41129.4	37376.7	37485.5	37600.1	37089.4	37524.7	37564.7
6p12	11570.5	11221.3	14602.2	13797.3	12916.3	12817.3	8048.1	7821.4	8254.7	8047.1	8214.1	7970.8
6p15	17356.6	17263.8	21627.3	21183.2	20966.1	18603.5	9917.4	9774.7	9853.3	10154.4	9996.5	9941.8
6p18	23289.9	22307.4	27821.6	27188.7	26449.5	27009.9	11236.2	11217.3	11291	11495.1	11347.8	11269.8
Avg	20458.1	20276.4	22322.1	21885.0	21541.3	21405.3	18313.0	18320.8	18416.5	18333.4	18374.8	18328.4
RSE	15.6	14.5	26.1	23.6	21.7	20.9	3.4	3.5	4	3.6	3.8	3.5
	Tournament						Tournament					
4p20	8397.7	8397.8	8397.8	8398	8397.6	8397.9	8397.4	8397.4	8397.4	8397.4	8397.4	8397.4
4p30	13154.6	13154.5	13154.8	13154.5	13154.6	13154.5	13154.2	13154.2	13154.2	13154.1	13154.2	13154.2
4p40	16811.2	16811.0	16811.5	16811.1	16811.0	16811.1	16810.2	16810.3	16810.2	16810.2	16810.3	16810.2
4p50	20709.4	20709.5	20710.0	20709.8	20709.8	20709.3	20706.5	20706.6	20706.7	20706.7	20706.7	20706.5
5p15	21428.8	21696.6	21883.0	21699.0	21745.0	21714.2	21423	21423	21423.3	21423.2	21423.1	21423.0
5p18	23448.5	23654.7	24596.5	25071.9	24307.6	24429.9	23945.3	23942.9	23389.3	23513.2	23372.1	23513.7
5p25	32824.2	30920.0	35888.0	35183.3	33184.5	32460.9	31383.0	31016.7	31383.5	31449.4	31813.5	31017.8
5p30	40104.0	39074.8	43442.8	43648.6	42383.1	42450.1	36704.8	37272.8	36936.4	37069.0	37620.4	38572.3
6p12	13248.3	12212.4	15569.1	15116.3	15430.0	14218.1	8093.5	8161.2	8273.1	8410.2	8095.8	8182.5
6p15	19463.9	19229.0	23572.0	22360.3	21930.2	20937.6	10183.1	10055.4	10033.6	10069.4	10308.8	10024.5
6p18	25515.2	24496.8	29148.5	28334.8	28623.4	27654.8	11536.2	10995.2	11451.7	11234.2	11484.7	11142.6
Avg	21373.3	20941.6	23015.8	22771.6	22425.2	22085.3	18394.3	18357.8	18359.9	18385.2	18471.5	18449.5
RSE	20.7	18.3	30	28.6	26.7	24.7	3.9	3.7	3.7	3.8	4.3	4.2
	Roulette wheel						Roulette wheel					
4p20	8397.8	8397.9	8397.9	8397.8	8397.9	8397.9	8397.4	8397.4	8397.4	8397.4	8397.4	8397.4
4p30	13154.7	13154.7	13154.6	13154.8	13154.6	13154.5	13154.2	13154.2	13154.2	13154.2	13154.2	13154.2
4p40	16810.8	16810.9	16811.5	16811.5	16811.1	16811.2	16810.3	16810.1	16810.3	16810.3	16810.3	16810.3
4p50	20710.0	20709.6	20710.2	20710.2	20709.7	20709.6	20706.6	20706.5	20706.6	20706.7	20706.7	20706.5
5p15	21424.0	21543.9	21741.3	21740.8	21860.9	21860.9	21422.8	21422.8	21423.0	21423.0	21423.1	21422.9
5p18	23492.4	23510.2	24389.6	23915.6	23602.3	24081.9	23371.5	23371.6	23512.7	23371.6	23942.8	23371.8
5p25	30859.5	31530.6	32609.0	32860.1	32205.1	31568.9	30346.2	30630.4	30779.3	31878.0	30888.8	30532.6
5p30	38479.1	38223.4	40895.7	41565.1	40233.0	39435.9	36811.9	36273.4	36816.1	37024.5	37242.6	37799.2
6p12	10379.4	11092.0	11504.3	12163.2	11936.7	11046.8	7750.3	7856.7	8160.9	7953.3	8125.2	7857.8
6p15	15423.0	13915.9	16535.4	17123.8	15788.5	16233.2	9841.1	9581.2	9932.6	9555.6	9850.9	9700.2
6p18	18579.6	16463.7	20411.1	18797.7	19929.2	18233.0	11304.9	10640.7	10833.1	11084.3	11428.1	10979.9
Avg	19791.8	19577.5	20651.0	20658.2	20420.8	20139.4	18174.3	18076.8	18229.7	18305.4	18360.9	18248.4
RSE	11.8	10.6	16.6	16.7	15.3	13.8	2.7	2.1	3.0	3.4	3.7	3.1

In conclusion, we consider that the most difficult families of instances are Clique and Square Root for which the use of SMA combined with either SDV3 or DV3 can be considered. For the rest of the cases the only use of SDV3 or DV3 provide high quality solutions. We want to highlight that higher running times of our SMA even can result in better results. We let all those analysis as future work since for our purposes we have a strong set of algorithms, heuristics and meta-heuristics to solve a wide variety of instances of MAP.

Chapter 4

Personnel assignment problems: the school timetabling problem as a case of study

In different problem contexts it is required to assign people to objects, such as employees to jobs, employees to offices, professors to courses, job seekers to vacant positions, etc. Each assignment has a value and, depending on the perspective, we wish either to minimize the total value, in which case the value is a cost, or maximize it, in which case the value is a benefit. The correct assignment of people will increase the productivity of the involved process.

The set of problems in which is required the assignment of people to resources are known as *personnel assignment problems* (PAP).

Consider the case when a university has n_1 professors to fill n_2 courses. Based on the aptitude and experience of each professor, such as previously taught courses, researching line, approval rating, number of times that such a class was taught, among others. In addition, can it be required to assign the professor into a classroom between a set of n_3 classrooms. Some aspects as the equipment of the classroom, if the classroom have computers, number of blackboards, etc, could be considered to perform the assignment. Finally, time restrictions of the professor or for the required time to teach the class can be considered from a set of n_4 time slots. The objective is to identify an assignment of professors to courses to classrooms to time slots that minimizes the total cost overall possible assignments. This problem is named *school timetabling problem* (STC).

Each problem of assignment of personnel has its own restrictions and considerations and can it be a difficult task to identify them and, even more, to weight them.

We will tackle the school timetabling problem as a case of study of personnel assignment problems, however the same considerations and restrictions can be applied to other problems.

4.1 State of the art

There are many works that deal with different personnel assignment problems, however, just a few modeled them as multidimensional assignment problems. Here we decided to focused our study of the state-of-the-art in the personnel assignment problem related to the school timetabling problem.

Early in the sixties, [Lewis, 1961] introduced the school timetabling problem as a problem in which some pupils required to be assigned into classes, commonly between 4 and 6 classes in total. In these formulation the teachers and classrooms were relevant for the assignment.

In the late sixties, [Wolfenden and Johntson, 1969] presented more requirements of a program for timetabling. They form a list in which each entry specifies a list of items which must be available simultaneously for certain number of time slots. An item can be a professor, a class, a classroom and, a time slot.

In the same years, [Lawrie, 1969] described an integer linear programming model o a school timetabling but considering each combination of items or sets under bipartite graphs, then by combining the solutions get a set timetables. They solved their model by applying some branching procedures.

In the middle seventies, [Dempster et al., 1975] provided a brief description of the development of computer-based school timetabling systems in the UK. This is a good document for starting to evaluate between a set of different approaches to tackle the STP. Ideas like preassignments (some professor/class must occur on certain time slot), preference preassignments (some professor/class must occur in one of some preselected time slots), consecutive periods (some professor/class must occur within a certain section of time slots), setting requirements (a set of professors/classes must occur at the same time slots), special rooms (some professor/class must occur in s special room). We adopt some of this ideas in order to added them into our formulation.

In the middle eighties, [de Werra, 1985] provided some models with an emphasis on graph theoretical models. This work is another excellent reference for staring to model a STP in many different ways adopting the best one according to the problem restrictions. In their most relevant model, they described a model aimed to assign classes to teachers to time slots through a couple of bipartite graphs. Instead of solving a 3AP they join the bipartite graph of classes to teachers to the bipartite graph of teachers to time slots and solved through a flow network by adding a source vertex to the classes vertices and a sink vertex from the time slots vertices. Such formulation have been commonly used by many authors since the complexity of solving this problem is similar to solve two normal assignment problems.

In the late nineties, [Burke et al., 1997] provided a new analysis of the state of the art for the STP and provided ideas for larger size instances. They divided the constraints provided by other authors in hard constraints and soft constraints for the STP. A timetable which breaks a hard constraints is not a feasible solution, soft constraints can be violated but with an involved cost. They provided some basic ideas

for different types of heuristics to solve the STP such as genetic algorithms, memetic algorithms, simulated annealing, tabu search and constraint logic programming.

In the middle 2000's, [Cambazard et al., 2005] proposed an interactive constraint programming model in which can be either added or removed constraints. This type of techniques are called dynamic constraint satisfaction problem in which we have a sequence of static constraint satisfaction problems where each constraint satisfaction problem is the consequence of addition or retraction of a constraint in the preceding problem.

In 2007, [Abdullah et al., 2007b] proposed a randomized iterative improvement algorithm with composite neighborhood structures for the STP. A composite neighborhood structure subsumes two or more neighborhood structures. The advantage of such process is to move along neighborhoods that two structures cannot reach by itself. Given an initial randomly generated feasible solution and given a set of neighborhood structures, at each iteration of the process all the neighborhood structures are evaluated and if some provides a better solution it is accepted with some preestablished probability. This process is similar to a local search heuristic with the difference that the probability of acceptance of a solution makes the process stochastic. In the same year, [Abdullah et al., 2007a] also proposed a memetic algorithm for the STP which consisted on a basic genetic algorithm combined with their previously developed heuristic.

In 2008, [Cerdeira-Pena et al., 2008] proposed a memetic algorithm which consisted of a basic genetic algorithm combined with a $2-$opt based local search heuristic. They applied several selection operators, e. g. tournament and elitist, and a mutation operators that changes dynamically its probability of be applied when no better solutions are found. In the same year, [Jat and Yang, 2008] proposed other memetic algorithm which consisted of a basic genetic algorithm combined with two different local search heuristics, under the claim that such combination provided higher quality results. At the same time, [Lara et al., 2008] proposed other evolutionary algorithm but of a different type: a bee algorithm. A bee algorithm is a population based search algorithm that somehow measure the topological distance between solutions. This type of algorithm performs a local search combined with a random search and, by evaluating the solutions using a fitness function, determine a new population of bees. All these heuristics were tested on real life instances of their corresponding universities, claiming practical good results for the timetable generation.

In recent years, several genetic and memetic algorithms have been proposed for the STP, e. g. [Raghavjee and Pillay, 2009], [Qaurooni, 2011], [Budiono and Wong, 2011], [Doulaty et al., 2013], [Fonseca and Santos, 2013], but they only change in the type of hard and soft restrictions to consider and in the basic genetic operators as well as in the way to generate the initial population. However, no one of such procedures solve the timetabling problem modeled as a 4AP.

4.2 Modeling the school timetabling problem as a MAP

The school timetabling problem can be formulated as a 4−dimensional assignment problem (4AP). Given a set of n_1 professors, n_2 courses, n_3 classrooms and n_4 time slots and the corresponding 4-dimensional matrix of costs $C^{n_1 n_2 n_3 n_4}$ where the entry c_{ijkm} is the cost of assigning the professor p_i to the class c_j to the classroom r_k to the time slot t_m with $1 \leq i \leq n_1$, $1 \leq j \leq n_2$, $1 \leq k \leq n_3$, $1 \leq m \leq n_4$ the 0-1 integer linear programming formulation is:

$$\begin{aligned}
\min & \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{n_3}\sum_{m=1}^{n_4} c_{ijkm}b_{ijkm} \\
\text{subject to}: & \sum_{j=1}^{n_2}\sum_{k=1}^{n_3}\sum_{m=1}^{n_4} b_{ijkm} = 1 \text{ for } i \text{ with } 1 \leq i \leq n_1 \\
& \sum_{i=1}^{n_1}\sum_{k=1}^{n_3}\sum_{m=1}^{n_4} b_{ijkm} = 1 \text{ for } j \text{ with } 1 \leq j \leq n_2 \\
& \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{m=1}^{n_4} b_{ijkm} = 1 \text{ for } k \text{ with } 1 \leq k \leq n_3 \\
& \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{n_3} b_{ijkm} = 1 \text{ for } m \text{ with } 1 \leq m \leq n_4
\end{aligned} \tag{4.1}$$

where $b_{ijkm} \in \{0, 1\}$ for all $1 \leq i \leq n_1, 1 \leq j \leq n_2, 1 \leq k \leq n_3, 1 \leq m \leq n_4$.

The first set of restrictions establishes that a professor will be assigned to one class into a classroom at some time slot. The second set of restrictions establishes that each class will be taught by one professor into a classroom at some time slot. The third set of restriction establishes that each classroom will be used by one professor teaching his corresponding class at some time slot. The last set of restrictions establishes that each time slot it is only available for one professor teaching some class into a classroom.

By considering that not necessarily $n_1 = n_2 = n_3 = n_4$ then some elements from each set will not be present in the final assignment.

In practice, the set of professors and courses will be greater than the set of classrooms and time slots. However, each classroom frequently can be used at all of the time slots and a time slot can be used to taught simultaneously several courses. The only case when the same time slot cannot be used more than once is when all the courses need to be taken for a particular group of people. In particular, when all the classrooms can be used at any time slot then we can reduce this problem in one dimension and to solve either n_3 times a 3AP or n_4 times a 3AP.

Even when a 3AP remains NP-hard we already have a strong machinery (the MAP-Gurobi) that is able to solve instances of 3AP up to 130 vertices by dimension in a few minutes whereas instances of 4AP can be solved up to 20 or 30 vertices by dimension in some minutes. Anyway if the required instance is greater than this sizes the heuristics we introduced can provided a good approach.

4.2.1 Setting up the costs matrix for the STP

One of the main problems is to obtain the corresponding costs matrix for the sAP to solve. We propose two criteria to perform this task.

4.2.1.1 Binary costs matrix

The simplest way for setting up the cost of all the vectors or hyperedges of the instance is to set a 0 value for the valid combinations and a 1 for the invalid ones. The idea is to identify the set of valid combinations and set the rest as invalids.

In practice, each professor is able to taught a very small number of courses, usually ≤ 10. Moreover, each professor is only able to taught such class in some of the available time slots. For instance, in some universities time slots use to be of one hour, one hour and a half and, two hours. When time slots are of one hour then a day can have until 15 time slots (considering a school day from 7:00 a.m. until 10 p.m.). Commonly professors are only available until eight hours from a predefined schedule time, which represents only the half of such time slots.

From the perspective of the courses, some of them need to be imparted in a classroom with special requirements such as laboratory instrumental, computer machines, projectors and classroom capacity. Such restrictions can help to reduce significantly the number of available combinations.

Depending on the logistic of each school more restrictions can be considered and can help to reduce the number of possible valid combinations.

In order to perform this task in an easier way we propose to generate a set of bipartite graphs which set the possible valid combination in a simple way, aimed to generated an instance in a similar way to the Clique family of instances for the MAP.

In this case we require the next bipartite graphs:

1. Professors-courses.
2. Professors-time slots.
3. Courses-classrooms.

There are other combinations, however them do not represent realistic restrictions or are already cover by the provided combinations, for example, is not common to have a professor with a preference over some classroom at least some special requirement

must be covered, in which case such restriction will be excluded from the assignment problem.

The final 4-dimensional matrix will have a 0 value for the combinations allowed according with the given bipartite graphs.

For example, suppose to have $n_1 = 4$, $n_2 = 4$, $n_3 = 4$ and $n_4 = 4$. Suppose that professor p_2, is only able to give the course c_1 and c_4, in the time slots t_1, t_3 and t_4. In addition, suppose that the class c_1 requires to be set in the classrooms r_1 and r_4 whereas the class c_2 requires to be set in the classrooms r_1 and r_3.

If we were allowing all the possible combinations for each professor then a total of 64 would be set for p_2, however, under the given restrictions just a set of 12 combinations are valid.

$$\begin{pmatrix} (p_2, c_1, r_1, t_1) & (p_2, c_1, r_1, t_3) & (p_2, c_1, r_1, t_4) \\ (p_2, c_1, r_4, t_1) & (p_2, c_1, r_4, t_3) & (p_2, c_1, r_4, t_4) \\ (p_2, c_2, r_1, t_1) & (p_2, c_2, r_1, t_3) & (p_2, c_2, r_1, t_4) \\ (p_2, c_2, r_4, t_1) & (p_2, c_2, r_4, t_3) & (p_2, c_1, r_4, t_4) \end{pmatrix}.$$

All these vectors will have a 0 cost in the $4-$dimensional costs matrix and the rest that consider professor p_2 will have a 1 value.

4.2.1.2 Priority costs matrix

The binary costs matrix has the disadvantage that all the valid vectors or combinations have the same weight whereas in practice some restrictions can have a higher priority over others.

For example can be more important to set professors to courses, then professors to time slots and, finally, courses to classrooms.

A very simple way to model a priority of a bipartite graph over others is as follows:

- Set a priority to each bipartite graph from 1 to P where P is the number of bipartite graphs. P denotes the highest priority and 1 the lowest priority.
- Fill each entry of the costs matrix with 2^P-1.
- For each entry of the costs matrix subtract 2^{j-1} with $1 \leq j \leq P$ depending on the priorities that apply to such entry based on its corresponding vector from the bipartite graphs.

At the end the lower costs entries will have a higher chance to be selected. This methodology provides an option to measure the cost of setting infeasible options under the given restrictions.

The advantage of this method is that we can generate $P!$ different orders for the priorities and, indeed, we will have $P!$ candidates to be choose the final assignment instead of have just one solution as in the binary costs matrix definition.

4.2.2 Dealing with a different number of vertices by set

To simplify this part we decided to change the size of each dimension to the size of the higher dimensions by adding some dummy vertices in order to equilibrate them. The only required rule is to set a very high cost (depending on the selected model to create the matrix costs) for those vectors that consider at least one dummy vertex.

This will allow us to use our algorithms and heuristics exactly in the same way that they already work, however this will increase considerable the complexity of an instance.

Suppose to have $n_1 > n_2 > n_3 > n_4 \geq 1$. The final assignment will be of size n_1 and we will have some vector in the final assignment that are invalid because they consider some dummy vertices. Such vector should be taken out of the solution as well as the invalid vectors due to the corresponding restrictions according with the selected model to build the matrix costs.

4.3 A real life instance: classes scheduling at UAM

We obtained a real data set which corresponds to the information of 9 years of classes scheduling of the faculty of basic sciences and engineering of the Universidad Autonoma Metropolitana campus Azcapotzalco (UAM-A). Here we summarize some important aspects and considerations about the classes scheduling in this faculty.

- This faculty is divided in five departments: basic sciences, systems, electronics, energy and materials.
- The faculty can require some courses from other faculties and departments of the university.
- Each department is in charge of some specific courses and each course is dispensed by only one department.
- At UAM, a year is divided in three quarters called YYI, YYP and YYO for winter, spring and autumn correspondingly. For example, the quarter of winter of 2016 is denoted as 16I.
- A course is called UEA (from the Spanish teaching-learning unit).
- For each quarter there is a minimum requirement of courses that should be covered.
- Some courses, can be required to appear in more than one class.
- Each course is required to be taught a specific number of hours in a week.

- Usually, most of courses are taught a minimum of 90 minutes or some multiples of 90 minutes.
- The most common is to have courses of 270 minutes and 180 minutes and the less common are of 90, 360 or 450 minutes in a week.
- Courses with more than 450 minutes use to be referred to some self-studying courses and should not be taught in some specific time slot so they are not considered as part of the assignment process since the function of the professor is more as an advisor.
- Each course has assigned a number of credits that counts for the students record in the university.
- The number of credits multiplied by ten is the desired number of hours that the student should dedicate to the studying of such course.
- The possible notes to obtain for a student are MB (10), B (8), S (6) and NA (not approved).
- The university opens at 7:00 a.m. and closes at 10:00 p.m.
- Each professor can be assigned to zero, one or more courses limited only by the working day of each person, however the average of classes taught by each professor is between 1 and 2.
- Each professor is assigned to some class by considering its experience or if he claims to know the topics of specific courses.
- In most of the cases, a course should preferably assigned at the same time slot among the week days.

The real data set is presented in a big table in a sheet of an Excel file. The table is composed of the next columns:

- Id. Some identifier of the row for each set of quarters.
- Record. The number of the registry corresponding with such class.
- UEA. The identifier of the course.
- UEA name. The name of the course.
- Group. The identifier of the group.
- Quarter. The identifier of the quarter (as YYI, YYP or YYO)
- Credits. The number of credits of the course.

- Theory hours. The number of hours of theory that should be taught by some professor.
- Laboratory hours. The number of hours of the practical laboratory.
- Department. The name of the department.
- MB. The number of student who got MB.
- B. The number of student who got B.
- S. The number of student who got S.
- NA. The number of not approved students.
- Capacity. The maximum number of students in such class.
- Number of declines. The number of people that decline to the course.
- Approval count. The number of students that approve the course. It corresponds with the sum of MB,
- Not approval count. The number of students that do not approve the course.
- Professor. The name of the professor.
- Five columns for the day of week from Monday to Friday. Each day contains the time period for the corresponding course or a dash if the course is not taught such day.

This data set has 42604 records. There are 1372 professors, 1346 courses and 34 departments among all records. For the purposes of our assignment we only consider the columns: UEA, UEA name, quarter, theory hours, laboratory hours, department, professor and the five columns for the day of the week. The other columns could be also useful as by giving a weight based on the approval count or the number of declines, however we are considering a more general case in which maybe these aspects are not relevant. If they are, then they can be incorporated as part of the cost in some particular way depending of the requirements and the value added for the assignment. In this moment we do not have some information about the value added of other variables for the assignment.

4.3.1 Solving the school timetabling problem at UAM

In order to solve this particular STP we make the following assumptions:

- We solve the assignment problem for each department independently. We analyzed that just approximately less than the 10% of professors are assigned to courses in two departments and less than the 1% are in more than two departments. Anyway we will avoid to assign a professor to more than one course at the same time.
- In order to have the UEA requirement we can consider the assigned courses for a the corresponding quarter. This means that we will perform assignments over the quarters from the data set.
- For the available working day of each professor for a quarter we consider the times at which such professor was assigned in at least one class during the previous nine quarters. In this way we do not direct the our assignment to something that already was applied.
- For the set of courses for which a professor will be available we consider those courses that were taught during the previous nine quarters to the quarter to solve.
- We consider time slots of 90 minutes starting at 7:00 a.m. and finishing at 10:00 p.m. This give us a total of 10 time slots in a day to set courses.
- The starting point of each time slot is fixed according with the Table 4.1:

Table 4.1: Starting times for the 10 time slots considered at UAM

Time slot	1	2	3	4	5	6	7	8	9	10
Starting time	7:00	8:30	10:00	11:30	13:00	14:30	16:00	17:30	19:00	20:30

- In order assign a class in one step, for the courses that require an assignment with more than one time slot is designed a schema to create time periods which allow to consider a group of time slots as a unique time period.
- Each course should be assigned from 1 to 5 time slots so, each course will be assigned to the same time slots over the week according with its number of required time slots.
- In addition to divide the data by department, we will divide the courses for each department according with their number of time slots required. This means that will be solved by approximately 34×5 (|departments|×maximum time slots by course) multidimensional assignment problems for design a complete courses scheduling for a quarter.
- We do not have the information about the rooms so, the corresponding MAPs can be modeled as 3AP.

One of the most difficult parts of this modeling was to create the general schema for those courses that should be assigned in more than one time slot over different days of the week (many as the number of time slots). The number of all the possible combinations can be really high and unrealistic for most of the cases. For example, a course that should be set at two time slots can be resulted assigned a Monday at 7:00 hours and a Friday at 19:00 hours which is not common to occur. The time periods schema that we consider reduce the number of combinations under the assumption that all the time slots of a course should be given at the same starting points among the days of the week. For example, a course with four time slots can be assigned at any of the 10 starting points and assigned to one of five possible combinations which are described at Table 4.2, additionally, we show a binary representation of the considered days of the week for the required time slots where the most left bit corresponds to Monday and the rightmost to Friday. A bit set to 1 indicates that such day will be part of the time period for the course. The binary representation of time periods helps to deal with the generation of the combinations and its handling in an algorithm.

Table 4.2: Five possible options for a course with four time slots

Considered days	Binary representation
Monday, Tuesday, Wednesday, Thursday	11110
Monday, Tuesday, Wednesday, Friday	11101
Monday, Tuesday, Thursday, Friday	11011
Monday, Wednesday, Thursday, Friday	10111
Tuesday, Wednesday, Thursday, Friday	01111

There are a total of $\binom{5}{1} = 5$, $\binom{5}{2} = 10$, $\binom{5}{3} = 10$, $\binom{5}{4} = 5$ and $\binom{5}{5} = 1$ possible combinations for courses with 1, 2, 3, 4, and 5 time slots correspondingly. By considering the starting points for the time slots, this give us a maximum of 10×10 unique time periods among which the courses can be assigned.

By considering all the previous assumptions we proposed a new solution to solve the STP at UAM-A that is based on the resolution of several 3AP to satisfy the UEA requirement of a quarter at the faculty of Basic sciences and engineering at UAM-A. Algorithm 25 shows our proposed solution.

The process showed in Algorithm 25 is as follows: first the list of assignments is set to empty. The cycle at the line 3 allows to divide the UEA requirements by department and the cycle at the line 4 allows to divide the requirements according to the number of required time slots for the courses, both divisions are considered at the same time at the line 5. At the line 6 the time slots are converted into time periods according with the rules previously described. The cycle at the line 8 will be executed whereas we have required courses to assign or until the process can not set a required course. At the line 10 we obtain all the possible feasible assignments based on the list of ProfessorsToCourses, the professor availability and the given time periods. Then, at the line 11 we generate the binary costs matrix for the 3AP by adding the required

Algorithm 25: A new solution for the STP at UAM-A based on the 3AP.

Input:
UEAReqByD. The requirements of UEA by department.
Departments. The list of departments.
ProfessorToCourses. A list of professors with their list of courses.
ProfessorAvailability. The working day of professors for the whole week.
TimeSlotsByCourse. The number of time slots by course.
Result: classesScheduling: The classes scheduling description.

```
1  Set ListOfAssignments := ∅;
2  foreach department ∈ Departments do
3      for blocks from 5 to 1 do
4          CoursesToSet := GetCoursesWithBlocks(UEAReqByD[department],
           TimeSlotsByCourse, blocks);
5          TimePeriods := GetCombinationsOfTimePeriods(blocks);
6          MissingCourses := |CoursesToSet|+1;
7          while MissingCourses > |CoursesToSet| and |CoursesToSet| > 0 do
8              MissingCourses := |CoursesToSet|;
9              possibleAssignments := getAssignments(ProfessorsToCourses,
               ProfessorAvailability, TimePeriods);
10             costsMatrix := generateBinaryCostsMatrix(possibleAssignments);
11             finalAssignment := SolveMAP(costsMatrix);
12             validAssignment := GetFeasibleAssignments(finalAssignment);
13             UpdateProfessorAvailability(ProfessorAvailability,
               validAssignment);
14             ListOfAssignments := ListOfAssignments ∪ validAssignment;
15             UpdateCoursesToSet(CoursesToSet, validAssignment);
16 Set classesScheduling := getCompleteScheduling(ListOfAssignments);
17 return {classesScheduling};
```

dummy vertices if required. At the line 12 the corresponding 3AP is solved and then at the line 13 we get the list of valid assignments from the corresponding solution to the 3AP. Since the problem is solved by applying the technique of modeling the 3AP through a binary costs matrix the only valid assignments are those whose cost is equal to zero. At the lines 14, 15 and 16 we update the professor availability, the missing list of uea requirements and the final ListOfAssignments. The deepest cycle can be executed by as many times ar the number of the required courses in the worst case. However in practices just a few executions were required before to full the requirements or not find more feasible assignments. Finally, we build the classes scheduling based on the ListOfAssignments found by this procedure.

We decided to start with the courses with the highest number of time slots because such courses usually are taught by a few number of professors therefore they have a

higher priority to be assigned at first, however different orders for such process can provide different global solutions.

4.3.2 Results on a real data set

We evaluated our solution for the STP at UAM-A by proposing classes scheduling for the quarters of 15P, 15O and 16I for all the departments considered in the data set. It is difficult to determine if our solution is better than the proposed classes scheduling for the corresponding quarters because we do not have a way to measure how good were the actual classes scheduling for such quarters. However, we proposed a metric that calculates the percentage of satisfied courses. This metric allow us to measure the effectiveness of the assignment by itself to cover all the set of UEA requirement for each quarter.

In order to solve the 3AP instances involved in our solution for the STP at UAM-A we considered our two best techniques:

1. MAP-Gurobi. It is used for the cases when the number of vertices is lower or equal than 120.
2. Simple Memetic Algorithm (SMA). It is used for the cases when the number of vertices is greather than 120.

Our solution for the STP at UAM-A was implemented in the programming language R and its performance was evaluated on a platform with an Intel Core i5-3210M 2.5 GHz processor with 4 GB of RAM under Windows 8.

It is important to mention that our solution is not satisfying the 100.0% of the UEA requirement in all the cases because we are only considering the historical time periods and the historical taught courses for each professor but no the actual time periods and the list of preferred courses (which may consider additional time periods and courses to those considered in the historical reference) for the corresponding quarter that we are solving, since such historical information is not available. In addition, our formulation to generate time periods do not consider some special cases of time periods, for example when a class should be scheduling at different days and at different start times or when a course with two time slots should be set in two consecutive time slots at the same day (because it is a laboratory course) or courses with three time slots that should be set in two days of 1.5 time slot each due to the working day professor requirements. This reasons derive in the problem that our solution cannot find feasible options when an assignment with such time periods is required. Another special case is when professors were hired just in the quarter in which the assignment is solved and the cases when some professors are teaching some courses by first time in the quarter to solve, which is again due to the fact that we are only considering just the historical data for the last nine quarters to create the valid assignment. If the model receive such new information the the quality of the assignment can be higher.

Table 4.3: Percentage of satisfied courses at UAM-A for the quarter 15P.

Department	UEA requirements	Satisfied courses	Missing courses	Percentage of satisfied courses
ADMINISTRACION	3	2	1	66.6
CIENCIAS BASICAS	377	375	2	99.4
DERECHO	8	8	0	100.0
DIR CYAD	1	0	1	0.0
DIRECCION DE LA	263	197	66	74.9
ECONOMIA	5	3	2	60.0
ELECTRONICA	198	194	4	97.9
ENERGIA	279	273	6	97.8
EVA. DISENO EN E	1	0	1	0.0
HUMANIDADES	5	2	3	40.0
INV.CONOCIMIENTO	3	3	0	100.0
MATERIALES	156	155	1	99.3
OFICINAS DE LA R	6	4	2	66.6
PROCESOS TEC. RE	3	1	2	33.3
SECRETARIA ACADE	21	5	16	23.8
SISTEMAS	169	168	1	99.4
SOCIOLOGIA	4	3	1	75.0
Total	1502	1393	109	92.7

Tables 4.3, 4.4 and 4.5 show a summary of the classes scheduling found by our solution for the STP at UAM-A considering the metric of the percentage of satisfied courses. We show the number of UEA requirements, then the number of satisfied courses, the number of not satisfied courses and the percentage of satisfied courses. We can observe that in general the percentage of the total satisfaction of the UEA requirement was greater than the 86.0%, even when in our assignment were not considered some special cases of time periods. The quarter of higher demand usually is the YYO.

It is important to highlight that this results were obtained just by only considering the historical information about previously taught courses and historical observed time periods of availability at the university for each professor. In order to get a more realistic solution it is necessary to have the information of the options of courses to teach by each professor for the quarter to solve as well as the corresponding availability of time slots in a week. Our tool is providing a solution that by using the historical information is able to determine if the actual UEA requirements can be covered with the currently hired personal or if it is necessary to hire some additional positions in order to cover the missing requirement.

We omitted the results for the version of our solution that considers the creation of a priority costs matrix because such formulation is more realistic for the cases when the information about preferred courses to teach and availability of time slots is the actual considered information for the quarter to solve and not the historical data.

Table 4.4: Percentage of satisfied courses at UAM-A for the quarter 15O.

Department	UEA requirements	Satisfied courses	Missing courses	Percentage of satisfied courses
ADMINISTRACION	118	104	14	88.1
CIENCIAS BASICAS	550	547	3	99.4
DEPTO. DE ESTUDI	6	1	5	16.6
DEPTO. DE PROCES	4	2	2	50.0
DEPTO. DE TECNOL	8	1	7	12.5
DERECHO	210	187	23	89.0
DIR CSH.	34	25	9	73.5
DIR CYAD	47	32	15	68.0
DIRECCION DE LA	190	168	22	88.4
ECONOMIA	150	121	29	80.6
ELECTRONICA	249	249	0	100.0
ENERGIA	315	299	16	94.9
EVA. DISENO EN E	156	104	52	66.6
FILOSOFIA	3	1	2	33.3
HUMANIDADES	78	70	8	89.7
INV.CONOCIMIENTO	118	90	28	76.2
MATEMATICAS	5	3	2	60.0
MATERIALES	222	222	0	100.0
MEDIO AMBIENTE	74	42	32	56.7
OFICINAS DE LA R	5	4	1	80.0
PROCESOS TEC. RE	222	162	60	72.9
PRODUCCION ECONO	3	3	0	100.0
SECRETARIA ACADE	4	4	0	100.0
SISTEMAS	204	202	2	99.0
SOCIOLOGIA	145	61	84	42.0
TEORIA Y ANALISI	3	1	2	33.3
Total	3123	2705	418	86.6

Table 4.5: Percentage of satisfied courses at UAM-A for the quarter 16I.

Department	UEA requirements	Satisfied courses	Missing courses	Percentage of satisfied courses
ADMINISTRACION	3	3	0	100.0
CIENCIAS BASICAS	247	246	1	99.5
DERECHO	10	10	0	100.0
DIR CSH.	1	0	1	0.0
DIRECCION DE LA	176	127	49	72.1
ECONOMIA	3	3	0	100.0
ELECTRONICA	135	133	2	98.5
ENERGIA	173	166	7	95.9
EVA. DISENO EN E	2	2	0	100.0
HUMANIDADES	5	5	0	100.0
MATERIALES	101	100	1	99.0
OFICINAS DE LA R	4	2	2	50.0
PROCESOS TEC. RE	4	4	0	100.0
SECRETARIA ACADE	65	2	63	3.0
SISTEMAS	110	110	0	100.0
SOCIOLOGIA	4	4	0	100.0
Total	1043	917	126	87.9

Chapter 5

Conclusions

We can conclude several things about two different lines of researching, by one side all the results about algorithms and heuristics for the resolution of the multidimensional assignment problem and, by the other side about the modeling and resolution of personnel assignment problems, in particular the school timetabling problem, through the multidimensional assignment problem.

Even when such lines are related we decided to divide our perspective in both lines in order to show the progress on each researching line as well as the possible future work.

5.1 The assignment problem

We determined that the Hungarian method is not the most suitable algorithm to solve some particular types of instances of the assignment problem, in particular for the three general variants of assignment problems: perfect assignments, imperfect assignments and incremental assignments.

For the purposes of solving a 2-dimensional assignment problem as part of a s-dimensional assignment problem with $s \geq 3$ we determined that the most suitable technique is a state-of-the-art auction algorithm aimed to solve perfect assignment which is the particular problem to solve for this case. This auction algorithm is called ϵ-scaling auction algorithm and it is more than 20 times faster than a state-of-the-art version of the Hungarian method, which was a very important fact that helped us to obtain a high quality solution in our simple memetic algorithm but in lower running times in comparison with the more complex state-of-the-art memetic algorithm for the multidimensional assignment problem.

Other contribution is that we found that a very important factor that has an impact on the resolution time is the distribution of the weights among the edges of the bipartite graph of each instance. Those instances whose distribution of weights among the edges were uniformly generated at random are the most difficult to solve

by the ϵ-scaling auction algorithm in comparison to the other studied distributions. The distribution that we care for the purposes of the multidimensional assignment problem were those uniformly generated at random, but even in such case the ϵ-scaling auction algorithm was approximately 27 times faster than the Hungarian method.

5.2 The multidimensional assignment problem

The multidimensional assignment problem it is known to be a NP-hard problem and even when better algorithms for this problem can be proposed in the future, unless P = NP will be difficult to reach a progress that allows to solve exactly instances of real life because the complexity of the problem grows exponentially and the real life problem size are considerably larger to the currently reached progress.

We summarize the main exact techniques that solve this problem and propose a new one, our MAP-Gurobi, which even defeat a state-of-the-art technique for the particular cases of instances of 3AP until 100 vertices under the evaluated families of instances proposed by several authors at different researches, finding optimal solutions for some instances where only feasible solutions were known.

We proposed some naive basic local search heuristics and the generalization of state-of-the-art local search heuristics, such as the generalization of the dimension-wise variation heuristics and the generalized local search heuristic, which allow us to obtain new local searches as DV3 and DVH3 that are competitive against the state-of-the-art metaheuristic (a relative complex memetic algorithm) that uses as part of its machinery some lower quality solution techniques with respect to our proposed local searches. We determined that the techniques DV3 and DVH3 provide quality solutions that obtained optimal results for the evaluated families of instances Random and Geometric, results pretty near to the optimal for the family of instances Product, with a relative solution error of 0.4%, and very competitive results for the families of instances Clique and Square Root, with relative solution errors of approximately 2.0% and 2.6% correspondingly. Part of this results were accepted for its publication in the Journal of Computational Intelligence.

We proposed a new state-of-the-art simple memetic algorithm for this problem which is competitive against the previously state-of-the-art memetic algorithm in terms of the complexity of the structure of the procedure as well as in obtaining similar quality solutions in lower running times. We determined that the quality of the memetic algorithm depends on the local search used as well as depend on the combination of the used method for the selection function, the crossover operator and the mutation operator, where the contribution is that among the evaluated selection functions and operators, the elitist selection function, the cycled crossover operator and the inversion mutation provide us the results of higher quality under the evaluated families of instances. The relative solution error for the families of instances Clique and Square Root under our simple memetic algorithm, which were the most difficult families for DV3 and DVH3, were of 0.3% and 0.7% correspondingly after

running times of 30 seconds which is similar to the reported relative solution errors of 0.2% and 0.4% for the state-of-the-art memetic algorithm but after running times of 300 seconds. Part of this results were accepted for its publication in the 2017 14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2017) through a medium length paper.

We obtained as state-of-the-art software for solving instances of the multidimensional assignment problem in as many dimensions and vertices as the current restrictions of memory space allow us to have.

5.3 Personnel assignment problems

We determined that any problem that deals with the assignment of personnel can be modeled as a through a multidimensional assignment problem for the cases where are involved assignments between persons and two or more other disjoint sets of things of concepts. We focused our study of personnel assignment problems on the case of the school timetabling problem where the assignment between elements from more than two sets is required.

Once we have a nice software to solve instances of the multidimensional assignment problem the only concern is to design a methodology for the establishing of weights for the hyperedges of the corresponding sAP. We proposed two criteria for this task: the binary costs matrix and the priority costs matrix. The first criteria is good for the cases where only feasibility is important whereas the second try to measure priorities over the assignment. This two criteria can be applied to any type of personnel assignment problem even when they are originally proposed for the school timetabling problem, we only need to model the corresponding problem according with the given methodologies and the proper restrictions and considerations of the problem to solve.

We considered the problem at the department of Basic Sciences at the Universidad Autonoma Metropolitana campus Azcapotzalco as a particular case of a school timetabling problem. We proposed a new solution for this problem which considered the historical data about previously taught courses by professors and their availability time periods at the university in order to determine if, based on such historical data, the university is able to satisfy the UEA requirement for the a particular quarter. The more realistic case is consider the actual requirements for a new quarter and the actual restrictions of the professors, however by the time of this thesis such data were not available. We show that the effectiveness of our methodology from the perspective of the percentage of the satisfied UEA requirements is higher than the 98% by considering historical data. We consider that the use of the proposed solution with not only the historical data, but with the actual data to be used for the classes scheduling of a new quarter can give even more better results and can be useful for the making decisions about if are required new hirings in order to satisfy the UEA requirements or they even can be solved with the available personnel at that

moment. We are able to add several restrictions to our solution and obtain results in a few minutes whereas the actual mechanism can take more than hours and days depending on the restrictions for each quarter. We conclude that our tool is able to be tested and applied for future scenarios of classes scheduling at UAM-A.

5.4 Future work

There are several lines that even can be explored.

In the case of the MAP we believe that should be possible to develop a better particular algorithm to solved it because the proposed solution, even when represents a new state-of-the-art algorithm, is based on a generic machinery. We only explored the versions of DV3 and DVH3 however we believe that other heuristics like a DV4, DVH4, or even more, DVH($s-1$) can provide higher quality solutions. We believe that smarter local searches derived from the k-opt heuristics also can be proposed in order to provide higher quality solutions than the brute force version of k-opt. We considered that our simple memetic algorithm also can explore over other selection functions, crossover and mutation operators such that the development of other more specific for MAP can even provide higher quality solutions. In addition, it can be considered the combination of our simple memetic algorithm with stronger local searches as DV3 or DV($s-1$) in order to compare its performance against the current state-of-the-art which considers local searches of lower quality solution.

In the case of the STP new ways to build the costs matrix for the corresponding sAP even can be proposed as well as other ways for the creation of time periods from time slots, which may include some particular cases not considered in our solution. It is necessary to apply our solution by considering a realistic scenario instead of historical information, this will provide a better metric for our solution. It is necessary to consider the information about rooms since out solution could not test those scenarios since in the information of real data the number of rooms was not available. More complex but realistic versions for the STP at UAM-A should consider the original demand of courses for the quarter to solve as well as preferences of students in order to obtain a solution that satisfies more aspects of the problem.

Bibliography

[Abdullah et al., 2007a] Abdullah, S., Burke, E. K., and McCollum, B. (2007a). A hybrid evolutionary approach to the university course timetabling problem. In *2007 IEEE Congress on Evolutionary Computation*, pages 1764–1768.

[Abdullah et al., 2007b] Abdullah, S., Burke, E. K., and McCollum, B. (2007b). *Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for the University Course Timetabling Problem*, pages 153–169. Springer US, Boston, MA.

[Ahrens and Dieter, 1982] Ahrens, J. H. and Dieter, U. (1982). Computer generation of poisson deviates from modified normal distributions. *ACM Trans. Math. Softw.*, 8(2):163–179.

[Ahuja et al., 1994] Ahuja, R. K., Orlin, J. B., Stein, C., and Tarjan, R. E. (1994). Improved algorithms for bipartite network flow. *SIAM Journal on Computing*, 23(5):906–933.

[Aiex et al., 2005] Aiex, R. M., Resende, M. G. C., Pardalos, P. M., and Toraldo, G. (2005). Grasp with path relinking for three-index assignment. *INFORMS Journal on Computing*, 17(2):224–247.

[Andrijich and Caccetta, 2001] Andrijich, S. M. and Caccetta, L. (2001). Solving the multisensor data association problem. *Nonlinear Analysis: Theory, Methods and Applications*, 47(8):5525 – 5536. Proceedings of the Third World Congress of Nonlinear Analysts.

[Appa et al., 2004] Appa, G., Magos, D., and Mourtos, I. (2004). A branch & cut algorithm for a four-index assignment problem. *Journal of the Operational Research Society*, 55(3):298–307.

[Appa et al., 2006] Appa, G., Magos, D., and Mourtos, I. (2006). A new class of facets for the latin square polytope. *Discrete Applied Mathematics*, 154(6):900 – 911.

[Balas and Saltzman, 1991] Balas, E. and Saltzman, M. J. (1991). An algorithm for the three-index assignment problem. *Operations Research*, 39(1):150–161.

[Bandelt et al., 1994] Bandelt, H.-J., Crama, Y., and Spieksma, F. C. (1994). Approximation algorithms for multi-dimensional assignment problems with decomposable costs. *Discrete Applied Mathematics*, 49(1):25 – 50.

[Bekker et al., 2005] Bekker, H., Braad, E. P., and Goldengorin, B. (2005). *Using Bipartite and Multidimensional Matching to Select the Roots of a System of Polynomial Equations*, pages 397–406. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bellman, 1962] Bellman, R. (1962). Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63.

[Bertsekas, 1981] Bertsekas, D. P. (1981). A new algorithm for the assignment problem. *Mathematical Programming*, 21(1):152–171.

[Bertsekas, 1988] Bertsekas, D. P. (1988). The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14(1):105–123.

[Bertsekas, 1992] Bertsekas, D. P. (1992). Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1(1):7–66.

[Bertsekas, 2009] Bertsekas, D. P. (2009). *Auction algorithms*, pages 128–132. Springer US, Boston, MA.

[Bertsekas and Castanon, 1993] Bertsekas, D. P. and Castanon, D. A. (1993). A forward/reverse auction algorithm for asymmetric assignment problems. *Computational Optimization and Applications*, 1:277–297.

[Bertsekas et al., 1993] Bertsekas, D. P., n on, D. A. C., and Tsaknakis, H. (1993). Reverse auction and the solution of inequality constrained assignment problems. *SIAM Journal on Optimization*, 3(2):268–297.

[Bozdogan and Efe, 2008] Bozdogan, A. O. and Efe, M. (2008). Ant colony optimization heuristic for the multidimensional assignment problem in target tracking. In *2008 IEEE Radar Conference*, pages 1–6.

[Budiono and Wong, 2011] Budiono, T. A. and Wong, K. W. (2011). Memetic algorithm behavior on timetabling infeasibility. In *TENCON 2011 - 2011 IEEE Region 10 Conference*, pages 93–97.

[Burkard et al., 2009] Burkard, R., Dell'Amico, M., and Martello, S. (2009). *Assignment Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[Burkard et al., 1996] Burkard, R. E., Rudolf, R., and Woeginger, G. J. (1996). Three-dimensional axial assignment problems with decomposable cost coefficients. *Discrete Applied Mathematics*, 65(1):123 – 139.

[Burke et al., 1997] Burke, E., Jackson, K., Kingston, J. H., and Weare, R. (1997). Automated university timetabling: The state of the art. *The Computer Journal*, 40(9):565.

[Cambazard et al., 2005] Cambazard, H., Demazeau, F., Jussien, N., and David, P. (2005). *Interactively Solving School Timetabling Problems Using Extensions of Constraint Programming*, pages 190–207. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Cerdeira-Pena et al., 2008] Cerdeira-Pena, A., Carpente, L., Farina, A., and Seco, D. (2008). New approaches for the school timetabling problem. In *2008 Seventh Mexican International Conference on Artificial Intelligence*, pages 261–267.

[Crama and Spieksma, 1992] Crama, Y. and Spieksma, F. C. (1992). Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research*, 60(3):273 – 279.

[Croes, 1958] Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812.

[Dasgupta et al., 2011] Dasgupta, S., Papadimitriou, C., and Vazirani, U. (2011). *Algorithms*. McGraw-Hill Education (India) Pvt Limited.

[de Werra, 1985] de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research*, 19(2):151 – 162.

[Dempster et al., 1975] Dempster, M., Lethbridge, D., and Ulph, A. (1975). School timetabling by computer a technical history. *Educational Research*, 18(1):24–31.

[Doulaty et al., 2013] Doulaty, M., Derakhshi, F., and Abdi, M. (2013). Timetabling: A state-of-the-art evolutionary approach. *Journal of Machine Learning and Computing*, 3(3):255–258.

[Duan and Su, 2012] Duan, R. and Su, H.-H. (2012). A scaling algorithm for maximum weight matching in bipartite graphs. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1413–1424, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

[Dyer and Frieze, 1986] Dyer, M. E. and Frieze, A. M. (1986). Planar 3dm is np-complete. *J. Algorithms*, 7(2):174–184.

[Easterfield, 1946] Easterfield, T. E. (1946). A combinatorial algorithm. *Journal of the London Mathematical Society*, s1-21(3):219–226.

[Edmonds and Karp, 1972] Edmonds, J. and Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264.

[Feltl and Raidl, 2004] Feltl, H. and Raidl, G. R. (2004). An improved hybrid genetic algorithm for the generalized assignment problem. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 990–995, New York, NY, USA. ACM.

[Ferland and Roy, 1985] Ferland, J. A. and Roy, S. (1985). Timetabling problem for university as assignment of activities to resources. *Computers and Operations Research*, 12(2):207 – 218.

[Fonseca and Santos, 2013] Fonseca, G. H. G. and Santos, H. G. (2013). Memetic algorithms for the high school timetabling problem. In *2013 IEEE Congress on Evolutionary Computation*, pages 666–672.

[Frieze, 1983] Frieze, A. (1983). Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13(2):161 – 164.

[Frieze, 1974] Frieze, A. M. (1974). A bilinear programming formulation of the 3-dimensional assignment problem. *Mathematical Programming*, 7(1):376–379.

[Gabow and Tarjan, 1989] Gabow, H. N. and Tarjan, R. E. (1989). Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036.

[Garbow, 1985] Garbow, H. N. (1985). Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31(2):148 – 168.

[Garey and Johnson, 1979] Garey, M. and Johnson, D. (1979). *Computers and intractability: A guide to the theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA.

[Gimadi et al., 2000] Gimadi, E. K., Kairan, N. M., and Serdyukov, A. I. (2000). On the solvability of a multi-index axial assignment problem on one-cycle permutations. *Russian Math. (Iz. VUZ)*, 44:21–26.

[Goldberg, 1995] Goldberg, A. V. (1995). Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504.

[Goldberg and Kennedy, 1995] Goldberg, A. V. and Kennedy, R. (1995). An efficient cost scaling algorithm for the assignment problem. *Mathematical Programming*, 71(2):153–177.

[Goldberg and Kennedy, 1997] Goldberg, A. V. and Kennedy, R. (1997). Global price updates help. *SIAM Journal on Discrete Mathematics*, 10(4):551–572.

[Goldberg and Tarjan, 1988] Goldberg, A. V. and Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940.

[Goldberg and Lingle, 1985] Goldberg, D. E. and Lingle, Jr., R. (1985). Alleslociand the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 154–159, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

[Grundel et al., 2004] Grundel, D. A., Oliveira, C. A. S., and Pardalos, P. M. (2004). Asymptotic properties of random multidimensional assignment problems. *Journal of Optimization Theory and Applications*, 122(3):487–500.

[Grundel and Pardalos, 2005] Grundel, D. A. and Pardalos, P. M. (2005). Test problem generator for the multidimensional assignment problem. *Computational Optimization and Applications*, 30(2):133–146.

[Gurobi Optimization, 2016] Gurobi Optimization, I. (2016). Gurobi optimizer reference manual.

[Gutin et al., 2007] Gutin, G., Goldengorin, B., and Huang, J. (2007). Worst case analysis of max-regret, greedy and other heuristics for multidimensional assignment and traveling salesman problems. In Erlebach, T. and Kaklamanis, C., editors, *Approximation and Online Algorithms*, volume 4368 of *Lecture Notes in Computer Science*, pages 214–225. Springer Berlin Heidelberg.

[Gutin and Karapetyan, 2009] Gutin, G. and Karapetyan, D. (2009). A memetic algorithm for the multidimensional assignment problem. In *Proceedings of the Second International Workshop on Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, SLS '09, pages 125–129, Berlin, Heidelberg. Springer-Verlag.

[Hall, 1935] Hall, P. (1935). *On Representatives of Subsets*, pages 58–62. Birkhäuser Boston, Boston, MA.

[Held and Karp, 1962] Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210.

[Huang and Lim, 2003] Huang, G. and Lim, A. (2003). A simple yet effective heuristic framework for optimization problems.

[Huang and Lim, 2006] Huang, G. and Lim, A. (2006). A hybrid genetic algorithm for the three-index assignment problem. *European Journal of Operational Research*, 172(1):249 – 257.

[Jat and Yang, 2008] Jat, S. N. and Yang, S. (2008). A memetic algorithm for the university course timetabling problem. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 427–433.

[Jiang et al., 2008] Jiang, H., Xuan, J., and Zhang, X. (2008). An approximate muscle guided global optimization algorithm for the three-index assignment problem. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2404–2410. IEEE.

[Johnsson et al., 1998] Johnsson, M., Magyar, G., and Nevalainen, O. (1998). On the euclidean 3-matching problem. *Nordic J. of Computing*, 5(2):143–171.

[Kachitvichyanukul and Schmeiser, 1985] Kachitvichyanukul, V. and Schmeiser, B. (1985). Computer generation of hypergeometric random variates. *Journal of Statistical Computation and Simulation*, 22(2):127–145.

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

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

[Karapetyan et al., 2009] Karapetyan, D., Gutin, G., and Goldengorin, B. (2009). Empirical evaluation of construction heuristics for the multidimensional assignment problem. In *London Algorithmics 2008: Theory and Practice*, Texts in algorithmics, pages 107–122, London, UK. College Publications.

[Karp, 1972] Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

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

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

[Kuhn, 1956] Kuhn, H. W. (1956). Variants of the hungarian method for assignment problems. *Naval Research Logistics Quarterly*, 3(4):253–258.

[Kuroki and Matsui, 2009] Kuroki, Y. and Matsui, T. (2009). An approximation algorithm for multidimensional assignment problems minimizing the sum of squared errors. *Discrete Applied Mathematics*, 157(9):2124 – 2135. Optimal Discrete Structures and AlgorithmsODSA 2006.

[Lara et al., 2008] Lara, C., Flores, J. J., and Calderón, F. (2008). *Solving a School Timetabling Problem Using a Bee Algorithm*, pages 664–674. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Lawrie, 1969] Lawrie, N. L. (1969). An integer linear programming model of a school timetabling problem. *The Computer Journal*, 12(4):307.

[Lewis, 1961] Lewis, C. F. (1961). The school timetable. *Cambridge University Press*.

[Magos, 1996] Magos, D. (1996). Tabu search for the planar three-index assignment problem. *Journal of Global Optimization*, 8(1):35–48.

[Magos and Mourtos, 2009] Magos, D. and Mourtos, I. (2009). Clique facets of the axial and planar assignment polytopes. *Discrete Optimization*, 6(4):394 – 413.

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

[Moscato, 1989] Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms.

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

[Murphey et al., 1998] Murphey, R., Pardalos, P., and Pitsoulis, L. (1998). A greedy randomized adaptive search procedure for the multitarget multisensor tracking problem. In Pardalos, P. and Du, D.-Z., editors, *Network design: Connectivity and facilities location*, volume 40 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 277–301. American Mathematical Society.

[Murphey et al., 1999] Murphey, R. A., Pardalos, P. M., and Pitsoulis, L. (1999). *A Parallel Grasp for the Data Association Multidimensional Assignment Problem*, pages 159–179. Springer New York, New York, NY.

[Naparstek and Leshem, 2016] Naparstek, O. and Leshem, A. (2016). Expected time complexity of the auction algorithm and the push relabel algorithm for maximum bipartite matching on random graphs. *Random Structures and Algorithms*, 48(2):384–395.

[Nguyen et al., 2014] Nguyen, D. M., Le Thi, H. A., and Pham Dinh, T. (2014). Solving the multidimensional assignment problem by a cross-entropy method. *Journal of Combinatorial Optimization*, 27(4):808–823.

[Orlin and Ahuja, 1992] Orlin, J. B. and Ahuja, R. K. (1992). New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical Programming*, 54(1):41–56.

[Pierskalla, 1968] Pierskalla, W. P. (1968). Letter to the editor: The multidimensional assignment problem. *Operations Research*, 16(2):422–431.

[Qaurooni, 2011] Qaurooni, D. (2011). A memetic algorithm for course timetabling. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 435–442, New York, NY, USA. ACM.

[Raghavjee and Pillay, 2009] Raghavjee, R. and Pillay, N. (2009). Evolving solutions to the school timetabling problem. In *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)*, pages 1524–1527.

[Ramshaw and Tarjan, 2012a] Ramshaw, L. and Tarjan, R. E. (2012a). On minimum-cost assignments in unbalanced bipartite graphs. Technical report, HP Labs technical report HPL-2012-40R1, www. hpl. hp. com/techreports/HPL-2012-40R1. html.

[Ramshaw and Tarjan, 2012b] Ramshaw, L. and Tarjan, R. E. (2012b). A weight-scaling algorithm for min-cost imperfect matchings in bipartite graphs. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*.

[Robertson, 2001] Robertson, A. J. (2001). A set of greedy randomized adaptive local search procedure (grasp) implementations for the multidimensional assignment problem. *Comput. Optim. Appl.*, 19(2):145–164.

[Schell, 1955] Schell, E. (1955). Distribution of a product by several properties. *Proceddings of the second symposium in Linear Programming*, pages 615–642.

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

[Thorup, 2004] Thorup, M. (2004). Integer priority queues with decrease key in constant time and the single source shortest paths problem. *J. Comput. Syst. Sci.*, 69(3):330–353.

[Tsidulko, 2014] Tsidulko, O. Y. (2014). On solvability of the axial 8-index assignment problem on single-cycle permutations. *Journal of Applied and Industrial Mathematics*, 8(1):115–126.

[Vargas, 2011] Vargas, M. (2011). Some algorithms for the assignment problem. Master's thesis, Center for Research and Advanced Studies of the National Polytechnic Institute.

[Vargas, 2017] Vargas, M. (2017). *Matchings in bipartite graphs and assignment problems*. PhD thesis, Center for Research and Advanced Studies of the National Polytechnic Institute.

[Votaw and Orden, 1953] Votaw, D. F. and Orden, A. (1953). The personnel assignmnet problem. *Symposium on Linear Inequalities and Programming*, abs/1003.3564:155–163.

[Wolfenden and Johntson, 1969] Wolfenden, K. and Johntson, H. C. (1969). Computer aided construction of scool timetables.