

**Universidad
Autónoma
Metropolitana**



Casa abierta al tiempo **Azcapotzalco**

Posgrado en Optimización

**Problemas de asignación de recursos
humanos a través del problema de
asignación multidimensional**

Tesis Doctoral

para obtener el título de

Doctor en Optimización

Presentada por

M. en C. Sergio Luis Pérez Pérez

Asesores de Tesis

Dr. Carlos Enrique Valencia Oleta

Dr. Francisco Javier Zaragoza Martínez

Ciudad de México, México

Septiembre de 2017

Resumen

El problema de asignación de personal aparece en diversas 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, \dots, X_s una colección de $s \geq 3$ conjuntos disjuntos, considere todas las combinaciones que pertenecen al producto Cartesiano $X = X_1 \times \dots \times X_s$ tal que cada vector $x \in X$, donde $x = (x_1, \dots, x_s)$ 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 $A = (x^1, \dots, x^n)$ de n vectores si $x_k^i \neq x_k^j$ 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 PAs consiste en encontrar una asignación de peso mínimo.

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. Se describen algunos algoritmos y se propone uno nuevo que resuelve instancias de tamaño medio para PAM. Se propone la generalización de las conocidas heurísticas de variación de dimensión así como una búsqueda local generalizada que proporciona un nuevo estado del arte de búsquedas locales para PAM. Adicionalmente, se propone un algoritmo memético con una estructura sencilla pero efectiva y que es competitivo con el mejor algoritmo memético conocido para PAM.

Finalmente, se presenta un caso particular de problema de asignación de personal: el Problema de Asignación de Horarios (PAH). El PAH considera la asignación de personal a uno, dos o más conjuntos de objetos, por ejemplo puede ser requerida la asignación de profesores a cursos a periodos de tiempo a salones, para determinados grupos de estudiantes. Primero, se presenta el PAH así como una breve descripción de su estado del arte. Luego, se propone una nueva forma de modelar este problema a través de la resolución de PAM y se aplica sobre el PAH en la Universidad Autónoma Metropolitana unidad Azcapotzalco (UAM-A). Se describen las consideraciones particulares del PAH en la UAM-A y proponemos una nueva solución para éste. Nuestra solución se basa en la resolución de múltiples PA3 a través de los algoritmos y heurísticas propuestos.

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

**Universidad
Autónoma
Metropolitana**



Casa abierta al tiempo **Azcapotzalco**

Graduate program on Optimization

**Personnel assignment problems through
the multidimensional assignment problem**

Doctoral Thesis

to obtain the title of

Doctor of Optimization

Presented by

M.Sc. Sergio Luis Pérez Pérez

Thesis Advisors

Dr. Carlos Enrique Valencia Oleta

Dr. Francisco Javier Zaragoza Martínez

Mexico City, Mexico

September 2017

Abstract

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

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

In this thesis we make an in depth study of MAP beginning with the state-of-the-art algorithms, heuristics, and metaheuristics for solving it. We describe some algorithms and we propose a new one for solving optimally medium size instances of MAP. We propose the generalization of the called dimensionwise variation heuristics for MAP and a new generalized local search heuristic that provides new state-of-the-art 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 study a particular case of personnel assignment problem: the School Timetabling Problem (STP). The STP considers the assignment of personnel to other two or more sets, for example the assignment of professors to courses to time slots to rooms can be required. First, we provide a brief description of the state-of-the-art for STP. Then, we introduce a new approach for modeling this problem through the resolution of several MAP and we apply our solution on a real life case of study: STP at the Universidad Autonoma Metropolitana campus Azcapotzalco (UAM-A). We provide the particular aspects for STP at UAM-A and we provide a new solution for this problem. Our approach is based on solving several 3AP considering the introduced model and our proposed techniques.

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

Acknowledgements

This work is not only the result of a personal effort, it is also the result of the effort of all the people that helped to me during my doctoral studies. First, I want to thank to my wife Shantal for her patience while I was working in this thesis and for all the motivation she always gave me to successfully finish this work. To my beautiful children Fatima, Ilian, Fernando, and the little Sergio (who is in the way) who were the main engine that impelled me to reach this objective in my life. To my parents, Lourdes and Alfredo, who always supported me and were patient about me without matter the hard decisions I took while I was doing my doctoral studies at the University. To my brother Cristian for the cigarettes that we used to share while I was working on this. To my grandparents, aunts, uncles and cousins who always believed in me.

I want to thank to all my friends, mainly to Carlos, Gualberto, Luis, Marcos and Rodrigo who always helped me to the realization of this work and because they always gave me his friendship without restrictions. To my colleagues among the different jobs that I had at some periods of my studies (those from UAM-Cuajimalpa, Aftercollege, and Banco de Mexico), they also believed in me. To Pano Santos because he inspired me to apply theoretical knowledge into real world problems. To my thesis advisor Carlos for all his support and excellent guide, his knowledge about assignment problems was fundamental to conclude this work. To my thesis advisor Francisco who always believes in me and, in many ways, always supported my work. To the reviewers of this thesis, Laura, Antonin, and Edgar whose comments and suggestions added a big value to this work. To Rafael and Javier for being patient about me and my decisions while I was on this program. To all the people who conforms the Graduate program on Optimization, really thanks.

Also, I want to thank to COMECyT and CONACyT for the economical support that they provided to me in some periods of my doctoral studies. Finally, I want to say thank you to all the people and institutions who helped me in some way to reach this goal in my life.

Sergio Perez

Contents

1	Introduction	13
1.1	Motivation	14
1.2	Problem description	15
1.3	Methodology	15
1.4	Thesis structure	16
2	The assignment problem	17
2.1	State of the art	20
2.1.1	Auction algorithms	20
2.1.2	Weight scaling algorithms	21
2.1.3	Push relabel algorithms	22
2.2	Selection of an efficient algorithm	22
2.2.1	The Hungarian method	22
2.2.2	The FlowAssign algorithm	23
2.2.3	The ϵ -scaling Auction algorithm	24
2.3	Families of instances for the AP	25
2.3.1	Uniform distribution	26
2.3.2	Normal distribution	26
2.3.3	Poisson distribution	27
2.3.4	Binomial distribution	27
2.3.5	Hypergeometric distribution	27
2.4	Performance results	28
2.5	Conclusions	32
3	The multidimensional assignment problem	35
3.1	State of the art	37
3.1.1	Exact algorithms	38
3.1.2	Approximate algorithms	39

3.1.3	Local search heuristics	40
3.1.4	Evolutionary algorithms	42
3.2	Families of instances for the MAP	43
3.2.1	The Random family	43
3.2.2	The Clique family	44
3.2.3	The Square Root family	44
3.2.4	The Geometric family	44
3.2.5	The Product family	45
3.3	Exact algorithms	45
3.3.1	Brute force	46
3.3.2	Wise brute force	47
3.3.3	Dynamic programming reduction	47
3.3.4	MAP-Gurobi	50
3.3.5	Experimental evaluation	52
3.4	Local search heuristics	56
3.4.1	Basic local search heuristics	58
3.4.2	Dimensionwise variation heuristics	66
3.4.3	The generalized dimensionwise variation heuristics	68
3.4.4	The k -opt heuristic	71
3.4.5	Combined heuristics	73
3.4.6	A generalized local search heuristic	73
3.4.7	Multi-start local search	74
3.4.8	Iterated local search	75
3.4.9	Experimental evaluation	75
3.5	A new simple memetic algorithm for the MAP	89
3.5.1	Genetic representation and fitness function	96
3.5.2	The selection function	96
3.5.3	The crossover operator	99
3.5.4	The mutation operator	102
3.5.5	Experimental evaluation	105
3.6	General results	114

4 Personnel assignment problems: the school timetabling problem as a case study **117**

4.1	State of the art	118
4.2	Modeling the school timetabling problem as a MAP	120

4.2.1	Setting up the costs matrix for the STP	121
4.2.2	Dealing with a different number of vertices per set	122
4.3	A real life instance: scheduling classes at UAM	123
4.3.1	Solving the school timetabling problem at UAM	125
4.3.2	Results on a real data set	128
5	Conclusions	133
5.1	The assignment problem	133
5.2	The multidimensional assignment problem	134
5.3	Personnel assignment problems	135
5.4	Future work	136
A	Best known solutions	137

List of Figures

2.1	Representation of an assignment as a matching in a bipartite graph. .	18
3.1	Representation of a 3-dimensional assignment as a matching in a multipartite hypergraph.	37
3.2	Application Programming Interface (API) of Gurobi.	50
3.3	Complexity curves (in logarithmic scale) of exact algorithms for 3AP.	53
3.4	Complexity curves (in logarithmic scale) of exact algorithms for 4AP.	54
3.5	Complexity curves (in logarithmic scale) of exact algorithms for 5AP.	54
3.6	Structure of the LSGA proposed by Huang and Lim (2006).	94

List of Tables

2.1	Number of successes under Jarque-Bera test for normality.	30
2.2	Averaged running times for the five families of instances for the AP solved through Auction, FlowAssign and Hungarian algorithms. . . .	31
2.3	Comparative results for the sum of running times of Auction, FlowAssign and Hungarian over the families of instances for the AP.	32
3.1	Computational results for the family of instances of Balas and Saltzman under exact algorithms	55
3.2	Computational results for the family of instances <i>saxialn</i> under Magos' algorithm and MAP-Gurobi	56
3.3	Computational results for the family of instances Random under MAP-Gurobi	57
3.4	Computational results for the family of instances Clique under MAP-Gurobi	57
3.5	Computational results for the family of instances Square Root under MAP-Gurobi	58
3.6	Random, Clique and SquareRoot under basic heuristics.	78
3.7	Geometric and Product under basic heuristics.	79
3.8	Summary of relative solution error for basic heuristics.	79
3.9	Running times for Random, Clique, Square Root, Geometric, and Product under basic heuristics.	80
3.10	Random, Clique and SquareRoot under k -opt heuristics.	82
3.11	Geometric and Product under k -opt heuristics.	83
3.12	Summary of relative solution error for k -opt heuristics.	83
3.13	Running times for Random, Clique, Square Root, Geometric, and Product under k -opt heuristics.	84
3.14	Random, Clique and Square Root under DVH.	85
3.15	Geometric and Product under DVH.	86
3.16	Summary of relative solution error for DVH.	86

3.17	Running times for Random, Clique, Square Root, Geometric, and Product under DVH.	87
3.18	Random, Clique and Square Root under basics combined with MLS and ILS.	88
3.19	Geometric and Product under basics combined with MLS and ILS.	89
3.20	Random, Clique and Square Root under k -opt and DV2 combined with MLS and ILS.	90
3.21	Geometric and Product under k -opt and DV2 combined with MLS and ILS.	91
3.22	General results for Random, Clique and Square Root under the best local searches.	92
3.23	General results for Geometric and Product under the best local searches.	93
3.24	Averaged best known solutions.	105
3.25	Random under SMA combined with SDV2 and DV2.	106
3.26	Clique under SMA combined with SDV2 and DV2.	107
3.27	Square Root under SMA combined with SDV2 and DV2.	108
3.28	Geometric under SMA combined with SDV2 and DV2.	109
3.29	Product under SMA combined with SDV2 and DV2.	110
3.30	Relative solution errors for all combinations of operators for SMA.	111
3.31	Random, Clique, Square Root, Geometric, and Product under LSGA.	112
3.32	Running times for Random, Clique, Square Root, Geometric, and Product under LSGA.	113
3.33	Random, Clique and Square Root under memetic algorithms.	114
3.34	Geometric and Product under memetic algorithms.	115
3.35	General results for Random, Clique and Square Root under the best techniques.	116
4.1	Starting times for the 10 time slots considered at UAM	126
4.2	Five possible options for a course with four time slots	127
4.3	Percentage of satisfied courses at UAM-A for the quarter 15P.	130
4.4	Percentage of satisfied courses at UAM-A for the quarter 15O.	131
4.5	Percentage of satisfied courses at UAM-A for the quarter 16I.	132
A.1	Optimal solutions for the family of instances Random.	138
A.2	Best known solutions for the family of instances Clique.	138
A.3	Best known solutions for the family of instances Square Root.	139
A.4	Best known solutions for the family of instances Geometric.	140
A.5	Best known solutions for the family of instances Product.	140

List of Algorithms

1	Improved version of the Hungarian method (by Ramshaw and Tarjan)	23
2	FlowAssign algorithm (by Ramshaw and Tarjan)	24
3	The ϵ -scaling Auction algorithm (by Bertsekas)	25
4	Jarque-Bera test for the composite hypothesis of normality	28
5	Instance generator for a variety of families of instances for the AP	29
6	The brute force algorithm for MAP.	46
7	The wise brute force algorithm for MAP.	48
8	The dynamic programming reduction for MAP.	49
9	The MAP-Gurobi algorithm for MAP.	52
10	The simple swap heuristic for MAP.	59
11	The inversion heuristic for MAP.	61
12	The circular rotation heuristic for MAP.	63
13	The k -vertex permutation heuristic for MAP.	65
14	The dimensionwise variation heuristic for MAP.	67
15	The generalized dimensionwise variation heuristic for MAP.	69
16	The k -opt heuristic for MAP.	72
17	The multi-start local search for MAP.	75
18	The iterated local search for MAP.	76
19	The general structure of a genetic algorithm.	94
20	A new memetic algorithm for MAP.	95
21	The deterministic selection function.	97
22	The tournament selection function.	97
23	The roulette wheel selection function.	98
24	The partially mapped crossover.	100
25	The cycle crossover.	102
26	The ordered crossover.	103
27	The swapping mutation.	104

28 The inversion mutation. 104

29 A new solution for the STP at UAM-A based on the 3AP. 128

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, in some cases, 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 a few heuristics have been proposed 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 hand, 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 arguably one of the most important problems in operations research, and the multidimensional assignment problem arises naturally 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 some heuristics and a memetic algorithm were published for the multi-dimensional assignment problem for the cases with an arbitrary number of dimensions ([Karapetyan and Gutin, 2011a] and [Karapetyan and Gutin, 2011b]). One of the main goals 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 it is still 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 could also 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 courses 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. This will give us the maximum number of possible assignments.

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. 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 and comparison of algorithms for the assignment problem.
- The study of algorithms and heuristics for the multidimensional assignment problem. We are focused on the techniques that were proven to be the best ones (in terms of quality solution and complexity).
- Development of some algorithms for the multidimensional assignment problem.

- Development of several local search heuristics for the multidimensional assignment problem and their experimental evaluation.
- Development of a memetic algorithm for the multidimensional assignment problem and its experimental evaluation.
- The study of the school timetabling problem as a particular case of personnel assignment problems.
- The procurement of a real application of the school timetabling problem.
- Modeling and solving this real timetabling problem through the resolution of several multidimensional assignment problems.
- Analysis of results, conclusions and proposal of future work.

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 to solve this problem. In Chapter 3 we stated the multidimensional assignment problem and we propose several algorithms, heuristics and a memetic algorithm for solving it, then we compare such procedures against state of the art heuristics and meta-heuristics. In Chapter 4 we study the school timetabling problem as a case study for personnel assignment problems and we propose a new solution based on the resolution of several 3AP for a real application. In Chapter 5 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 dimensional version and then extended it to higher dimensions. On the other hand, it is necessary because some of the 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 is 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 $\varphi(1)$, \dots , 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:

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

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 satisfies 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 disjoint 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 meets 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.

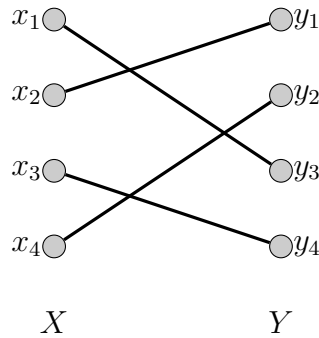


Figure 2.1: Representation of an assignment as a matching in a bipartite graph.

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 elements, 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 associated with the cost matrix C then the corresponding 0-1 integer linear programming formulation is:

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

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:

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

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 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 and 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 $1, 2, \dots, \nu(G)$ 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 the perfect assignment problem given that imperfect assignments can be reduced to perfect assignment as is described in [Vargas, 2011]. 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 complexity was $O(2^n n^2)$, however the problem was described in terms of a combinatorial problem rather 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 it was determined that its 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 on a set of offered products to multiple buyers.

An auction algorithm is an iterative procedure where we compare a set of offers and then a sale 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 explained

in detail [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. The 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. The complexity of parallel auction algorithms for a problem with n vertices 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 can 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. Complexity of the first weight scaling algorithm was $O(\sqrt{nm} \log nC)$ where n is the number of vertices, m is the number of edges and C is the maximum weight among all the edges.

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 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 [Gabow, 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 in the middle nineties by [Goldberg and Kennedy, 1995].

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 recent algorithms were experimentally evaluated.

2.2.1 The Hungarian method

The implemented Hungarian method consists on an 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

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 .

```

1 Set  $A$  to the empty matching;
2 Set prices at  $X$  to 0, at  $Y$  to  $C$ ;
3 Let  $\nu(G)$  be the max size of any matching at  $G$ . for  $s$  in  $0 : \nu(G)$  do
4   use Dijkstra to build a shortest path forest with roots at all  $x \notin A$ ;
5   if some  $y \notin A$  was reached then
6     Raise prices to tighten the tree path to  $y$ ;
7     Augment  $A$  along that tight path;
8   else
9     return  $A$  of size  $\nu(G)$ ;

```

G is reached. This algorithm has an overall complexity of $O(ms + s^2 \log r)$ where m 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 other algorithms are, in comparison to this technique.

2.2.2 The FlowAssign 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

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 $s \leq t$.

- 1 Set $(M, s) = \text{HopcroftKarp}(G, t)$;
- 2 Convert M into an integral flow f on N_G with $|f| = s$;
- 3 Set $\epsilon = \bar{\epsilon}$;
- 4 $\forall v \in N_G$ set the prices $p_d(v) = 0$;
- 5 **while** $\epsilon > \underline{\epsilon}$ **do**
- 6 $\epsilon = \epsilon/q$;
- 7 Refine(f, p, ϵ) builds a shortest path forest, finds a maximal set of augmenting paths and augments along them;
- 8 Round prices to integers that make all arcs proper;

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 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 an overall 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 than perfect assignments but easier than incremental assignments, this is why its complexity is better than the one for the Hungarian method.

2.2.3 The ϵ -scaling Auction algorithm

In order to evaluate a faster algorithm, we consider the ϵ -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 it performs local improvements in order to reach a global optimum whereas the other ones perform global updates for the same goal. The ϵ -scaling Auction algorithm operates like a real auction. The core of this algorithm is that, at each step a condition called *complementary slackness* should be kept. The complementary slackness states that it is possible to obtain an optimal solution to the dual when only an optimal solution to the primal is known. It 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 algorithm.

Algorithm 3: The ϵ -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 .

```

1 Set  $A = \emptyset$ ;
2 Let  $p$  the prices that will satisfy the  $\epsilon$ - Complementary Slackness condition
  (such that  $w(x_1y_1) - p(y_1) \leq \min_{y_2 \in N(x_1)} \{w(x_1y_2) - p(y_2)\} + \epsilon$ );
3 while is some  $x \notin A$  do
4   Consider some  $x_1 \notin A$ ;
5   Find the edges  $x_1y_1, x_1y_2$  with the two minimum costs;
6   if If  $u$  has only 1 neighbor then
7     Set  $\gamma = \infty$ ;
8   else
9     Set  $\gamma = (w(x_1y_1) - p(y_2)) - (w(x_1y_2) - p(y_1))$ ;
10  Set  $p(y_1) = p(y_1) - \gamma - \epsilon$ ;
11 return  $A$ 

```

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 ϵ -scaling Auction 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 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.

A way to generate a test bed for the family of instances ENV is to consider a complete bipartite graph with uniformly random generated weights in the closed in-

terval $[a, b]$. 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 hand, 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 comparison between each solver several distributions will be considered.

The next subsections describe the types of distributions used 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 occurs with the same probability. This distribution is described by the next probability density function:

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

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:

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

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. Anyway, we

will use a normality testing to experimentally show that truncating values to integers has no affectation over the distribution curve.

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 density:

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

This family of instances will be generated by setting $\lambda = 50$ such that the generated values tend to something similar like in the normal distribution. A formulation proposed by [Ahrens and Dieter, 1982] will be used for this distribution. They 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 it provides a competitive method for generation of 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 density:

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

This family of instances will be generated by setting the parameters $n = 100$ and $p = 0.25$ such that the generated values tend to 25, 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:

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

This family of instances will be generated by considering the parameters $N = 200$, $K = 50$, $n = 50$ such that the generated values are within the closed interval of $[0, 50]$ and the behavior of data has the desired curve. A formulation proposed by [Kachitvichyanukul and Schmeiser, 1985] will be used for this distribution.

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.

We applied Jarque-Bera test ([Jarque and Bera, 1987]) to verify that, for the chosen parameters, our generated sample data do not have the skewness and kurtosis matching a normal distribution. Algorithm 4 shows the created function to apply Jarque-Bera test over a generated sample data. The probability value *pvalue* is the probability, for a given statistical model, of accepting or rejecting null hypothesis at that significance level. In [Jarque and Bera, 1987] is recommended a *pvalue* = 0.1 however we decided to use *pvalue* = 0.05 which allows to have a higher significance level for the sizes of the generated sample data. The input *sampleData* is then evaluated under Jarque-Bera test; it rejects the null hypothesis (returns 0) when *pvalue* < 0.05, which means that *sampleData* do not have the skewness and kurtosis matching a normal and it accepts the null hypothesis (returns 1) otherwise.

Algorithm 4: Jarque-Bera test for the composite hypothesis of normality

Input: NameDistribution: name of the distribution to verify

sampleData: a numeric vector of data values

Result: 1 if the behaviour is the expected or 0 otherwise

```

1 Set pvalue := 0.1;
  // The maximum value at which the hypothesis is rejected
2 Set test := jb.norm.test(sampleData, nrepl = 100);
3 if test$.p.value ≥ pvalue then
4   return 1 // The sample data is approximated by a normal
   distribution
5 return 0 // The sample data is not approximated by a normal
   distribution
```

Algorithm 5 shows the code used to generate the five families of instances for the AP as well as the applied test. Each instance file is generated with a header value n that indicates the number of vertices on the bipartite graph (with $n = |X| = |Y|$); then following are n^2 integer values corresponding with the n^2 edge costs given in the order $w(x_1, y_1), \dots, w(x_1, y_n), w(x_2, y_1), \dots, w(x_2, y_n), \dots, w(x_n, y_1), \dots, w(x_n, y_n)$. This format is commonly used to store complete bipartite graphs. Five

instances for each combination of *distribution* and N were generated for a total of 125 instances. The 25 instances for each distribution were tested under Jarque-Bera test. The variable *test_ok* shows the number of instances that accepted the null hypothesis for each distribution.

Algorithm 5: Instance generator for a variety of families of instances for the AP

Input:

Result: A total of $(\text{number_of_instances} * \text{length}(\text{n_sampleData}) * 5)$ instances for the AP under the distributions uniform, normal, Poisson, binomial, and hypergeometric.

```

1 set.seed(0);
2 Set n_sampleData := c(64, 128, 256, 512, 1024);
3 Set number_of_instances := 5;
4 Set test_ok := c(0, 0, 0, 0, 0);
5 for iteration in 1:number_of_instances do
6   for n in n_sampleData do
7     Set edges := n * n;
8     Set sampleData := round(runif(edges, min = 1, max = 100));
9     Set test_ok[1]:= test_ok[1] + testNormal("Uniform", sampleData);
10    write(c(n, sampleData), file = sprintf("ap_unif_n%d_k%d.in", n,
iteration), ncolumns = 1);
11    Set sampleData := abs(round(rnorm(edges, 50, 10)));
12    Set test_ok[2]:= test_ok[2] + testNormal("Normal", sampleData);
13    write(c(n, sampleData), file = sprintf("ap_norm_n%d_k%d.in", n,
iteration), ncolumns = 1);
14    Set sampleData := rpois(edges, 50);
15    Set test_ok[3]:= test_ok[3] + testNormal("Poisson", sampleData);
16    write(c(n, sampleData), file = sprintf("ap_pois_n%d_k%d.in", n,
iteration), ncolumns = 1);
17    Set sampleData := rbinom(edges, 100, 0.25);
18    Set test_ok[4]:= test_ok[4] + testNormal("Binom", sampleData);
19    write(c(n, sampleData), file = sprintf("ap_binom_n%d_k%d.in", n,
iteration), ncolumns = 1);
20    Set sampleData := rhyper(edges, 50, 150, 50);
21    Set test_ok[5]:= test_ok[5] + testNormal("Hyper", sampleData);
22    write(c(n, sampleData), file = sprintf("ap_hyper_n%d_k%d.in", n,
iteration), ncolumns = 1);
23 print(test_ok)

```

Table 2.1 shows the results of applying Jarque-Bera test for normality to the generated instances. The number of instances that accepted the null hypothesis was

25 in the case of the instances generated under Normal distribution and very near to 0 for the rest of the cases. Hence, for the given parameters for each distribution the obtained sample data do not have the skewness and kurtosis matching a normal distribution except in the case of normal distribution as is expected.

Table 2.1: Number of successes under Jarque-Bera test for normality.

Distribution	Number of successes	Ratio
Binomial	2	8%
Hypergeometric	1	4%
Normal	25	100%
Poisson	0	0%
Uniform	0	0%

The format name for the instance files is `ap_`*distribution*`_` n `_`*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 an improved version of the Hungarian method, the FlowAssign algorithm, and the ϵ -scaling Auction algorithm. These algorithms were implemented in C++ and their performance was evaluated on a platform with an Intel Core i5-3210M 2.5 GHz processor with 4 GB of RAM under Windows 8.

Table 2.2 shows the running times for the described families of instances. 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 ϵ -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 better at solving a different category of the assignment problem: the Hungarian method for incremental assignments (which is the most difficult category of AP), the FlowAssign algorithm for imperfect assignments, and the ϵ -scaling Auction algorithm for perfect assignments (which corresponds to the easiest category of AP). All the generated instances corresponds with the problem of perfect assignments. This is why, the ϵ -scaling Auction algorithm outperformed the other two algorithms.

By considering the average of the total time results reported in Table 2.2, it can be observed that the ϵ -scaling Auction algorithm is approximately 10 times faster than the FlowAssign algorithm whereas it is 3 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 category of perfect assignments is required as part of some heuristics that will be described in the next chapter. Those heuristics are able to get better

Table 2.2: 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	0.006	0.006	0.006
	128	0.003	0.025	0.031
	256	0.009	0.119	0.187
	512	0.034	0.481	1.044
	1024	0.166	2.688	7.297
Hypergeometric	64	0.000	0.006	0.012
	128	0.000	0.019	0.031
	256	0.006	0.103	0.166
	512	0.041	0.500	1.081
	1024	0.247	2.750	7.604
Normal	64	0.000	0.009	0.009
	128	0.000	0.025	0.038
	256	0.013	0.116	0.203
	512	0.038	0.490	1.066
	1024	0.175	2.656	7.163
Poisson	64	0.003	0.010	0.003
	128	0.006	0.028	0.028
	256	0.013	0.100	0.169
	512	0.041	0.500	1.043
	1024	0.169	2.613	7.016
Uniform	64	0.000	0.006	0.012
	128	0.000	0.025	0.031
	256	0.015	0.109	0.181
	512	0.066	0.522	1.288
	1024	0.394	2.209	10.154
Ave. time		0.058	0.645	1.835

solutions if they 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 solution quality in a shorter period of time.

Table 2.3 shows the average of the running times by each family of distribution and algorithm showed in Table 2.2. The row Average shows the average overall running times for each algorithm and the row Standard dev. shows the standard deviation σ among the running times for all the families of instances under the same algorithm. It can be observed that the families of instances Binomial, Hypergeometric, Normal and Poisson have a similar level of difficulty for all the algorithms whereas the level

of difficulty of the Uniform family is significantly different. The Uniform family is the easiest family for FlowAssign algorithm whereas is the most difficult for ϵ -scaling Auction algorithm and Hungarian method. It is explained because ϵ -scaling Auction algorithm and Hungarian method are local-like update algorithms which performs better when distribution of weights is different. In the case of ϵ -scaling Auction algorithm the auctions will be easier to decide since distribution of weights is variable and will require less scaling phases. In the case of Hungarian method the augmenting paths can be easier to obtain due to variability at weights distribution. FlowAssign is a global-like update technique, so that at each iteration the optimization is performed uniformly being instances with Uniform distributions over the weights the easiest cases to solve. Even so, the ϵ -scaling Auction algorithm is the faster algorithm in all the cases which is the relevant fact for our purposes.

Table 2.3: 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	0.044	0.664	1.713
Hypergeometric	0.059	0.676	1.779
Normal	0.045	0.659	1.696
Poisson	0.046	0.650	1.652
Uniform	0.095	0.574	2.333
Average	0.058	0.645	2.333
Standard dev.	0.022	0.040	0.282

Based on the experimental results of Table 2.3 we can conclude that the weights distribution have an impact on the algorithmic performance. In the case of the ϵ -scaling Auction algorithm and the FlowAssign algorithm the complexity function has an explicit relation with the weights but, in the case of the Hungarian method, such relation is not part of the complexity function.

2.5 Conclusions

There is a wide variety of algorithms that solve different categories of AP. It is recommendable to use the better algorithm according with the category problem to solve.

Here were implemented some of the best algorithms to solve each category of linear assignment problems in order to show the expected performance in practice for the category that we care, which are perfect assignments.

Five families of instances were proposed 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 ϵ -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.

This analysis is relevant because some of the heuristics that we improved implemented the Hungarian method in their heuristics as in [Huang and Lim, 2006] and [Karapetyan and Gutin, 2011a] whereas we implemented the ϵ -scaling Auction algorithm, which is a better option for our heuristics. Such results are presented in the next chapter.

Chapter 3

The multidimensional assignment problem

The multidimensional assignment problem (MAP), also known as s AP 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, \dots, X_s be a collection of s disjoint sets, without loss of generality we assume that $n = |X_1| = |X_2| = \dots = |X_s|$ (otherwise we add some dummy elements to equilibrate them), a s AP could be equivalently stated in one of several ways.

An 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 , \dots , 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:

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

where 1 is mapped to $\varphi_1(1)$, \dots , and n is mapped to $\varphi_1(n)$, and $\varphi_1(1)$ is mapped to $\varphi_2(1)$, \dots , 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:

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

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

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

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

Furthermore, a permutation matrix satisfies the system of linear equations:

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

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

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 $G = (X_1, X_2, \dots, X_s; E)$ be a multipartite hypergraph with disjoint vertex sets X_1, X_2, \dots, X_s and with hyperedges $E \subseteq X_1 \times X_2 \times \dots \times X_s$. 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 $x^2 \in X_2, \dots$, to some $x^s \in X_s$. Let $C = (c_{i^1 i^2 \dots i^s})$ be a matrix of size n^s , where $c_{i^1 i^2 \dots i^s}$ is the cost of assigning $x_{i^1}^1$ to $x_{i^2}^2$ to \dots to $x_{i^s}^s$ for all i^1, i^2, \dots, i^s with $1 \leq i^1, i^2, \dots, i^s \leq n$.

Given the assignment problem as a combination of permutations $\varphi_1, \varphi_2, \dots, \varphi_{s-1}$, 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:

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

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

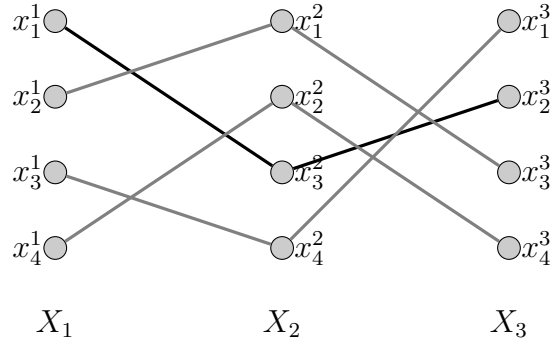


Figure 3.1: Representation of a 3-dimensional assignment as a matching in a multipartite hypergraph.

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

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

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:

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

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

3.1 State of the art

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

Early in the 1970s, [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. In the middle 1970s, [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 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 1990s, [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 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 MAP. The advantage of this generator is that it guarantees the existence of a unique solution. Its main disadvantage falls in its 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 for the multidimensional assignment problem it was created a general classification of the developed techniques. 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 designed aimed to find optimal solutions in shorter running times, any of them will have an expo-

ponential complexity, unless $P = NP$. Even if we have algorithm with an exponential complexity two possible advantages of one over other can be: a lower complexity, for example $O(2^n) < O(3^n) < O(n!)$, and a small multiplicative constant (for small cases of n).

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. They applied 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 $n \in \{4, 6, \dots, 24, 26\}$ vertices and uniformly random generated weights $w(x) \in \{0, \dots, 100\}$, obtaining running times from some seconds to some minutes. The 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 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 shown 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 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 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 techniques developed evaluate several neighborhoods in some predefined way and end when no improvement is obtained.

Some of the first local searches were proposed by [Balas and Saltzman, 1991] for 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 explained in detail in the same work. All the heuristics were tested on instances with $n \in \{20, 25, \dots, 65, 70\}$ vertices and uniformly random generated weights $w(x) \in \{0, \dots, 1000\}$, obtaining running times of less than a minute. The 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 MAP. It considers as part of its machinery four of the 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 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 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 s AP 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 values used 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 the problem sizes with $s = \{3, 4, 5, 6, 7, 8\}$ and with $n = \{150, 50, 30, 18, 12, 8\}$, respectively. The results reported shown a relative solution error of approximately 5% in most of the instances. Dimensionwise variation heuristics were evaluated as the best local searches for MAP.

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 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 results obtained 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 most known family of instances was provided by [Balas and Saltzman, 1991] for 3AP. It includes 60 test instances with the problem size $n \in \{4, 6, \dots, 24, 26\}$. 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.

Other 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 $n \in \{10, 11, \dots, 19, 20\}$; $s = 5$ and $n \in \{7, 8, \dots, 13, 14\}$; $s = 6$ 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 $G = (X_1 \cup \dots \cup X_s, E)$. 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:

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

A set of instances for this family was provided by [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 $scqn$ 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 family of instances Clique. The main difference is that it considers the square root of a sum of squares of the involved weights, as in Equation 3.6.

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

A set of instances for this family was provided by [Karapetyan and Gutin, 2011b]. It considers the same problem sizes as the family of instances Clique. 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 family of instances Clique. In this case, the sets X_1, \dots, 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.

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

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

A set of instances for this family was proposed by [Karapetyan and Gutin, 2011a]. It 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 this family, the weight of a vector x is defined as follows:

$$w_P(x) = \prod_{j=1}^s a_{x_j}^j \quad (3.8)$$

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

A set of instances for this family was proposed by [Karapetyan and Gutin, 2011a]. It considers the same problem sizes as the family of instances Geometric. 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 our state of the art implementation to solve 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 $\{1, \dots, n\}$.

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 complexity of $O(n!^{s-1})$ and it requires a space of $O(n \cdot s)$.

Algorithm 6 shows a generic implementation through recursion of the Brute Force algorithm for the MAP which generates all the possible assignments of a s AP 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 s AP (at least in comparison with more complex techniques).

Algorithm 6: The brute force algorithm for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP of size n .

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

```
1 Set  $A_{n \times s} := 0$ ;  
2 Set  $A^1 := \{1, \dots, n\}$ ;  
3 Set bestAssignment := A;  
4 Call BruteForce(s, A);  
5 BruteForce  $\leftarrow$  function(currentDimension, A){  
6   if currentDimension = 1 then  
7     if calculateCost(bestAssignment) > calculateCost(A) then  
8       Set bestAssignment := A;  
9     return;  
10  Set  $x_{currentDimension} := \{1, \dots, n\}$ ;  
11  Set origin :=  $x$ ;  
12  repeat  
13    Set  $A^{currentDimension} = x_{currentDimension}$ ;  
    //  $A^i$  refers to the column  $i$  of the matrix A  
14    Call BruteForce(currentDimension - 1, A);  
15    Set  $y := next\_permutation(x)$ ;  
16  until  $x = origin$ ;  
17 };  
18 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 reduction from $O(n!^{s-1})$ to $O(n!^{s-2}n^3)$, and the space that it requires changes from $O(n \cdot s)$ to $O(n \cdot s + n^2)$.

Algorithm 7 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 in a similar way as in the dynamic programming solution (proposed simultaneously by [Bellman, 1962] and [Held and Karp, 1962]) for the Traveling Salesman Problem.

Let S_1, \dots, S_s be sets of n vertices each and let $X = S_1 \times S_2 \times \dots \times S_s$ be all members of the Cartesian product between the elements of all the sets. X is composed by vectors from all the possible combinations of vertices from the sets S_i . Lets denote by S_i^k a set of any k vertices of S_i , for $1 \leq i \leq n$, and let $X^k = \{k\} \times S_2^k \times \dots \times S_s^k$ 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 $x^k \in X^k$. The corresponding vertices $x_i^k \in x^k$ should be deleted from each set S_i^k , then an optimal partial solution is calculated for the rest of the vertices. The optimal solution consists in choosing the vector x^k whose sum with its corresponding optimal partial solution is minimum. Then the appropriate subproblem is established.

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

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

In order to define $C(k, S_2^k, \dots, S_s^k)$ in terms of smaller subproblems, one vector $x^k = (k, x_2^k, \dots, x_s^k)$ should be selected such that $x_i^k \in S_i^k$ 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

Algorithm 7: The wise brute force algorithm for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP of size n .

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

```

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

```

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

A recursive function could be stated in the following way:

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

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 balanced. 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 $x_i^k \in S_i^k$ 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 memorization technique we can avoid repeated recursive calls. The 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 8 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 subsets such that the repeated subsets of vertices for the next iterations are avoided. 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 8: The dynamic programming reduction for MAP.

Input: $G = (X_1, \dots, X_s; E)$. 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  $n \times s$ ;
2 Set bitSetS :=  $(1 \ll (n \times s)) - 1$ ;
   // This turned on all the bits of bitSetS
3 Set result := DynamicProgrammingReduction(bitSetS);
4 DynamicProgrammingReduction  $\leftarrow$  function(bitSetS){
5   if bitSetS = 0 then
6     return  $\{0, \emptyset\}$ ;
7   Set bestCost :=  $\infty$ ;
8   Set bestPartialA :=  $\emptyset$ ;
   // The function AvailableTuples extract all the available tuples
   // considering the turned on bits from the given bitset
9   for  $x \in AvailableTuples(bitSetS)$  do
10    Turn off the corresponding bits of bitSetS according to the elements in  $x$ ;
11    Set result := DynamicProgrammingReduction(bitSetS);
12    if result.cost + cost( $x$ ) < bestCost then
13      Set bestCost := result.cost;
14      Set bestPartialA := result.assignment  $\cup$   $x$ ;
15   return  $\{bestCost, bestPartialA\}$ ;
16 };
17 return result.assignment;
```

3.3.4 MAP-Gurobi

The Gurobi Optimizer is a commercial optimization solver for linear programming, quadratic programming, quadratically constrained programming, mixed-integer linear programming, mixed-integer quadratic programming, and mixed integer quadratically constrained programming. 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. Image 3.2 shows the general Application Programming Interface (API) of Gurobi. Even when Gurobi Optimizer is a commercial software, it is easy to get a free academic version for purposes of research [Gurobi Optimization, 2017a].

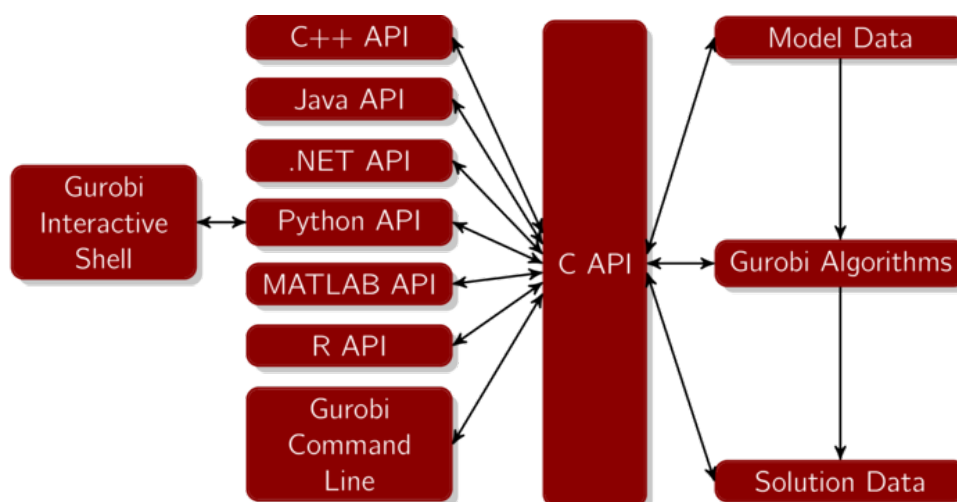


Figure 3.2: Application Programming Interface (API) of Gurobi.

We use Gurobi to specifically solve our 0-1 integer linear programming formulation provided in Equation 4.1. Gurobi performs the next steps for solving linear programming [Gurobi Optimization, 2016] formulations:

1. Pre solve process. The goal is to reduce the problem size. In this step, redundant constraints are eliminated and it is applied the substitution of variables.
2. Resolution with primal and dual simplex method. The goal is to solve the model. First, Karush-Kuhn-Tucker [Kuhn and Tucker, 1951] (KKT) conditions for linear programming optimality are verified. Then, the model is partitioned into basic and non-basic variables, non-basic structural variables corresponds to tight bounds whereas non-basic slack variables correspond to tight constraints. Next, the model is solved for basic variables and for non-basic is applied the next process:
 - (a) Pick a non basic variable to enter the basis. A variable with a negative reduced cost should be picked.

- (b) Push one variable out of the basis.
 - (c) Update primal and dual variables, reduced costs, basis, basis factors, etcetera.
 - (d) Apply simplex pivots until no more negative reduced cost variables exist. This yields optimality.
3. Apply parallel Barrier method with crossover. The goal is also to solve the model. An interior point method (also referred as Barrier method) is applied. This method is a class of algorithm to solve linear and nonlinear convex optimization problems. In this case, the Karmarkar's algorithm [Karmarkar, 1984] is applied. It consists on modifying the KKT conditions. For simplicity it considers the all-inequality version of a nonlinear optimization problem:

$$\begin{aligned} & \min && f(x) \\ & \text{subject to :} && \\ & && c_i(x) \geq 0 \quad \text{for } i \in \{1, \dots, m\} \end{aligned} \tag{3.10}$$

Then, a so called logarithmic Barrier function (see Equation 3.11) is associated with the model.

$$B(x, \mu) = f(x) - \mu \sum_{i=1}^m \log(c_i(x)) \tag{3.11}$$

Here μ is a small positive scalar. As μ converges to zero the minimum of $B(x, \mu)$ should converge to a solution of the model.

As we can see, Gurobi applies both simplex method and parallel Barrier simultaneously. The optimal solution is provided by the first technique that finishes. The Gurobi reference manual can be found at [Gurobi Optimization, 2017b].

We called MAP-Gurobi to our implementation of our 0-1 integer linear programming formulation. Algorithm 9 shows the implemented model. First a Gurobi environment object is created. Then the environment object is associated with a new Gurobi model. At line 3, we have to declare a Gurobi variable p that associates the multipartite hypergraph G with a s -dimensional binary matrix p . Since the number of dimensions s can vary, we decided to consider p as a vector of size n^s to keep the values of all the variables. The function `AllocateMemoryGRBVar` allows to assign the required memory for p based on G . At line 5, the objective function is defined and it is assigned to a Gurobi expression. At line 6, the model is defined to be minimized. The main cycle at the line 7 allows to add all the restrictions to the model. Finally the model is solved and the assignment is retrieved according to the turned on variables in p .

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. Due to the internal functionality of Gurobi, it is difficult to provide the complexity and the space required for this algorithm, however we decided to perform an experimental

Algorithm 9: The MAP-Gurobi algorithm for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP of size n .

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

```

1 Set GRBEnv env := GRBEnv();
  // Gurobi enviroment object
2 Set GRBModel model := GRBModel(env);
  /* Create variables */
3 Set GRBVar p := NULL;
4 AllocateMemoryGRBVar(p, G, model);
5 Set GRBLinExpr objectiveFuntion := GenerateObjectiveFunction(G, p);
6 Set model.setObjective(objectiveFuntion, GRB_MINIMIZE);
  /* Add constraints */
7 for s in 1:G.getDimensions() do
8   for i in 1:G.getSize(s) do
9     Set GRBLinExpr expr := addRestriction(G, s, i, p);
10    Set model.addConstr(expr == 1);
  /* Optimize model */
11 Set model.optimize();
12 Set assignment := getGurobiSolution(G, p, model);
13 return assignment;
```

evaluation in order to provide a fair approach for it. Two 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 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 Experimental evaluation

The three first algorithms were implemented specifically for the 3AP. Table 3.1 shows the results for Balas-Saltzman family of instances solved by these algorithms. The brute force algorithm becomes intractable for $n > 8$, the wise brute force algorithm for $n > 10$ and the dynamic programming reduction for $n > 14$. Even when these algorithms are exponential, there is a considerable difference between the complexity of each algorithm. Figures 3.3, 3.4, and 3.5 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 grow of the curves. In particular, for 5AP it can be observed that the behavior of the brute

force algorithm and the wise brute force algorithm tend 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.

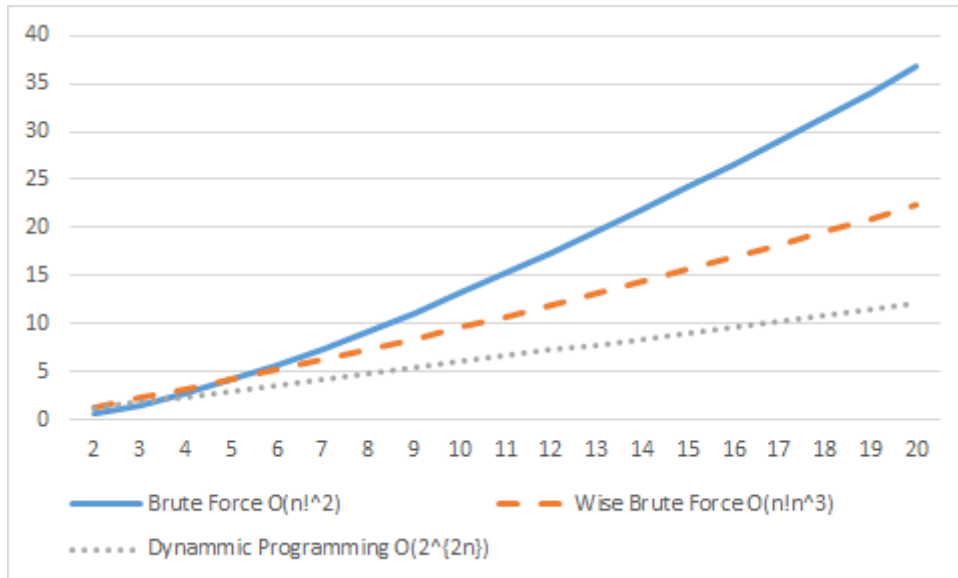


Figure 3.3: Complexity curves (in logarithmic scale) of exact algorithms for 3AP.

In order to evaluate the performance of the MAP-Gurobi implementation several 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 *sbsn* ([Balas and Saltzman, 1991]). 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 in opposite with the first three algorithms. 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.

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 because we did not implement their algorithm. The 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

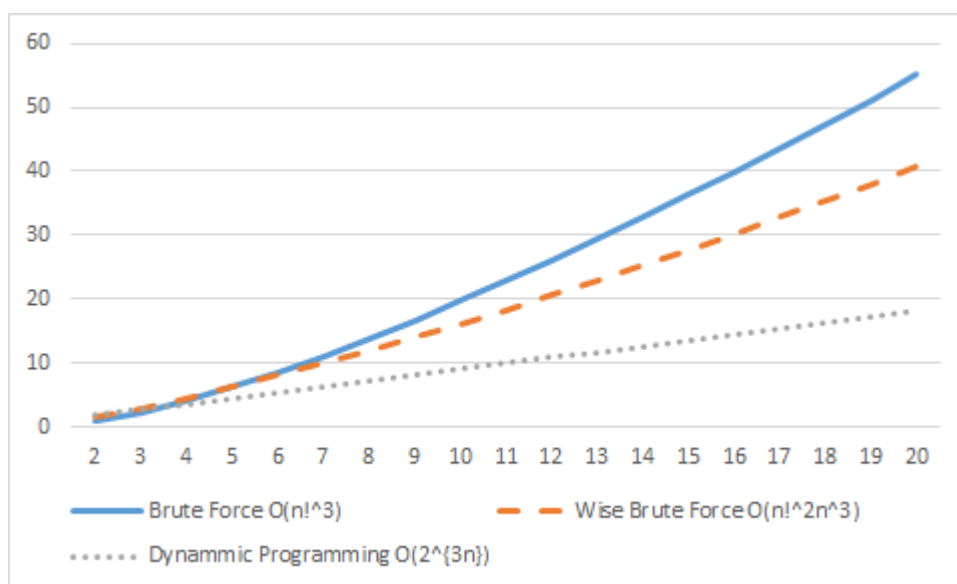


Figure 3.4: Complexity curves (in logarithmic scale) of exact algorithms for 4AP.

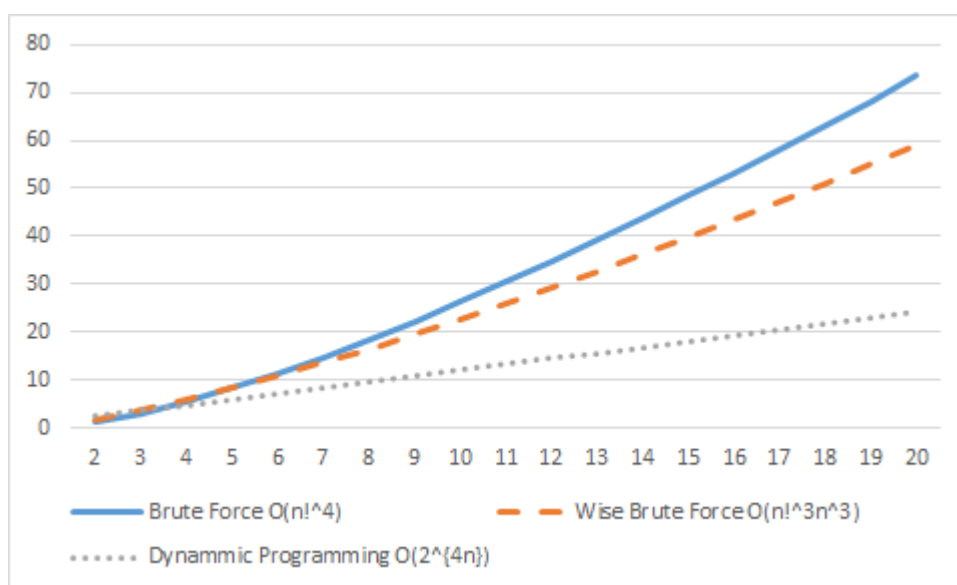


Figure 3.5: Complexity curves (in logarithmic scale) of exact algorithms for 5AP.

the number of vertices is one. For the family of instances $4axialn$ 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 $5axialn$ 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 ins-

Table 3.1: Computational results for the family of instances of Balas and Saltzman under exact algorithms

Instance	Optimum	Variables	Brute force	Wise brute force	Dynamic programming	MAP-Gurobi
3_bs_4	42.2	64	0.0	0.1	0.1	0.1
3_bs_6	40.2	216	0.1	0.1	0.1	0.1
3_bs_8	23.8	512	60.0	0.4	0.1	0.1
3_bs_10	19.0	1000	-	43.4	0.1	0.1
3_bs_12	15.6	1728	-	-	2.0	0.1
3_bs_14	10.0	2744	-	-	48.1	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

tances $6axialn$ 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 $s \in \{4, 5\}$.

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 the best known 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 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 $saxialn$ was good enough to obtain the required complexity approach.

In conclusion, considering the running times for the family of instances $saxialn$ a fair approximation to the 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 hand, according to the running times for the families of instances provided by [Karapetyan and Gutin, 2011b] it 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

Table 3.2: Computational results for the family of instances *saxialn* under Magos' algorithm and MAP-Gurobi

Instance	Optimum	Variables	Magos' algorithm		MAP Gurobi	
			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	-

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 $18!^4 < 30!^3$.

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 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

Table 3.3: Computational results for the family of instances Random under 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	-	-

Table 3.4: Computational results for the family of instances Clique under MAP-Gurobi

s	n	Variables	Previous best known	Optimum found	Seconds
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	-	-

MAP and this is why we study them extensively in this work.

Each of the next heuristics requires an initial feasible solution $A = (x^1, x^2, \dots, x^n)$ generated somehow. 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 an artificial instance with $s = 4$ dimensions and $n = 5$ vertices in order to illustrate the next heuristics to describe. Let e_i^j be the i -th element

Table 3.5: Computational results for the family of instances Square Root under 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	-	-

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 $w(e_1^1, e_1^2, e_1^3, e_1^4) = 1$, $w(e_1^1, e_1^2, e_1^3, e_2^4) = 2$, \dots , $w(e_5^1, e_5^2, e_5^3, e_5^4) = 625$. For simplicity the vectors $(e_a^1, e_b^2, e_c^3, e_d^4)$ will be written as (a, b, c, d) since the dimension to which the vertex belongs can be deduced 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 swap

Our simple swap heuristic is based on the 2-opt technique 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, this 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 $i, j \in \{1, \dots, n\}$ with $i < j$. If the new solution is better than the previous one then the feasible solution is updated with the new vectors.

Algorithm 10 shows the pseudo-code of the simple swap 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 10: The simple swap heuristic for MAP.

Input: $G = (X_1, \dots, X_s; E)$. 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  $A := \text{FeasibleRandomGeneratedSolutionForMAP}(G)$ ;
2 Set  $improved := \text{true}$ ;
3 Set  $iterations := 1$ ;
4 while  $improved = \text{true}$  and  $iterations \leq iter$  do
5     Set  $improved := \text{false}$ ;
6     Set  $iterations := iterations + 1$ ;
7     foreach  $d$  in  $\{1, \dots, s\}$  do
8         foreach  $i = 1$  to  $n$  do
9             foreach  $j = i + 1$  to  $n$  do
10                Set  $A' := A$ ;
11                 $swap(x_i^d, x_j^d)$  from  $A'$ ;
12                if  $assignmentCost(A') < assignmentCost(A)$  then
13                    Set  $improved := \text{true}$ ;
14                    Set  $A := A'$ ;
15 return  $\{A\}$ ;

```

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

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

The corresponding weights are:

$$\left[\begin{array}{l} 24 \rightarrow 60 \\ 120 \rightarrow 48 \end{array} \right].$$

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

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

The search space for one iteration of this local search is $O(sn^2)$ which is too small in comparison with the whole search space ($O(n!^{s-1})$). It can be a disadvantage because the number of neighbors evaluated at each iteration is small. However, the main advantage of this heuristic is that it can move from any assignment to any other through some path of simple swaps. In general, it is easy to show that we can start at any assignment and to move to any other by applying an algorithm of depth first search (DFS) or breadth first search (BFS) that explores the whole search space through simple swaps. It is obvious that such algorithms will be exponential but the point is to expose that even when this local search is a bit simple it can be able to reach high quality solutions.

3.4.1.2 Inversion

The inversion heuristic is a generalization of the simple swap, 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 vertices indexed by $i, j \in \{1, \dots, n\}$ with $i < j$. The sub-array of vertices $x_i^d, x_{i+1}^d, \dots, x_{j-1}^d, x_j^d$ is reversed as $x_j^d, x_{j-1}^d, \dots, x_{i+1}^d, x_i^d$ 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 11 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.

Algorithm 11: The inversion heuristic for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP 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  $A := \text{FeasibleRandomGeneratedSolutionForMAP}(G)$ ;
2 Set  $\text{improved} := \text{true}$ ;
3 Set  $\text{iterations} := 1$ ;
4 while  $\text{improved} = \text{true}$  and  $\text{iterations} \leq \text{iter}$  do
5   Set  $\text{improved} := \text{false}$ ;
6   Set  $\text{iterations} := \text{iterations} + 1$ ;
7   foreach  $d$  in  $\{1, \dots, s\}$  do
8     foreach  $i = 1$  to  $n$  do
9       foreach  $j = i + 1$  to  $n$  do
10        Set  $A' := A$ ;
11         $\text{reversingVertices}(x_i^d, x_j^d)$  from  $A'$ ;
12        if  $\text{assignmentCost}(A') < \text{assignmentCost}(A)$  then
13          Set  $\text{improved} := \text{true}$ ;
14          Set  $A := A'$ ;
15 return  $\{A\}$ ;

```

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

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

The corresponding weights are:

$$\left[\begin{array}{l} 24 \rightarrow 12 \\ 18 \rightarrow 30 \\ 120 \rightarrow 72 \\ 8 \rightarrow 16 \end{array} \right].$$

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

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

The search space for one iteration of this heuristic is also $O(sn^2)$ but in comparison with the simple swap heuristic it explores a different neighborhood. Whereas this heuristic is able to perform changes that can affect at all the vectors, the simple swap heuristic performs changes that only affects at two vectors.

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 applying it 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 $i, j \in \{1, \dots, n\}$ with $i < j$ and rotate the sub-array of vertices $x_i^d, x_{i+1}^d, \dots, x_{j-1}^d, x_j^d$, 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 $x_{i+1}^d, x_{i+2}^d, \dots, x_j^d, x_i^d$. Each time that the new solution is better than the previous one, the feasible solution is updated with the new vectors.

Algorithm 12 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 complexity of 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.

For example, consider the feasible solution A , for our artificial 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:

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

Algorithm 12: The circular rotation heuristic for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP 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                Set A' := A;
11                Set x_orig := x_j^d;
12                repeat
13                    rotateVertices(x_i^d, x_j^d) from A';
14                    // either left rotation or right rotation
15                    if assignmentCost(A') < assignmentCost(A) then
16                        Set improved := true;
17                        Set A := A';
18                until x_orig ≠ x_i^d;
19 return {A};

```

The corresponding weights are:

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

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

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

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, the direction can be any, what survives is the best configurations among all the rotations.

The search space for one iteration of this heuristic is $O(sn^3)$.

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 applying it 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 $i, i + k - 1 \in 1, \dots, n$, that is $x_i^d, x_{i+1}^d, \dots, x_{i+k-1}^d, x_{i+k}^d$, 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 13 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.

For example, consider the feasible solution A , for our artificial 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:

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

The corresponding weights are:

$$\left[\begin{array}{cccccc} 24 \rightarrow 24, & 24, & 36, & \mathbf{36}, & 60, & 60 \\ 18 \rightarrow 18, & 30, & 12, & \mathbf{30}, & 12, & 18 \\ 120 \rightarrow 120, & 72, & 120, & \mathbf{48}, & 72, & 48 \end{array} \right].$$

Algorithm 13: The k -vertex permutation heuristic for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP 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  $A := \text{FeasibleRandomGeneratedSolutionForMAP}(G)$ ;
2 Set improved := true;
3 Set iterations := 1;
4 while improved = true and iterations  $\leq$  iter do
5     Set improved := false;
6     Set iterations := iterations + 1;
7     foreach  $d$  in  $\{1, \dots, s\}$  do
8         foreach  $i = 1$  to  $n$  do
9             foreach  $j = i + 1$  to  $n$  do
10                Set  $A' := A$ ;
11                Set  $x_{orig} := x_j^d$ ;
12                foreach Permutation  $P$  of the vertices  $x_i^d, \dots, x_j^d$  do
13                    Replace the vertices  $x_i^d, \dots, x_j^d$  from  $A'$  with the
                    corresponding permutation  $P$ ;
14                    if assignmentCost( $A'$ ) < assignmentCost( $A$ ) then
15                        Set improved := true;
16                        Set  $A := A'$ ;
17 return  $\{A\}$ ;
```

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

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

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 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. Initially, [Huang and Lim, 2006] applied this technique for 3AP and in the work of [Karapetyan and Gutin, 2011a] they were extended 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 $F \subset \{1, \dots, s\}$ 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:

$$v_{i,j}^d = \begin{cases} x_i^d & \text{if } d \in F \\ x_j^d & \text{if } d \notin F \end{cases} \quad \text{for } 1 \leq d \leq s . \quad (3.12)$$

The corresponding 2AP can be solved in $O(n^3)$ time and gives a local optimum for the corresponding neighborhood. In the works proposed by [Huang and Lim, 2006] and [Karapetyan and Gutin, 2011a] was proposed to use the Hungarian method, however we suggest to use the ϵ -scaling Auction algorithm or an even faster algorithm for it.

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 14 shows the pseudo-code of a dimensionwise variation heuristic. A complete iteration of this heuristic consists in trying every possible non empty subset $F \in \wp(\{1, \dots, s\})$. 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 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.

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

$$\left[\begin{array}{ccccc} (1, 2, 3, 4) & (2, 2, 1, 4) & (\mathbf{3}, \mathbf{2}, \mathbf{4}, \mathbf{4}) & (4, 2, 2, 4) & (5, 2, 5, 4) \\ (1, 3, 3, 3) & (2, 3, 1, 3) & (3, 3, 4, 3) & (\mathbf{4}, \mathbf{3}, \mathbf{2}, \mathbf{3}) & (5, 3, 5, 3) \\ (\mathbf{1}, \mathbf{5}, \mathbf{3}, \mathbf{2}) & (2, 5, 1, 2) & (3, 5, 4, 2) & (4, 5, 2, 2) & (5, 5, 5, 2) \\ (1, 1, 3, 1) & (2, 1, 1, 1) & (3, 1, 4, 1) & (4, 1, 2, 1) & (\mathbf{5}, \mathbf{1}, \mathbf{5}, \mathbf{1}) \\ (1, 4, 3, 5) & (\mathbf{2}, \mathbf{4}, \mathbf{1}, \mathbf{5}) & (3, 4, 4, 5) & (4, 4, 2, 5) & (5, 4, 5, 5) \end{array} \right] \rightarrow$$

Algorithm 14: The dimensionwise variation heuristic for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP 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  $A := \text{FeasibleRandomGeneratedSolutionForMAP}(G)$ ;
2 Set  $\text{improved} := \text{true}$ ;
3 Set  $\text{bestCost} := \text{assignmentCost}(A)$ ;
4 Set  $\text{iterations} := 1$ ;
5 while  $\text{improved} = \text{true}$  and  $\text{iterations} \leq \text{iter}$  do
6     Set  $\text{improved} := \text{false}$ ;
7     Set  $\text{iterations} := \text{iterations} + 1$ ;
8     foreach  $F \in \{\wp(\{1, \dots, s\}) \setminus \emptyset\}$  do
9         By considering the Equation 3.12, obtain the matrix  $M_{n \times n}$  for the
10        corresponding 2AP;
11        Set  $A' := \text{SolutionFor2AP}(M_{n \times n})$ ;
12        // We suggest the  $\epsilon$ -scaling Auction algorithm
13        if  $\text{assignmentCost}(A') < \text{bestCost}$  then
14            Set  $\text{improved} := \text{true}$ ;
15            Set  $\text{bestCost} := \text{assignmentCost}(A')$ ;
16            Map the solution  $A'$  to the corresponding feasible solution onto  $A$ ;
17 return  $\{A\}$ ;

```

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

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 s AP to a 2AP can be applied for any s AP with $s \geq 3$. Here we propose a generalization of this heuristic which consists in reducing a s AP to a t AP 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, \dots, F_{t-1} non empty proper subsets of $\{1, \dots, s\}$ such that $F_1 \cap \dots \cap F_{t-1} = \emptyset$ and $F_1 \cup \dots \cup F_{t-1} \subset \{1, \dots, s\}$. At one step of the heuristic all the dimensions but $F_1 \cup \dots \cup F_{t-1}$ are fixed and a t -dimensional matrix M^t of size $n^1 \times \dots \times n^t$ (recall the simplification $n^i = n$ for $1 \leq i \leq t$) with entries $M_{i^1, \dots, i^t}^d = w(v_{i^1, \dots, i^t}^d)$ is generated. Let v_{i^1, \dots, i^t}^d denote the d -th element of the vector v_{i^1, \dots, i^t} , all the vectors are built according to the next function:

$$v_{i^1, i^2, \dots, i^{t-1}, i^t}^d = \begin{cases} x_{i^1}^d & \text{if } d \in F_1 \\ x_{i^2}^d & \text{if } d \in F_2 \\ \dots & \\ x_{i^{t-1}}^d & \text{if } d \in F_{t-1} \\ x_{i^t}^d & \text{otherwise} \end{cases} \quad \text{for } 1 \leq d \leq s. \quad (3.13)$$

The corresponding t AP instance can be solved by using some exact technique, in this case with the MAP-Gurobi. Algorithm 15 shows the pseudo-code of the generalized dimensionwise variation heuristic. A complete iteration tries every possible combination of t subsets of $\{1, \dots, s\}$. 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.

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 our artificial instance and pick $t = 3$ and the sets $F_1 = \{2, 4\}, F_2 = \{3\}$ such that $F_1 \cup F_2 \subset$

Algorithm 15: The generalized dimensionwise variation heuristic for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP 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  $F_1, \dots, F_{t-1} \in \{CombinationSets(\{1, \dots, s\})$  do
9         By considering the Equation 3.15, obtain the matrix  $M^t$  for the
            corresponding  $t$ AP;
10        Set  $A' := \text{MAP-Gurobi}(M^t)$  ;
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  $\{A\}$ ;

```

$\{1, \dots, s\}$, then M^3 is obtained as:

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

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

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

$$\begin{array}{c}
\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, 5, 1, 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, 1, 5, 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, 4, 3, 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}
\end{array}$$

In this example, we can observe that the vertices of the dimensions in the set $\{1, \dots, s\} \setminus \{F_1 \cup F_2\}$ 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 t AP 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 s AP to a t AP and considers $|F_i| = 1$ for all $1 \leq i < t$. We called DVt a dimensionwise variation heuristic that reduces a s AP to a t AP 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 s AP 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 s AP subproblem with k vertices on each dimension. The corresponding s AP 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 16 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 complexity of $O(\binom{n}{k}k!^{s-1})$. By applying Gurobi-MAP an iteration has a complexity of $O(\binom{n}{k}(s-1)^k)$. As in the case for DVH a bound of ten iterations is fixed.

Algorithm 16: The k -opt heuristic for MAP.**Input:** $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP 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  $A := \text{FeasibleRandomGeneratedSolutionForMAP}(G)$ ;
2 Set  $\text{improved} := \text{true}$ ;
3 Set  $\text{iterations} := 1$ ;
4 while  $\text{improved} = \text{true}$  and  $\text{iterations} \leq \text{iter}$  do
5   Set  $\text{improved} := \text{false}$ ;
6   Set  $\text{iterations} := \text{iterations} + 1$ ;
7   foreach combination  $P$  of vectors in  $\binom{n}{k} \in A$  do
8     Set  $P' := \text{BruteForce}(P)$ ; // MAP-Gurobi( $P$ ) is suggested for
       $k \geq 4$ 
9     if  $\text{assignmentCost}(P') < \text{assignmentCost}(P)$  then
10      Set  $\text{improved} := \text{true}$ ;
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 our artificial 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:

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

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:

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

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 us 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 order of execution is: simple swap, 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, \dots, F_{r-1} non empty proper subsets such that $F_1 \cap \dots \cap F_{r-1} = \emptyset$ and $F_1 \cup \dots \cup F_{r-1} \subset \{1, \dots, s\}$. 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 hold:

$$\begin{aligned} 2 &\leq r \leq s \\ 2 &\leq k \leq n \\ r + k &< s + n \end{aligned} \quad . \quad (3.14)$$

At one step of this heuristic all the dimensions but $F_1 \cup \dots \cup F_{r-1}$ 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 $k^1 \times \dots \times k^r$ (recall the simplification $k^i = k$ for $1 \leq i \leq r$) with entries $M_{i^1, \dots, i^r} = w(v^{i^1, \dots, i^r})$ is generated. Let v_{i^1, \dots, i^r}^d denote the d -th element of the vector v_{i^1, \dots, i^r} , all the vectors are built according to the next function:

$$v_{i^1, i^2, \dots, i^{r-1}, i^r}^d = \begin{cases} x_{i^1}^d & \text{if } d \in F_1 \\ x_{i^2}^d & \text{if } d \in F_2 \\ \dots & \\ x_{i^{r-1}}^d & \text{if } d \in F_{r-1} \\ x_{i^r}^d & \text{otherwise} \end{cases} \quad \text{for } 1 \leq d \leq s . \quad (3.15)$$

The corresponding r AP instance with k vertices at each dimension can be solved using some exact technique, again, we suggest the MAP-Gurobi implementation. The search space of this heuristic is $O(k^{r-1}!)$.

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. A possible advantage of GLSH over GDVH is its flexibility. Whereas GLSH can consider any subset of vectors of A , GDVH always considers the complete list of vectors of A . 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 either 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 tuning 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]. This heuristic and some results obtained were published in a work entitled *A new Local Search Heuristic for the Multidimensional Assignment Problem* [Pérez Pérez et al., 2017a].

3.4.7 Multi-start local search

A multi-start local search (MLS) is a strategy in which one can start multiple instances of a local search, and allocate computational resources, e. g. processing time, to the instances depending on their behavior. This multi-strategy has to decide when to allocate additional resources to a particular instances and when to start new instances. This heuristic is explained in detail in [Kocsis and György, 2009]. We decided to implement a very simple idea for this strategy.

Algorithm 17 shows the structure of an MLS for MAP. We designed our MLS to be executed for a predefined time. First, a feasible solution *bestSolution* will be generated at random. It will be our first best solution. The main loop will be executed for the required time. At each iteration, a new feasible solution A will be generated at random and it will be improved through some local search. If the improved solution A is better than *bestSolution*, then the latter solution will be updated.

We decided to evaluate all the described local searches in combination with MLS.

Algorithm 17: The multi-start local search for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP of size n .
 maxSec . Number of seconds for running the MLS.

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

```

1 Set bestSolution := FeasibleRandomGeneratedSolutionForMAP(G);
2 Set seconds := 0;
3 while seconds < maxSec do
4   Set A := FeasibleRandomGeneratedSolutionForMAP(G);
5   Set A := LocalSearchForMAP(A);
6   if assignmentCost(A) < assignmentCost(bestSolution) then
7     Set bestSolution := A;
8   Set seconds := updateRunningTime();
9 return {bestSolution};

```

3.4.8 Iterated local search

An Iterated Local Search (ILS) is a modification of a local search for solving optimization problems. A local search can get into a local minimum such that, in some cases, it can not be performed an improvement through the available neighbors. An ILS consists on performing a soft random perturbation once the local minimum is reached, then the local search is applied again. The idea is modify the local minimum reached in order to move into a new neighborhood which will not be so different from the previous one.

Algorithm 17 shows the structure of an ILS for MAP. We designed our ILS to be executed for a predefined time. First, a feasible solution A will be generated at random. It will be our first best solution. The main loop will be executed for the required time. At each iteration, A will be improved through some local search. If the improved solution A is better than bestSolution , then the latter solution will be updated. The improved solution A will be perturbed and it will be used for the next iteration.

We decided to evaluate all the described local searches, with the exception of SDV3 and DV3, in combination with this technique. We decided to exclude SDV3 and DV3 of our analysis because an ILS requires several iterations to obtain higher quality solutions. Furthermore, SDV3 and DV3 are very expensive in terms of computational time. We believe that the combination of ILS with SDV3 and DV3 can be more competitive if the execution time of SDV3 and DV3 is reduced. We let such evaluation as future work.

3.4.9 Experimental evaluation

Several local search heuristics were compared in our experiments:

Algorithm 18: The iterated local search for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP of size n .
 maxSec . Number of seconds for running the ILS.

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

```

1 Set bestSolution := FeasibleRandomGeneratedSolutionForMAP(G);
2 Set seconds := 0;
3 while seconds < maxSec do
4   Set A := LocalSearchForMAP(A);
5   if assignmentCost(A) < assignmentCost(bestSolution) then
6     Set bestSolution := A;
7   Set A := Perturbate(A);
8   Set seconds := updateRunningTime();
9 return {bestSolution};

```

1. Basic heuristics. We implemented the simple swap, 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.
5. Multi-start local searches. We implemented a multi-start local search combined with all our basic heuristics, the k -opt heuristics, and the SDV2 and the DV2 proposed by [Karapetyan and Gutin, 2011a].
6. Iterated local searches. We implemented an iterated local search combined with all our basic heuristics, the k -opt heuristics, and the SDV2 and the DV2 proposed by [Karapetyan and Gutin, 2011a].

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 they 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 in 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:

$$\frac{w(A) - w(opt)}{w(opt)} \times 100 \quad (3.16)$$

Each local search was included as part of a multi-start local search which consists on executing them a certain number of times z , instead of some execution time, 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.

3.4.9.1 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 Geometric family of instances.

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 swap is the most effective heuristic among the basic heuristics. However, all the results are pretty far away from the optimal value, except for the family of instances Geometric under the simple swap, 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

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

Instance	Best known	Simple swap	Inversion	Circular rotation	6-vertex permutation	Basics 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	344.0	175.0
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	6089.0	5256.9	3668.5
4cq50	3032.6	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	4671.7	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	3696.4	4450.7	6008.0	6091.9	5406.5	4391.4
RSE	-	20.4	62.5	64.8	46.3	18.8
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	1491.1	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	1828.1	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	1432.8	1807.5	2425.1	2452.9	2181.1	1778.7
RSE	-	26.2	69.3	71.2	52.2	24.1

instances under the basic heuristics. The basic combined heuristic provides the lowest relative solution error for this techniques followed by the simple swap.

Table 3.9 shows the running times for all the basic heuristics. The k -vertex permutation heuristic and, in consequence, the basics combined heuristic are computationally the most expensive heuristics.

Table 3.7: Geometric and Product under basic heuristics.

Instance	Best known	Simple swap	Inversion	Circular rotation	6-vertex permutation	Basics combined
4g20	2380.5	2383.2	3079.0	3173.0	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	4815.0	4970.4	4578.9	3806.2
5g25	4594.9	4615.4	6620.4	6827.4	6286.7	4611.6
5g30	5036.8	5078.1	7813.0	8066.9	7443.1	5072.4
6g12	4483.7	4487.3	5148.2	5262.6	4920.0	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	13159.0	13237.2	13284.8	13241.1	13159.0
4p40	16810.0	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	29693.0	43046.5	43517.1	43928.8	43390.9	43051.0
5p30	34799.4	50234.6	50993.1	51318.1	52256.7	50232.8
6p12	7421.0	30578.3	32048.7	33045.3	32400.1	30436.8
6p15	8888.9	39227.4	42392.0	42291.5	42485.3	39115.5
6p18	9600.2	47585.2	52434.2	51415.7	52785.9	47384.8
Avg	17660.3	29691.1	30832.8	30911.0	31054.2	29649.9
RSE	-	68.1	74.6	75.0	75.8	67.9

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

Family of instances	Simple swap	Inversion	Circular rotation	6-vertex permutation	Basics combined
Random	193.5	615.3	661.7	344.0	175.0
Clique	20.4	62.5	64.8	46.3	18.8
SquareRoot	26.2	69.3	71.2	52.2	24.1
Geometric	0.4	40.4	43.9	33.2	0.4
Product	68.1	74.6	75.0	75.8	67.9

3.4.9.2 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 3.10 and 3.11 show the experimental results for the k -opt heuristics. It can be observed that the solutions obtained for the k -opt heuristics have a simi-

Table 3.9: Running times for Random, Clique, Square Root, Geometric, and Product under basic heuristics.

Inst.	Simple swap	Invers.	Circ. rot.	6-vertex perm.	Basics comb.	Inst.	Simple swap	Invers.	Circ. rot.	6-vertex perm.	Basics comb.
4r20	0.0	0.0	0.3	3.9	4.2	4g20	0.1	0.1	0.5	6.2	6.2
4r30	0.2	0.2	1.6	9.5	10.9	4g30	0.2	0.2	1.7	11.8	12.5
4r40	0.4	0.4	4.4	15.5	20.7	4g40	0.4	0.4	4.5	17.9	21.9
4r50	1.4	1.2	11.5	30.3	44.1	4g50	0.9	0.8	10.0	28.7	37.8
5r15	0.0	0.0	0.2	3.1	2.9	5g15	0.0	0.0	0.2	4.3	3.7
5r18	0.1	0.1	0.4	4.7	4.7	5g18	0.1	0.1	0.4	6.2	5.7
5r25	0.7	0.6	1.6	9.5	10.4	5g25	0.6	0.6	1.7	12.8	12.5
5r30	3.7	2.5	4.9	23.5	26.6	5g30	1.5	1.5	3.5	20.1	20.5
6r12	0.2	0.2	0.2	2.7	2.4	6g12	0.2	0.2	0.3	4.0	3.5
6r15	0.8	0.8	1.0	6.0	5.9	6g15	0.7	0.7	0.9	7.4	6.6
6r18	2.5	2.4	2.9	10.8	10.6	6g18	2.5	1.8	2.3	12.5	11.4
Avg	0.9	0.8	2.6	10.9	13.0	Avg	0.6	0.5	2.3	11.9	12.9
4cq20	0.3	0.2	0.7	7.9	8.0	4p20	0.1	0.1	0.4	5.2	4.4
4cq30	0.6	0.3	2.6	17.1	19.4	4p30	0.2	0.2	1.7	12.4	10.6
4cq40	0.9	0.7	7.2	26.8	37.1	4p40	0.5	0.5	5.1	20.6	18.6
4cq50	1.8	1.5	17.3	46.3	70.9	4p50	2.4	1.1	10	40.8	30.1
5cq15	0.1	0.0	0.2	4.1	3.9	5p15	0.3	0.2	0.4	6.0	5.0
5cq18	0.1	0.1	0.5	6.2	6.3	5p18	0.4	0.3	0.8	9.2	7.4
5cq25	0.6	0.6	1.9	12.7	13.9	5p25	1.3	1.2	2.6	19.3	15.2
5cq30	2.9	2.2	5.2	28.2	32.2	5p30	2.7	2.6	5.5	28.8	23.7
6cq12	0.2	0.1	0.3	3.9	3.4	6p12	0.4	0.3	0.4	5.2	4.2
6cq15	0.7	0.6	0.8	7.2	6.7	6p15	1.3	1.0	1.3	9.3	7.5
6cq18	1.9	1.8	2.2	11.5	11.3	6p18	4.6	3.0	3.4	14.9	12.7
Avg	0.9	0.7	3.5	15.6	19.3	Avg	1.3	1.0	2.9	15.6	12.7
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	17.0	23.4						
4sr50	1.9	1.4	17.7	46.4	70.8						
5sr15	0.0	0.0	0.2	4.0	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						
Avg	0.8	0.6	3.2	13.6	16.9						

lar quality solution independently of the implementation, either through brute force as in [Karapetyan and Gutin, 2011a] or through our MAP-Gurobi, however MAP-Gurobi is computationally 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 -dimensional matrix and to build a model for the MAP-Gurobi, which is a process expensive computation-

ally. 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 they are still far from optimal solutions, but not in the case of the family of instances Geometric, in which case 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 greater 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 3.12 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.

Table 3.13 shows the corresponding running times for the k -opt heuristics. It can be observed that running times for the 2-opt and 3-opt heuristics of MAP-Gurobi version are high whereas running times of brute force version are relatively shorter.

3.4.9.3 Computational results for dimensionwise variation heuristics

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

Tables 3.14 and 3.15 show the experimental results for some versions of the DVH. We can observe that by combining DV2 + 3-opt the quality of the results obtained are increased considerably, such that the relative solution error is lower than 10% for all the families of instances. By the other hand, 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 s AP to a $(s - 1)$ AP can even provide higher quality solutions, however we let such analysis as future work.

Table 3.16 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 such analysis as future work.

Table 3.10: Random, Clique and SquareRoot under k -opt heuristics.

Instance	Best known	2-opt (GK)	3-opt (GK)	2-opt MAP-Gurobi	3-opt MAP-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	-	106.0	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	3032.6	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	4671.7	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	3696.4	4369.7	3979.1	4366.0	3973.6
RSE	-	18.2	7.6	18.1	7.5
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	1491.1	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	1828.1	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	1432.8	1764.2	1579.0	1759.0	1577.0
RSE	-	23.1	10.2	22.8	10.1

Table 3.17 shows 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 $2AP$. We claim that the computational cost in comparison with the benefit obtained is excellent because for some families of instances our DV3 is able to find the optimal solutions and, for the rest of families,

Table 3.11: Geometric and Product under k -opt heuristics.

Instance	Best known	2-opt (GK)	3-opt (GK)	2-opt MAP-Gurobi	3-opt MAP-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	29693.0	37232.6	30475.0	35341.0	30508.1
5p30	34799.4	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	9600.2	17181.3	11379.1	16506.6	10411.9
Avg	17660.3	21345.3	18245.3	20539.8	18001.4
RSE	-	20.9	3.3	16.3	1.9

Table 3.12: Summary of relative solution error for k -opt heuristics.

Family of instances	2-opt (GK)	3-opt (GK)	2-opt MAP-Gurobi	3-opt MAP-Gurobi
Random	106.0	8.1	98.0	4.0
Clique	18.2	7.6	18.1	7.5
SquareRoot	23.1	10.2	22.8	10.1
Geometric	0.3	0.1	0.3	0.0
Product	20.9	3.3	16.3	1.9

the relative solution error is approximately of 3% 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

Table 3.13: Running times for Random, Clique, Square Root, Geometric, and Product under k -opt heuristics.

Instance	2-opt (GK)	3-opt (GK)	2-opt Gurobi	3-opt Gurobi	Instance	2-opt (GK)	3-opt (GK)	2-opt Gurobi	3-opt Gurobi
4r20	0.4	2.2	21.3	614.8	4g20	0.2	1.2	16.9	370.4
4r30	0.7	7.9	53.1	2532.5	4g30	0.3	4.9	42.1	1402.2
4r40	1.3	19.3	91.8	6261.3	4g40	0.7	13.7	85.0	4002.1
4r50	1.5	35.1	114.1	10998.2	4g50	1.3	29.4	135.5	7860.4
5r15	0.4	3.2	16.2	1135.5	5g15	0.4	2.4	15.0	1090.5
5r18	0.4	6.9	27.2	2491.8	5g18	0.5	5.6	29.2	2318.3
5r25	1.2	16.7	49.6	6533.4	5g25	1.1	16.8	61.9	5891.4
5r30	2.9	36.3	98.6	18209.3	5g30	2.3	31.7	95.2	9285.7
6r12	0.6	10.9	25.2	613.7	6g12	0.4	8.9	22.3	350.7
6r15	1.5	23.5	44.6	1201.3	6g15	1.3	19.6	38.5	739.2
6r18	3.9	42.9	69.5	2666.6	6g18	3.1	37.6	60.5	1394.6
Avg	1.4	18.7	55.6	4841.7	Avg	1.1	15.6	54.7	3155.0
4cq20	0.3	2.3	26.3	287.6	4p20	0.2	0.9	13.3	393.1
4cq30	0.6	10.0	70.0	1130.1	4p30	0.3	4.1	42.0	1145.2
4cq40	1.1	28.7	127.1	3486.4	4p40	0.7	10.7	79.3	3608.8
4cq50	1.8	52.3	174.6	9938.3	4p50	1.3	22.1	129.1	7085.8
5cq15	0.2	3.7	19.1	1126.3	5p15	0.3	3.1	26.4	671.4
5cq18	0.4	7.2	31.5	2493.5	5p18	0.4	6.1	46.6	1420.0
5cq25	1.1	23.8	66.4	7007.8	5p25	1.3	18.1	105.1	4477.8
5cq30	3.0	52.2	129.8	15830.2	5p30	4.8	396.7	158.0	6520.8
6cq12	0.6	10.8	24.7	429.9	6p12	0.5	10.8	34.8	650.5
6cq15	1.1	25.8	43.0	973.3	6p15	1.1	24.3	60.4	1404.8
6cq18	3.1	48.5	64.6	2161.5	6p18	3.0	45.7	98.2	3068.1
Avg	1.2	24.1	70.6	4078.6	Avg	1.3	49.4	72.1	2767.8
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					
Avg	1.3	25.8	75.7	4954.9					

of their machinery, the simplest versions of the family of heuristics DVH, that is the SDV2 and DV2.

3.4.9.4 Computational results for MLS and ILS

For this section we decided to compare the results obtained between MLS and ILS.

Table 3.14: Random, Clique and Square Root under DVH.

Instance	Best known	SDV2 (GK)	DV2 (GK)	DV2+ 3-opt	SDV3	DV3	DV3+ 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	3032.6	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	4671.7	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	3696.4	3995.3	3976.5	3920.1	3791.8	3781.3	3780.8
RSE	-	8.1	7.6	6.1	2.6	2.3	2.3
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	1491.1	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	1828.1	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	1432.8	1585.5	1574.4	1552.6	1479.1	1477.4	1476.4
RSE	-	10.7	9.9	8.4	3.2	3.1	3.0

We implemented all the basic heuristics as part of MLS and ILS. Tables 3.18 and 3.19 show the experimental results obtained for these combinations. For the case of the k -vertex permutation heuristic we considered $k = 4$. We considered an execution time of 10 seconds for both MLS and ILS. We observed that more than 10 seconds did not provide us with higher quality solutions. We can observe a significant difference

Table 3.15: Geometric and Product under DVH.

Instance	Best known	SDV2 (GK)	DV2 (GK)	DV2+ 3-opt(GK)	SDV3	DV3	DV3+ 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	29693.0	43044.7	34971.2	30448.2	34333.0	30258.4	30258.0
5p30	34799.4	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	9600.2	46417.1	13125.6	10842.4	12085.9	9610.1	9610.1
Avg	17660.3	29346.7	19950	18156.7	19271.0	17774.0	17774.0
RSE	-	66.2	13.0	2.8	9.1	0.6	0.6

Table 3.16: Summary of relative solution error for DVH.

Family of instances	SDV2 (GK)	DV2 (GK)	DV2+ 3-opt	SDV3	DV3	DVH3+ 3-opt
Random	72.2	34.7	7.7	0.4	0.0	0.0
Clique	8.1	7.6	6.1	2.6	2.3	2.3
SquareRoot	10.7	9.9	8.4	3.2	3.1	3.0
Geometric	0.0	0.0	0.0	0.0	0.0	0.0
Product	66.2	13.0	2.8	9.1	0.6	0.6

between the quality solution of the basic heuristics combined with a MLS and their combination with an ILS. In all the cases the combination of the basic heuristics with an ILS provide us with higher quality solutions. The most competitive results were obtained with the Simple Swap heuristic combined with an ILS.

We implemented the 2-opt, 3-opt, SDV2, and DV2 heuristics as part of MLS and ILS. Tables 3.20 and 3.21 show the experimental results obtained for these combi-

Table 3.17: Running times for Random, Clique, Square Root, Geometric, and Product under DVH.

Inst.	SDV2 (GK)	DV2 (GK)	DV2 3-opt	SDV3	DV3	DV3 3-opt	Inst.	SDV2 (GK)	DV2 (GK)	DV2 3-opt	SDV3	DV3	DV3 3-opt
4r20	0.2	0.3	2.5	177.8	178.1	179.4	4g20	0.1	0.1	0.8	85.3	82.9	54.1
4r30	0.3	0.5	8.6	875.4	1297.0	866.0	4g30	0.2	0.2	2.6	224.9	208.9	172.8
4r40	0.9	0.8	21.4	2474.2	2895.3	2112.8	4g40	0.4	0.5	6.7	660.9	646.0	515.8
4r50	1.3	1.4	43.7	3035.8	3541.7	4011.4	4g50	1.2	1.6	18.0	1224.9	1011.7	1028.4
5r15	0.3	0.2	4.3	65.2	120.9	125.1	5g15	0.5	0.1	3.2	28.7	73.5	123.9
5r18	0.5	0.4	10.3	200.1	383	388.8	5g18	1.4	0.2	5.5	68.5	173.1	304.6
5r25	1.1	1.2	29.8	678.9	2013.5	1739.1	5g25	0.9	0.9	19.6	233.7	926.3	797.8
5r30	2.7	2.7	54.4	1514.6	4984.7	4813.6	5g30	2.6	2.9	33.1	563.0	1431.9	1312.2
6r12	0.4	0.3	9.6	38.9	159.9	183.1	6g12	0.4	0.4	8.4	35.9	215.7	255.7
6r15	1.0	0.9	21.5	77.0	269.6	303.7	6g15	1.1	1.1	18.7	65.3	406.5	486.0
6r18	2.5	103.1	36.7	183.9	676.6	645.4	6g18	3.2	3.2	40.9	159.9	1001.2	1133.5
Avg	1.0	10.2	22.1	847.4	1501.8	1397.1	Avg	1.1	1.0	14.3	304.6	561.6	562.3
4cq20	0.5	0.2	2.0	120	122.2	124.1	4p20	0.1	0.2	1.9	485.6	117.2	153.6
4cq30	0.4	0.4	6.1	431.3	426.7	432.0	4p30	0.5	0.5	6.1	849.0	827.2	840.0
4cq40	0.7	0.8	17.5	2128.8	1550.9	1540.8	4p40	0.7	0.9	15.7	2010.8	1770.1	1961.5
4cq50	1.3	1.6	38.4	2478.1	2505.5	2520.7	4p50	1.4	1.9	30.9	7837.3	8506.3	9160.0
5cq15	0.3	0.3	4.2	52.5	128.4	132.3	5p15	0.1	0.3	6.6	92.1	191.8	295.5
5cq18	0.4	0.4	7.1	129.9	296.7	300.6	5p18	0.4	0.6	12.3	286.4	637.2	831.6
5cq25	1.6	1.7	21.8	527.5	832.8	861.8	5p25	1.2	1.7	28.3	1015.9	2836.1	3108.4
5cq30	2.4	2.1	36.9	602.8	1334.2	1364.8	5p30	2.8	3.9	57.8	3655.8	7110.8	7033.6
6cq12	0.4	0.5	11.6	41.7	231.2	240.1	6p12	0.5	0.7	21.7	75.1	341.4	438.4
6cq15	1.1	1.2	26.6	80.1	442.9	457.2	6p15	1.3	1.9	44.9	163.6	639.5	842.9
6cq18	3.1	3.0	54.5	295.3	1099.0	1193.4	6p18	7.1	3.6	96.7	346.4	1544.7	2703.6
Avg	1.1	1.1	20.6	626.2	815.5	833.4	Avg	1.5	1.5	29.4	1528.9	2229.3	2488.1
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							
Avg	1.1	1.0	18.3	561.6	956.5	905.7							

nations. We only considered the brute force implementations for 2-opt and 3-opt heuristics. We considered an execution time of 30 seconds for both MLS and ILS. We observed that more than 30 seconds did not provide us with higher quality solutions. In all the cases the combination of these local searches with an ILS obtained the best quality solutions. We can observed that ILS combined with DV2 provides us with the most competitive heuristics for these cases.

We conclude that there is a high advantage about the use of an ILS over a MLS. We believe that other similar techniques as simulated annealing can provide higher quality solutions.

Table 3.18: Random, Clique and Square Root under basics combined with MLS and ILS.

Inst.	Best known	Simple swap		Inversion		Circular		4-vertex perm.		Basics combined	
		MLS	ILS	MLS	ILS	MLS	ILS	MLS	ILS	MLS	ILS
4r20	20	48.4	36.3	97.2	57.0	123.6	91.1	99.1	65.5	57.0	46.2
4r30	30	74.8	60.2	177.4	121.4	215.7	178.3	169.9	118.1	83.3	70.3
4r40	40	95.0	77.1	264.4	179.6	316.4	275.1	237.1	169.7	110.9	94.0
4r50	50	113.7	94.4	358.0	243.4	422.6	371.8	323.8	236.8	131.6	118.3
5r15	15	32.3	26.9	57.7	41.9	73.7	59.0	54.0	44.2	36.9	30.8
5r18	18	39.0	32.4	77.7	53.0	97.3	85.3	72.8	60.0	44.9	40.3
5r25	25	55.9	47.5	130.8	95.3	153.2	139.2	118.8	93.1	64.2	57.7
5r30	30	67.3	57.1	166.6	116.1	204.1	165.6	156.8	108.7	73.7	64.9
6r12	12	23.2	20.1	33.4	29.3	45.8	41.8	38.1	30.2	25.9	23.9
6r15	15	30.2	26.4	53.7	45.4	67.7	61.1	55.0	44.2	35.5	33.8
6r18	18	37.8	34.9	72.9	59.2	95.2	79.3	67.0	57.9	43.6	40.9
Avg	24.8	56.1	46.7	135.4	94.7	165.0	140.7	126.6	93.5	64.3	56.5
RSE	-	126.2	88.3	446.0	281.9	565.3	467.3	410.5	277.0	159.3	127.8
4cq20	1901.8	2036.3	1929.1	2741.9	2215.0	2898.1	2532.3	2843.1	2275.8	2109.3	1970.0
4cq30	2281.9	2823.5	2440.5	4181.5	3405.6	4418.4	4145.4	4298.9	3560.8	2870.9	2618.8
4cq40	2606.3	3539.3	3015.4	5706.4	4797.8	6089.0	5773.6	5933.1	4999.9	3656.4	3454.5
4cq50	3032.6	4312.2	3696.5	7243.4	6363.7	7572.0	7396.3	7497.8	6402.3	4479.5	4219.2
5cq15	3110.7	3233.4	3136.3	3887.1	3369.1	4072.5	3795.1	3990.1	3545.7	3296.2	3163.2
5cq18	3458.6	3727.1	3509.4	4685.1	4066.4	4935.8	4630.7	4810.1	4268.8	3832.3	3602.9
5cq25	4192.7	4906.2	4380.2	6631.6	5857.2	6905.0	6574.2	6763.2	6033.0	5024.0	4649.7
5cq30	4671.7	5703.5	5028.6	7999.5	7169.3	8312.9	7996.4	8223.6	7187.4	5859.4	5412.7
6cq12	4505.6	4600.4	4529.7	5171.1	4763.0	5309.9	4973.7	5239.1	4853.8	4691.7	4588.0
6cq15	5133.4	5441.9	5189.4	6400.2	5853.6	6612.8	6273.5	6515.3	5972.2	5530.9	5296.3
6cq18	5765.5	6307.3	5956.2	7718.4	7001.7	7993.2	7669.5	7836.8	7151.7	6387.1	6008.9
Avg	3696.4	4239.2	3891.9	5669.7	4987.5	5920.0	5614.6	5813.7	5113.8	4339.8	4089.5
RSE	-	14.7	5.3	53.4	34.9	60.2	51.9	57.3	38.3	17.4	10.6
4sr20	929.3	1023.1	946.0	1361.8	1120.8	1443.0	1290.7	1425.9	1177.9	1068.8	1002.6
4sr30	1118.0	1427.1	1218.8	2094.0	1773.7	2212.6	2015.8	2179.5	1823.0	1491.3	1352.3
4sr40	1271.4	1821.4	1536.1	2855.4	2467.9	3033.3	2905.3	2933.5	2462.6	1909.1	1779.1
4sr50	1530.1	2246.5	1920.1	3632.1	3210.4	3833.5	3760.5	3722.4	3169.8	2334.6	2182.4
5sr15	1203.9	1274.3	1225.1	1525.2	1345.1	1578.8	1484.3	1566.9	1378.2	1306.0	1239.2
5sr18	1343.9	1479.1	1394.6	1847.0	1621.6	1939.4	1792.8	1892.3	1671.8	1516.2	1407.0
5sr25	1627.5	1957.7	1767.6	2608.6	2357.2	2711.4	2567.1	2657.7	2365.6	2016.0	1879.2
5sr30	1852.4	2308.7	2013.3	3159.7	2839.7	3240.3	3149.0	3166.0	2871.0	2340.3	2174.3
6sr12	1436.8	1480.0	1451.5	1635.0	1532.7	1704.3	1598.6	1685.0	1546.1	1492.6	1452.6
6sr15	1654.6	1773.6	1696.9	2076.1	1917.1	2120.1	2025.8	2091.5	1948.8	1793.1	1714.8
6sr18	1856.3	2051.9	1923.0	2477.4	2266.7	2556.0	2470.8	2507.0	2337.8	2088.6	1974.9
Avg	1432.8	1713.0	1553.9	2297.5	2041.2	2397.5	2278.2	2348.0	2068.4	1759.7	1650.8
RSE	-	19.6	8.5	60.4	42.5	67.3	59.0	63.9	44.4	22.8	15.2

3.4.9.5 General results

Here we compare the results obtained by the best local searches. The best heuristics from each section were: Simple Swap, 3-opt, DV3, and the ILS combined with SDV2.

Table 3.19: Geometric and Product under basics combined with MLS and ILS.

Inst.	Best known	Simple swap		Inversion		Circular		4-vertex perm.		Basics combined	
		MLS	ILS	MLS	ILS	MLS	ILS	MLS	ILS	MLS	ILS
4g20	2380.5	2380.5	2380.5	2790.8	2439.1	2981.0	2676.9	3118.7	2530.1	2380.5	2380.5
4g30	3015.2	3015.7	3015.2	4252.0	3420.9	4560.2	4079.9	4727.0	3595.9	3020.0	3016.1
4g40	3523.8	3533.2	3524.8	5734.8	4676.1	6193.4	5704.3	6389.6	4831.9	3551.2	3529.7
4g50	4102.3	4124.1	4105.7	7273.2	6161.3	7888.6	7536.2	8110.4	6369.2	4158.2	4124.7
5g15	3423.7	3423.7	3423.7	3791.4	3456.8	3964.3	3686.6	4092.8	3575.8	3423.7	3423.7
5g18	3799.3	3799.3	3799.3	4483.5	3985.8	4739.1	4436.1	4889.8	4162.0	3799.7	3799.3
5g25	4594.9	4595.3	4596.8	6209.4	5387.9	6617.3	6213.8	6916.6	5688.7	4597.6	4600.5
5g30	5036.8	5043.3	5036.8	7438.4	6431.7	7865.8	7460.8	8265.1	6661.0	5060.1	5044.5
6g12	4483.7	4483.7	4483.7	4502.5	4743.2	4675.7	4939.6	4630.3	5090.4	4483.7	4483.7
6g15	5242.4	5242.4	5242.4	5461.9	5874.1	5927.4	6250.4	5688.9	6433.5	5242.4	5243.2
6g18	5767.4	5767.4	5767.4	6279.2	6996.0	7065.2	7423.4	6568.9	7661.2	5772.4	5770.6
Ave	4124.5	4128.1	4125.1	5292.5	4870.3	5679.8	5491.6	5763.5	5145.4	4135.4	4128.8
RSE	-	0.1	0.0	28.3	18.1	37.7	33.1	39.7	24.8	0.3	0.1
4p20	8397.3	8397.8	8397.4	8412.5	8401.2	8430.9	8415.7	8432.3	8408.0	8399.8	8398.4
4p30	13154.1	13155.2	13154.6	13190.5	13164.6	13244.5	13214.4	13243.7	13176.9	13156.4	13155.5
4p40	16810.0	16813.1	16810.6	16883.5	16842.1	16989.5	16942.1	17008.8	16864.3	16816.1	16815.7
4p50	20705.6	20712.2	20709.4	20831.5	20759.0	20986.1	20895.1	21027.2	20790.4	20716.8	20714.1
5p15	21422.8	22069.9	21590.8	22206.2	21844.3	23552.7	23700.0	23437.7	26873.4	23314.9	23104.7
5p18	23371.4	25469.1	24498.1	26176.2	25579.6	27205.5	30502.7	27961.8	29535.5	26972.0	29379.4
5p25	29693.0	36186.1	35200.8	37974.5	35536.0	39923.5	46782.5	41242.7	46310.8	38633.6	44389.0
5p30	34799.4	45473.7	42278.1	46339.3	43980.5	49689.9	53578.0	51106.3	52296.7	49222.3	51445.5
6p12	7421.0	18035.0	11242.8	19809.0	12511.0	22605.5	15384.5	25573.4	15088.4	20607.5	14164.8
6p15	8888.9	24992.8	14636.1	27116.2	17007.1	32411.4	23906.0	34078.7	22705.4	30613.3	21627.7
6p18	9600.2	32754.3	18541.8	37470.3	21715.2	41529.8	30234.1	46548.8	29696.8	38673.8	26970.7
Ave	17660.3	24005.4	20641.9	25128.2	21576.4	26960.8	25777.7	28151	25613.3	26102.4	24560.5
RSE	-	35.9	16.9	42.3	22.2	52.7	46.0	59.4	45.0	47.8	39.1

We also decided to add the results of the ILS combined with Simple Swap and 3-opt in order to have a better reference about the improvement obtained with these combination.

Tables 3.22 and 3.23 show the general results obtained among the best heuristics from each type. The 3-opt and DV2 correspond with the local searches proposed by [Karapetyan and Gutin, 2011a]. The rest of the heuristics were proposed in this thesis. We can observe that the DV2 (ILS) and the DV3 provided us with the best local searches to solve MAP. DV2 (ILS) obtained the best results for the families of instances Clique and Square Root whereas DV3 obtained the best results for the families of instances Product, Random, and Geometric.

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

Table 3.20: Random, Clique and Square Root under k -opt and DV2 combined with MLS and ILS.

Instance	Best known	2-opt		3-opt		SDV2 (GK)		DV2 (GK)	
		MLS	ILS	MLS	ILS	MLS	ILS	MLS	ILS
4r20	20	37.4	28.7	23.2	22.4	31.0	27.8	28.2	25.2
4r30	30	57.1	43.0	33.9	32.8	42.3	38.1	38.2	36.0
4r40	40	74.0	58.3	43.8	42.7	51.1	46.9	46.5	44.3
4r50	50	89.9	71.7	53.3	52.6	60.4	55.7	54.9	53.7
5r15	15	21.8	18.2	15.2	15.1	23.7	20.9	18.8	18.0
5r18	18	26.5	22.3	18.1	18.0	27.3	24.0	21.8	21.4
5r25	25	37.0	32.1	25.2	25.1	35.2	32.4	29.5	29.3
5r30	30	45.5	39.1	30.2	30.0	40.2	37.8	34.2	33.7
6r12	12	13.1	12.6	12.0	12.0	17.8	16.7	13.5	13.7
6r15	15	18.0	16.9	15.0	15.0	23.1	22.0	17.2	17.1
6r18	18	22.3	20.4	18.0	18.0	26.0	25.0	20.2	20.0
Avg	24.8	40.2	33.0	26.2	25.8	34.4	31.6	29.4	28.4
RSE	-	62.1	33.1	5.6	4.0	38.7	27.4	18.5	14.5
4cq20	1901.8	1999.2	1921.5	1937.4	1919.2	1914.8	1907.7	1918.2	1910.9
4cq30	2281.9	2728.1	2391.0	2490.0	2378.3	2405.6	2317.8	2421.7	2331.9
4cq40	2606.3	3392.0	2899.9	3047.7	2923.2	2892.8	2697.2	2878.8	2705.9
4cq50	3032.6	4189.5	3520.1	3688.7	3481.6	3408.0	3182.8	3415.0	3166.6
5cq15	3110.7	3183.1	3129.6	3131.6	3135.8	3137.1	3111.8	3130.0	3116.7
5cq18	3458.6	3628.6	3488.4	3544.8	3514.4	3518.4	3491.1	3526.0	3485.2
5cq25	4192.7	4754.4	4346.8	4559.4	4397.0	4439.6	4289.0	4449.6	4268.8
5cq30	4671.7	5526.4	4974.1	5242.5	5033.0	5045.3	4772.5	5080.4	4843.3
6cq12	4505.6	4529.6	4516.9	4550.4	4521.9	4522.3	4516.4	4528.4	4521.9
6cq15	5133.4	5315.1	5178.4	5287.8	5213.0	5221.3	5156.4	5223.1	5179.0
6cq18	5765.5	6140.0	5879.8	6081.5	5952.3	5968.6	5826.3	5991.4	5852.3
Avg	3696.5	4126.0	3840.6	3960.2	3860.9	3861.3	3751.7	3869.3	3762.0
RSE	-	11.6	3.9	7.1	4.5	4.5	1.5	4.7	1.8
4sr20	929.3	1006.6	942.7	947.9	945.1	939.2	936.8	942.9	937.5
4sr30	1118.0	1376.6	1194.5	1253.5	1173.5	1209.5	1134.8	1202.3	1146.5
4sr40	1271.4	1749.7	1464.7	1525.6	1437.7	1451.7	1340.3	1444.7	1340.6
4sr50	1491.1	2156.3	1767.6	1854.1	1760.5	1711.5	1564.6	1724.7	1572.4
5sr15	1203.9	1238.5	1211.2	1223.4	1213.7	1219.4	1209.8	1214.6	1207.9
5sr18	1343.9	1440.3	1363.4	1397.8	1358.8	1389.4	1358.5	1379.3	1358.7
5sr25	1627.5	1888.0	1720.2	1800.9	1728.5	1755.9	1685.3	1757.2	1675.5
5sr30	1828.1	2208.9	1966.0	2083.3	2007.7	2017.2	1890.8	2020.2	1897.4
6sr12	1436.8	1458.3	1442.8	1464.1	1446.9	1442.2	1441.5	1445.6	1439.8
6sr15	1654.6	1732.4	1672.8	1728.1	1686.0	1691.8	1671.0	1700.7	1668.0
6sr18	1856.3	2011.6	1898.4	1985.2	1933.1	1932.5	1886.4	1950.3	1897.1
Avg	1432.8	1660.7	1513.1	1569.4	1517.4	1523.7	1465.4	1525.7	1467.4
RSE	-	15.9	5.6	9.5	5.9	6.3	2.3	6.5	2.4

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

Table 3.21: Geometric and Product under k -opt and DV2 combined with MLS and ILS.

Instance	Best known	2-opt		3-opt		SDV2 (GK)		DV2 (GK)	
		MLS	ILS	MLS	ILS	MLS	ILS	MLS	ILS
4g20	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5
4g30	3015.2	3015.4	3018.4	3016.6	3021.3	3015.2	3015.2	3015.2	3015.2
4g40	3523.8	3526.7	3527.1	3534.2	3540.7	3523.8	3523.8	3523.8	3523.8
4g50	4102.3	4115.7	4104.4	4114.6	4113.8	4102.3	4102.3	4102.3	4102.3
5g15	3423.7	3423.7	3423.7	3423.7	3424.7	3423.7	3423.7	3423.7	3423.7
5g18	3799.3	3799.3	3799.3	3799.3	3801.1	3799.3	3799.3	3799.3	3799.3
5g25	4594.9	4594.9	4595.4	4600.1	4600.6	4594.9	4594.9	4594.9	4594.9
5g30	5036.8	5040.9	5037.2	5048.9	5058.5	5036.9	5036.8	5036.8	5036.8
6g12	4483.7	4483.7	4483.7	4483.7	4486.4	4483.7	4483.7	4483.7	4483.7
6g15	5242.4	5242.4	5242.4	5244.4	5248.4	5242.4	5242.4	5242.4	5242.4
6g18	5767.4	5767.4	5771.1	5772.8	5785.5	5767.4	5767.4	5767.4	5767.4
Avg	4124.5	4126.4	4125.7	4129.0	4132.9	4124.6	4124.5	4124.5	4124.5
RSE	-	0.0	0.0	0.1	0.2	0.0	0.0	0.0	0.0
4p20	8397.3	8397.4	8397.4	8397.4	8397.3	8398.3	8397.5	8397.4	8397.4
4p30	13154.1	13154.2	13154.2	13154.2	13154.2	13154.9	13154.6	13154.2	13154.2
4p40	16810.0	16810.9	16810.3	16810.2	16810.2	16811.8	16810.4	16810.4	16810.2
4p50	20705.6	20708.2	20707.0	20706.2	20705.9	20710.4	20707.9	20707.2	20706.1
5p15	21422.8	21424.5	21423.0	21423.1	21422.8	22428.2	22221.3	21424.2	21423.2
5p18	23371.4	23460.9	23457.9	23371.8	23508.0	25476.4	24108.7	24080.9	23457.9
5p25	29693.0	31755.9	30681.7	30474.7	30334.0	36165.7	35753.5	32973.6	32772.1
5p30	34799.4	38305.3	36380.8	35668.9	36264.7	45499.9	43844.8	39066.4	38584.5
6p12	7421.0	8688.1	7844.9	7842.1	7749.9	19155.9	11931.6	9029.1	8232.8
6p15	8888.9	11527.9	10274.0	10336.8	10216.6	25725.7	15527.8	10861.9	10602.2
6p18	9600.2	13682.4	11667.5	11666.4	10623.7	32859.3	16444.1	12220.0	11436.0
Avg	17660.3	18901.4	18254.4	18168.3	18107.9	24217.0	20809.3	18975.0	18688.8
RSE	-	7.0	3.4	2.9	2.5	37.1	17.8	7.4	5.8

to direct the set of feasible solutions to a local optimum since it allows to explore several neighborhoods at the same time.

Algorithm 19 shows the general structure of a genetic algorithm. Let *generations* be the number of evolutionary steps, let N be the required size for the population among the evolutionary process and, let *mutationProbability* be the probability of mutating an individual. A genetic algorithm works as follows: at first, all the individuals are created according to the required size N . Then a total of *generations* iterations are executed. At each iteration, we perform first a selection of the best individuals. Based on the selected individuals, a set of new individuals is created. Finally, with a low probability, some mutations are applied to the individuals. It is expected to have better individuals as long as the generations pass.

In a memetic algorithm the main idea is to mutate the individuals by improving them instead of by performing random changes over them. Such improvements can be done by a local search heuristic. The disadvantage is that the technique can direct the

Table 3.22: General results for Random, Clique and Square Root under the best local searches.

Instance	Best known	Simple swap	S. swap (ILS)	3-opt	3-opt (ILS)	DV2	DV2 (ILS)	DV3
4r20	20	66.8	36.3	25.7	22.4	28.2	25.2	20.0
4r30	30	92.4	60.2	35.3	32.8	38.2	36.0	30.0
4r40	40	110.4	77.1	44.0	42.7	46.5	44.3	40.0
4r50	50	132.2	94.4	53.3	52.6	54.9	53.7	50.0
5r15	15	50.8	26.9	16.5	15.1	18.8	18.0	15.0
5r18	18	55.4	32.4	19.0	18.0	21.8	21.4	18.0
5r25	25	72.3	47.5	25.4	25.1	29.5	29.3	25.0
5r30	30	81.4	57.1	30.1	30.0	34.2	33.7	30.0
6r12	12	39.8	20.1	12.0	12.0	13.5	13.7	12.0
6r15	15	47.0	26.4	15.0	15.0	17.2	17.1	15.0
6r18	18	52.8	34.9	18.0	18.0	20.2	20.0	18.0
Avg	24.8	72.8	46.7	26.8	25.8	29.4	28.4	24.8
RSE	-	193.5	88.3	8.1	4.0	18.5	14.5	0.0
4cq20	1901.8	2235.4	1929.1	1983.5	1919.2	1918.2	1910.9	1909.0
4cq30	2281.9	3012.3	2440.5	2549.9	2378.3	2421.7	2331.9	2322.6
4cq40	2606.3	3664.5	3015.4	3055.3	2923.2	2878.8	2705.9	2714.2
4cq50	3032.6	4441.9	3696.5	3670.3	3481.6	3415.0	3166.6	3144.8
5cq15	3110.7	3438.0	3136.3	3185.3	3135.8	3130.0	3116.7	3128.0
5cq18	3458.6	3962.4	3509.4	3597.3	3514.4	3526.0	3485.2	3507.9
5cq25	4192.7	5134.1	4380.2	4562.1	4397.0	4449.6	4268.8	4337.2
5cq30	4676.4	5948.4	5028.6	5237.3	5033.0	5080.4	4843.3	4886.1
6cq12	4505.6	4842.5	4529.7	4577.6	4521.9	4528.4	4521.9	4532.6
6cq15	5133.4	5702.1	5189.4	5309.8	5213.0	5223.1	5179.0	5216.7
6cq18	5765.5	6575.8	5956.2	6041.8	5952.3	5991.4	5852.3	5895.5
Avg	3696.4	4450.7	3891.9	3979.1	3860.9	3869.3	3762.0	3781.3
RSE	-	20.4	5.3	7.6	4.5	4.7	1.8	2.3
4sr20	929.3	1123	946	979.5	945.1	942.9	937.5	937.4
4sr30	1118.0	1524.2	1218.8	1272.5	1173.5	1202.3	1146.5	1153.2
4sr40	1271.4	1914.7	1536.1	1527.9	1437.7	1444.7	1340.6	1337.0
4sr50	1491.1	2320.9	1920.1	1852.0	1760.5	1724.7	1572.4	1551.6
5sr15	1203.9	1360.2	1225.1	1248.6	1213.7	1214.6	1207.9	1215.4
5sr18	1343.9	1582.6	1394.6	1416.6	1358.8	1379.3	1358.7	1369.3
5sr25	1627.5	2071.4	1767.6	1814.0	1728.5	1757.2	1675.5	1703.9
5sr30	1828.1	2414.8	2013.3	2080.0	2007.7	2020.2	1897.4	1935.3
6sr12	1436.8	1557.0	1451.5	1468.3	1446.9	1445.6	1439.8	1447.6
6sr15	1654.6	1862.3	1696.9	1730.9	1686.0	1700.7	1668.0	1689.1
6sr18	1856.3	2151.4	1923.0	1979.0	1933.1	1950.3	1897.1	1911.9
Avg	1432.8	1807.5	1553.9	1579	1517.4	1525.7	1467.4	1477.4
RSE	-	26.2	8.5	10.2	5.9	6.5	2.4	3.1

individuals to some local optimum because the diversification provided by random mutations is lost. In some cases, one can apply some random changes over a low proportion of the individuals in order to avoid losing the diversity.

Table 3.23: General results for Geometric and Product under the best local searches.

Instance	Best known	Simple swap	S. swap (ILS)	3-opt	3-opt (ILS)	DV2	DV2 (ILS)	DV3
4g20	2380.5	2383.2	2380.5	2380.5	2380.5	2380.5	2380.5	2380.5
4g30	3015.2	3026.9	3015.2	3017.3	3021.3	3015.2	3015.2	3015.2
4g40	3523.8	3553.0	3524.8	3529.6	3540.7	3523.8	3523.8	3523.8
4g50	4102.3	4148.9	4105.7	4112.9	4113.8	4102.3	4102.3	4102.3
5g15	3423.7	3427.2	3423.7	3423.7	3424.7	3423.7	3423.7	3423.7
5g18	3799.3	3807.7	3799.3	3799.3	3801.1	3799.3	3799.3	3799.3
5g25	4594.9	4615.4	4596.8	4596.5	4600.6	4594.9	4594.9	4594.9
5g30	5036.8	5078.1	5036.8	5040.2	5058.5	5036.8	5036.8	5036.8
6g12	4483.7	4487.3	4483.7	4483.7	4486.4	4483.7	4483.7	4483.7
6g15	5242.4	5252.8	5242.4	5243.1	5248.4	5242.4	5242.4	5242.4
6g18	5767.4	5785.7	5767.4	5767.4	5785.5	5767.4	5767.4	5767.4
Avg	4124.5	4142.4	4125.1	4126.7	4132.9	4124.5	4124.5	4124.5
RSE	-	0.4	0.0	0.1	0.2	0.0	0.0	0.0
4p20	8397.3	8402.9	8397.4	8397.4	8397.3	8397.4	8397.4	8397.3
4p30	13154.1	13159.0	13154.6	13154.2	13154.2	13154.2	13154.2	13154.1
4p40	16810.0	16817.4	16810.6	16810.2	16810.2	16810.4	16810.2	16810.0
4p50	20705.6	20716.1	20709.4	20706.3	20705.9	20707.2	20706.1	20705.6
5p15	21422.8	27070.6	21590.8	21423.4	21422.8	21424.2	21423.2	21422.8
5p18	23371.4	29763.9	24498.1	24012.4	23508.0	24080.9	23457.9	23371.8
5p25	29693.0	43046.5	35200.8	30475.0	30334.0	32973.6	32772.1	30258.4
5p30	34799.4	50234.6	42278.1	35811.6	36264.7	39066.4	38584.5	35231.3
6p12	7421.0	30578.3	11242.8	8172.2	7749.9	9029.1	8232.8	7664.0
6p15	8888.9	39227.4	14636.1	10356.6	10216.6	10861.9	10602.2	8888.9
6p18	9600.2	47585.2	18541.8	11379.1	10623.7	12220.0	11436.0	9610.1
Avg	17660.3	29691.1	20641.9	18245.3	18107.9	18975.0	18688.8	17774.0
RSE	-	68.1	16.9	3.3	2.5	7.4	5.8	0.6

The first memetic algorithm for the MAP was proposed by [Huang and Lim, 2006]. It was called LSGA (Local Searching Genetic Algorithm) and consisted on a basic structure of a genetic algorithm with an initial population of 100 individuals randomly generated, a crossover operator partially mapped crossover, the SDV2 instead of a mutation operator, a basic selection similar to the elitist selection, and a stopping criteria consisting on ending either after 10 generations of no improvement or when in the set of individuals there are many duplicates. Figure 3.6 shows the general structure of an iteration of the LSGA.

We introduce a Simple Memetic Algorithm (SMA) similar to the LSGA. The main difference is that we evaluated several crossover operators, selection functions, local search heuristics, and some mutations, in order to obtain a more robust memetic algorithm that provided us higher quality solutions. In addition, the structure of our SMA is a bit more sophisticated.

Algorithm 20 shows the structure of our SMA. SMA returns the best individual found among all the iterations. In the initialization, the population is randomly

Algorithm 19: The general structure of a genetic algorithm.

Input: *generations*: The number of generations to iterate.

N: 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(N);
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)};

```

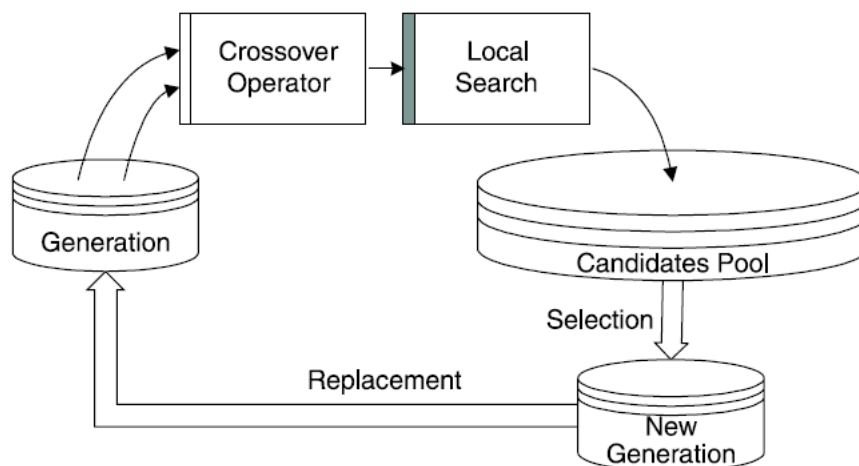


Figure 3.6: Structure of the LSGA proposed by Huang and Lim (2006).

generated and improved by a local search heuristic, then the first best individual is obtained. At each iteration, we first apply a crossover method but, instead of obtaining an offspring equal to the required size of the population, we just generate a proportion determined by the input parameter *survivorPerc*. These individuals will be the first part of the next generation. After the crossing, each individual from the offspring may 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.

Algorithm 20: A new memetic algorithm for MAP.

Input: $G = (X_1, \dots, X_s; E)$. Weighted hypergraph of a s AP 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 :=  $\emptyset$ ;
2 while |individuals|  $\leq N$  do
3   Set new_individual := FeasibleRandomGeneratedSolutionForMAP(G);
4   Set new_individual := LSH(individual, LSHMethod);
5   Set individuals := individuals  $\cup$  new_individual;
6 Set best_individual := GetBestOfIndividuals(individuals);
7 Set iterations := 1;
8 while iterations  $\leq$  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  $\cup$  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  $\cup$  best_individual);
18  Set iterations := iterations+1;
19 return {best_individual};

```

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 resulting from the crossover function. By performing this process we allow a higher diversification as well as increase the fitness of the population. Even

when a good individual can be crossed with an individual of low quality, the local search heuristic allows to improve the resulted individual.

3.5.1 Genetic representation and fitness function

A 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.

The most common representation of a solution is as a binary array. In our case 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 of 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 cases, produces infeasible solutions which must be repaired.

The fitness function is the way to evaluate how close to the optimum is a feasible solution. Our fitness function consists on summing the weights of the hyperedges of each feasible solution. Let c_i be the solution value of some individual i , its fitness value is given by $f_i = 1/c_i$. A solution A is more apt than a solution B if the sum of the weights of the vectors of A is lower than the sum of the weights of the vector of B . The process of evaluating each solution takes $O(sn)$ time.

3.5.2 The selection function

In our case, the selection function allows 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: deterministic, tournament, and roulette wheel selection.

3.5.2.1 Deterministic selection

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

Algorithm 21 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.

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.

Algorithm 21: The deterministic 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};
-

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 small, 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 22 shows this procedure. In the first step, we perform a random permutation of the elements from $\{1, \dots, N\}$. 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 22: 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 $N := |\text{individuals}|$;
 - 2 Set permutation := Make a random permutation of the elements from $1, \dots, N$;
 - 3 Set survivors := \emptyset ;
 - 4 Set index := 1;
 - 5 Set permutation := permutation || permutation// || is for concatenation
 - 6 **while** $|\text{survivors}| < \text{survivorPerc} * |\text{individuals}|$ **do**
 - 7 Set winner := MakeTournament(individuals[permutation[index]],
 individuals[permutation[index + 1]]);
 - 8 Set survivors := survivors \cup winner;
 - 9 Set index := index + 2;
 - 10 **return** {survivors};
-

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

from the previous generation.

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 be the fitness of the individual i of the current population, then its probability of being selected is:

$$p_i = \frac{f_i}{\sum_{i=1}^N f_i} \quad (3.17)$$

Algorithm 23 shows this procedure. In the first step, the individuals are sorted in descending order by its fitness value. The total fitness sum $fitSum$ is calculated and an array dp stores the cumulative fitness sum until each sorted individual. Then a set of $survivorPerc \cdot N$ iterations are performed. At each iteration, a uniformly random value α is generated from the reals 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 23: 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 Set 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   Set fitnessSum := fitnessSum + fitness(orderedIndividuals[i]);
7   Set dp[index] := fitnessSum;
8 Set survivors :=  $\emptyset$ ;
9 while  $|survivors| < survivorPerc * |orderedIndividuals|$  do
10  Set alpha := random() % fitnessSum;
11  Set index := binarySearch(dp, dp + N, alpha);
12  // returns the lowest index with  $alpha \leq dp[index]$ 
13  Set survivors := survivors  $\cup$  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 take 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 $survivorPerc \cdot N$ iterations are required, therefore the 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 a process analogous to biological reproduction. A crossover takes some characteristics from two or more parents and combines them to create a new individual.

Formerly, the selection function is used to select the individuals to cross, however we decided to choose uniformly at random two elements from the current generation to cross them, even if they have very different fitness values. The main reason to apply this criteria is that a new individual is always 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 the works proposed by [Huang and Lim, 2006] as well as in the proposed by [Karapetyan and Gutin, 2011b] they performed this type of crossing obtaining good results. In addition, in our experimental evaluation we observed that this allows us to have a higher diversity among the generations. We observed that this criteria is good in this case because the local searches that we use as improvement use to provide high quality solutions, the main problem occurs when two good solutions are combined, since they are very similar then in most of the cases we observed that it is not obtained a better solution. However, by combining two solutions that are different it allows to create a very different individual that is able to consider a completely new neighborhood. Further analysis could be performed but in our experience this alternative provided us with really good results this is why we decided to follow a similar line as other authors.

We implemented three basic crossover techniques: partially mapped crossover, cycle crossover, and order crossover. All this crossover techniques were implemented to work over permutations. Recall that 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 must be repaired.

Algorithm 24 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.

Algorithm 24: 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};

```

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:

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

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

3.5.3.2 Cycle crossover

The cycle 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 25 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 $ind2 = (3, 1, 2, 4, 5)$ 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:

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

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 with 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 26 shows the OX procedure. After the swapping process, as in PMX, the repairing process is performed. All the duplicated elements out of the exchanged

Algorithm 25: The cycle crossover.

Input: $ind1, ind2$. The parents to cross.

Result: $o1 \cup o2$. The two new individuals after the crossing.

```

1 Set  $p1 := \text{random}() \% \text{ind1.assignment.size}$ ;
2 Set  $o1 := \text{ind1}$ ;
3 Set  $o2 := \text{ind2}$ ;
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  $\text{indexes} :=$  An array of  $n$  elements;
8   foreach  $index$  in  $1:n$  do
9     [ Set  $\text{indexes}[\text{o1.matching}[i][dim]] := index$ ;
10    Set  $\text{start\_vertex} := \text{o1.matching}[\text{point}][dim]$ ;
11    Set  $index := \text{point}$ ;
12    while  $\text{start\_vertex} \neq \text{o2.matching}[index][dim]$  do
13      [ Set  $\text{next\_point} := \text{o2.matching}[index][dim]$ ;
14      swap( $\text{o1.assignment}[index][dim]$ ,  $\text{o2.assignment}[index][dim]$ );
15      Set  $index := \text{indexes}[\text{next\_point}]$ ;
16    swap( $\text{o1.assignment}[index][dim]$ ,  $\text{o2.assignment}[index][dim]$ );
17 return  $\{o1 \cup o2\}$ ;
```

segment are replaced with the missing elements of the corresponding permutation according with their position in the opposite parent, going from left to right.

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:

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

Then the repairing process is as follows:

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

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 diversity from one generation to the next. In a mutation process the original solution can be changed completely.

Algorithm 26: 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
12    // Set true if  $p1 \leq i$  and  $i \leq p2$ , otherwise set false
13    Set flag[ o1.matching[index][dim] ] := ( $p1 \leq i$  and  $i \leq p2$ );
14    Set vertices_order[index] := ind2.matching[index][dim];
15  Set next := 0;
16  foreach index in 0 : ( $n - 1$ ) do
17    if index < p1 or index > p2 then
18      while next <  $n$  and flag[vertices_order[next]] = True do
19        Set next := next + 1;
20        Set o1.matching[index][dim] = vertices_order[next];
21        Set next := next + 1;
21 Repair o2 analogous to o1;
22 return {o1  $\cup$  o2};

```

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 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 swap heuristic. This operator consists on choosing

two arbitrary indices from each permutation and swapping them.

Algorithm 27 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 27: 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 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 indices from each permutation and reversing the whole interval (including the chosen indices) of vertices.

Algorithm 28 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 28: 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};
```

Table 3.24: 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	3032.6	1491.1	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	29693.0
5	30	30.0	4671.7	1828.1	5036.8	34799.4
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	9600.2
Avg	-	24.8	3696.4	1432.8	4124.5	17660.3

There are s inversions performed in $O(n)$ time each, so the 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 memetic algorithm. In the case of our SMA 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. The population size, mutation probability and survivors probability were selected based on our experience after testing a test bed that considered all the combinations from $N \in \{10, 50, 100\}$, $mp \in \{2, 4, 6, 8, 10\}$ and $sp \in \{25, 50\}$. We evaluated our SMA under two variants of DVH: SDV2 and DV2.

Table 3.24 shows the best known values for the 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 in bold the corresponding average. Table 3.24 can be used as reference to verify how close are the results obtained by SMA to the best known solutions.

Tables 3.25, 3.26, and 3.27 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. We can observe that the RSE is high for the family of instances Random. This occurs because SDV2

and DV2 provide a low quality solution error for this family of instances. In the case of the families of instances Clique and Square Root our SMA provides similar quality results to those reported by the best known memetic algorithm for MAP. The RSE of SMA combined with the SDV2 for the family of instances Clique, considering the deterministic selection, CX, and IM operators, is approximately 0.5 whereas for the best known memetic algorithm is 0.1. The RSE of SMA combined with the SDV2 for the family of instances Square Root, considering the deterministic selection, CX and IM operators, is approximately 1.1 whereas for the best known memetic algorithm is 0.1. In general, the results obtained by SMA combined with SDV2 and DV2 outperformed those obtained by SDV3 and DV3.

Table 3.25: 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
	Deterministic						Deterministic					
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 3.26: 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
	Deterministic						Deterministic					
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	0.8	0.7	0.6	0.5	1.6	1.5	1.1	1.0	0.9	0.9	1.8	1.9
	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	1.0	0.9	0.8	0.6	1.8	1.8	1.3	1.2	1.0	0.9	2.1	2.2
	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	1.0	0.8	0.7	0.7	1.5	1.5	1.2	1.2	1.0	0.9	1.7	1.9

Tables 3.28 and 3.29 show the results obtained for the families of instances Geometric and Product. In the case of the family of instances Geometric, the SMA combined with either SDV2 or DV2 is able to find the optimal solutions in all the evaluated instances for any combination of operators. The RSE of SMA combined with the DV2 for the family of instances Product, considering the roulette wheel selection, PMX and IM operators, is approximately 2.4. For the family of instances Product 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 0.6.

It is important to mention that the RSE of the best known memetic algorithm

Table 3.27: Square Root 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
	Deterministic						Deterministic					
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	1.7	1.4	1.2	1.1	2.8	2.6	2.4	2.2	2.0	1.8	3.3	3.3
	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	1.9	1.8	1.5	1.2	3.3	3.2	2.5	2.4	2.3	2.1	3.7	3.5
	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	2.1	2.0	1.7	1.6	3.0	2.9	2.8	2.4	2.3	2.2	3.6	3.5

were obtained after running times of 300 seconds whereas ours were obtained within 30 seconds. This is why we decided to execute one more time our SMA but now considering a running time of 300 seconds and the combination of the operators that obtained the best results.

In order to determine the best combination of operators we compared the RSE obtained for all the families of instances. Table 3.30 shows a summary of the RSE obtained for each combination of operators for the SMA combined with SDV2 and DV2. The last set of rows, called “Wins”, is the number of cases in which the corresponding combination obtained the best results among all the combinations. In

Table 3.28: 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
	Deterministic						Deterministic					
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

the case of the family of instances Geometric we can observe that all the combinations of operators obtained a RSE equal to zero. The maximum number of victories for each combination of operators is 6, however the number of wins for any combination was at most 3. The combination of operators that obtained the best results was a deterministic selection combined with a cycle crossover and an inversion mutation. These combination is only better on the cases of the families of instances Clique and Square Root. For the families of instances Random and Product is not clear which of our three selections is the best one, however the combination of a partially mapped crossover with an inversion mutation provided us with the best results.

Table 3.29: 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	OX	IM	SM	IM	SM	IM	SM	OX	IM
	Deterministic							Deterministic						
4p20	8397.5	8397.5	8397.8	8397.7	8397.7	8397.6	8397.4	8397.4	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	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	16810.3	16810.2	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	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	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	23513.9	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	30859.8	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	37564.7	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	7970.8	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	9941.8	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	11269.8	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	18328.4	18328.4
RSE	15.8	14.8	26.4	23.9	22.0	21.2	3.7	3.7	4.3	3.8	4.0	3.8	3.8	3.8
	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	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	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	16810.2	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	20706.5	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	21423.0	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	23513.7	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	31017.8	31017.8
5p30	40104.0	39074.8	43442.8	43648.6	42383.1	42450.1	36704.8	37272.8	36936.4	37609.0	37620.4	38572.3	38572.3	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	8182.5	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	10024.5	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	11142.6	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	18449.5	18449.5
RSE	21.0	18.6	30.3	28.9	27.0	25.1	4.2	3.9	4.0	4.1	4.6	4.5	4.5	4.5
	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	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	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	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	20706.5	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	21422.9	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	23371.8	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	30532.6	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	37799.2	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	7857.8	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	9700.2	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	10979.9	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	18248.4	18248.4
RSE	12.1	10.9	16.9	17.0	15.6	14.0	2.9	2.4	3.2	3.7	4.0	3.3	3.3	3.3

According with our experience from local searches, DV3 is able to solve exactly the test bed of the family of instances Random and it obtained a RSE of 0.6 for the family of instances Product. This is an example in which the results obtained by DV3 outperformed the results obtained by a memetic algorithm. However, the RSE of DV3 for the families of instances Clique and Square Root are 2.3 and 2.7 respectively. This is why that for the final evaluation we decided to apply the combination of the operators that are better solving the families of instances Clique and Square Root.

We implemented the LSGA proposed by [Huang and Lim, 2006] in order to compare their results against ours. Since LSGA was designed for 3AP we generalized it for sAP. We applied the same considerations: a population size $N = 100$, a PMX as crossover and an improvement through the solving of a reduction of a 3AP to a 2AP instead of a mutation. We implemented two versions of the LSGA, the difference is on the algorithm to solve the corresponding 2AP. The first version considers the Hungarian method and the second version considers the ϵ -scaling Auction algorithm. Both versions are equivalent in terms of quality solution, however the version that considers ϵ -scaling Auction algorithm is faster. We verified that we had replicated the results from their paper.

Table 3.30: Relative solution errors for all combinations of operators for SMA.

Family of inst.	Selection	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
Random	Deterministic	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	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	30.2	29.8	31.9	31.9	31.0	31.0	13.7	13.3	14.9	14.5	13.7	14.1
Clique	Deterministic	0.8	0.7	0.6	0.5	1.6	1.5	1.1	1.0	0.9	0.9	1.8	1.9
	Tournament	1.0	0.9	0.8	0.6	1.8	1.8	1.3	1.2	1.0	0.9	2.1	2.2
	Roulette	1.0	0.8	0.7	0.7	1.5	1.5	1.2	1.2	1.0	0.9	1.7	1.9
Square	Deterministic	1.7	1.4	1.2	1.1	2.8	2.6	2.4	2.2	2.0	1.8	3.3	3.3
	Tournament	1.9	1.8	1.5	1.2	3.3	3.2	2.5	2.4	2.3	2.1	3.7	3.5
	Roulette	2.1	2.0	1.7	1.6	3.0	2.9	2.8	2.4	2.3	2.2	3.6	3.5
Geometric	Deterministic	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	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Product	Deterministic	15.8	14.8	26.4	23.9	22.0	21.2	3.7	3.7	4.3	3.8	4.0	3.8
	Tournament	21.0	18.6	30.3	28.9	27.0	25.1	4.2	3.9	4.0	4.1	4.6	4.5
	Roulette	12.1	10.9	16.9	17.0	15.6	14.0	2.9	2.4	3.2	3.7	4.0	3.3
Wins	Deterministic	1	2	1	3	1	1	1	1	2	3	2	1
	Tournament	1	1	1	1	1	1	1	2	1	2	1	1
	Roulette	1	2	1	1	1	1	1	2	1	2	1	1

Table 3.31 shows the results obtained for all the families of instances under LSGA with the Hungarian method (HM) and the ϵ -scaling Auction algorithm (AA). It can be observed that the RSE obtained by the two versions of LSGA are very similar, so we consider both versions are equivalent. Table 3.32 shows the corresponding running times. It can be observed that the version of LSGA with the AA is approximately 5 times faster than the version with the HM.

We implemented the memetic algorithm of [Karapetyan and Gutin, 2011b] but we could not replicate their results through our implementation. We also asked to the authors for their implementation but it was not provided. In order to compare our results against such implementation we decided to consider the RSE obtained by both heuristics. We considered the best results that they reported which were obtained after a running time of 300 seconds.

Tables 3.33 and 3.34 show the results for all the families of instances under LSGA (with AA), our SMA (with SDV3 and DV3), and the memetic algorithm of [Karapetyan and Gutin, 2011b] (MA). All the algorithms were executed for a running time of 300 seconds. For the SMA we considered the parameters $N = 100$, $m = 10$, $sp = 50$, and the combination of the operators deterministic selection, cycle crossover and inversion mutation as we decided after the evaluation of all the combinations of operators. Even when we could not replicate the implementation of MA we are considering their reported results. Their results were obtained through an implementation in Visual C++ and their instances were evaluated on a platform based on AMD Athlon 64 X2 3.0 GHz processor which is a better architecture compared to our Intel Core i5-3210M 2.5 GHz processor. We calculated the results obtained by MA (GK) based on their reported RSE. In the case of the family of instances Random the MA (GK) outperformed the results of our SMA, however our SDV3 and DV3 are

Table 3.31: Random, Clique, Square Root, Geometric, and Product under LSGA.

Inst.	Best known	HM	AA	RSE		Inst.	Best known	HM	AA	RSE	
				HM	AA					HM	AA
4r20	20.0	34.2	33.8	71.0	69.0	4g20	2380.5	2380.5	2380.5	0.0	0.0
4r30	30.0	45.7	45.1	52.3	50.3	4g30	3015.2	3015.2	3015.2	0.0	0.0
4r40	40.0	56.1	55.2	40.3	38.0	4g40	3523.8	3523.8	3523.8	0.0	0.0
4r50	50.0	63.8	63.1	27.6	26.2	4g50	4102.3	4102.3	4102.3	0.0	0.0
5r15	15.0	27.8	26.2	85.3	74.7	5g15	3423.7	3423.7	3423.7	0.0	0.0
5r18	18.0	30.8	30.8	71.1	71.1	5g18	3799.3	3799.3	3799.3	0.0	0.0
5r25	25.0	39.4	39.0	57.6	56.0	5g25	4594.9	4594.9	4594.9	0.0	0.0
5r30	30.0	44.0	44.8	46.7	49.3	5g30	5036.8	5036.8	5037.1	0.0	0.0
6r12	12.0	20.9	21.1	74.2	75.8	6g12	4483.7	4483.7	4483.7	0.0	0.0
6r15	15.0	23.8	25.6	58.7	70.7	6g15	5242.4	5242.4	5242.4	0.0	0.0
6r18	18.0	29.2	28.7	62.2	59.4	6g18	5767.4	5767.4	5767.4	0.0	0.0
Avg	24.8	37.8	37.6	52.4	51.6	Avg	4124.5	4124.5	4124.6	0.0	0.0
4cq20	1901.8	1909.8	1911.2	0.4	0.4	4p20	8397.3	8398.7	8398.7	0.0	0.0
4cq30	2281.9	2298.0	2304.6	0.7	0.9	4p30	13154.1	13155.3	13155.4	0.0	0.0
4cq40	2606.3	2726.6	2718.5	4.6	4.3	4p40	16810.0	16812.9	16812.6	0.0	0.0
4cq50	3032.6	3344.4	3310.6	10.2	9.1	4p50	20705.6	20711.9	20711.0	0.0	0.0
5cq15	3110.7	3114.3	3113.1	0.1	0.0	5p15	21422.8	22035.5	21559.1	2.8	0.6
5cq18	3458.6	3463.1	3465.3	0.1	0.1	5p18	23371.4	23801.6	23495.6	1.8	0.5
5cq25	4192.7	4228.3	4233.2	0.8	0.9	5p25	29693.0	30273.4	30154.1	1.9	1.5
5cq30	4671.7	4714.8	4739.3	0.9	1.4	5p30	34799.4	35985.9	35732.1	3.4	2.6
6cq12	4505.6	4513.0	4511.8	0.1	0.1	6p12	7421.0	10960.6	10861.6	47.6	46.3
6cq15	5133.4	5145.4	5145.3	0.2	0.2	6p15	8888.9	14480.4	16656.9	62.9	87.3
6cq18	5765.5	5795.9	5809.7	0.5	0.7	6p18	9600.2	20561.2	20611.6	114.1	114.6
Avg	3696.4	3750.3	3751.1	1.4	1.4	Avg	17660.3	19743.4	19831.7	11.7	12.2
4sr20	929.3	933.6	936.2	0.4	0.7						
4sr30	1118.0	1136.0	1137.0	1.6	1.6						
4sr40	1271.4	1365.6	1395.2	7.4	9.7						
4sr50	1491.1	1673.2	1680.1	12.2	12.6						
5sr15	1203.9	1207.7	1209.5	0.3	0.4						
5sr18	1343.9	1349.7	1349.1	0.4	0.3						
5sr25	1627.5	1647.0	1647.6	1.1	1.2						
5sr30	1828.1	1872.8	1884.9	2.4	3.1						
6sr12	1436.8	1440.4	1440.6	0.2	0.2						
6sr15	1654.6	1666.9	1664.0	0.7	0.5						
6sr18	1856.3	1872.4	1869.0	0.8	0.6						
Avg	1432.8	1469.6	1473.9	2.5	2.8						

able to obtain the same results within lower running times in some cases (it can be observed at Table 3.17). In the case of the family of instances Clique our results are practically equivalent to theirs. In the case of the family of instances Square Root our results are very near to their results, they obtained a RSE of 0.1 whereas ours was 0.3. Even when the families of instances Geometric and Product were proposed by the same authors in [Karapetyan and Gutin, 2011a] they did not evaluate such instances under their MA (GK). At any case, we can observe that our SMA combined

Table 3.32: Running times for Random, Clique, Square Root, Geometric, and Product under LSGA.

Instance	HG	AA	Instance	HG	AA	Instance	HG	AA
4r20	15.9	2.4	4cq20	11.0	2.3	4sr20	11.3	3.8
4r30	46.3	5.5	4cq30	51.4	8.7	4sr30	71.5	14.4
4r40	65.5	13.5	4cq40	91.0	22.0	4sr40	97.5	22.6
4r50	115.6	17.9	4cq50	146.1	34.5	4sr50	130.4	29.5
5r15	9.0	2.5	5cq15	6.1	2.3	5sr15	8.3	2.7
5r18	14.0	3.6	5cq18	12.4	3.7	5sr18	19.1	5.6
5r25	31.2	9.2	5cq25	48.0	10.7	5sr25	58.9	14.9
5r30	47.3	16.3	5cq30	111.8	23.3	5sr30	127.3	29.7
6r12	8.7	2.7	6cq12	4.9	2.1	6sr12	6.6	3.3
6r15	14.9	6.6	6cq15	12.4	4.3	6sr15	15.8	7.4
6r18	23.5	13.9	6cq18	21.2	9.0	6sr18	32.4	11.7
Avg	35.6	8.5	Avg	46.9	11.1	Avg	52.6	13.2
4g20	5.3	1.6	4p20	22.9	4.7			
4g30	23.5	3.8	4p30	50.7	9.7			
4g40	52.8	7.2	4p40	140.7	14.1			
4g50	88.9	10.7	4p50	216.1	24.9			
5g15	4.7	1.2	5p15	21.7	4.6			
5g18	8.8	2.2	5p18	34.0	8.6			
5g25	22.9	5.9	5p25	98.2	21.6			
5g30	35.6	9.1	5p30	165.7	24.2			
6g12	2.5	1.2	6p12	21.9	7.1			
6g15	8.6	3.0	6p15	36.6	8.9			
6g18	15.9	8.4	6p18	51.4	14.9			
Avg	24.5	4.9	Avg	78.1	13.0			

with the DV2 outperformed to the LSGA.

In conclusion, we consider that the most difficult families of instances are Clique and Square Root for which the use of SMA combined with our DV2 is the most effective option. We suggest the use of the operators deterministic, PMX and IM as the best combination of operators for our SMA. For the rest of cases, the use of SDV3 or DV3 provides higher quality solutions. We consider that a most exhaustive selection of the parameters N , mp , and sp can be performed as well as the evaluation of other operators to be considered as part of our SMA. We let such analysis as future work since for our purposes we have developed a strong set of algorithms, heuristics, and meta-heuristics to solve a wide variety of instances of MAP.

Some of the results obtained by the SMA were accepted for its publication in the 14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2017) [Pérez Pérez et al., 2017b].

Table 3.33: Random, Clique and Square Root under memetic algorithms.

Inst.	Best known	RSE							
		LSGA (AA)	SMA-1 (SDV2)	SMA-2 (DV2)	MA (GK)	LSGA	SMA-1	SMA-2	MA
4r20	20	33.8	27.2	24.7	20.0	69.0	36.0	23.5	0.0
4r30	30	45.1	38.0	34.4	30.0	50.3	26.7	14.7	0.0
4r40	40	55.2	47.0	43.8	40.0	38.0	17.5	9.5	0.0
4r50	50	63.1	56.2	52.2	-	26.2	12.4	4.4	-
5r15	15	26.2	19.8	16.9	15.0	74.7	32.0	12.7	0.0
5r18	18	30.8	23.5	19.8	18.0	71.1	30.6	10.0	0.0
5r25	25	39.0	31.8	27.0	25.0	56.0	27.2	8.0	0.0
5r30	30	44.8	36.5	32.0	-	49.3	21.7	6.7	-
6r12	12	21.1	15.7	12.6	12.0	75.8	30.8	5.0	0.0
6r15	15	25.6	18.9	15.6	15.0	70.7	26.0	4.0	0.0
6r18	18	28.7	23.0	18.7	18.0	59.4	27.8	3.9	0.0
Avg	24.8	37.6	30.7	27.1	24.8	58.2	26.2	9.3	0.0
4c20	1901.8	1911.2	1903.7	1901.8	1901.8	0.5	0.1	0.0	0.0
4c30	2281.9	2304.6	2290.2	2289.2	2297.9	1.0	0.4	0.3	0.7
4c40	2606.3	2718.5	2633.8	2632.3	2618.0	4.3	1.1	1.0	0.5
4c50	3032.6	3310.6	3048.6	3043.6	-	9.2	0.5	0.4	-
5c15	3110.7	3113.1	3110.7	3110.7	3110.7	0.1	0.0	0.0	0.0
5c18	3458.6	3465.3	3458.6	3459.4	3458.6	0.2	0.0	0.0	0.0
5c25	4192.7	4233.2	4217.7	4202.8	4195.2	1.0	0.6	0.2	0.1
5c30	4671.7	4739.3	4704.7	4705.2	-	1.4	0.7	0.7	0.0
6c12	4505.6	4511.8	4505.6	4505.6	4505.6	0.1	0.0	0.0	0.0
6c15	5133.4	5145.3	5134.9	5133.6	5133.4	0.2	0.0	0.0	0.0
6c18	5765.5	5809.7	5785.3	5777.8	5769.0	0.8	0.3	0.2	0.1
Avg	3696.4	3751.1	3708.5	3705.6	3699.5	0.9	0.2	0.1	0.1
4sr20	929.3	936.2	931.3	929.8	929.3	0.7	0.2	0.1	0.0
4sr30	1118.0	1137.0	1122.5	1120.8	1119.5	1.7	0.4	0.3	0.1
4sr40	1271.4	1395.2	1286.5	1282.2	1276.6	9.7	1.2	0.8	0.4
4sr50	1491.1	1680.1	1504.6	1494.8	-	12.7	0.9	0.2	0.0
5sr15	1203.9	1209.5	1204.7	1204.2	1203.9	0.5	0.1	0.0	0.0
5sr18	1343.9	1349.1	1346.7	1346.6	1343.9	0.4	0.2	0.2	0.0
5sr25	1627.5	1647.6	1639.4	1636.6	1629.8	1.2	0.7	0.6	0.1
5sr30	1828.1	1884.9	1834.8	1839.3	-	3.1	0.4	0.6	0.0
6sr12	1436.8	1440.6	1437.5	1436.8	1436.8	0.3	0.0	0.0	0.0
6sr15	1654.6	1664.0	1657.7	1660.4	1654.6	0.6	0.2	0.4	0.0
6sr18	1856.3	1869.0	1860.1	1861.7	1857.0	0.7	0.2	0.3	0.0
Avg	1432.8	1473.9	1438.7	1437.6	1433.7	1.8	0.4	0.3	0.1

3.6 General results

We have developed a wide variety of algorithms, heuristics, and meta-heuristics for MAP. We have compared our results against the state of the art techniques obtaining a new set of techniques that allows to obtain competitive results. In order to provide a better idea about the results obtained we summarize all our best results in this section.

Table 3.34: Geometric and Product under memetic algorithms.

Instance	Best known	LSGA (AA)	SMA-1 (SDV2)	SMA-1 (DV2)	RSE		
					LSGA	SMA-1	SMA-2
4g20	2380.5	2380.5	2380.5	2380.5	0.0	0.0	0.0
4g30	3015.2	3015.2	3015.2	3015.2	0.0	0.0	0.0
4g40	3523.8	3523.8	3523.8	3523.8	0.0	0.0	0.0
4g50	4102.3	4102.3	4102.3	4102.3	0.0	0.0	0.0
5g15	3423.7	3423.7	3423.7	3423.7	0.0	0.0	0.0
5g18	3799.3	3799.3	3799.3	3799.3	0.0	0.0	0.0
5g25	4594.9	4594.9	4594.9	4594.9	0.0	0.0	0.0
5g30	5036.8	5037.1	5036.8	5036.8	0.0	0.0	0.0
6g12	4483.7	4483.7	4483.7	4483.7	0.0	0.0	0.0
6g15	5242.4	5242.4	5242.4	5242.4	0.0	0.0	0.0
6g18	5767.4	5767.4	5767.4	5767.4	0.0	0.0	0.0
Avg	4124.5	4124.6	4124.5	4124.5	0.0	0.0	0.0
4p20	8397.3	8398.7	8397.5	8397.3	0.0	0.0	0.0
4p30	13154.1	13155.4	13154.3	13154.2	0.0	0.0	0.0
4p40	16810.0	16812.6	16810.6	16810.2	0.0	0.0	0.0
4p50	20705.6	20711.0	20708.7	20706.2	0.0	0.0	0.0
5p15	21422.8	21559.1	21595.8	21423.0	0.6	0.8	0.0
5p18	23371.4	23495.6	24314.5	23371.8	0.5	4.0	0.0
5p25	29693.0	30154.1	32346.3	30583.5	1.6	8.9	3.0
5p30	34799.4	35732.1	39773.5	36412.0	2.7	14.3	4.6
6p12	7421.0	10861.6	14898.6	8021.7	46.4	100.8	8.1
6p15	8888.9	16656.9	20724.8	9619.7	87.4	133.2	8.2
6p18	9600.2	20611.6	26376.0	10814.9	114.7	174.7	12.7
Avg	17660.3	19831.7	21736.4	18119.5	23.1	39.7	3.3

Table 3.35 shows the results for the families of instances Random, Clique and Square Root under MAP-Gurobi, the iterated local search combined with DV2, DV3, SMA combined with DV2 and the memetic algorithm proposed by Gutin and Karapetyan MA (GK). In the case of the family of instances Random, it can be observed that DV3 and MA (GK) provide the best results. In the case of the family of instances Clique, it can be observed that MA (GK) provides the best results, however the results obtained by our SMA (DV2) are very competitive against MA results. In the case of the family of instances Square Root, it can be observed that MA (GK) provides the best results, however the results obtained by our SMA (DV2) are also competitive against MA results.

The running times for MAP-Gurobi can be consulted at the Tables 3.3, 3.4, and 3.5 whereas running times for DV3 can be consulted at the Table 3.17. In the case of DV2 (ILS) the results reported correspond with running times of 30 seconds (recall more than this time did not provided better results). In the case of the memetic algorithms SMA and MA the results reported correspond with running times of 300 seconds. We focused our attention in the quality solution but not in the execution time to get a solution. Even when we have a considerable difference in seconds for the compared

Table 3.35: General results for Random, Clique and Square Root under the best techniques.

Instance	Best known	MAP-Gurobi	DV2 (ILS)	DV3	SMA (DV2)	MA (GK)
4r20	20	20.0	25.2	20.0	24.7	20.0
4r30	30	30.0	36.0	30.0	34.4	30.0
4r40	40	40.0	44.3	40.0	43.8	40.0
5r15	15	15.0	18.0	15.0	16.9	15.0
5r18	18	18.0	21.4	18.0	19.8	18.0
5r25	25	-	29.3	25.0	27.0	25.0
6r12	12	12.0	13.7	12.0	12.6	12.0
6r15	15	-	17.1	15.0	15.6	15.0
6r18	18	-	20.0	18.0	18.7	18.0
Avg	21.4	-	25.0	21.4	23.7	21.4
RSE	-	-	16.8	0.0	10.7	0.0
4cq20	1901.8	1901.8	1910.9	1909.0	1901.8	1901.8
4cq30	2281.9	-	2331.9	2322.6	2289.2	2297.9
4cq40	2606.3	-	2705.9	2714.2	2632.3	2618.0
5cq15	3110.7	3110.7	3116.7	3128.0	3110.7	3110.7
5cq18	3458.6	3458.6	3485.2	3507.9	3459.4	3458.6
5cq25	4192.7	-	4268.8	4337.2	4202.8	4195.2
6cq12	4505.6	4505.6	4521.9	4532.6	4505.6	4505.6
6cq15	5133.4	-	5179.0	5216.7	5133.6	5133.4
6cq18	5765.5	-	5852.3	5895.5	5777.8	5769.0
Avg	3661.8	-	3708.1	3729.3	3668.1	3665.6
RSE	-	-	1.3	1.8	0.2	0.1
4sr20	929.3	929.3	937.5	937.4	929.8	929.3
4sr30	1118.0	-	1146.5	1153.2	1120.8	1119.5
4sr40	1271.4	-	1340.6	1337.0	1282.2	1276.6
5sr15	1203.9	1203.9	1207.9	1215.4	1204.2	1203.9
5sr18	1343.9	1343.9	1358.7	1369.3	1346.6	1343.9
5sr25	1627.5	-	1675.5	1703.9	1636.6	1629.8
6sr12	1436.8	1436.8	1439.8	1447.6	1436.8	1436.8
6sr15	1654.6	-	1668.0	1689.1	1660.4	1654.6
6sr18	1856.3	-	1897.1	1911.9	1861.7	1857.0
Avg	1382.4	-	1408.0	1418.3	1386.6	1383.7
RSE	-	-	1.9	2.6	0.3	0.1

techniques, we consider that this difference is negligible because all the compared techniques solve instances in a few minutes, which matches with the purposes of this thesis. We believe that can be developed faster techniques, however we have covered our purposes which were to provide techniques that were able to obtain high quality solutions for the MAP. We let the study of faster techniques as future work.

Chapter 4

Personnel assignment problems: the school timetabling problem as a case 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 it is required to assign 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, research line, approval rating, number of times that such a class was taught, among others. In addition, it can 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 has 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 called *school timetabling problem* (STP).

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

We will tackle the school timetabling problem as a case 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 focus our study in personnel assignment problems 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 year, [Lawrie, 1969] described an integer linear programming model of a school timetabling but considered each combination of items or sets under bipartite graphs, then combined the solutions to get a set of timetables. He solved his model by applying some branching procedures.

In the mid 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 to start to evaluate 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 a special room). We adopt some of this ideas in order to add them into our formulation.

In the mid eighties, [de Werra, 1985] provided some ideas with an emphasis on graph theoretical models. This work is another excellent reference to start to model an STP in many different ways adopting the best one according to the problem restrictions. In his most relevant model, he described a model aimed to assign classes to teachers to time slots through a couple of bipartite graphs. Instead of solving a 3AP he joined 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 has been commonly used by many authors, since the complexity of solving this problem is similar to solving 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 a cost involved. 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 mid 2000's, [Cambazard et al., 2005] proposed an interactive constraint programming model in which one can either add or remove constraints. This type of techniques are called dynamic constraint satisfaction problems in which we have a sequence of static constraint satisfaction problems where each constraint satisfaction problem is the consequence of the 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 themselves. 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 pre-established 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, for example tournament and elitist, and a mutation operators that changes dynamically its probability of being applied when no better solutions are found. In the same year, [Jat and Yang, 2008] proposed another 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 measures 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, determines 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.

Recently, several genetic and memetic algorithms have been proposed for the STP. For example [Raghavjee and Pillay, 2009], [Qaurooni, 2011], [Budiono and Wong, 2011], [Doulaty et al., 2013], [Fonseca and Santos, 2013], but they only vary 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, none of these procedures solves 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 s -dimensional assignment problem (sAP). For example, for a given group of students, it can be required the assignment of n_1 professors to n_2 courses to n_3 time slots. The corresponding 3-dimensional matrix of costs is $C^{n_1 n_2 n_3}$ where the entry c_{ijk} is related to the cost of assigning the professor p_i to the class c_j to the time slot t_k with $1 \leq i \leq n_1$, $1 \leq j \leq n_2$, and $1 \leq k \leq n_3$. The goal is to find an assignment of minimum cost, then the corresponding 0-1 integer linear programming formulation is:

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

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

The first set of restrictions establishes that a professor will be assigned to one class at some time slot. The second set of restrictions establishes that each class will be taught by one professor at some time slot. The last set of restrictions establishes that each time slot will be available for one professor teaching some course. It can be considered that not necessarily $n_1 = n_2 = n_3$, hence some elements from each set will not belong to the final assignment.

In practice, the set of professors and courses will be larger than the set of time slots. In addition, if other sets are involved we can consider them as extra dimensions, for example sometimes should be considered the set of classrooms and groups of students as the dimensions fourth and five. However, each classroom can frequently be used at every time slot and a time slot can be used to teach 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 by a particular group of people, which is the case that we are considering.

Even when a 3AP remains NP-hard, our MAP-Gurobi is able to solve exactly instances of 3AP up to 130 vertices by dimension in some minutes, whereas our Simple Memetic Algorithm provides an excellent meta-heuristic to obtain feasible solutions of high quality for 3AP with more than 130 vertices by dimension.

4.2.1 Setting up the costs matrix for the STP

One of the main difficulties is to propose a reasonable 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 teach a very small number of courses, usually less than 10. Moreover, each professor is only able to teach such class in a subset of the available time slots. For instance, in some universities time slots use to be one hour, one hour and a half or two hours. When time slots are one hour then a day can have at most 15 time slots (considering the length of a school day from 7:00 a.m. until 10 p.m.). Usually professors are only available for eight hours from a predefined schedule time, which represents only the half of such time slots.

In terms of the courses, some of them need to be taught 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 that can helps 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 generate an instance in a similar way to the Clique family of instances for the MAP.

In this case we require the following bipartite graphs:

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

There are other combinations, however they do not represent realistic restrictions or are already cover at the provided combinations, for example, it is not common to have a professor with a preference over some classroom (although sometimes a special requirement must be satisfied) in which case such restriction will be excluded from the assignment problem.

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

For example, suppose we have $n_1 = 4$, $n_2 = 4$, $n_3 = 4$ and $n_4 = 4$. Suppose that professor p_2 is only able to teach the courses 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 or r_4 whereas the class c_2 requires to be set in the classrooms r_1 or 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 than others.

For example it may 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 another 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 from, instead of having just one solution as in the binary costs matrix definition.

4.2.2 Dealing with a different number of vertices per 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 balance 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 considerably the complexity of an instance.

Suppose we 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 vectors in the final assignment that are invalid because they consider some dummy vertices. Such vectors should be taken out of the solution as well as the invalid vectors due to the corresponding restrictions according to the selected model to build the matrix costs.

4.3 A real life instance: scheduling classes at UAM

We obtained a real data set which corresponds to the information of 9 years of classes scheduling at the Faculty of Basic Sciences and Engineering of the Universidad Autónoma Metropolitana campus Azcapotzalco (UAM-A). Here we summarize some important aspects and considerations about the classes scheduling in this faculty.

- This faculty is divided into 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 *unidad de enseñanza-aprendizaje*).
- For each quarter there is a minimum requirement of courses that should be covered.
- Some courses can be required to have more than one class.
- Each course is required to be taught a specific number of hours in a week.
- Most courses are taught a multiple of 90 minutes.
- The most common cases are to have courses of 270 minutes or 180 minutes and the less common cases are of 90, 360 or 450 minutes in a week.

- Courses with more than 450 minutes are usually 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 studying such course.
- The possible notes obtained by 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 number 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 the cases, a course should preferably be assigned at the same time slot on each week day that it uses.

The real data set is presented in a large spreadsheet. The table is composed by the following 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.
- 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. The data considers records from many other departments because there are courses from other Faculties (such as the Faculty of Social Sciences and Humanities and the Faculty of Sciences and Arts for Design) that are shared by students from this Faculty. In total, 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 by giving a weight based on the approval count or the number of declines, however we are considering a more general case in which these aspects may not be 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 fewer than 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 assigning a professor to more than one course at the same time slot.
- In order to have the UEA requirement we can consider the assigned courses for the corresponding quarter. This means that we will perform assignments over the quarters from the data set.

- In order to obtain the available working day of each professor for a quarter to solve, we consider the time slots at which such professor was assigned in at least one class during the previous nine quarters. In practice, these data must be part of the input, however by this moment we do not have the records of such historical data. Such data must belongs to the department of Human Resources or could be collected from the preferred time slots of each professor.
- For the set of courses for which a professor will be available we consider those courses that were taught during the previous nine quarters.
- We consider time slots of 90 minutes starting at 7:00 a.m. and finishing at 10:00 p.m. This gives 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 a schema is designed to create time periods which allow us 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. We have 5 categories of courses, it means those required in either 1, 2, 3, 4 or 5 time slots. This means that we will be solving approximately 34×5 (number of departments \times number of categories of courses) 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 (as many as the number of time slots). The number of all possible combinations can be really high and unrealistic for most cases. For example, a course that should be set at two time slots could be assigned on Monday at 7:00 and on Friday at 19:00, which is not desirable. The time periods schema that we consider reduces 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 on Table 4.2. Additionally, we show a binary representation of the days of the week considered for the required time slots where the left most bit corresponds to Monday and the right most bit 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 respectively. By considering the starting points for the time slots, this gives us a maximum of 10×10 unique time periods among which the courses could be assigned.

By considering all the previous assumptions, we proposed a new approach to solve the STP at UAM-A that is based on the resolution of several 3AP to satisfy the quarterly UEA requirement at the faculty of Basic Sciences and Engineering at UAM-A. Algorithm 29 shows our proposed solution.

The process shown in Algorithm 29 is as follows: first the list of assignments is set to empty. The cycle on line 3 allows to divide the UEA requirements by department and the cycle on 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 on line 5. On line 6 the time slots are converted into time periods according with the rules previously described. The cycle on line 8 will be executed while we have required courses to assign or until the process can not set a required course. On line 10 we obtain all the possible feasible assignments based on the list of ProfessorsToCourses, the professor availability, and the given time periods. Then, on line 11 we generate the binary costs matrix for the 3AP by adding the required dummy vertices if necessary. On line 12 the corresponding 3AP is solved and then on 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. On 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 as many times as the number of required courses in the worst case. However in practice, just a few executions were required before filling the requirements or not finding more feasible

Algorithm 29: 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 :=  $\emptyset$ ;
2 foreach department  $\in$  Departments do
3   for blocks from 5 to 1 do
4     CoursesToSet := GetCoursesWithBlocks(UEAReqByD[department],
5     TimeSlotsByCourse, blocks);
6     TimePeriods := GetCombinationsOfTimePeriods(blocks);
7     MissingCourses := |CoursesToSet|+1;
8     while MissingCourses > |CoursesToSet| and |CoursesToSet| > 0 do
9       MissingCourses := |CoursesToSet|;
10      possibleAssignments := getAssignments(ProfessorsToCourses,
11      ProfessorAvailability, TimePeriods);
12      costsMatrix := generateBinaryCostsMatrix(possibleAssignments);
13      finalAssignment := SolveMAP(costsMatrix);
14      validAssignment := GetFeasibleAssignments(finalAssignment);
15      UpdateProfessorAvailability(ProfessorAvailability,
16      validAssignment);
17      ListOfAssignments := ListOfAssignments  $\cup$  validAssignment;
18      UpdateCoursesToSet(CoursesToSet, validAssignment);
19 Set classesScheduling := getCompleteScheduling(ListOfAssignments);
20 return {classesScheduling};

```

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 are taught usually by a small number of professors therefore they have a higher priority to be assigned 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 when the number of vertices is at most 130.
2. Simple Memetic Algorithm (SMA). It is used when the number of vertices is greater than 130.

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 does not satisfy the 100.0% of the UEA requirement in all the cases because we are only considering the historical time periods and the historical courses taught for each professor. The actual historical information used for the corresponding quarter to solve 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 scheduled on different days and at different starting times or when a course with two time slots should be set in two consecutive time slots on 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 for the 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 received such new information then the quality of the assignment could be higher.

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, 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 86.0%, even when in our assignment we did not consider 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 by only considering the historical information about previously taught courses and historically observed

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

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. We let such evaluation as future work.

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 research, first the results about algorithms and heuristics for the resolution of the multidimensional assignment problem and, second, the modeling and resolution of personnel assignment problems, in particular the school timetabling problem, through the multidimensional assignment problem. Even if such lines are related, we decided to divide our perspective into both lines in order to show the progress on each research 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 an 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.

Another 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 this case the ϵ -scaling auction algorithm is approximately 30 times faster than the Hungarian method.

5.2 The multidimensional assignment problem

The multidimensional assignment problem is known to be an NP-hard problem and even when better algorithms for this problem can be proposed in the future, unless $P = NP$, it 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 sizes 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 defeated a state-of-the-art technique for the particular cases of instances of 3AP up to 130 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 allows 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 very high 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 0.6%, and very competitive results for the families of instances Clique and Square Root, with relative solution errors of approximately 1.8% and 2.6% correspondingly. Some of these results were accepted for its publication on Studies in Computational Intelligence [Pérez Pérez et al., 2017a]. We also determined that the combination of an ILS with DV2 provides even higher quality solutions for the families of instances Clique and Square Root. This combination obtained a relative solution error of 1.3% for the family of instances Clique and of 1.9% for the family of instances Square Root, which outperformed the results obtained by DV3 for these families of instances. We propose the combination of an ILS with DV3 as future work.

We proposed a new state-of-the-art simple memetic algorithm for the MAP 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 on the combination of the method used 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 errors 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 0.2% and 0.3% respectively after running times of 300 seconds, which is similar to the reported relative solution errors of 0.1% and 0.1% for the state-of-the-art memetic algorithm after the same running time. Some of these results were accepted for its publication in the 14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2017) [Pérez Pérez et al., 2017b].

We constructed a 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 through a multidimensional assignment problem for the cases where assignments between persons and two or more other disjoint sets of things of concepts are involved. 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 criterion is good for the cases where only feasibility is important whereas the second criterion tries to measure priorities over the assignment. These 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 to the given methodologies and the proper restrictions and considerations of the problem to solve.

We considered the problem of the Faculty of Basic Sciences and Engineering 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 could be able to satisfy the UEA requirement for the a particular quarter. The more realistic case is to consider the actual requirements for a new quarter and the actual restrictions of the professors however, while this thesis was written, 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 90% by considering historical data. We suppose that the use of the proposed solution, not only using historical data, but with the actual

data to be used for the classes scheduling of a new quarter can give even better results and can be also useful to decide whether new hirings are required in order to satisfy the UEA requirements or whether these can be satisfied with the available personnel at that moment. We were able to add several restrictions to our solution and to obtain results in a few minutes, whereas the current mechanism can take more than a few days depending on the restrictions for each quarter. We conclude that our tool is able to be tested and applied to future scenarios of classes scheduling at UAM-A.

5.4 Future work

In the case of MAP, we believe that should be possible to develop a better algorithm than MAP-Gurobi for solving it because even when MAP-Gurobi represents a new state of the art algorithm, it is based on a generic machinery. We only considered the versions SDV2, DV2, SDV3, and DV3 of GDVH, however we believe that other heuristics like SDV4, DV4, or in general SDV($s - 1$) and DV($s - 1$) could 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 can also explore over other selection functions, crossover, and mutation operators such that the development of other more specific for MAP could provide even higher quality solutions. In addition, one could consider 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 solutions.

In the case of the STP, new ways to build the costs matrix for the corresponding sAP 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 our 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.

Appendix A

Best known solutions

Here we show the best known solutions for the families of instances Random, Clique, Square Root, Geometric and Product which were considered in our experimental evaluations. It includes the original instances for the families Random, Clique and Square Root provided by Gutin and Karapetyan in [Karapetyan and Gutin, 2011b] and the extra data set with $(s = 3, n = 130)$, $(s = 4, n = 50)$, and $(s = 5, n = 30)$. The last column of each table is the average value for the best known solutions (or optimal in some cases) for the ten samples of each type of instance, defined for a pair of s and n and for the corresponding family of instances. The average value of the best known solutions (or optimal in some cases) is the reference that we used for calculating the relative solution error in our reported results. In the next tables (except for the family of instances Random), we show a row colored in green for those solutions that correspond with the optimal value. For the cases where we know the optimal solutions they were found thanks to our MAP-Gurobi. The rest of values that we have as the best known solutions were obtained from the best solution found after all the executions that we performed over each problem instance considering all our developed techniques. In most of the cases we obtained the same averages for each set of ten instances as those obtained in [Karapetyan and Gutin, 2011b].

Table A.1 shows the optimal solutions for the family of instances Random. This is the only case in which we know that the values of the best solutions obtained correspond with the optimal solution without considering our MAP-Gurobi. The weight of any hyperedge of an instance for this family is not lower than one. Since the costs of the best solutions obtained are equal to the number of vertices n then the best solutions obtained correspond with the optimal solutions. We included our 30 samples generated for 3r130, 4r50, and 5r30.

Table A.2 and shows the best known solutions for the family of instances Clique. We included our 30 samples generated for 3cq130, 4cq50, and 5cq30. It is important to mention that in the case of the set of instances 3cq70 the best solution reported was 1158.4 whereas we found that the optimum value corresponds with 1157.1. We also found that for the set of instances 3cq100 the best solutions reported was 1368.1 whereas we found that the optimum value corresponds with 1345.9.

Table A.1: Optimal solutions for the family of instances Random.

Instance	1	2	3	4	5	6	7	8	9	10	Avg
3r40	40	40	40	40	40	40	40	40	40	40	40
3r70	70	70	70	70	70	70	70	70	70	70	70
3r100	100	100	100	100	100	100	100	100	100	100	100
3r130	130	130	130	130	130	130	130	130	130	130	130
4r20	20	20	20	20	20	20	20	20	20	20	20
4r30	30	30	30	30	30	30	30	30	30	30	30
4r40	40	40	40	40	40	40	40	40	40	40	40
4r50	50	50	50	50	50	50	50	50	50	50	50
5r15	15	15	15	15	15	15	15	15	15	15	15
5r18	18	18	18	18	18	18	18	18	18	18	18
5r25	25	25	25	25	25	25	25	25	25	25	25
5r30	30	30	30	30	30	30	30	30	30	30	30
6r12	12	12	12	12	12	12	12	12	12	12	12
6r15	15	15	15	15	15	15	15	15	15	15	15
6r18	18	18	18	18	18	18	18	18	18	18	18

Table A.2: Best known solutions for the family of instances Clique.

Instance	1	2	3	4	5	6	7	8	9	10	Avg
3cq40	944	927	948	1022	849	926	957	900	950	976	939.9
3cq70	1089	1133	1195	1111	1224	1204	1122	1183	1153	1157	1157.1
3cq100	1333	1394	1340	1357	1357	1336	1330	1357	1314	1341	1345.9
3cq130	1494	1604	1632	1575	1601	1596	1585	1521	1644	1446	1569.8
4cq20	1764	2043	2019	1923	1876	1905	1928	1803	1923	1834	1901.8
4cq30	2274	2254	2279	2377	2345	2289	2222	2325	2270	2188	2281.9
4cq40	2745	2500	2673	2559	2682	2610	2648	2600	2576	2597	2606.3
4cq50	2986	2983	3085	3037	2918	3059	3126	2980	3059	3093	3032.6
5cq15	3220	3126	3069	3158	2983	3193	3400	3100	2871	2987	3110.7
5cq18	3651	3324	3710	3529	3556	3292	3420	3265	3433	3406	3458.6
5cq25	4242	4171	4245	4228	4063	4175	4230	4065	4326	4182	4192.7
5cq30	4794	4593	4647	4900	4662	4759	4497	4616	4676	4573	4671.7
6cq12	4683	4363	4418	4269	4607	4562	4360	4698	4605	4491	4505.6
6cq15	5232	5203	5162	4967	5178	5271	5217	4885	5045	5174	5133.4
6cq18	5689	5811	5823	5718	5784	5907	5751	5890	5598	5700	5765.5

Table A.3 shows the best known solutions for the family of instances Square Root. We included our 30 samples generated for 3sr130, 4sr50, and 5sr30. In the case of the sets of instances 3cq40, 3sr70 and 3sr100 the best solutions reported were 610.6, 737.1, and 866.3 whereas we found that the optimum values correspond with 606.9, 733.6, and 838.1.

Table A.3: Best known solutions for the family of instances Square Root.

Instance	1	2	3	4	5	6	7	8	9	10	Avg
3sr40	603	592	607	664	561	595	629	586	597	635	606.9
3sr70	684	728	772	707	778	768	703	750	726	720	733.6
3sr100	838	853	831	846	859	826	828	849	816	835	838.1
3sr130	935	1014	982	1006	969	988	947	996	906	986	972.9
4sr20	860	979	959	956	917	929	983	867	944	899	929.3
4sr30	1111	1125	1097	1162	1127	1134	1076	1171	1110	1067	1118.0
4sr40	1330	1236	1302	1281	1297	1288	1306	1288	1275	1252	1271.4
4sr50	1469	1476	1525	1455	1463	1513	1526	1489	1512	1483	1491.1
5sr15	1242	1201	1194	1249	1156	1253	1312	1173	1078	1181	1203.9
5sr18	1406	1308	1429	1353	1371	1292	1331	1285	1321	1343	1343.9
5sr25	1661	1631	1675	1651	1581	1634	1634	1593	1669	1582	1627.5
5sr30	1874	1810	1806	1889	1820	1881	1769	1801	1845	1786	1828.1
6sr12	1495	1383	1410	1356	1464	1446	1395	1515	1456	1448	1436.8
6sr15	1652	1683	1698	1618	1670	1688	1668	1599	1616	1654	1654.6
6sr18	1846	1897	1897	1823	1837	1881	1863	1879	1811	1831	1856.3

Table A.4 shows the best known solutions for the family of instances Geometric. All these instances were designed for the purposes of this thesis. This family of instances is one of the easiest families. Most of the problem instances were solved through our MAP-Gurobi.

Table A.5 shows the optimal solutions for the family of instances Product. All these instances were designed for the purposes of this thesis. This family of instances is one of the most difficult families. Most of the problem instances could not be solved through our MAP-Gurobi.

Table A.4: Best known solutions for the family of instances Geometric.

Instance	1	2	3	4	5	6	7	8	9	10	Avg
3g40	1857	1471	1752	1761	1482	1697	1420	1730	1998	2143	1731.1
3g70	2530	2876	2263	2102	2155	2571	2229	2277	2530	2420	2395.3
3g100	2744	2930	2401	2837	3273	3054	3059	3115	3494	2730	2963.7
3g130	3059	3545	4138	4186	3521	3453	3671	3250	3529	3022	3537.4
4g20	2459	2149	2270	2392	2223	2842	2199	2963	2158	2150	2380.5
4g30	3117	2934	2602	2542	2886	3148	2747	2633	3480	4063	3015.2
4g40	3603	4042	3528	3226	3332	3238	3696	4265	2858	3450	3523.8
4g50	4327	4283	3923	4082	3660	4012	4521	4551	3967	3697	4102.3
5cq15	3220	3126	3069	3158	2983	3193	3400	3100	2871	2987	3110.7
5cq18	3651	3324	3710	3529	3556	3292	3420	3265	3433	3406	3458.6
5cq25	4242	4171	4245	4237	4063	4175	4230	4065	4326	4182	4193.6
5cq30	4794	4625	4669	4900	4662	4767	4497	4616	4676	4573	4677.9
6cq12	4683	4363	4418	4269	4607	4562	4360	4698	4605	4491	4505.6
6cq15	5232	5203	5162	4967	5178	5271	5217	4885	5045	5174	5133.4
6cq18	5689	5811	5823	5718	5784	5907	5751	5890	5598	5700	5767.1

Table A.5: Best known solutions for the family of instances Product.

Instance	1	2	3	4	5	6	7	8	9	10	Avg
3p40	3841	4083	3811	4898	2960	3444	2773	4713	3187	4591	3830.1
3p70	6306	6196	8185	5111	6636	6083	7508	5309	7033	5609	6397.6
3p100	9729	7735	8964	8641	10306	8114	9657	8125	7178	11292	8974.1
3p130	13675	13902	9926	13309	11241	11316	10539	12767	9554	13234	11946.3
4p20	7405	8375	7925	11299	11733	12828	4308	5596	8734	5770	8397.3
4p30	13162	14102	12997	18177	9280	11353	8522	17114	10260	16574	13154.1
4p40	15320	16670	23027	14868	13984	13912	18139	19578	19014	13588	16810.0
4p50	19394	15872	32191	15110	16594	23106	25597	18544	19720	20928	20705.6
5p15	17542	20300	25710	15693	21278	40216	14237	17960	31012	10280	21422.8
5p18	17405	27204	25689	17504	37318	19650	17606	30716	16514	24108	23371.4
5p25	26182	28026	28177	39686	33103	28061	23175	35135	20066	35319	29693.0
5p30	32478	20904	27964	50082	38184	25262	45056	32557	42027	33480	34799.4
6p12	6612	7904	5260	6878	4920	12846	13234	5024	3312	8220	7421.0
6p15	7909	8992	6312	6304	14266	12434	5402	13188	7066	7016	8888.9
6p18	8049	8372	9022	11526	15522	3916	10761	11682	7844	9308	9600.2

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 randomized 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 traveling 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., Castanon, D. A., 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., Carpenste, 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, 1985] Gabow, H. N. (1985). Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31(2):148 – 168.
- [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.
- [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). Alleles loci and 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). Algorithms in gurobi.
- [Gurobi Optimization, 2017a] Gurobi Optimization, I. (2017a). Gurobi optimizer.
- [Gurobi Optimization, 2017b] Gurobi Optimization, I. (2017b). 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 Kaklamani, 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.

- [Jarque and Bera, 1987] Jarque, C. M. and Bera, A. K. (1987). A test for normality of observations and regression residuals. *International Statistical Review / Revue Internationale de Statistique*, 55(2):163–172.
- [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.
- [Karmarkar, 1984] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395.
- [Karp, 1972] Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.
- [Kocsis and György, 2009] Kocsis, L. and György, A. (2009). *Efficient Multi-start Strategies for Local Search Algorithms*, pages 705–720. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [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.
- [Kuhn and Tucker, 1951] Kuhn, H. W. and Tucker, A. W. (1951). Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, Calif. University of California Press.
- [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 Algorithms ODSA 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. Technical Report C3P 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena CA.
- [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.
- [Pérez Pérez et al., 2017a] Pérez Pérez, S. L., Valencia, C. E., and Zaragoza Martínez, F. J. (2017a). *A New Local Search Heuristic for the Multidimensional Assignment Problem*, pages 183–202. Springer International Publishing, Cham.
- [Pérez Pérez et al., 2017b] Pérez Pérez, S. L., Valencia, C. E., and Zaragoza Martínez, F. J. (2017b). A simple but effective memetic algorithm for the multidimensional assignment problem. *2017 14th International Conference on Electrical Engineering, Computing Science and Automatic Control*, pages 1–6.
- [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. *Proceedings 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 assignment 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 school timetables. *IFIP Conference*.