POLITECNICO DI TORINO

SCUOLA INTERPOLITECNICA DI DOTTORATO

Dottorato in Computer and Control Engineering

Tesi di Dottorato

# Exact and Heuristic Hybrid Approaches for Scheduling and Clustering Problems




**Michele Garraffa**

**Tutore**
prof. Federico Della Croce di Dojola

**Coordinatore del corso di dottorato**
prof. Matteo Sonza Reorda

April 2016

# Acknowledgements

In November 2015, I attended an enlightening talk given by prof. Luciano Floridi from Oxford University. My being there was just a coincidence: I was waiting to receive the annual award for the bests PhD students of my institution. It was definitely not a waste of time. On the contrary, I found a lot of interesting remarks about what does it mean to do scientific research. The title of the talk was: “How do you get good ideas?". As said by prof. Floridi, the key point to find an answer to this tricky question is “the problems". A researcher should enjoy them, see them from each different sides and finally a brillant solution will arise naturally. Tackling interesting scientific problems is fundamental for any researcher, because they help us to improve from a scientific point of view, just like everyday difficulties help us to be better people. Well, I am proud to say that I learned a lot in the last three years, because I loved to spend time with combinatorial problems and I wanted to learn from all the difficulties I went through.

I had the chance to meet a huge number of people thanks to the PhD activities, and I would like to thank all of them. Knowing that it will be impossible to be exahustive, I try to provide a quite representative list. First, I want to thank my supervisor prof. Federico Della Croce, for being an excellent guide along all these years and for all the opportunities he gave me. Furthermore, I want to thank the supervisors I had in the research period I spent in Belgium (Ghent) and France (Tours): prof. Greet Vanden Berghe and prof Vincent T’kindt, respectively. I visited Greet’s research group at the end of 2012 (right before the start of my PhD) and in 2015 for a few days. I am grateful for the fact that she believed in me and she allowed me to have an exceptional research experience in the beginning of my academic career. I spent most of the 2015 in Tours, where I worked with Vincent and Lei Shang for almost 7 months. A special thank goes to Vincent for the support

he gave me in this research period and for the interesting topic he proposed me: the design of exact exponential algorithms for scheduling problems. I was enthusiastic to work on such a theoretical field, and it was really rewarding to obtain some results. In the months spent in Tours, Lei was an excellent research partner and a very good friend. I learned a lot from him and I feel honestly indebted. I want to thank all the other members of the ALCO group at Politecnico di Torino, in particular Marco Ghirardi, Fabio Salassa and Rosario Scatamacchia. Thanks for their support and help to all the colleagues and friends I met in the labs at Torino, Ghent and Tours, among which: Luca, Francesca, Elisa, Mauro, Edoardo, Wim, Tony, Pieter, Jannes, Gaetan, Faiza, Frédéric, Zeina.

Special thanks are due also to the friends I had “outside academia" that somehow gifted me something important in these years. Calogero has been a very important friend with whom I shared every difficulty and happy moment. My cousin Massimilano helped me thousands of times with his wise hints. Thanks to Barbara for being a very supportive friend and for all the precious hours she spent talking with me on Skype. Thanks to Maria for still helping me with my poor English. I am also grateful to my gorgeous girlfriend Clara, for being so sweet and supportive in the last 2 years and a half and loving me like nobody has ever done before.

Most importantly, none of this could have happened without my family. I want to spend the final paragraph to thank them: my mom Anna, my brother Giuseppe and finally my grandparents Michele e Sina, who passed away in the last years. My mother and my brother lovingly supported me in all my studies from the beginning. The love I feel for them cannot be expressed in words. Finally, thanks to my grandparents who had a lot of infuence on me. They will keep living in my heart. All the merit for my achievements should be somehow shared with all of them.

# Contents

**Acknowledgements** II

**1 Introduction** 1

1.1 Matheuristics . . . 6

1.1.1 Local Branching . . . 7

1.1.2 Variable Partitioning Local Search . . . 9

1.1.3 Continuous Relaxation Based Matheuristics . . . 10

1.1.4 Column Generation Based Heuristics . . . 11

1.2 Semidefinite Programming for Combinatorial Optimization . . . 13

1.3 The Design of Exact Exponential Algorithms . . . 17

1.3.1 The Branch and Reduce Paradigm . . . 18

1.3.2 Dynamic Programming across the Subsets . . . 19

1.3.3 Other Techniques . . . 20

1.4 Outline of the Thesis . . . 20

**I Scheduling Problems** **22**

**2 A Domestic Energy Management Problem** 23

2.1 Problem Description and Formulation . . . 25

2.1.1 MIP Model . . . 26

2.2 A Matheuristic Approach . . . 29

2.3 Computational Results . . . 31

2.4 Conclusions . . . 35

**3 The Cutting Stock Problem with Sequence-Dependent Cut Losses** 36
3.1 Problem Formulation . . . 37
3.1.1 Related Work and Similar Problems . . . 41
3.1.2 An Approximation by 1D-CSP . . . 42
3.2 Algorithms . . . 43
3.2.1 An Exact Enumerative Pattern Based Approach . . . 43
3.2.2 Heuristic Approach . . . 45
3.3 Computational Study . . . 48
3.3.1 Experimental Setup . . . 48
3.3.2 Experimental Results and Discussion . . . 49
3.4 Conclusions . . . 53

**4 An Exact Exponential Branch and Merge Approach for the Total Tardiness Problem** 54
4.1 A Branch and Reduce Approach . . . 55
4.2 A Branch and Merge Algorithm . . . 59
4.2.1 Merging Left-Side Branches . . . 61
4.2.2 Merging Right-Side Branches . . . 69
4.2.3 Complete Algorithm and Analysis . . . 79
4.3 Conclusions . . . 83

**II Clustering Problems** **85**

**5 The Max-Mean Dispersion Problem** 86
5.1 Mathematical formulations . . . 88
5.2 A Semidefinite Programming Approach . . . 92
5.2.1 The Semidefinite Programming Relaxation . . . 92
5.2.2 The Branch and Bound Framework . . . 95
5.2.3 Reducted SDP model . . . 96
5.3 A Three-Phase Hybrid Heuristic . . . 100
5.3.1 PHASE ONE: using the QIP solver to compute initial solutions 101
5.3.2 PHASE TWO: local branching . . . 104

5.3.3 PHASE THREE: path relinking . . . 105
5.4 Computational Experiments . . . 106
5.4.1 Tests for the Branch and Bound Algorithm . . . 107
5.4.2 Tests for the Hybrid Heuristic . . . 111
5.5 Conclusions . . . 116

**6 The Multi-Meter Covering Problem** 118
6.1 Problem Description . . . 119
6.2 Mathematical Formulation . . . 121
6.3 Literature . . . 122
6.4 The Proposed Algorithm . . . 123
6.5 Computational Experiments . . . 126
6.6 Conclusions . . . 133

**7 Conclusions and Future Developments** 134

**Bibliography** 136

# List of Tables


2.1 Results on the first dataset . . . 32
2.2 Results on the extended dataset . . . 34
3.1 Length and demand for each item . . . 37
3.2 Cut losses matrix . . . 38
3.3 Results on instances with $IS = 80$. . . . 51
3.4 Results on instances with $IS = 125$. . . . 51
3.5 Results on instances with $IS = 170$. . . . 52
3.6 Summary of results on the second set of generated instances. . . . 52
4.1 The time complexity of TTBM for values of $k$ from 3 to 20 . . . 83
5.1 Solutions computed in PHASE ONE . . . 104
5.2 Values of $T_{max}$ used in the tests . . . 108
5.3 Results on [76]'s small instances . . . 109
5.4 Results on [76]'s medium instances . . . 110
5.5 Results grouped by instance size . . . 111
5.6 Results of phase 2 on the instances of [76] with $n = 500$ for different values of $\delta$ with $k = 5$ and time distribution 1 . . . 112
5.7 Results on the instances of [76] of Type II with $n = 500$ for different distributions . . . 112
5.8 Results on the instances of [76] of Type II with $n = 500$ for different distributions . . . 113
5.9 Results on the large instances for different CPU time limits . . . 114
5.10 Comparing GRASP, XPRESS SLP and the hybrid algorithm on the large instances . . . 115
6.1 Telecom vs Polito solutions for Asti instance . . . 129
6.2 Telecom vs Polito solutions for Torino instance . . . 129

6.3 Detailed Polito results for Asti instance . . . . . 130
6.4 Detailed Polito results for Torino instance . . . . . 131

# List of Figures


1.1 The Local Branching scheme . . . . . 9
1.2 LP and SDP in the general taxonomy for convex programs . . . . . 14
2.1 Washing Machine Power Profile . . . . . 26
2.2 Dish Machine Power Profile . . . . . 26
2.3 Dynamic Tariff Profile . . . . . 27
2.4 Three Tier Tariff Profile . . . . . 27
2.5 All the active variables in the complete MIP model . . . . . 30
2.6 All the active variables in the first phase . . . . . 30
2.7 All the active variables in the second phase . . . . . 30
3.1 An example of a problem instance . . . . . 38
3.2 First solution, 4 cutting patterns used . . . . . 39
3.3 Second solution, 3 cutting patterns used . . . . . 39
3.4 Third solution, 3 cutting patterns used and larger leftovers . . . . . 40
4.1 The branching scheme of TTBR1 at the root node . . . . . 60
4.2 Left-side branches merging at the root node . . . . . 62
4.3 Merging for a generic left-side branch . . . . . 66
4.4 An example of right-side branches merging for $k = 3$ . . . . . 69
4.5 Generic right-side merging at the root node . . . . . 70
4.6 The right branches of $P_n$ have been modified when performing right-merging from $P$ . . . . . 73
5.1 An example showing three steps of PHASE ONE . . . . . 102
6.1 Flowchart of the procedure `COMPUTE_UB` . . . . . 125
6.2 Asti area with all the concentrators . . . . . 127
6.3 Asti area with the concentrators included in the optimal solution . . . 128
6.4 Comparing Telecom - Polito solutions on Asti instance . . . . . 132

6.5 Comparing Telecom - Polito solutions on Torino instance . . . . . 132

# Chapter 1

# Introduction

In the modern society, the importance of taking smart decisions is universally acknowledged in any different context. Companies, nations, individuals aim at being efficient and productive in order to achieve their goals. As a consequence, they need to find good solutions for problems of different nature, often consisting in looking for the best decisions among a finite number of alternatives. In operational research, the discipline that studies how to choose the best object in a finite set is called combinatorial optimization. An impressive number of combinatorial optimization problems are of practical interest in many different fields like logistics, data mining, finance, networking and telecommunication, computational biology and so on.

One of the biggest challenges of the IT era is to solve these problems automatically by means of electronic computing devices and proper optimization algorithms. Unluckily, most of the interesting combinatorial optimization problems are NP-hard [47]. For this class of problems, it is commonly recognized that the existence of an exact solution algorithm with polynomial complexity is very unlikely. More specifically, unless P=NP, there are no algorithms with the following requirements:

1. solve any istance of an NP-hard problem;
2. to optimality;
3. in polynomial time.

All the algorithmic studies on NP-hard problems are performed by discarding at least one of the requirements above. Such studies date back to the 60s and involve

both practical and theoretical aspects. On the one hand, decision makers aim at defining efficient practical approaches that manage large real world instances in a reasonable time. On the other hand, theoretical computer scientists have shown that different NP-hard problems have a different intrinsic complexity from a theoretical point of view.

From a theoretical point of view, several algorithmic studies are mainly devoted to achieve a deep understanding of the inner complexity of NP-hard problems. In general, these algorithms may or may not be efficient and interesting from a practical point of view. Three research lines can be identified in this context: *exact exponential algorithms*, *parameterized algorithms* and *approximation algorithms*. Exact exponential algorithms [46] correspond to the study of exact approaches for NP-hard problems where the focus is on their worst case complexity. Obviously, requirement (3) is discarded in this context. Traditionally, the problems belonging to the NP-hard class can be solved to optimality with different worst case complexity. Pushing the barrier of the best worst case complexity achievable for an NP-hard problem gives a better understanding of its intrinsic complexity, whether the resulting algorithm may be inefficient in practice. In the field of parameterized complexity, fixed-parameter algorithms [26, 38] solve to optimality NP-hard problems in polynomial time with respect to the input size of the instance, but in exponential time with respect to a certain input parameter. The polynomial complexity is achieved as soon as the specific parameters are fixed, then we loose the generality of the considered instances that can be handled and requirement (1) is excluded. Approximation algorithms [95, 93] replace requirement (2) with a quality guarantee on the solution. In simple words, the solution provided by the algorithm may be not optimal, but it can not be worse than the optimal solution more than a specific threshold. There are several interconnections among these three research lines. As an example, a brand new research field arises from matching approximation theory with exact computation with respect to the worst-case complexity. This is the field of *moderately exponential approximation* [78]: it allows to achieve approximation factors that are not achievable by means of polynomial approximation algorithms. Here, both requirement (2) and requirement (3) are discarded.

The two most important families of practical approaches are *exact methods* and *heuristics/ metaheuristics*. Practical exact methods for combinatorial optimization

address the challenge of solving to optimality large instances in a limited amount of time. In this case, requirements (1) and (2) hold, while requirement (3) does not. As a consequence, the worst case complexity of these approaches grows exponentially with the input size as for exact exponential algorithms focused on the worst case complexity. The difference between these two classes is the focus of the resulting approach: practical efficiency or worst case complexity. Heuristics and metaheuristics satisfy the need of finding high quality solution very quickly. They are definitely faster than exact methods because the algorithm structure does not preserve the optimality, nor any quality guarantee, of the solutions provided. Usually, well-performing exact methods and heuristics are approaches tailored on a specific problem, whose properties and structure are exploited in order to improve the algorithm performances. The common aim is to manage real world instances: depending on the case, the accent is put on the optimality requirement or on the need of managing very large instances.

Different techniques have been used across the years to design efficient exact methods and metaheuristics for NP-hard problems. Metaheuristics define a high-level problem-independent strategy that guides heuristic algorithms in finding good solutions. Tons of different metaheuristics with different structures have been designed so far. A general taxonomy for metaheuristics distinguish constructive metaheuristics (GRASP, etc), local-search metaheuristics (simulated annealing, tabu search, multistart algorithms, etc) and population based metaheuristics (genetic algorithms, scatter search, path relinking, etc). For an introduction to the main types of metaheuristics, the reader is referred to [49].

In the state of the art, the great majority of exact methods (both practical and theoretical) are search tree approaches. In fact, practical exact methods are usually branch and bound algorithms, where the bounding technique is used to identify branchings that can be pruned because they will not lead to any improving solution. The efficiency of such approaches is generally a direct consequence of the tradeoff between the tightness of the bounds and their computational cost. Roughly speaking, we often end up in computing a weak bound in a short time (several nodes will be opened in the branch and bound tree), or in computing a tight and time consuming bound (a few nodes will be opened, but a lot of time will be required by the bounding steps). In general, the bounding procedure can be of

different nature and consists in solving a relaxation of the original problem. The simplest and more common choice is to use an LP relaxation of the mathematical formulation, which can be generally solved very quickly, for example via the simplex method. In this context, it is possible to compute the relaxation in a node of the tree by reusing the optimal continuous solution found at the father node (*warm start*). Although such solution may be infeasible for the current node, the reoptimization can be performed very quickly by using the dual simplex algorithm. This scheme requires the use of a linear model for the problem considered. In other cases, convex optimization algorithms can be exploited to obtain high quality bounds for non-linear formulations. There exists a class of interior point methods which are able to efficiently solve such relaxations with an arbitrary small error from the optimal solution. The resulting bounds are often heavier to compute than LP bounds, but their quality leads to efficient branch and bound algorithms in several cases. In the worst case complexity analysis of an exact approach, bounding techniques are shown so far to be fruitless. As a consequence, the branch and bound framework is replaced by another paradigm (*branch and reduce*, described in Section 1.3.1) whenever the focus is on the worst case complexity of the exact approach.

A general and universally adopted approach to determine exact methods for combinatorial optimization is to use the state of the art solvers. These solvers take as an input a mathematical formulation and an instance of the problem, and provide an optimal solution as an output. There are specific mathematical programming solvers for different types of formulations (linear, quadratic, fractional, etc). As an example, the current Integer Linear Programming (ILP) solvers include commercial solvers like Cplex [7] or Xpress [5] and free solvers like Symphony [3] and CBC [2]. Thanks to mathematical solvers, today many operational researchers can focus on the structure of the problem formulations to obtain exact solution approaches for the problem. Since today solvers are really developed and well-optimized, the implementation of an exact ad-hoc algorithm from scratch is often useless.

In the last years, mathematical solvers have been used also in a heuristic context. This has led to the definition of a new class of metaheuristics: the so called *matheuristics* [74]. Such algorithms arise from the hybridization of exact methods and metaheuristics and try to exploit the advantages of both approaches. Even though matheuristics avoid the complete enumeration of the solution space (as for

metaheuristics), the efficiency of commercial solvers can be exploited as for exact methods. Section 1.1 is devoted to briefly describe the most important topics in the context of matheuristics.

As a generalization of this concept, we refer to hybrid methods as the algorithms (exact or heuristics) that mix ingredients coming from different mathematical programming techniques. In this thesis, hybrid algorithms will reuse pieces of software (mathematical solvers) in order to provide heuristic or exact solution for real world combinatorial problems. As an example, this thesis provides both applications of linear and non-linear models in the design of effective matheuristics and the use of a convex relaxation (more specifically, a *semidefinite programming* relaxation) to design an efficient branch and bound algorithm. Finally, an exact exponential algorithm is proposed. It is based on a new technique that hybridize the branch and reduce paradigm (Section 1.3.1) with concepts coming from another well-established exact approach (dynamic programming across the subsets, Section 1.3.2).

As evidenced by the title, this thesis tackles combinatorial problems belonging to two important families of combinatorial problems: scheduling and clustering problems. Scheduling problems concern with the allocation of a limited amount of resources to some activities over time. Due to the huge number of applications in industry, several studies have been conducted for scheduling problems since the first developments of the OR discipline, and an impressive number of books have been written to summarize the most important practical and theoretical results (such as [98]). Clustering problems consist in grouping a set of objects in several subgroups (cluster) such that the most similar/different objects are in the same cluster. In particular, this thesis deals with clustering problems where the aim is to find a single cluster, the one that maximizes a specific objective. As an example, many applications of clustering problems can be found in data mining, image processing and machine learning.

In the next sections, the main techniques used in the thesis are introduced and illustrated with respect to the state of the art. Section 1.1 is devoted to briefly introduce the most important topics in the context of matheuristics. Section 1.2 describes the general use of semidefinite programming in the context of combinatorial optimization. Section 1.3 provides a brief introduction to the most important techniques for the design of exact exponential algorithms. Finally, Section 1.4 gives

a general overview on the content of this thesis and summarizes the main scientific contributions.

## 1.1 Matheuristics

The intuitive idea behind the concept of matheuristics is, as described previously, the use of mathematical solvers in a heuristic context. Since mathematical solvers work with a model of the problem, matheuristics are also known as *model-based* heuristics. Although the general idea is simple, any try to derive a formal definition for matheuristics leads to statements that are pretty vague or not completely correct. In fact, there is not a universally accepted formal definition in the literature. As a matter of fact, even the main important point needs to be rediscussed in some circumstances: mathematical solvers may be replaced by some other ad hoc exact algorithm for the specific problem, used in a hybrid fashion. As a consequence, the aim of this section is to delineate the features that occur most frequently in the past works.

Generally speaking, the most important families of solvers that are exploited so far are LP and ILP solvers. Indeed, there exists very efficient solvers for this type of problems. The basic algorithms used for their solution are primal/dual simplex and branch and bound. However, a lot of algorithmic improvements have led these software to be more practically efficient by far than a standard implementation. Recently, the focus has been extended to other families of problems such as quadratic problems, whose continuous relaxation may not be convex. As an example, last versions of CPLEX implement a very efficient Quadratic Integer Programming solver, which is also able to manage non-convexity by means of some convexification techniques. It is very likely that other classes of combinatorial problems are going to be managed by new mathematical programming solvers in the upcoming decades. As a consequence, more and more research could likely be conducted in the area of matheuristics, in order to exploit such new solvers.

Similarly as metaheuristics, there no exists a unique framework that works well for any type of problem. Contrarily, there are several distinct strategies that have been successfully adopted in several cases. A first classification is analogous to the one that is traditionally performed for heuristics. In fact, we can distinguish

local-search based matheuristics and constructive matheuristics. The firsts take as an input a starting solution $\mathbf{x}_{in}$ generated by means of another algorithm, and produce as an output a possibly improved solution $\mathbf{x}_{out}$. The improvement steps are performed thanks to a mathematical model which describes the neighborhood of $\mathbf{x}_{in}$, thus the exploration of the neighborhood is performed by the mathematical solver. Constructive matheuristics create the output solution $\mathbf{x}_{out}$ from scratch and exploit mathematical programming techniques to generate it.

Here follows an introduction to four different matheuristic strategies studied in the state of the art and successfully applied in this thesis. As an example of local search matheuristics, local branching and variable fixing are described, while two ideas on how to obtain good feasible solutions from scratch are discussed subsequently. These ideas consist of creating a feasible integer solution from a continuous relaxation and by means of a column generation formulation. For more details on matheuristics, we refer to [74, 32].

### 1.1.1 Local Branching

Given a solution $\hat{\mathbf{x}}$ of a combinatorial problem and an integer $k$, a possible neighborhood $\mathcal{N}_k(\hat{\mathbf{x}})$ consists of all the solutions such that the Hamming distance from $\hat{\mathbf{x}}$ is less than or equal to $k$. This corresponds to the well-known k-opt neighborhood traditionally defined for the Travelling Salesman Problem (TSP). The exploration of such neighborhood can be performed by enumeration of the solutions, but this may lead to poor performances when $k$ increases. Whenever the problem can be formulated as an ILP, the local search can be performed by running an ILP solver on the model of the problem, with the addition of the following constraint:

$$\Delta(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{j:\hat{x}_j=0} x_j + \sum_{j:\hat{x}_j=1} (1 - x_j) \leq k$$

where $\hat{x}_j$ are the components of $\hat{\mathbf{x}}$. Today ILP solvers can explore $\mathcal{N}_k(\hat{\mathbf{x}})$ in a very aggressive way, such that good heuristic solutions are available at a very early stage. However, such computation needs to be integrated with a high level strategy that avoids to fall in local optima and properly explores the promising parts of

the solution space. The resulting method is Local Branching [43], a general purpose matheuristic which integrates small neighborhood explorations and strategic branchings.

As highlighted by its name, the branching rule covers a fundamental role in Local Branching. At the very first step, the algorithm classifies solutions into two categories: the ones with an Hamming distance from $\hat{\mathbf{x}}$ less than or equal to $k$, and the remaining ones. This induces the following branching scheme:

$$\Delta(\mathbf{x}, \hat{\mathbf{x}}) \leq k \tag{1.1}$$

$$\Delta(\mathbf{x}, \hat{\mathbf{x}}) \geq k \tag{1.2}$$

The value of the parameter $k$ is chosen such that the size of $\mathcal{N}_k(\hat{\mathbf{x}})$ is small enough to be completely explored in a reasonable time and large enough to contain new improving solutions. Then, the larger subproblem where $\Delta(\mathbf{x}, \hat{\mathbf{x}}) \geq k$ needs to be solved. This is done by applying the same branching scheme where the solution $\hat{\mathbf{x}}$ is replaced by the improving solution $\hat{\mathbf{x}}'$ found in $\mathcal{N}_k(\hat{\mathbf{x}})$. Note that if $\mathcal{N}_k(\hat{\mathbf{x}})$ does not contain any improving solution, the search is extended to a larger neighborhood, in order to find at least one improving solution. This scheme is applied recursively as depicted in Figure 1.1.

The method is exact in nature, since the branching scheme can be recursively applied until all the solution space is explored. However, the heuristic efficiency can be improved by setting ad hoc defined time limits for the solution of the problem induced by Equation 1.1. Another important remark is that the whole procedure is not problem dependent and can be easily applied to a wide class of problems. Even the problem formulation may be linear or not, but the efficiency of the resulting approach will be strongly affected by the efficiency of the mathematical solver used. Local branching has been successfully applied in several problems in the last few years, as an example in [82, 29, 102].

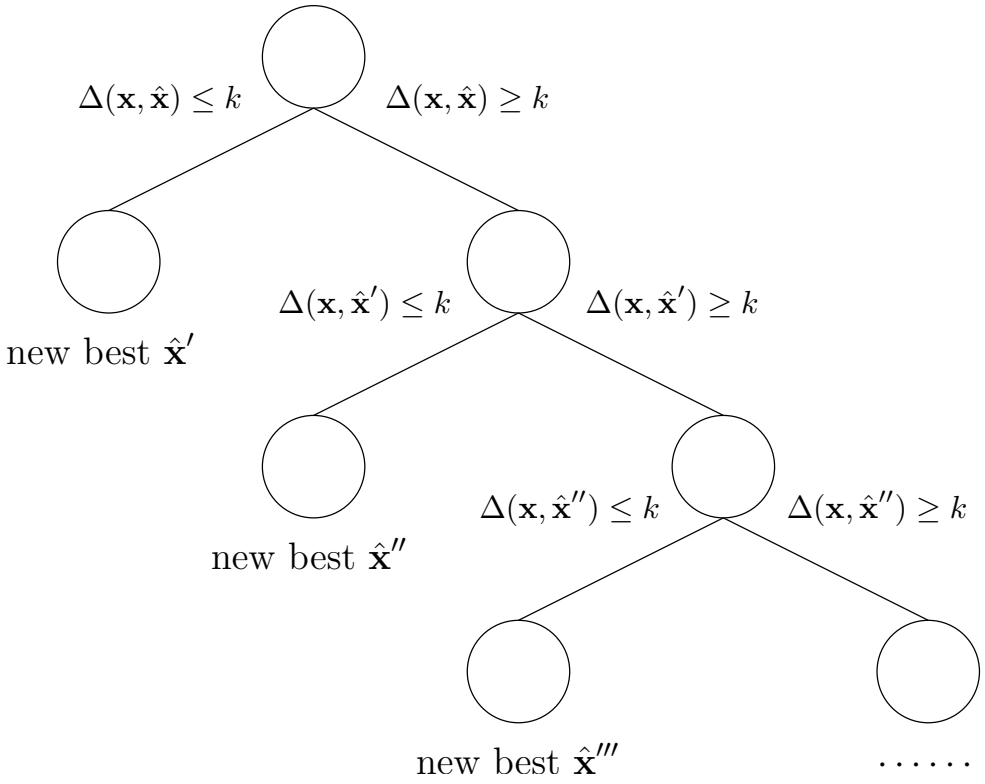



Figure 1.1: The Local Branching scheme

### 1.1.2 Variable Partitioning Local Search

Given a certain mathematical model of a combinatorial problem, a simple measure of the hardness of an instance is the number of free variables. MIP solvers generally try to exploit some properties such that variables can be fixed at some values without loss of optimality as soon as possible in the branch and bound tree. This is the aim of techniques such as variable probing and reduced cost fixing (see page 31 of [12] for a short review on these topics). The same idea can be exploited in a heuristic fashion, as an example by fixing variables at a specific value a priori, before the solution of the model. This can be done according to several heuristic criteria that can be based on the specific properties of the problem considered. When a sufficiently high number of variables are fixed, the corresponding reduced problem is much easier to be handled by an integer programming solver.

The size of such problem corresponds to the number of remaining variables. Let us consider the subset $S$ of the indexes set $\{1, \ldots, n\}$, that refers to the indexes

associated to the unfixed variables. Given a starting solution $\hat{\mathbf{x}}$, whose components are the scalars $\hat{x}_j$ $j = 1 \dots n$, we define the neighborhood $N_S(\hat{\mathbf{x}})$ of the solution $\hat{\mathbf{x}}$, such that all the solutions in $N_S(\hat{\mathbf{x}})$ are feasible and have the j-th component $(\forall j \notin S)$ equal to $\hat{x}_j$. As a consequence, the following formulation is valid for $N(\hat{\mathbf{x}})$:

$$N(\hat{\mathbf{x}}) = \{\mathbf{x} \in F | x_j = \hat{x}_j \forall j \notin S\}$$

where $F$ is the feasible solution space of the problem. The neighborhood $N_S(\hat{\mathbf{x}})$ can be very large and the problem of exploring such part of the solution space is sometimes referred to as *core problem*. In such case, it is usually explored by using a MIP solver approximately, by setting a limit on the time required for its solution. Since different subsets $S$ are related to different neighborhoods $N_S(\hat{\mathbf{x}})$, further improvement can be established by using the MIP solver on several of these sets. Similarly as for Local Branching, this general technique has been exploited to derive matheuristics for several combinatorial problems [30, 34, 28].

### 1.1.3 Continuous Relaxation Based Matheuristics

A traditional approach for dealing with a combinatorial problem is to solve the problem formulation obtained by discarding the integrality constraint. In this case, we obtain a continuous version of the problem that is easier to be solved. In fact, instead of using a branch and bound approach, efficient algorithms (such as interior point methods for the convex case and simplex methods for the linear case) can be used to find an optimal solution for the continuous problem. Given the continuous solution, the easiest way to obtain a feasible integer solution is to round each fractional component to the nearest integer. This is very easy to implement, but often the solution obtained is not feasible for the integer problem. In fact, the rounded solution may not belong to the feasible set, since some constraints may not be verified anymore after the rounding operation. This issue is solved by means of a problem-independent method called Feasibility Pump [42].

### 1.1.4 Column Generation Based Heuristics

Column Generation (CG) [37, 72] is a general technique to solve large LPs with an exponential number of variables in the input size. In such case, most of the variables assume a value of zero in the optimal solution, then even if the number of variables is very large, only a few of them should be considered in the solution of the LP. The idea behind this technique is very similar to the one adopted in the simplex method, where variables enter in the basis when they can improve the objective according to their reduced costs. Analogously, in CG a column is generated only when it is needed, i.e. when it can contribute to achieve a better solution.

Let us consider the following LP, called in this context *master problem* (MP):

$$\min \sum_{i \in J} c_j \lambda_j$$

subject to:

$$\sum_{i \in J} \mathbf{a}_j \lambda_j \geq \mathbf{b}$$

$$\lambda_j \geq 0 \ \forall j \in J$$

In MP, the number of variables $\lambda_j \in \mathbb{R}$ is equal to $|J|$ and is very large, while the number of constraints is $m$. Furthermore, we have $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{a}_j \in \mathbb{R}^m$, $c_j \in \mathbb{R}$ for all $j \in J$. In each iteration of the simplex method, we look for a non-basic variable with negative reduced cost to enter the basis. When $|J|$ is large, this operation may be too costly. For this reason, in column generation we work with the so-called *restricted master problem* (RMP), that is the previous problem where the number of columns considered is restricted and $J$ is replaced by $\tilde{J} \subseteq J$.

At each iteration, we want to add a column with the most negative reduced cost in $\tilde{J}$, that means performing the *pricing* step. As in the simplex method, this is repeated until no columns in $J$ can improve the objective. Let $\boldsymbol{\lambda}, \boldsymbol{\pi}$ be the current primal-dual optimal solution of the current RMP. The pricing operation can be performed by solving the following *pricing problem* (PP):

$$\min_{j \in J} c_j - \boldsymbol{\pi}^t \mathbf{a}_j$$

When the optimal solution of PP is positive, there are not columns with negative

reduced costs and the current $\boldsymbol{\lambda}$ is optimal for the original master problem. In this decomposition scheme, it may not be clear why solving PP it is so helpful instead of solving the pricing by enumeration. The reason is that PP often results in a well structured and studied optimization problem. Besides, there are many algorithmic choices that arise in this context and lead to different computational performances. As an example, the pricing step can be performed such that several columns are selected via some heuristic criteria. Same considerations can be done for the initialization of the set $\tilde{J}$, that can be done by means of different problem-specific choices. In general, many studies have been performed to improve the convergence of the method, that include considerations about the dual point of view [17].

Nowadays, CG is often used as a bounding procedure in branch and bound to solve large MIPs. The resulting method is called *branch and price* (B&P)[60], since at each node the pricing problem is solved to compute the optimal solution of the relaxation. Branch and price is one of the most important advances in mathematical programming of the last years and has many important applications in a wide class of combinatorial problems, among which vehicle routing problems and cutting/packing problems.

Although CG and B&P are originally used to solve LPs and ILPs to optimality, several studies have been performed to use such techniques in a heuristic fashion (see [61] for a review). Certainly, the most natural way to use CG to derive a heuristic for an integer problem is to solve its continuous relaxation and then round the corresponding solution as described in Section 1.1.3. This is often known as a *rounding* CG heuristic. Another choice is to explore the branch and price tree by means of a heuristic depth-first rule. In this context, the re-optimization of the master problem computed at each node does not even need to be solved to optimality. This type of heuristic is called *diving* CG heuristic. Finally, the rescricted master problem, with the restricted set $\tilde{J}$ generated according to a specific heuristic rule, can be solved as a static integer program. Even if RMP is solved to optimality, the solution provided is a heuristic solution for the master problem, since only a subset of all the possible columns is considered. This class of heuristics is known as *restricted master* CG heuristics. Note that in this case the set $\tilde{J}$ needs to be created such that at least a feasible solution for RMS exists.

As a conclusion, CG techniques perfectly fit in the context of hybrid methods

for combinatorial optimization. In fact, in large LPs and MIPs we need to call multiple times a mathematical solver in order to implement the specific CG technique. The last decades showed that these techniques are effective for many classes of combinatorial problems.

## 1.2 Semidefinite Programming for Combinatorial Optimization

A fundamental point for the efficiency of a branch and bound algorithm is the compromise between the tightness of the bound and the time required to compute it. Both convex and linear relaxations have been successfully used to this aim. Admittedly, there is a deep connection between these two worlds. In fact, the tightest convex relaxation for a combinatorial problem is linear and describes its convex hull. Unfortunately, such relaxation is not practical to be computed in several cases, due to the high number of linear constraints to be considered. The alternative is to find a convex relaxation that approximates well the convex hull and for which there exist efficient solution algorithms.

In the last decades, the field of non-linear relaxations have received much attention in this direction. In particular, Semidefinite Programming (SDP) [97] has been considered as a powerful tool to derive tight relaxations for combinatorial problems that are reasonably fast to be computed. SDP is the natural extension of LP: it consists in minimizing a linear function over the intersection of the cone of positive semidefinite matrices with an affine space. An SDP can be expressed as follows:

$$\min \mathrm{Tr}(CX)$$

subject to

$$\mathrm{Tr}(A_i X) = b_i \ \forall i = 1, ..., m$$

$$X \succeq 0$$

where $X$, $C$ and $A_i \ \forall i = 1, ..., m$ are n-dimensional symmetric square matrices, $b_i \ \forall i = 1, ..., m$ are real coefficients and $\mathrm{Tr}(\cdot)$ is the trace function. Contrarily to LP, the feasible region of an SDP is not polyhedral, but it has a shape that is a

spectrahedron. Although the SDP constraint is non-linear, it turns out to be convex: as a consequence, SDPs belong to the more general family of convex optimization problems (see Figure 1.2). Conversely, SDP generalizes some classes of optimization problems, such as LPs, quadratic convex programs (QCPs) and second order cone programs (SOCPs).

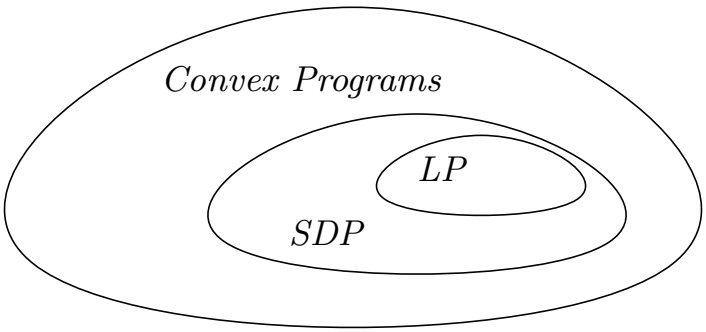



Figure 1.2: LP and SDP in the general taxonomy for convex programs

Several engineering problems can be formulated as SDPs, they arise in fields like control theory, signal processing, eigenvalue optimization and, finally, combinatorial optimization. As a matter of fact, a consistent part of the great popularity of SDP is due to its application in combinatorial optimization. The key point is that, in spite of the higher generality of SDPs with respect to LPs, SDPs are not much harder to solve, whether their generality allows to design more powerful relaxations. In other words, SDP relaxations can be used to better approximate the convex hull of the feasible region of a combinatorial problems and, in many cases, it does not require an excessive computational effort. This is due to the fact that interior point methods for linear programming can be directly extended to the semidefinite programming case [10], and their complexity is still polynomial. Besides, other efficient approaches have been adopted (as an example, the Specral Bundle method [56]) and several SDP solvers have been implemented (CSDP, SDPA, ConicBundle, etc).

The applications of SDP in combinatorial optimization are based on deriving tight SDP relaxations for combinatorial problems. Deriving such relaxations is often not intuitive for the people out from the SDP community, then, as an introductory example, we describe the steps that lead to the definition of a basic semidefinite relaxation for a very well known combinatorial problem: the Quadratic Knapsack

Problem (QKP) [58, 79]. First, we recall the definition of the problem, that is, basically, an extension of the knapsack problem where a quadratic objective is considered. Let us consider a knapsack of size $K$, and a set $N = \{1, ..., n\}$ of items whose weights are $w_j \ \forall j = 1, ..., n$, and a matrix $P$ whose components $p_{i,j}$ indicate that the profit $p_{i,j} + p_{j,i}$ is obtained when the items $i$ and $j$ are selected. The QKP calls for selecting the subset of items such that they fit in the knapsack, meaning that the sum of their weights should be less than $K$, and the overall profit is maximized. The following quadratic programming formulation follows:

$$\max \sum_{i \in N} \sum_{j \in N} p_{i,j} x_i x_j$$

subject to:

$$\sum_{i \in N} w_i x_i \leq K$$

$$x_i \in \{0,1\} \ \forall i \in N$$

Let $\mathbf{x}$ be the vector of the decision variables $x_i$. The objective can be reformulated by using the trace operator and considering the profit matrix $P$ and the matrix $X = \mathbf{x}\mathbf{x}^t$, whose components model the products $x_i x_j \ \forall i, j \in N$. In fact, it is possible to verify that $\mathrm{Tr}(PX) = \sum_{i \in I} \sum_{j \in N} p_{i,j} x_i x_j$. Since $x_i x_i = x_i^2 = x_i$ for binary variables, we also have that $\mathrm{Tr}(IX) = \sum_{i \in N} w_i x_i$, where $I$ is the identity n-dimensional square matrix. As a conclusion, the previous formulation for the QKP can be rewritten as follows:

$$\max \mathrm{Tr}(PX)$$

subject to:

$$\mathrm{Tr}(IX) \leq K$$

$$X = \mathbf{x}\mathbf{x}^t$$

Now, the interesting point is that all the model is convex (even linear) but the

constraint $X = \mathbf{x}\mathbf{x}^t$. In order to provide an SDP relaxation, such constraint is relaxed by the weaker SDP constraint $X \succeq \mathbf{x}\mathbf{x}^t$, that is still non-linear, but convex. Thus, the following SDP relaxation holds for the QKP:

$$\max \operatorname{Tr}(PX)$$

subject to:

$$\operatorname{Tr}(IX) \leq K$$

$$X \succeq \mathbf{x}\mathbf{x}^t$$

that, by applying Schur's complement theorem, can be written as follows:

$$\max \operatorname{Tr}(PX)$$

subject to:

$$\operatorname{Tr}(IX) \leq K$$

$$\begin{bmatrix} X & \mathbf{x} \\ \mathbf{x}^t & 1 \end{bmatrix} \succeq 0$$

As for linear relaxations, SDP relaxations as the one above can be tightened by considering sets of valid inequalities. In the context of SDP, they can be linear inequalities and non-linear convex inequalities that can be expressed as Linear Matrix Inequalities (LMI). Several valid inequalities have been considered for the QKP polytope (see [58] for a review). A branch and bound based on such relaxation lead to the definition of an efficient branch and bound approach for the QKP, and thanks to the power of SDP relaxations, several other branch and bound approaches have been designed for many well-studied combinatorial optimization problems, such as the Quadratic Assignment Problem [104]. These studies involve specifically many problems whose natural formulation is quadratic in the objective. Recently, all such

advancements made the definition of a general purpose quadratic integer programming (QIP) solver based on SDP possible. We are talking about BiqCrunch [1]. It integrates the possibility of using a general purpose solver with some problem specific techniques for a wide class of quadratic problems (Max-Cut, k-cluster, Maximum Independent Set, Quadratic Stable Set, Exact Quadratic Knapsack).

Another appreciated application of SDP has mainly theoretical interest, that is the design of approximation algorithm. Basically, such algorithms are based on rounding the result given by an SDP relaxation of the problem, then similarly as in LP based approximation algorithms, a specific approximation ratio can be determined after some geometrical considerations. Several theoretical results have been based on this scheme, pioneered by the seminal work of Goemans and Williamson on the Max-Cut problem [53].

## 1.3 The Design of Exact Exponential Algorithms

Since the beginning of this century, the design of exact exponential algorithms for NP-hard problems has been attracting more and more researchers. The research in this area consists in designing exact exponential algorithms for NP-hard problems, such that they have the lowest possible worst case complexity. Some of these problems have appeared to be solvable with a lower exponential complexity than others belonging to the same complexity class: these differences are not explained by the traditional complexity theory. Although the research in this area dates back to early 60s, the discovery of new design and analysis techniques has led to many new developments. In this section, we give a brief overview of the most important techniques used in this field.

The study of moderate exponential algorithms is also motivated by the fact that they may perform well for problem instances of small size. As a matter of fact, an algorithm running in $\mathcal{O}(1.01^n)$ performs better on instances of small size than a polynomial algorithm running in $\mathcal{O}(n^4)$.

When an exact exponential algorithm is designed, the key point is its worst case complexity and, more specifically, the exponential contribute. Given a polynomial $poly(n)$, an algorithm whose complexity is $\mathcal{O}(poly(n)2^n)$ has an asymptotic behaviour that is between $\mathcal{O}(2^n)$ and $\mathcal{O}((2+\epsilon)^n)$ for every $\epsilon \geq 0$. For this reason, a

specific complexity notation, the $\mathcal{O}^*(\cdot)$ notation traditionally used in the context of exact exponential algorithms, is used in this thesis. Given two functions $f(n)$ and $g(n)$, we write that $f(n) = \mathcal{O}^*(g(n))$ if $f(n) = \mathcal{O}(g(n)poly(n))$, where $poly(n)$ is, again, a polynomial. This means that all the polynomial factors are ignored in this notation.

In the following, Section 1.3.1 and Section 1.3.2 describe two very important techniques used to design exact exponential algorithms: branch and reduce, and dynamic programming across the subsets. Finally, Section 1.3.3 is dedicated to list some of the other techniques in the state of the art. For a deeper survey on the most effective techniques in designing exact exponential algorithms, readers are kindly referred to Woeginger's survey [96].

### 1.3.1 The Branch and Reduce Paradigm

The *branch and reduce* paradigm is one of the most common and oldest techniques in the design of exact exponential algorithms, pioneered in 1960 by Davis and Putnam [27]. The structure of branch and reduce algorithms is intrinsically recursive. Given a problem instance, these two steps are applied as follows:

1. *reduction rules*, which simplify the instance by reducing the number of free variables;

2. *branching rules*, which divide the current problem instance in a number of distinct subproblems.

Since this scheme is applied recursively, a branch and reduce algorithm is then univocally defined by the list of reduction rules and branching rules to be considered.

The space consumption of a branching algorithm is generally polynomial: it is achieved by exploring the search tree in depth-first order. The time complexity is usually studied by estimating the number of nodes in the search tree, usually it has the form $\mathcal{O}^*(c^n)$ with $c$ a possibly irrational constant. When a single branching rule is considered, the linear recursion induced by the branching rule can be solved to derive an upper bound to the worst case complexity of the algorithm. When multiple branching rules are considered, a simple upper bound can be determined by considering the branching rule that induces the highest worst case complexity.

In general, how to determine tight bounds on the time complexity of branch and reduce algorithms is an open problem. A very common approach is *measure and conquer*, see Chapter 6 of [46] for more information.

### 1.3.2 Dynamic Programming across the Subsets

Dynamic programming is a classical technique to solve combinatorial optimization problems to optimality (see typical books on algorithms such as [24]). The basic idea is to solve and store solutions of small subproblems and then progressively combine them to obtain solutions for larger and larger subproblems. Dynamic programming has been used to derive polynomial, pseudo-polynomial and exponential exact algorithms. Here, we discuss exponential algorithms based on dynamic programming, that have the main drawback of being exponential both in time and space, contrarily (generally) to branching algorithms.

In this section, the focus is on specific type of dynamic programming: *dynamic programming across the subsets*. It is specifically designed for permutation problems, for which a trivial enumeration algorithm runs in $\mathcal{O}(n!)$. It allows to decrease the complexity to $\mathcal{O}^*(c^n)$ with $c$ a constant parameter (usually $c = 2$). The general approach is invented by Bellman [16], who designed this approach to solve the Travelling Salesman Problem (TSP) in $\mathcal{O}^*(2^n)$.

In order to present the technique, we describe the original approach for the TSP. Given a set of cities $C = \{1, ..., n\}$, the distance between the i-th city and the j-th city is $d(i, j)$. We target to find the shortest possible route that visits each city in $C$ only once. For every subset $S$ of $\{2, ..., n\}$ and every city $i \in C$, we denote by $Opt[S, i]$ the optimal route that starts in 1, crosses all the cities in $S \setminus i$ in arbitrary order and ends in $i$. Obviously, $Opt[\{i\}, i] = d(1, i)$. Therefore, the following recurrence holds:

$$Opt[S, i] = \min\{Opt[S \setminus \{i\}, j] : j \in S \setminus \{i\}\} \tag{1.3}$$

The algorithm computes the solution of subsets increasingly. For each subset, a cost of $\mathcal{O}(n^2)$ is required because we need to consider all the possible values of $i$ and $j$ in Equation 1.3. The number of all the possible subsets of $C$ is $\mathcal{O}(2^n)$, then the time and space complexity of the approach is $\mathcal{O}(n^2 2^n) = \mathcal{O}^*(2^n)$.

Several linear ordering problems can be solved by means of dynamic programming across the subsets, such as the TSP, the Treewidth Problem and the Cutwidth Problem. A complete list of these problem can be found in [18]. These problems can be solved also by extending the *divide and conquer* approach presented in [55].

### 1.3.3 Other Techniques

The most important technique for the design of exact exponential algorithms are discussed in [46]. As an example, a design technique is based on the *inclusion-exclusion* principle commonly used in combinatorics. As dynamic programming across the subsets, inclusion-exclusion based algorithms go through all possible subsets, but it does not require exponential space. Another paradigm is called *sort and search*, basically we consider instances of exponential size and then apply fast polynomial time algorithm on such instances.

## 1.4 Outline of the Thesis

The present thesis is organized in two parts: the first deals with scheduling problems, while the second focuses on clustering problems. Each chapter corresponds to the study of a combinatorial problem performed during my PhD. Finally, Section 7 reports some general conclusions and future developments for this work.

Let us start to describe Part I, about scheduling problems. Chapter 2 refers to a scheduling problem where the aim is to schedule home appliances in order to minimize the energy cost. An ILP formulation and a hybrid heuristic based on Variable Partitioning (Section 1.1.2) is presented. Cutting stock problems are special production scheduling problems where a set of items are cut from pieces of material. Chapter 3 considers a new cutting stock problem where the cuts determine a material loss which depends on the items order. A mathematical programming formulation for the problem is provided and a restricted master CG heuristic, according to the scheme discussed in Section 1.1.4, is presented to solve the problem efficiently. Chapter 4 describes an exact exponential algorithm for a classical scheduling problem: the Single Machine Total Tardiness Problem. Such algorithm runs in $\mathcal{O}^*(2^n)$ time and polynomial space, which improves the current result available in the state

of the art, and its design involves a new technique called *branch and merge*. The technique avoid solutions of equivalent subproblems in a branching tree: in a sense it includes the phylosophy of dynamic programming in a branching algorithm, then keeping the space requirement polynomial. Although the presentation of the resulting algorithm regards its theoretical relevance, Chapter 4 includes remarks about the extension of the technique to improve the practical performance of branching algorithms for scheduling problems.

Now, we describe the content of Part II about clustering. Chapter 5 is devoted to provide effective algorithms (a heuristic and an exact approach) for the Max-Mean Dispersion Problem, a clustering problem where the aim is to find the subset of a set whose average distance between selected elements is maximized. Two approaches are presented: the first is a heuristic algorithm based on a non-linear formulation of the problem, while the second is a branch and bound algorithm where the bounding procedure consists in solving a semidefinite relaxation of the problem. In such chapter, the techniques disccussed in Section 1.1.1 and Section 1.2 are used. Chapter 6 copes with the so-called Multi-Meter Covering Problem. Such problem arises in a smart city context where, given a number of concentrators located in the area, we aim at covering the overall area by minimizing the overall number of concentrators used. All the research works described in Chapter 2 and Chapter 6 have been carried out in a collaboration with Telecom Italia.

As a conclusion, the main contributions of this thesis are solution algorithms with practical relevance for scheduling and clustering combinatorial problems. Such contributions are certified by some journal/conference pubblications that are listed here, whose merits are shared with my co-authors. The work described in Chapter 2 was presented at Airo 2014. The research presented in Chapter 3 was preliminarily presented at Mista 2013 and subsequently published in International Transactions in Operational Research [48]. The research about the Total Tardiness Problem (Chapter 4) was presented at Mista 2015 and Airo 2015, and will be submitted to a theoretical computer science journal in March 2016. Chapter 5 includes two dinstict works that led to different pubblications. The heuristic described in such chapter was presented at Isco 2014 and Airo 2014, while the final work is published in Computers and Operations Research [29]. The exact approach was submitted at Journal of Combinatorial Optimization and currently in first revision.

# Part I

# Scheduling Problems

# Chapter 2

# A Domestic Energy Management Problem

A very interesting and trendy topic in the IT era is to deliver useful services to people inside buildings, in such a way that the economic cost and the environmental impact are lowered. The buildings that implement these functions are referred to as *smart buildings*. In smart buildings, a fundamental role is played by the energy management system, which aims at scheduling appliances at the most convenient time instant. This chapter deals with a real world combinatorial optimization problem arising in smart energy systems. It was studied jointly with Telecom Italia, in the context of the INTRePID project [8].

In general, electricity suppliers provide tariffs for which the costs strongly vary depending on the time of the day. At the same time, there are limits on the overall peak consumption, such that a given threeshold can not be surpassed during the day. As a consequence, it may not be possible to schedule all the appliances in the part of the day associated with the lowest costs. Then, the aim of the smart grid system is to schedule household appliances in order to lower the overall energy cost and avoid that the maximum peak consumption is exceeded. In this scenario, another important factor that may occur is the use of Renewable Energy Sources (RES), such as solar energy. This means that photovoltaic devices, usually located in the top of the building, generate an amount of energy directly from the sunlight. Since we consider the case where no batteries are used in the building, such energy

should be used on the fly. Nowadays, this case is still realistic due to the high costs of storing energy.

A few papers appeared recently on smart appliances scheduling, such as [14], [20], [85] and [103] where a house-size context is considered. Since many side conditions may be taken into account, all these works may significantly differ. In [14], these side conditions are batteries constraints and the possibility of selling unused energy. The use of batteries is also included in [44]. Another strongly different aspect, described also in [85], is that the total consumption of an appliance could be split over a variable number of timeslots. In other words, the amount of energy consumed by an appliance in each timeslot is a problem variable, and must be decided by the smart grid system.

Solution approaches are also different. In [14], a Mixed Integer Programming (MIP) model is proposed to deal with the problem as in [85] and [103] where optimization models, though with different variables definition, are used. In [20], a Particle Swarm Optimization (PSO) approach is proposed to obtain heuristic solutions for the problem. Finally, a matheuristic algorithm is presented in [44]. Matheuristic approaches have been successfully applied to other well-known scheduling problems (see [33] and [35]), a simple but effective one is presented in this chapter. This matheuristic is basically constructed by following the Variable Partitioning scheme described in Section 1.1.2.

In this chapter of the thesis, we study a specific smart grid energy problem with no appliance preemption and no possibility to store/sell energy. A more precise description of the problem together with a MIP formulation are provided in Section 2.1. Section 2.2 highlights the heuristic approach proposed, namely a matheuristic approach exploiting the MIP formulation described previously. In Section 2.3, a computational campaign, conducted by comparing the matheuristic approach to other approaches, is presented. As competitors, two algorithms are considered: the first is based on the solution of the MIP model via MIP solvers (CPLEX, XPRESS and SYMPHONY), the second is an enhanced version of the approach proposed in [20], henceforth denoted as RaPSOL, which was provided by Telecom Italia. Finally, some conclusions about this work are presented in Section 2.4.

## 2.1 Problem Description and Formulation

The problem can be formalized as follows. A set $A$ of $n$ appliances has to be scheduled during a defined time horizon. Each appliance $a \in A$ has to be switched on during the time horizon and, when active, it has to be on for exactly $d_a$ timeslots which must be contiguous, preemption is not allowed.

For each of these timeslots $t \in H_a = \{1, ..., d_a\}$, we have two kind of energy consumptions: an average energy consumption $e_{a,t}$ and a peak energy consumption $p_{a,t}$. The whole time horizon is 24 hours (represented by the set $H = \{1, ..., T\}$). The maximum energy consumption is $P$ for each timeslot of the time horizon. This refers to the peak energy consumption, given that the average energy consumption is smaller than the peak one ($e_{a,t} \leq p_{a,t}$). The cost for each energy unit is equal to $c_t \in \mathbb{R}_+$ which is calculated on the average energy consumption. The use of this couple of values for defining the consumption of an appliance is motivated by the realistic behaviour of the appliances. In fact, considering a minute as the timeslot size, the consumption within this time may vary a lot and peaks can be present for just milliseconds. Then, we consider peak values to guarantee that the maximum consumption is not exceeded, while we evaluate the average cost over single timeslots to compute the overall cost. The overall cost also depends on the energy cost per timeslot, which is given by the tariff considered and possibly varies every hour.

Moreover, an amount $S_t \geq 0$ of solar energy can be used for each timeslot $t \in H$. This additional energy influences both the maximum energy consumption $P$ and the tariff cost $c_t$. In fact, the use of solar energy may induce an increase to $P$ equal to $S_t$ and no costs, again, for the consumption up to $S_t$. In other words, a maximum quantity of $S_t$ $\forall t \in H$ can be used free of charge.

The aim of the problem is to schedule all requested appliances within the time horizon, at the minimum total cost, respecting the maximum power constraint. As an example, the power profiles of a washing machine and a dish machine are plotted in Figure 2.1 and Figure 2.2, where the grayscale distinguishes the peak power consumption and the average consumption. Besides, the values of two different tariffs are represented for the whole time horizon in Figure 2.3 and Figure 2.4. Here, the grayscale indicates how much expensive is the cost associated with a specific timeslot.

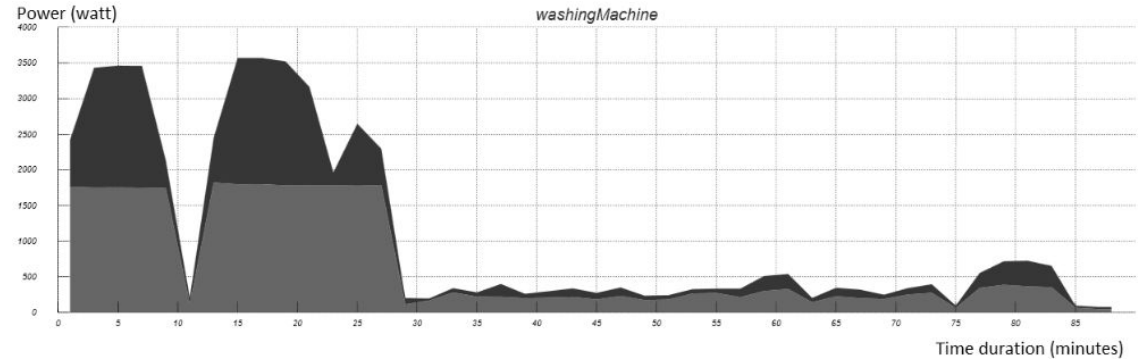



Figure 2.1: Washing Machine Power Profile

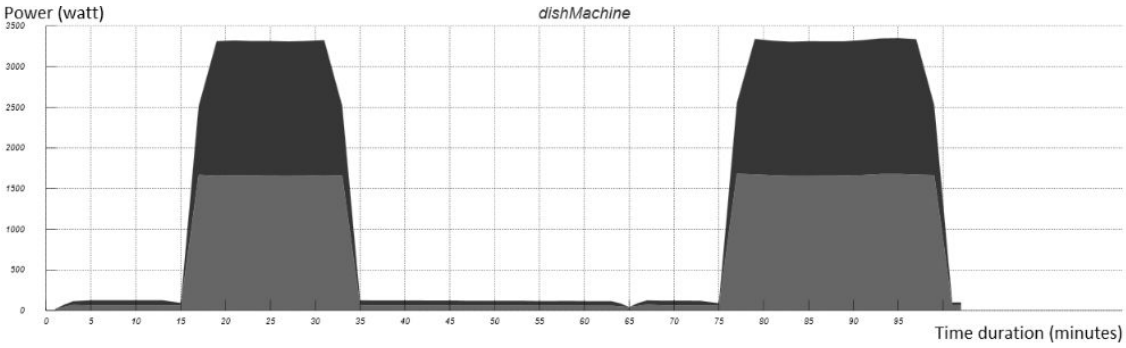



Figure 2.2: Dish Machine Power Profile

### 2.1.1 MIP Model

A solution of the problem is given by the starting time of each appliance, since durations are inputs for the problem. Given that, the overall cost of each appliance starting in every possible (feasible) timeslot can be precomputed. In such a way, a cumulative coefficient $C_{a,t}$, which represents the total cost of appliance $a$ starting at time $t$, is computed for all $a \in A, t \in H$. Each coefficient $C_{a,t}$ is a function of the different tariff coefficients $c_t$ and the mean power consumption $e_{a,t}$. In fact, the following equality holds:

$$C_{a,t} = \sum_{t' \in H_a} e_{a,t'} c_{t+t'}$$

The overall cost, due to the energy consumption, is the overall energy consumption of each appliance minus the solar energy consumed:

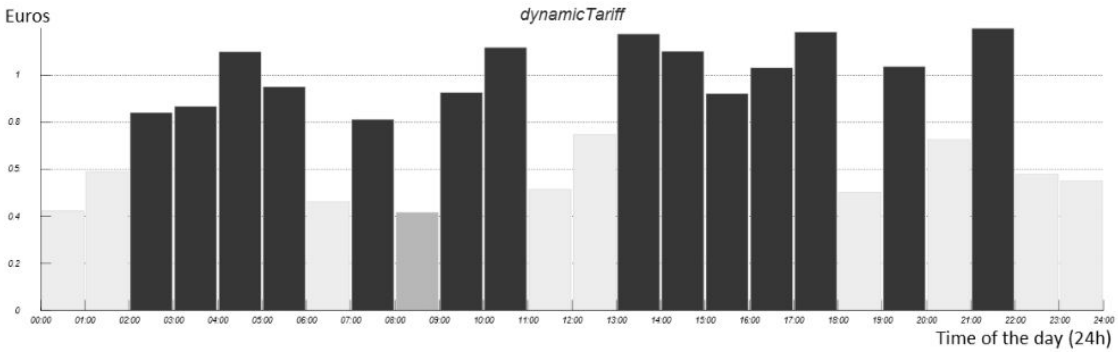



Figure 2.3: Dynamic Tariff Profile

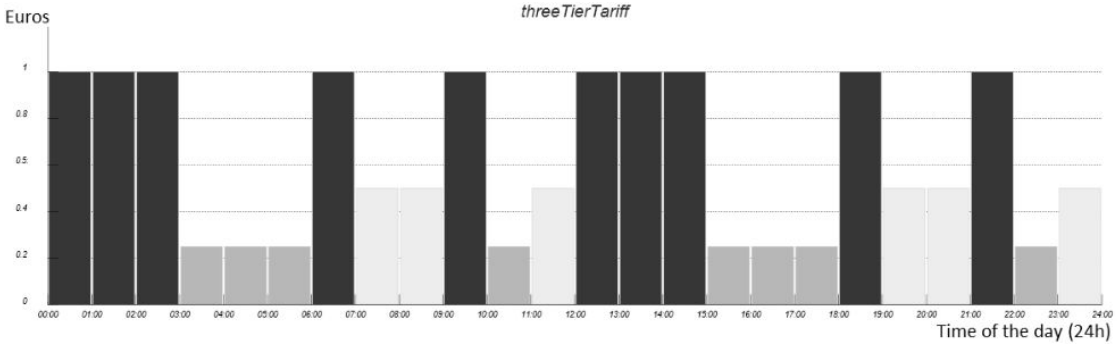



Figure 2.4: Three Tier Tariff Profile

$$\sum_{a \in A} \sum_{t \in H} C_{a,t} - \sum_{t \in H} c_t z_t$$

where $z_t$ is a set of continuous variables representing the minimum among the solar energy available $S_t$ at time $t$ and the sum of consumptions of all appliances again at time $t$. Consequently, the following inequalities hold for each $z_t$:

$$z_t \leq S_t \quad \forall t \in H$$

$$z_t \leq \sum_{a \in A} \sum_{t' \in H_a} e_{a,t'} x_{a,t-t'} \quad \forall t \in H$$

Given the previous considerations, for each $a \in A$ and $t \in H$, we define a variable

$x_{a,t}$ (as in [14]) such that:

$$x_{a,t} = \begin{cases} 1 & \text{if the appliance } a \text{ starts at time } t \\ 0 & \text{otherwise} \end{cases}$$

The complete MIP model follows.

$$\min f(x_{a,t}, z_t) = \sum_{a \in A} \sum_{t \in H} C_{a,t} x_{a,t} - \sum_{t \in H} c_t z_t \tag{2.1}$$

subject to:

$$\sum_{t \in H | t <= T - d_a} x_{a,t} = 1 \quad \forall a \in A \tag{2.2}$$

$$\sum_{a \in A} \sum_{t' \in H_a} p_{a,t'} x_{a,t-t'} - P - S_t \leq 0 \quad \forall t \in H \tag{2.3}$$

$$z_t \leq S_t \quad \forall t \in H \tag{2.4}$$

$$z_t \leq \sum_{a \in A} \sum_{t' \in H_a} e_{a,t'} x_{a,t-t'} \quad \forall t \in H \tag{2.5}$$

$$x_{a,t} \in \{0,1\} \quad \forall a \in A \quad \forall t \in H \tag{2.6}$$

$$z_t \geq 0 \quad \forall t \in H \tag{2.7}$$

The objective function 2.1 minimizes the overall cost of scheduling all appliances. Constraints 2.2 assign exactly one starting time to each of the appliances. Constraints 2.3 define the maximum power consumption for each timeslot, where the overall energy consumption must not exceed $P$ (plus the solar power). Constraints 2.4 and 2.5 set the values of $z_t$ as previously depicted. Finally, in Constraints 2.6 and 2.7, the variables domain is defined. The overall number of variables of the

model is $\mathcal{O}(nT)$ variables (proportional to the number of appliances and the size of the time horizon), while the overall number of constraints is $\mathcal{O}(T)$ (proportional to the size of the time horizon).

## 2.2 A Matheuristic Approach

In the following, a matheuristic approach, which can be used to tackle the problem, is described. Such approach is a two phase algorithm in which the first phase is dedicated to generate a feasible solution to the problem, while the second phase is devoted to improve the first solution in a local search framework. Both phases rely on the mathematical formulation of the problem and are solved by means of a MIP solver.

The first phase is based on the reduction of the number of variables (i.e. the cardinality of the set $H$) such that the MIP solver can carry out a smaller problem of controlled size. The reduction is made by sampling the possible starting times for each appliance within the whole time horizon. This implies that an appliance can be scheduled only on a subset of active time instant $H' \subset H$. The subset of active time instants is obtained by sampling elements from $H$ with a fixed size step $\alpha \in \mathbb{N}$.

$$H' = \{t \in H, k \in \mathbb{N} : t = k\alpha + 1\}$$

This is realized by introducing an additional constraint into the model 2.1–2.7 and by imposing that appliances can only start in $t \in H'$.

$$\sum_{t \notin H'} x_{a,t} = 0 \quad \forall a \in A \tag{2.8}$$

The second phase aims at improving the solution provided by the previous phase (given by the schedule time $t_a \in H$ of each appliance). A neighbourhood of $t_a$ is generated by activating a fixed size ($\beta \in \mathbb{N}$) “window" of variables, whose time instant is contiguous to $t_a$:

$$H'_a = \{t \in H : |t - t_a| \leq \beta\}$$

This is realized by modifying Constraints 2.8 such that only variables in $H'_a$ are

active.

$$\sum_{t \notin H'_a} x_{a,t} = 0 \quad \forall a \in A \tag{2.9}$$

Both phases are executed until a local optimum is reached or a time limit ($T_1$ and $T_2$ respectively) is reached. Algorithm 1 and Algorithm 2 depict the structure of the procedures associated with the first and the second phase of the matheuristic. Figures 2.5, 2.6 and 2.7 depict a simple example where two appliances must be scheduled within a time horizon of 12 timeslots. The timeslots filled by a dark color are the ones whose variables are active, while the other timeslots are not active. In this toy example, parameters $\alpha$ and $\beta$ of our heuristic are set to 4 and 1, respectively.


Figure 2.5: All the active variables in the complete MIP model


Figure 2.6: All the active variables in the first phase

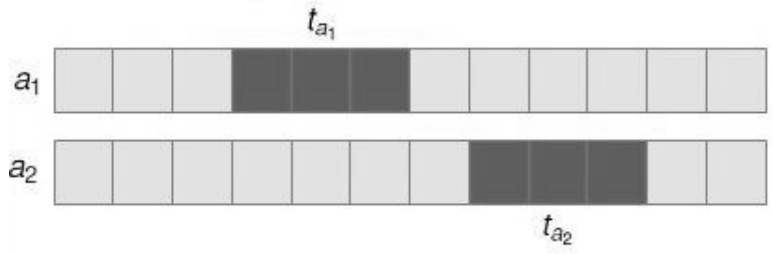



Figure 2.7: All the active variables in the second phase

Algorithm 1 and Algorithm 2 summarize the structure of the two phases of the algorithm.

**Algorithm 1** First phase

**Input:** Problem instance, $\alpha$
**for all** $a \in A$ **do**
  **for all** $t \notin H'$ **do**
    Set $x_{a,t}$ equal to 0
  **end for**
**end for**
$s_1 \leftarrow$ Solve model 2.1–2.7 with time limit $T_1$
**Output:** Solution $s_1 = \{(a, t_a)\}_{a \in A}$

**Algorithm 2** Second phase

**Input:** Problem instance, $s_1$, $\beta$
**for all** $a \in A$ **do**
  **for all** $t \notin H'_a$ **do**
    Set $x_{a,t}$ equal to 0
  **end for**
**end for**
$s_2 \leftarrow$ Solve model 2.1–2.7 with time limit $T_2$
**Output:** Solution $s_2$

## 2.3 Computational Results

A first dataset of small instances was used in order to asses the quality of the MIP model and to benchmark the results with the hybrid heuristic. Three different MIP solvers were considered on this first dataset. The tests were conducted using both commercial solvers (CPLEX 12.5 and FICO XPRESS 7.2) and an open source one (SYMPHONY 5.2 from COIN-OR). This because we wanted to evaluate the applicability of open source software to this real world problem. All the approaches were tested on instances with 3, 5, 10 and 15 appliances. The durations of the appliances were generated according to a uniform distribution between 80 and 200 minutes (timeslots). Analogously, the peak power consumption for each timeslot was generated according to a uniform distribution between 50 and 1600 Watts, while

the mean power consumption decreased the peak power of a percentage uniformly extracted between 5% and 10%. The dynamic tariff for the energy cost and the cloudy profile for the solar power were the only ones considered. Furthermore, different maximum power values were tested, namely 2KW, 3KW and 4KW (italian single home-sized values).

The three approaches considered (the MIP solvers, the hybrid heuristic and the RaPSOL approach provided by Telecom Italia) were tested with the same time limits, namely 10 and 60 seconds. The same time limit $TiLim = T_1 = T_2$ was set in both phases of the matheuristic. The results of the tests on the initial dataset are reported in Table 2.1. The values depicted with $*$ mean that the optimal value has been proved within the specific time limit. Empty entries occur whenever no result has been achieved within the time limit. Finally, column *bound 1h* tracks the value of the lower bound, given by CPLEX 12.5 after one hour of CPU time (again, entries with $*$ mean that the optimal solution was reached within 1 hour processing of CPLEX 12.5). The RaPSOL procedure is a randomized metaheuristic. Its performance is evaluated by performing 5 runs on each instance, then the average and the best objective function value are reported in the tables.

| | | | | XPRESS | CPLEX | SYMPHONY | Hybrid | | RaPSOL | |
|---|---|---|---|---|---|---|---|---|---|---|
| # Appl | Max Power | TiLim | Bound 1h | Obj | Obj | Obj | Obj | CPU (s) | Obj best | Obj mean |
| 3 | 4000 | 10 s | 2,5381* | 2,5384 | 2,5381* | 3,2278 | 2,5381 | 1,264 | 2,5381 | 2,5381 |
| | | 60 s | | 2,5381* | 2,5381* | 2,5381 | 2,5381 | 1,28 | 2,5381 | 2,5381 |
| 3 | 3000 | 10 s | 2,5381* | 2,5384 | 2,5381* | 3,2278 | 2,5381 | 1,419 | 2,5381 | 2,5381 |
| | | 60 s | | 2,5381* | 2,5381* | 2,5381 | 2,5381 | 1,435 | 2,5381 | 2,5381 |
| 3 | 2000 | 10 s | 2,5381* | 2,5381* | 2,5381* | 2,5425 | 2,5381 | 0,842 | 2,5381 | 2,5381 |
| | | 60 s | | 2,5381* | 2,5381* | 2,5381 | 2,5381 | 0,827 | 2,5381 | 2,5381 |
| 5 | 4000 | 10 s | 4,8642* | 4,8643 | 4,8643 | 4,8870 | 4,9091 | 2,871 | 4,8643 | 4,8656 |
| | | 60 s | | 4,8643 | 4,8643 | 4,8685 | 4,9091 | 2,87 | 4,8643 | 4,8643 |
| 5 | 3000 | 10 s | 4,8642* | 4,8643 | 4,8671 | 4,8925 | 4,8685 | 3,759 | 4,8643 | 4,9078 |
| | | 60 s | | 4,8643 | 4,8643 | 4,8925 | 4,8685 | 3,775 | 4,8643 | 4,8798 |
| 5 | 2000 | 10 s | 4,9544* | 4,9700 | 5,2445 | – | 4,95866 | 2,262 | 4,9551 | 4,9577 |
| | | 60 s | | 4,9700 | 4,9545 | – | 4,95866 | 2,278 | 4,9551 | 4,9557 |
| 10 | 4000 | 10 s | 8,6464 | 12,8896 | 9,7318 | – | 8,95512 | 6,255 | 9,1123 | 9,1497 |
| | | 60 s | | 9,1929 | 9,0141 | – | 8,95512 | 11,513 | 9,0149 | 9,07 |
| 10 | 3000 | 10 s | 8,7858 | 12,3398 | 12,1620 | – | 9,34792 | 7,581 | 9,6051 | 9,6804 |
| | | 60 s | | 10,0963 | 9,2056 | – | 9,25844 | 10,031 | 9,3841 | 9,5507 |
| 10 | 2000 | 10 s | 9,2177 | 11,4725 | 11,9863 | – | 10,52528 | 6,381 | 11,022 | 11,133 |
| | | 60 s | | 11,4561 | 11,6960 | – | 10,54349 | 31,684 | 10,809 | 10,95 |
| 15 | 4000 | 10 s | 13,2484 | – | 19,2274 | – | 14,27465 | 10,748 | 14,906 | 14,999 |
| | | 60 s | | 15,0148 | 15,0497 | – | 14,07361 | 42,526 | 14,681 | 14,823 |
| 15 | 3000 | 10 s | 13,5077 | – | 18,0702 | – | 15,72953 | 10,764 | 16,347 | 16,572 |
| | | 60 s | | 16,3381 | 16,8535 | – | 15,52666 | 60,778 | 16,172 | 16,398 |

Table 2.1: Results on the first dataset

As can be seen the hybrid heuristic appears to be slightly less competitive for the small instances (up to 5 appliances) with respect to RaPSOL and MIP solvers, while

for the larger ones (10-15 appliances) the hybrid heuristic performs better than the other approaches within the same time limit. We also generated the instance with 15 appliances and 2KW of maximum power, but no solution was found in 1 hour.

Given the results obtained on the first dataset, we decided to test the hybrid heuristic, the RaPSOL approach, CPLEX and XPRESS on a larger dataset, while we skept SYMPHONY solver because on larger instances it was not capable of providing any solution. The second dataset considers instances with 10, 20, 30 and 50 appliances. In this dataset, half of the appliances are washing machines and half are dish machines for any instance. Moreover, different solar profiles were taken into account, namely *cloudy*, *clearSky*, a larger solar power availability and *noSolar*, where power can be only drawn by the line. In addition, two different cost tariffs are used, the *dynamic* and the *three* which are depicted in Figure 2.3 and Figure 2.4, respectively. The maximum power consumption was modified accordingly in order to guarantee the existence of feasible solutions. The time limits were set according to the instance size. After preliminary testing, parameter $\alpha$ and $\beta$, as defined in Section 2.2 are set to 10 and 20 respectively.

In Table 2.2, the results show that the hybrid heuristic performed better than the RaPSOL procedure on all instances but two. CPLEX solver is strongly competitive on a specific class of instances, namely the ones with no solar power, but this dominance is limited to 30 appliances in size. XPRESS solver has a similar behaviour of CPLEX, i.e. it is competitive on no solar power instances, but, in this case, the dominance is limited to instances formed by 20 appliances at most. As the size of instances increases, the gap between the hybrid heuristic and the other approaches increases too, and the hybrid heuristic shows up to be the clear winner for large size instances.

| # | Appl. | Max Power | Tariff | Solar | Hybrid | RaPSOL Obj best | RaPSOL Obj mean | CPLEX | XPRESS | Time Lim. (s) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 3000 | dynamic | noSolar | 4,6980 | 4,6677 | 4,7011 | ***4,6505*** | ***4,6505*** | 60 |
| 2 | 10 | 3000 | dynamic | cloudy | ***1,8060*** | 1,8533 | 1,8885 | 1,8925 | 2,1118 | 60 |
| 3 | 10 | 3000 | dynamic | clearSky | ***0,3577*** | 0,4412 | 0,4575 | 0,5103 | 1,1752 | 60 |
| 4 | 10 | 3000 | threeTier | noSolar | 2,3255 | 2,3255 | 2,3658 | ***2,2165*** | ***2,2165*** | 60 |
| 5 | 10 | 3000 | threeTier | cloudy | ***0,8845*** | 0,9205 | 0,9286 | 0,9472 | 0,9298 | 60 |
| 6 | 10 | 3000 | threeTier | clearSky | 0,1956 | ***0,1840*** | 0,2129 | 0,2066 | 0,5231 | 60 |
| 7 | 10 | 4000 | dynamic | noSolar | 4,4791 | 4,6695 | 4,6948 | ***4,4545*** | ***4,4545*** | 60 |
| 8 | 10 | 4000 | dynamic | cloudy | ***1,8070*** | 1,8300 | 1,8810 | 1,8889 | 2,1926 | 60 |
| 9 | 10 | 4000 | dynamic | clearSky | ***0,3459*** | 0,4264 | 0,4468 | 0,6552 | 1,4057 | 60 |
| 10 | 10 | 4000 | threeTier | noSolar | ***2,1528*** | ***2,1528*** | 2,1528 | ***2,1528*** | ***2,1528*** | 60 |
| 11 | 10 | 4000 | threeTier | cloudy | ***0,8841*** | 0,9108 | 0,9231 | 0,9373 | 0,9193 | 60 |
| 12 | 10 | 4000 | threeTier | clearSky | ***0,1849*** | 0,1983 | 0,2110 | 0,2977 | 0,8405 | 60 |
| 13 | 20 | 3000 | dynamic | noSolar | 11,2363 | 11,6590 | 11,8730 | ***10,9634*** | 11,2726 | 120 |
| 14 | 20 | 3000 | dynamic | cloudy | ***5,7572*** | 6,2273 | 6,2863 | 6,2720 | 6,5549 | 120 |
| 15 | 20 | 3000 | dynamic | clearSky | ***3,8648*** | 4,3187 | 4,4007 | 4,1433 | 4,7256 | 120 |
| 16 | 20 | 3000 | threeTier | noSolar | 6,5332 | 7,3371 | 7,4904 | ***6,4152*** | 6,8619 | 120 |
| 17 | 20 | 3000 | threeTier | cloudy | ***2,9483*** | 3,1580 | 3,2937 | 3,1562 | 3,3282 | 120 |
| 18 | 20 | 3000 | threeTier | clearSky | ***1,7749*** | 2,0020 | 2,0834 | 2,0745 | 2,8823 | 120 |
| 19 | 20 | 4000 | dynamic | noSolar | 9,4391 | 9,5925 | 9,6905 | ***9,3281*** | ***9,3281*** | 120 |
| 20 | 20 | 4000 | dynamic | cloudy | ***5,5838*** | 5,9396 | 6,0112 | 5,8263 | 6,0444 | 120 |
| 21 | 20 | 4000 | dynamic | clearSky | ***3,6451*** | 3,9902 | 4,0895 | 4,2894 | 4,5495 | 120 |
| 22 | 20 | 4000 | threeTier | noSolar | 4,5887 | 5,0092 | 5,0668 | ***4,4295*** | 4,4304 | 120 |
| 23 | 20 | 4000 | threeTier | cloudy | ***2,9074*** | 2,9864 | 3,0194 | 3,0675 | 5,6774 | 120 |
| 24 | 20 | 4000 | threeTier | clearSky | ***1,7645*** | 1,8307 | 1,8937 | 1,8954 | 2,4962 | 120 |
| 25 | 30 | 4000 | dynamic | noSolar | 15,5035 | 16,3060 | 16,4600 | ***15,3603*** | 15,5729 | 180 |
| 26 | 30 | 4000 | dynamic | cloudy | ***10,2747*** | 10,6240 | 10,9590 | 10,7797 | 10,7979 | 180 |
| 27 | 30 | 4000 | dynamic | clearSky | ***8,3640*** | 8,8660 | 9,0229 | 8,6057 | 9,4602 | 180 |
| 28 | 30 | 4000 | threeTier | noSolar | ***8,5092*** | 9,4517 | 9,6183 | 8,5194 | 8,6925 | 180 |
| 29 | 30 | 4000 | threeTier | cloudy | ***5,2506*** | 6,0775 | 6,4360 | 5,6405 | 6,6524 | 180 |
| 30 | 30 | 4000 | threeTier | clearSky | ***4,0070*** | 4,6583 | 4,7961 | 11,2188 | 5,9905 | 180 |
| 31 | 30 | 5000 | dynamic | noSolar | 15,0684 | 15,6900 | 16,0450 | ***15,0199*** | 15,2949 | 180 |
| 32 | 30 | 5000 | dynamic | cloudy | ***10,1457*** | 10,6900 | 10,8420 | 10,5401 | 11,3998 | 180 |
| 33 | 30 | 5000 | dynamic | clearSky | ***8,3351*** | 8,8296 | 8,9401 | 8,8882 | 9,2533 | 180 |
| 34 | 30 | 5000 | threeTier | noSolar | 8,2077 | 8,8856 | 9,1870 | ***7,8718*** | 7,9708 | 180 |
| 35 | 30 | 5000 | threeTier | cloudy | ***5,1221*** | 5,4297 | 5,5386 | 6,2345 | 5,9731 | 180 |
| 36 | 30 | 5000 | threeTier | clearSky | ***3,8858*** | 4,2717 | 4,3767 | 4,4262 | 5,3123 | 180 |
| 37 | 50 | 4000 | dynamic | noSolar | ***30,8919*** | 33,4500 | 33,8060 | 31,8704 | 32,2769 | 300 |
| 38 | 50 | 4000 | dynamic | cloudy | ***22,9057*** | 24,8280 | 25,0610 | 27,6976 | 23,5963 | 300 |
| 39 | 50 | 4000 | dynamic | clearSky | ***20,3754*** | 22,1750 | 22,3770 | 20,9757 | 21,7216 | 300 |
| 40 | 50 | 4000 | threeTier | noSolar | ***19,5239*** | 24,1470 | 24,3960 | 22,5064 | 21,8103 | 300 |
| 41 | 50 | 4000 | threeTier | cloudy | ***13,9575*** | 17,5180 | 17,6360 | 20,1169 | 15,0825 | 300 |
| 42 | 50 | 4000 | threeTier | clearSky | ***11,6914*** | 14,9420 | 15,2290 | 18,6342 | 13,3780 | 300 |
| 43 | 50 | 5000 | dynamic | noSolar | ***30,2348*** | 31,8720 | 32,2410 | 32,5150 | 30,4633 | 300 |
| 44 | 50 | 5000 | dynamic | cloudy | ***22,1632*** | 23,2390 | 23,5990 | 23,2425 | 23,2187 | 300 |
| 45 | 50 | 5000 | dynamic | clearSky | ***19,8182*** | 20,9870 | 21,2990 | 20,3465 | 20,6428 | 300 |
| 46 | 50 | 5000 | threeTier | noSolar | ***18,2266*** | 21,8080 | 22,2260 | 24,6358 | 19,4974 | 300 |
| 47 | 50 | 5000 | threeTier | cloudy | ***12,4277*** | 14,0590 | 14,6510 | 14,0561 | 14,2489 | 300 |
| 48 | 50 | 5000 | threeTier | clearSky | ***10,8568*** | 12,6130 | 13,0880 | 12,3682 | 13,8075 | 300 |

Table 2.2: Results on the extended dataset

## 2.4 Conclusions

We considered the problem of scheduling smart appliances within a given time horizon, taking into account a threshold on the energy peak consumption and seeking for the minimization of the overall cost related to the average energy consumption of each appliance in each time instant. Two main approaches were investigated. The first approach explored a MIP formulation of the problem that is embedded into a hybrid matheuristic algorithm. The second approach (RAPSOL) consists of an enhanced particle swarm optimization algorithm.

The two approaches were tested on several real life data instances and compared with the direct solution of the MIP model by means of two state-of-the-art MIP solvers such as CPLEX 12.5 and FICO XPRESS 7.2. While the hybrid approach shows up to be globally the best performing algorithm (particularly on large size instances), it is interesting to note that RAPSOL reaches quite good results. This is particularly interesting, taking into account the fact that RAPSOL (contrarily to the integer linear programming based approaches described in this chapter) is substantially unaffected by a non linear modification of the cost function. As an example, this case occurs when the maximum power constraint is discarded and replaced by a strong non linear penality in the objective. Future research will be devoted to a specific study in this direction, i.e. in the design of an efficient mathematical programming technique that can manage an objective function that has a non linear component.

# Chapter 3

# The Cutting Stock Problem with Sequence-Dependent Cut Losses

Cutting stock problems arise in many different industries such as in textile, glass, steel, wood and paper. To reduce operating cost, companies strive to minimize waste of stock material when cutting stock down to customer orders. A great deal of research effort has thus focused on developing effective ways for improving operations.

The present chapter focusses on the one-dimensional cutting stock problem (1D-CSP). Many papers refer to material waste for a 1D-CSP as material that is not used in the cutting patterns (leftovers). However, this is not the only material loss that occurs in practical contexts. Another type of material loss is intrinsically due to the process of cutting the material (by use of a blade, laser, etc.). The overall loss due to cuts is usually negligible, thus it is normally not taken into account. Nevertheless, cut losses cannot always be ignored in practical cases and may even depend on the item order.

We generalize the 1D-CSP to include *sequence dependent cut losses* (SDCL) that may occur between any pair of adjacent items or at the start and end of a cutting pattern. This generalisation allows to further improve efficiency and reduce cut losses that may occur in some situations. The 1D-CSP with SDCL can be approximately solved by any 1D-CSP or 1D bin packing (1D-BPP) approach. We investigate the conditions that make it beneficial to consider the SDCL nature of the problem. To this end, we present a heuristic approach specifically considering

SDCL, and compare it with a 1D-BPP approach on a range of generated instances, with varying characteristics.

## 3.1 Problem Formulation

Consider the well known 1D-CSP: a set of items $I$, each $i \in I$ having length $l_i$ and demand $d_i$, is to be cut from an unbounded set of larger stock items, each of length $L$ (with $l_i < L \ \forall i \in I$). A feasible cutting pattern $p$ is a subset of items $i \in I$ with multiplicity (denoted $a_i$) for which the total length $\sum_{i \in p} a_i \cdot l_i$ is at most $L$. The objective is to find a minimum set of feasible cutting patterns that cover each item's demand.

The 1D-CSP with SDCL generalizes the problem to include sequence dependent cut losses between items: between each pair of adjacent items $i$ and $j$ in a cutting pattern, the presence of an additional *cut loss* $c_{ij}$ should be taken into account. A cutting pattern is then not only determined by the included items and their multiplicity, but also by the sequence of the items within the pattern. Consequently, feasible patterns are cutting patterns for which the sum of item lengths (with multiplicity) and the sum of the SDCL $c_{ij}$ between adjacent items is smaller than $L$. Finally, the problem also considers start and end cut losses, $c_{0i}$ and $c_{i0}$. These occur at the start (resp. end) of the pattern before the first (resp. after the last) item $i$ is cut and should also be considered in order to fit within $L$.

To illustrate the problem, consider the instance shown in Figure 3.1 with input data listed in Table 3.1 and Table 3.2.

| $\boldsymbol{i}$ | **1** | **2** | **3** | **4** |
|---|---|---|---|---|
| $\boldsymbol{l_i}$ | 4.5 | 3.1 | 4 | 6 |
| $\boldsymbol{d_i}$ | 2 | 2 | 2 | 1 |

Table 3.1: Length and demand for each item

The cut losses $c_{ij}$ are of the same order of magnitude as the smallest item and show a very high variability. Figures 3.2-3.4 show three feasible solutions for this instance: the cut losses are drawn with a dashed background, while the leftovers are coloured gray.

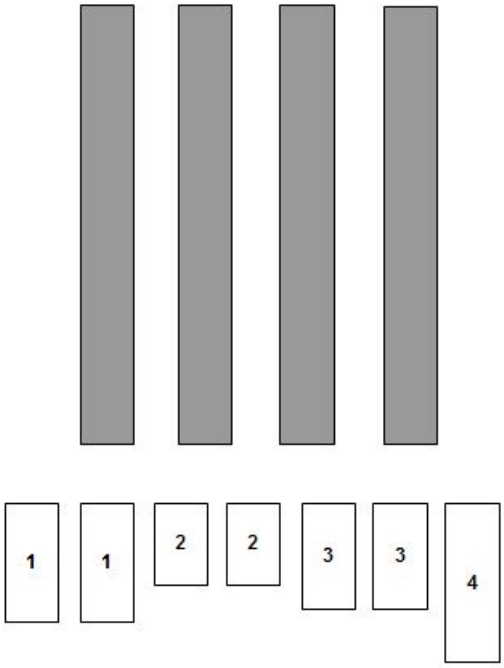



Figure 3.1: An example of a problem instance

$$\begin{bmatrix} 0 & 2.4 & 2.5 & 1.9 & 2.6 \\ 1 & 2.5 & 2.5 & 1.5 & 2.7 \\ 1.3 & 1 & 3 & 2.8 & 3 \\ 3.5 & 0.5 & 0.2 & 2.5 & 2.7 \\ 2.2 & 2.8 & 1.4 & 3 & 2.5 \end{bmatrix}$$

Table 3.2: Cut losses matrix

The example clearly illustrates that changing the order in which the items are cut determines a different amount of material loss due to the cut losses and leftovers. As a result, the solution in Figure 3.3 requires fewer stock material than the solution in Figure 3.2.

Several solutions require an equal amount of stock material. For example, Figure 3.4 shows another solution requiring three cutting patterns. The solution in Figure 3.4 is more suitable for more industrial purposes, as the leftover in the last pattern may be reusable. Larger leftovers are more likely to be reusable, allowing for further operating cost reductions. We therefore introduce a secondary objective, namely maximization of the sum of the squared leftovers produced by the solution, targeting

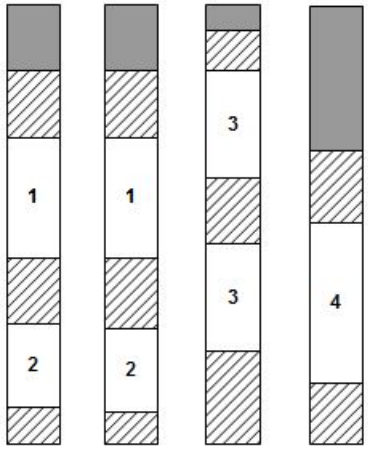



Figure 3.2: First solution, 4 cutting patterns used

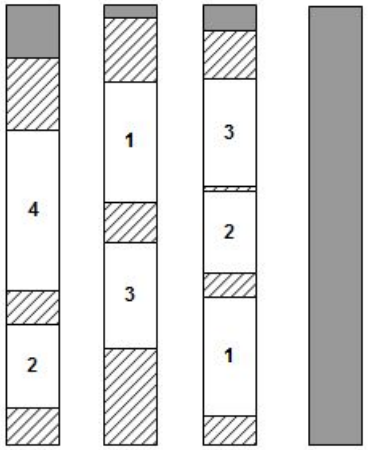



Figure 3.3: Second solution, 3 cutting patterns used

reusability of the leftover material. The square function is desirable because it allows distinction based on the leftover sizes as shown above. This objective was developed in analogy with [45], where maximization of the sum of square loads on a bin packing problem allows to find the optimal space utilization.

One possible (well known) integer linear programming formulation for the 1D-CSP with SDCL is based on the selection of a subset of patterns from the set of all possible feasible cutting patterns $P$, that covers each item's demand. A cutting pattern $p$ of $n_p$ items is defined as a sequence $(i_1, i_2, \ldots, i_{n_p})$ with $i_1, i_2, \ldots i_{n_p} \in I$. Feasible cutting patterns are patterns $(i_1, i_2, \ldots, i_{n_p})$ for which the following

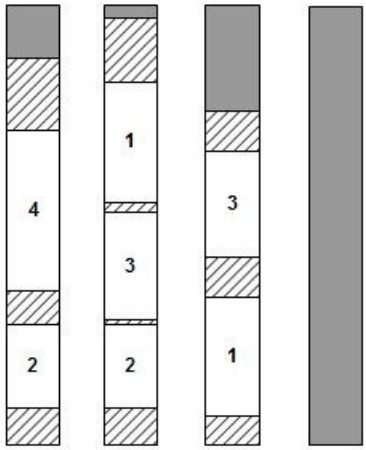



Figure 3.4: Third solution, 3 cutting patterns used and larger leftovers

inequality holds:

$$l_p = \sum_{j=1}^{n_p} l_i + \sum_{j=1}^{n_p} c_{i_j i_{j+1}} + c_{0i_1} + c_{i_{n_p}0} \leq L \tag{3.1}$$

with $l_p$ denoting the length of the pattern $p$, including SDCL.

The following model minimizes the number of patterns used, and then maximizes the sum of the squared leftovers. Let variables $x_p$ ($p \in P$) denote the number of occurrences of the pattern $p$ in the cutting plan. Let $r_p \in \mathbb{R}_+$ denote the leftover of pattern $p$ (i.e $r_p = L - l_p$), and $a_{ip} \in \mathbb{N}$ the multiplicity of the item $i$ in the pattern $p$. The integer model is then defined as:

$$\min \sum_{p \in P} (M - r_p^2) x_p \tag{3.2}$$

s.t. :

$$\sum_{p \in P} a_{ip} x_p = d_i \qquad \forall i \in I \tag{3.3}$$

$$x_p \in \mathbb{N} \qquad \forall p \in P \tag{3.4}$$

where $M \in \mathbb{R}_+$ is a suitably large constant.

### 3.1.1 Related Work and Similar Problems

According to Dyckhoff's typology [40], the 1D-CSP with SDCL can be categorised as 1/V/I/R. It is a one-dimensional problem with an unlimited supply of objects of identical size and a set of items to be cut. The problem can also be classified as a Single Stock Size Cutting Stock Problem (SSSCSP) according to the typology introduced in [94]. In both cases, however, the classification is not pure since the addition of sequence dependent waste should be included.

The literature on this type of problems is vast, starting as early as the 1960's with the seminal work of Gilmore et al. [50]. Many different models and approaches to the 1D-CSP have been studied since then, and many extensions to the problem definition include more practical considerations. Consideration of multiple stock lengths [59], sequencing and minimization of different cutting patterns to avoid machine setup costs [11, 99, 77], consideration of reusing leftovers [91, 25] have been of particular interest.

The inclusion of ordering significance within patterns has also been reported. Lewis et al. [69] studied the Truss Cutting Problem (TCP), a problem originating from the roofing industry that has strong connections to this work. Profiles of equal width having trapezoidal shapes, have to be cut from wooden boards with the aim of minimizing area waste. They show that the TCP is a special case of 2D-CSP and can actually be solved with a 1D packing approach. Sequence dependent cut losses (inter-item wastage) in this setting are specifically due to the shape of the items that may fit better together in certain arrangements. Given that the items have equal width, Lewis et al. solve the problem as a 1D grouping/packing problem using an algorithmic framework for bin packing problems. Feasibility of a grouping/packing for the TCP is determined by solving some form of a sequencing problem on a losses matrix. The authors understanding of SDCL is more abstract, and we do not restrict losses to the geometric considerations applicable to the TCP. In the same paper, the possibility of including two different orientations for each item is investigated. This, however, cannot be modelled by 1D-CSP with SDCL. Another fundamental difference with our work is that we include considerations about the size of the leftovers in the objective. The 1D-CSP with SDCL cannot be solved directly by applying the procedures in [69].

One dimensional CSP with SDCL presents interesting analogies with other combinatorial optimization problems. It can also be modelled as a distance constrained vehicle routing problem (DCVRP or DVRP). The three index formulation in [65] denoted VRP4 can be adapted to model 1D-CSP with SDCL, ignoring the capacity constraints on the vehicles. However, if the second objective function (sum of squared leftovers) is to be considered in the DCVRP model, it becomes nonlinear: since leftover sizes are variable depending on the patterns' sequence, the resulting objective function is quadratic. In this vehicle routing context, the stock material and the items of the 1D-CSP with SDCL are represented respectively by the vehicles and the cities (duplicated to cover the demand of each item) of the DVRP, while the dummy item 0 corresponds to the depot in vehicle routing. The cut losses matrix $C$ is replaced by an analogous matrix $C'$ indicating the distances between cities, increased by (half of) the corresponding item lengths (i.e. its components $c'_{ij} = \frac{l_i+l_j}{2} + c_{ij}$).

Moreover the problem can be modelled as a slight variation of the Multiple Travelling Salesman Problem ($mTSP$). Following the description in [15], the variations relevant for this work are related to the number of salesmen and to the so called emphspecial restrictions.

The presented problem also shares similarities with the parallel machine scheduling problem with sequence dependent setup times (see e.g. [71]). Sequence dependent setup times basically model the same idea as SDCL in a scheduling context. Main differences lie with the fact that for the parallel machine scheduling problem, the number of machines ($\sim$ corresponding to cutting patterns) is bounded, but no limit on the usage time for each machine is defined. Typically, the objective is to optimize some measure of throughput (total makespan, weighted tardiness, etc.), rather than minimizing the number of machines.

### 3.1.2 An Approximation by 1D-CSP

Under one specific assumption, the 1D-CSP with SDCL can also be solved as a standard 1D-CSP or 1D-BPP. Let $\overline{c_{ij}}$ denote the maximum cut loss in the cut loss matrix $c_{ij}$. If $l_i + \overline{c_{ij}} \leq L - \overline{c_{ij}}\ \forall i \in I$ holds, then a 1D-CSP with SDCL instance can be converted to a standard 1D-CSP instance by setting $l_i^* := l_i + \overline{c_{ij}}\ \forall i \in I$ and

$L^* := L - \overline{c_{ij}}$. Any cutting pattern or packing found by a 1D-CSP (or 1D-BPP) approach on this converted instance is feasible in any sequence. Consequently, the solution is also feasible for the 1D-CSP with SDCL. An important question addressed in Section 3.3 considers the conditions where an SDCL approach is better than a standard 1D-CSP approach.

## 3.2 Algorithms

We present a heuristic approach to the 1D-CSP with SDCL. Starting from an exact enumerative pattern based approach, presented in Section 3.2.1, a heuristic approach (denoted HSD) is developed that overcomes the shortcomings of the exact approach. This heuristic approach is described in Section 3.2.2.

### 3.2.1 An Exact Enumerative Pattern Based Approach

A general MILP solver was not able to efficiently solve the three index model [65] due to the subtour elimination constraints (SEC) and the item duplication required to cover demand. The following approach is based on the solution of the model 3.2 - 3.4 which is the classical set cover (SC) formulation where $r_p$ are pre-computed constants. Solving this model requires the computation of the set $P$ formed by all the efficient patterns. Efficient patterns are those for which the sequence of items within a pattern is optimal, i.e. that the total material loss incurred by the items and the cut losses is minimized. We refer to this optimization as *inner optimization*. The set $P$ includes all patterns that can be generated using the given set of items, having a total length smaller than $L$, and for which the inner optimization is optimal. Patterns for which the inner optimization of the pattern is not optimal, can be safely ignored due to consideration of the second objective. The pseudocode of this approach is presented in Algorithm 3.

The algorithm consists of two phases: generating all efficient patterns and solving model 3.2 - 3.4 to obtain the final solution. The pattern generation phase iterates over all possible subsets $S(I)$ (with repetition of items $i$ if $d_i > 1$) to check if they form feasible and efficient patterns; in which case they are added to the pattern set $P$.

**Algorithm 3** An enumerative approach

**Input:** $I$, $c_{ij}$, $L$, $d$
  $P \leftarrow \emptyset$
  **for all** $s \in S(I)$ **do** ▷ *For all item subsets s*
    **if LengthLowerBound**(s) < L **then**
      p $\leftarrow$ **TSPOPT**(s,$c_{ij}$) ▷ *Find optimal pattern for subset s*
      **if Length**(p) < L **then**
        $P \leftarrow P \cup \{p\}$
      **end if**
    **end if**
  **end for**
  Solution $\leftarrow$ **SC**($I$,$P$,$d$)
  **return** Solution

Checking for feasibility and efficiency requires solving a TSP problem on the cut losses matrix, restricted to the items (with multiplicity) in the considered pattern. If the optimal tour length (the minimal losses for the considered items), increased by the items' lengths, is lower than $L$, then the pattern is feasible. Clearly not all subsets of items need to be checked. A simple lower bound on the pattern length of the best permutation (**LengthLowerBound**) can be obtained by summing only the items' lenghts, without considering SDCL (or by considering that each loss is equal to the lowest value in the matrix $C$). The TSP optimization is only necessary when this lower bound does not exceed $L$.

Obviously, the main drawbacks of this algorithm are the memory needed for representing all the patterns, and the high computation time required to solve the TSP. The following heuristic approach tries to overcome these shortcomings.

### 3.2.2 Heuristic Approach

The main purpose of the heuristic is to modify the previous exact approach in order to overcome its drawbacks. The high memory required to represent the whole set of feasible patterns can be managed by reducing the number of patterns considered, at the expense of the optimality guarantee. The second critical point is the large time required to solve the TSP. A simple alternative is to solve the TSP heuristically, which leads to a sharp decrease in computational time required. In order to obtain a good quality TSP solution, we adopted an iterated version of dynasearch [23] followed by a 3-OPT [70] local search.

The pseudocode is presented in Algorithm 4. A solution of the 1D-CSP with SDCL is given by the patterns and the corresponding number of occurrences required to completely satisfy the item demand. If a part of the solution is fixed by imposing some patterns, the remaining problem consists in finding other patterns to include in the solution in order to fulfil the demand.

Let $U$ be the set of all the unitary patterns, formed by one item. If the set $U$ is a proper subset of the pattern set considered as input of the SC step, it will always provide a feasible solution. In the worst case, it will be given by adopting $d_i$ times each unitary pattern $\{i\}$ $\forall i \in I$. Let $B_{\mathbf{d}}$ be the set of the longest feasible patterns considering item set $I$ and demand vector $\mathbf{d}$.

**Algorithm 4** Heuristic approach

**Input:** $I$, $c_{ij}$ $L$, $\mathbf{d}$
  $U \leftarrow$ **GenerateUnitaryPatterns**($I$,$\mathbf{d}$,$L$)
  **d'** $\leftarrow$ **d** ▷ *Make a copy of the required demand*
  $k \leftarrow 1$
  **while TotalAmount**(**d'**) $> 0$ **do** ▷ *While demand not fully satisfied*
    $\hat{B}_{\mathbf{d}'} \leftarrow$ **GenerateLongPatterns**($I$, $c_{ij}$, **d'**,$U$)
    $P_k \leftarrow U \cup \hat{B}_{\mathbf{d}'}$
    $S_k \leftarrow \mathbf{SC}(I, \mathbf{d}', P_k)$
    Update **d'** considering all patterns $p \in S_k$ fixed
    $k \leftarrow k + 1$
  **end while**
  Solution $\leftarrow \mathbf{SC}(I, \mathbf{d}, \cup_k P_k)$
  **Output:** Solution

Each iteration $k$ of the algorithm generates a solution to the problem by applying the SC model to a set of patterns $P_k$, with $U \subseteq P_k$ in order to guarantee feasibility. More precisely, $P_k$ is determined as the union of $U$ and a set $\hat{B}_{\mathbf{d'}} \subseteq B_{\mathbf{d'}}$, which includes the best patterns of $B_{\mathbf{d'}}$ according to the leftover criterion. Note that **d'** is the vector representing the current unsatisfied demand for each item. In the pseudocode, the set $\hat{B}_{\mathbf{d'}}$ is computed by the function **GenerateLongPatterns**.

**GenerateLongPatterns** starts from the unitary patterns $U$ and iteratively generates good feasible patterns of increasing length. First, the unitary patterns are merged with themselves, generating all the possible patterns of length 2 (avoiding repetitions). Subsequently, the inner optimization algorithm is applied on the generated patterns and the best ones are kept according to the following criteria:

1. The ratio $\frac{LOAD}{WASTE}$, where $LOAD$ is the sum of the lengths of the items in the pattern and $WASTE$ is the sum of the cut losses. If it is high, it means that the large object is filled properly.

2. The value of $WASTE$.

3. The level of item coverage, such that the final set contains patterns formed by a wide variety of items.

The best patterns are merged with unitary patterns again considering all the possible combinations, and the selection of the best patterns is performed. These steps are iterated as long as feasible patterns can be generated, i.e. while patterns do not violate the stock material length constraint. In general, the first criterion is more suitable for long patterns (for which it is more important to maximize raw material utilization), while the second works better for small or medium size patterns (for which large leftovers are acceptable, and the goal is to minimize cut losses).

When the function **GenerateLongPatterns** ends, the set $\hat{B}_{\mathbf{d'}}$ is defined and $P_k$ can be computed as $U \cup \hat{B}_{\mathbf{d'}}$. At this point, the SC model is solved on $P_k$ and a feasible solution of the problem is provided. This solution is formed by two types of patterns: patterns belonging to $U$ and patterns belonging to $\hat{B}_{\mathbf{d'}}$. The latter are the best among the longest patterns that can be generated at iteration $k$. They are imposed as part of the final solution The remaining demand **d'** is updated considering this part of the solution as fixed.

These operations are iterated until the overall demand **d** is satisfied. Afterwards, the SC step is performed on all the generated patterns $\cup_k P_k$ in order to obtain a better solution.

The whole procedure can be seen as a column generation based primal heuristic, more precisely as a restricted master heuristic (see Section 1.1.4 for an introduction on column generation heuristics). The restricted master problem is model (3.2) - (3.4), and the initial columns given by the unitary patterns ensure feasibility. Additional columns are generated heuristically by the **GenerateLongPatterns** procedure.

## 3.3 Computational Study

The main research question in this work is to assess circumstances in which SDCL are relevant to consider, rather than disregarding them and applying a 1D-CSP or 1D-BPP approach. We therefore set up a computational study that compares the performance of the HSD heuristic and a 1D-BPP approach on a set of instances with varying characteristics. For the 1D-BPP approach, we opted for the MBS2+VNS heuristic from [45]. It performs well and considers the maximization of the sum of squared loads, thus aiming for very dense packings, and indirectly for reusable leftovers. This MBS2+VNS heuristic is applied to the 1D-BPP conversion of the problem described in Section 3.1.2.

The tests were executed on an Intel Core i5-3550 3.30 GHz and 4GB RAM, under Windows 7. The algorithms were implemented in C++ (except MBS2 + VNS which was coded in Java). CPLEX 12.4 was used as ILP solver. A time limit of 200 seconds was set for the SC steps of the HSD heuristic.

### 3.3.1 Experimental Setup

A set of instances was randomly generated in such a way that it includes instances with varying item size and cut losses. Firstly, all the instances have been generated such that the stock size is $L = 1000$. Two datasets have been generated. The main dataset covers a wide range of parameter values. The second, smaller, dataset consists of instances generated to study a specific feature.

Instances are uniquely identified by four parameters, denoted IS, N, CLV and ID. The first parameter IS establishes the probabilistic distribution of the item sizes, computed according to a truncated Poisson distribution with minimum and maximum values of respectively 50 and 200. The second parameter $N = \sum_{i \in I} d_i$ denotes the total number of items to be cut. The demands $d_i$ are generated according to a uniform distribution between 5 and 10, forcing that N is fixed. N can take values 25, 50, 100, 200 and 300. It defines a first approximation of the size of the instance. The third parameter CLV defines the variability of the cut losses. Both a high and low variability of the cut losses are studied. The cut losses are generated according to a uniform distribution; a setting of CLV = High denotes cut losses between 15 and 45, and CLV = Low denotes cut losses between 5 and 15. These distributions lead to a global cut losses variability corresponding respectively to 3% and 1% of the whole stock size.

Finally, five instances (distinguished by $ID$) with the same settings of the first three parameters have been generated. In what follows, we present the averaged results over these five instances, per parameter setting. The overall number of instances in this dataset is equal to 150.

The main dataset contains no instances with a very low cut loss variability. Consequently, the second set of instances was constructed with very small cut losses in order to compare the performances of HSD and 1D-CSP approaches under these circumstances. $N$ is fixed and equal to 100 and item sizes have been generated according to a uniform distribution between 100 and 150. Five instances are generated for five different values of absolute cut loss variability ($CL$ = 0,1,2,3,4). The cut losses vary uniformly between $[1,1 + CL]$.

### 3.3.2 Experimental Results and Discussion

The averaged results of the tests performed on the main dataset are reported in Tables 3.3, 3.4 and 3.5. HSD indicates the heuristic based on sequence dependencies introduced in Section 3.2.2, while MBS2+VNS refers to the heuristic in [45]. The tables report the following results for both HSD and MBS2+VNS: the averaged values of the first objective ($O1$), the second objective ($O2$) and the execution time of the algorithm ($T$, in seconds) per parameter setting, as well as the averaged lower

bound ($LB$) for those instances. Bold indicates best results. $LB$ was determined with the column generation procedure described in [92] that solves the linear relaxation of 1D-CSP with SDCL, considering only $O1$. In the tables, missing values for the lower bounds denote that the column generation model did not produce a bound with a time limit of 12 hours for at least one instance. The lower bound $LB_{BP}$ is the lower bound for the bin packing conversion of the problem. $LB_{BP}$ enables assessing the quality of the heuristic results (with respect to $O1$) for the bin packing version. For both HSD and MBS2+VNS, the average computation time required for finding these bounds is reported, as well as the number of optimal solutions found ($\#OPT$, maximum = 5). The time limit is set to 12 hours.

The computational results reveal that HSD is the best performing heuristic in terms of number of cutting patterns required. Considering SDCL is therefore relevant to find cutting patterns that make better use of raw material. Even a cut losses variability as small as 1% of the stock size and an item size as large as 30% of the stock size, should not be overlooked when considering SDCL. However, the consideration of SDCL comes at a computational cost, as the HSD heuristic requires significantly more computational time. It is possible to conclude that the computational effort is higher when item sizes are smaller (as in the case of $IS = 80$) and cut losses are smaller (as in the case of $CLV = Low$). In these cases, the inner optimization is more computationally intensive since patterns are formed by a larger number of items. Smaller cut losses make HSD less effective and increase its execution time.

Table 3.6 summarizes the results on the second set of instances that have very small cut losses (drawn uniformly from [1,1 + $CL$].). As expected, MBS2+VNS performs better than HSD if the cut losses are very small. HSD can only outperform MBS2+VNS if the absolute cut losses are larger and vary more (i.e. when cut losses are between [1,3] and larger). In the cases where the results on $O1$ are identical, HSD provides better solutions than MBS2+VNS.

| Main dataset | | HSD | | | | | | MBS2+VNS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | *CLV* | *LB* | $T_{LB}$ | *O*1 | *O*2 | *T* | # OPT. | $LB_{BP}$ | $T_{LB}$ | *O*1 | *O*2 | *T* | # OPT. |
| 25 | Low | 3.0 | 1156.0 | **3.0** | **526620.0** | 1.6 | 5 | 3.0 | 104.8 | 3.0 | 417392.6 | 0.0 | 5 |
| 25 | High | 3.0 | 327.6 | **3.0** | 124866.4 | 1.0 | 5 | 4.0 | 59.6 | 4.0 | 454319.4 | 0.0 | 5 |
| 50 | Low | 5.0 | 13316.6 | **5.0** | **449547.2** | 14.5 | 5 | 5.0 | 101.0 | 5.0 | 111841.2 | 0.2 | 5 |
| 50 | High | 5.8 | 5174.4 | **5.8** | 352242.6 | 7.8 | 5 | 7.2 | 74.8 | 7.2 | 320836.6 | 0.2 | 5 |
| 100 | Low | 9.4 | 104124.0 | **9.6** | 557879.0 | 63.4 | 4 | 10.2 | 164.2 | 10.2 | 374432.0 | 0.9 | 5 |
| 100 | High | 10.2 | 65795.8 | **10.4** | 218870.4 | 33.6 | 4 | 13.2 | 231.0 | 13.2 | 134662.4 | 0.1 | 5 |
| 200 | Low | 17.8 | 831582.6 | **17.8** | 433030.2 | 996.3 | 5 | 19.6 | 349.4 | 19.6 | 488221.4 | 2.1 | 5 |
| 200 | High | 20.2 | 3151317.8 | **20.6** | 255675.6 | 149.8 | 3 | 26.8 | 580.0 | 26.8 | 122297.6 | 0.2 | 5 |
| 300 | Low | 26.4 | 2327745.0 | **26.6** | 147403.6 | 13710.4 | 4 | 29.4 | 754.4 | 29.4 | 213160.4 | 0.3 | 5 |
| 300 | High | 29.8 | 4965184.4 | **30.6** | 533039.4 | 416.2 | 1 | 39.6 | 881.6 | 40.0 | 300060.4 | 0.4 | 3 |

Table 3.3: Results on instances with $IS = 80$.

| Main dataset | | HSD | | | | | | MBS2+VNS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | *CLV* | *LB* | $T_{LB}$ | *O*1 | *O*2 | *T* | # OPT. | $LB_{BP}$ | $T_{LB}$ | *O*1 | *O*2 | *T* | # OPT. |
| 25 | Low | 4.0 | 1131.4 | **4.0** | **398896.6** | 0.4 | 5 | 4.0 | 96.4 | 4.0 | 248966.6 | 0.0 | 5 |
| 25 | High | 4.8 | 302.0 | **4.8** | 579865.4 | 0.2 | 5 | 5.0 | 39.6 | 5.0 | 136972.0 | 0.0 | 5 |
| 50 | Low | 7.0 | 48908.0 | **7.0** | 116578.0 | 1.4 | 5 | 7.6 | 67.8 | 7.6 | 461525.6 | 0.1 | 5 |
| 50 | High | 8.0 | 45685.4 | **8.2** | 311025.8 | 1.4 | 4 | 9.8 | 48.8 | 9.8 | 263596.0 | 0.1 | 5 |
| 100 | Low | 13.8 | 2164429.2 | **14.0** | 213577.4 | 7.0 | 4 | 14.8 | 226.0 | 14.8 | 388055.6 | 0.3 | 5 |
| 100 | High | 15.0 | 1090221.0 | **15.4** | 379002.4 | 4.7 | 3 | 18.6 | 198.6 | 18.6 | 266930.8 | 0.1 | 5 |
| 200 | Low | - | 31528894.0 | **27.2** | 471917.8 | 76.7 | - | 28.8 | 519.8 | 28.8 | 478105.2 | 0.2 | 5 |
| 200 | High | 29.2 | 5925667.8 | **30.0** | 281166.0 | 24.0 | 1 | 36.8 | 320.0 | 37.6 | 506611.2 | 0.3 | 1 |
| 300 | Low | - | 40269585.2 | **40.8** | 310140.0 | 121.4 | - | 43.0 | 1098.8 | 43.4 | 432623.6 | 0.3 | 3 |
| 300 | High | - | 21508083.6 | **44.8** | 305606.8 | 62.0 | - | 55.8 | 670.8 | 56.4 | 495685.2 | 0.6 | 2 |

Table 3.4: Results on instances with $IS = 125$.

| Main dataset | | HSD | | | | | | MBS2+VNS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | $CLV$ | $LB$ | $T_{LB}$ | $O1$ | $O2$ | $T$ | # OPT. | $LB_{BP}$ | $T_{LB}$ | $O1$ | $O2$ | $T$ | # OPT. |
| 25 | Low | 5.0 | 89.0 | **5.0** | **204486.8** | 0.1 | 5 | 5.0 | 95.2 | 5.0 | 70145.6 | 0.0 | 5 |
| 25 | High | 5.4 | 77.6 | **5.6** | 277806.2 | 0.1 | 4 | 7.0 | 40.8 | 7.0 | 637881.2 | 0.0 | 5 |
| 50 | Low | 9.6 | 11609.2 | **9.8** | 183052.4 | 0.3 | 4 | 10.0 | 29.6 | 10.0 | 190604.4 | 0.1 | 5 |
| 50 | High | 10.2 | 891.0 | **10.4** | 222870.8 | 0.3 | 4 | 12.6 | 30.4 | 12.6 | 300492.6 | 0.2 | 5 |
| 100 | Low | 19.2 | 671442.6 | **19.6** | 257374.6 | 1.5 | 3 | 19.8 | 51.8 | 19.8 | 117312.6 | 0.2 | 5 |
| 100 | High | 20.0 | 510213.8 | **20.2** | 144705.6 | 1.0 | 4 | 24.6 | 54.6 | 24.6 | 467421.2 | 0.4 | 5 |
| 200 | Low | - | 36825450.0 | **37.6** | 439263.2 | 9.6 | - | 38.8 | 315.2 | 39.2 | 474066.6 | 0.4 | 3 |
| 200 | High | - | 23508436.6 | **40.0** | 135964.6 | 5.6 | - | 49.2 | 117.4 | 49.2 | 624402.4 | 2.8 | 5 |
| 300 | Low | - | 22080908.6 | **57.0** | 716090.4 | 35.4 | - | 58.6 | 606.2 | 59.0 | 522072.4 | 0.7 | 3 |
| 300 | High | - | 42738567.4 | **60.0** | 214577.6 | 17.4 | - | 74.6 | 176.8 | 74.6 | 1025290.8 | 4.1 | 5 |

Table 3.5: Results on instances with $IS = 170$.

| | HSD | | | | | | MBS2+VNS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CL | $LB$ | $T_{LB}$ | $O1$ | $O2$ | $T$ | # OPT. | $LB_{BP}$ | $T_{LB}$ | $O1$ | $O2$ | $T$ | # OPT. |
| 0 | 13.4 | 412.2 | 13.6 | 526327.8 | 19.1 | 4 | 13.4 | 333.6 | **13.4** | 464630.0 | 1.3 | 5 |
| 1 | 13.4 | 1224420.0 | 13.6 | 422938.8 | 16.4 | 4 | 13.4 | 260.4 | **13.4** | 311685.0 | 1.0 | 5 |
| 2 | 13.4 | 1297271.2 | **13.6** | **491371.8** | 17.2 | 4 | 13.6 | 267.2 | **13.6** | 370179.0 | 0.7 | 5 |
| 3 | 13.4 | 766664.8 | **13.4** | 290571.6 | 15.9 | 5 | 13.6 | 289.2 | 13.6 | 272759.4 | 1.4 | 5 |
| 4 | 13.4 | 1145944.6 | **13.6** | **386685.6** | 14.1 | 4 | **13.6** | 247.4 | 13.6 | 177527.4 | 0.8 | 5 |

Table 3.6: Summary of results on the second set of generated instances.

## 3.4 Conclusions

The present chapter discussed a one dimensional cutting stock problem with sequence dependent cut losses (1D-CSP with SDCL) that considers minimization of the number of raw materials required for cutting a set of items. As a secondary objective, the reusability of leftover material is considered.

It was shown that the problem can be approximately solved using a standard one dimensional cutting stock problem or one dimensional bin packing approach. Therefore, the main research question was to identify beneficial conditions to specifically consider SDCL.

A computational study on a set of generated instances with varying characteristics showed that it is clearly beneficial to consider SDCL, whenever the item size is small, i.e. up to 30% of the stock size, and when cut losses are not too small and have some variability, i.e. larger than 0.2% of the stock size.

The heuristic approach developed for this problem hybridizes a typical CG framework, such that it provides high quality heuristic solutions. In the approach, a mathematical solver runs on a SC model and a time limit is set such that it is used in a heuristic fashion. The effectiveness of the approach is validated in the computational experiment section, by comparing it with a standard bin packing approach.

# Chapter 4

# An Exact Exponential Branch and Merge Approach for the Total Tardiness Problem

Section 1.3 introduced the basic concepts in the design of exact exponential algorithms for combinatorial NP-hard problems. In spite of the growing interest on this field, few results are yet known on scheduling problems, see the survey of Lenté et al. [68]. In [67], Lenté et al. introduced the so-called class of multiple constraint problems and showed that all problems fitting into that class could be tackled by means of the Sort & Search technique. Further, they showed that several known scheduling problems are part of that class. However, all these problems required assignment decisions only and none of them required the solution of a sequencing problem.

This work focuses on a pure sequencing problem, the single machine total tardiness $1||\sum T_j$ problem. In this problem, a jobset $N = \{1,2,\ldots,n\}$ of $n$ jobs must be scheduled on a single machine. For each job $j$, a processing time $p_j$ and a due date $d_j$ are defined. The problem asks for arranging the jobset in a sequence $S = (1,2,\ldots,n)$ so as to minimize $T(N,S) = \sum_{j=1}^{n} T_j = \sum_{j=1}^{n} \max\{C_j - d_j,0\}$, where $C_j = \sum_{i=1}^{j} p_i$. The $1||\sum T_j$ problem is NP-hard in the ordinary sense [39]. It has been extensively studied in the literature and many exact procedures ([36, 66, 80, 88]) have been proposed. The current state-of-the-art exact method of [88] dates back to 2001 and

solves to optimality problems with up to 500 jobs. All these procedures are search tree approaches, but dynamic programming algorithms were also considered. On the one hand, in [66] a pseudo-polynomial dynamic programming algorithm was proposed running with complexity $\mathcal{O}(n^4 \sum p_i)$. On the other hand, the standard technique of doing dynamic programming across the subsets (see Section 1.3.2) applies and runs with complexity $\mathcal{O}^*(2^n)$ both in time and in space. Latest theoretical developments for the problem, including both exact and heuristic approaches can be found in the recent survey of Koulamas [63].

To the authors' knowledge, there is no available exact algorithm for this problem running in $\mathcal{O}^*(c^n)$ ($c$ being a constant) and polynomial space. Admittedly, one could possibly apply a divide-and-conquer approach as in [55] and [18]. This would lead to an $\mathcal{O}^*(4^n)$ complexity in time requiring polynomial space. Aim of this work is to present an improved exact algorithm exploiting known decomposition properties of the problem. Different versions of the proposed approach are described in Section 4.1. A final version making use of a new technique called branch and merge that avoids the solution of several equivalent subproblems in the branching tree is presented in Section 4.2. This version is shown to have a complexity that tends to $\mathcal{O}^*(2^n)$ in time and requires polynomial space. Finally, Section 4.3 discusses further developments of this work and possible extensions of the presented technique.

## 4.1 A Branch and Reduce Approach

First, we recall some basic properties of the total tardiness problem and introduces the notation used along the chapter. Given the jobset $N = \{1,2,\ldots,n\}$, let $(1,2,\ldots,n)$ be a LPT (Longest Processing Time first) sequence, where $i < j$ whenever $p_i > p_j$ (or $p_i = p_j$ and $d_i \leq d_j$). Let also $([1],[2],\ldots,[n])$ be an EDD (Earliest Due Date first) sequence, where $i < j$ whenever $d_{[i]} < d_{[j]}$ (or $d_{[i]} = d_{[j]}$ and $p_{[i]} \leq p_{[j]}$). As the cost function is a regular performance measure, we know that in an optimal solution, the jobs are processed with no interruption starting from time zero. Let $B_j$ and $A_j$ be the sets of jobs that precede and follow job $j$ in an optimal sequence. Correspondingly, $C_j = \sum_{k \in B_j} p_k + p_j$. Similarly, if job $j$ is assigned to position $k$, we denote by $C_j(k)$ the corresponding completion time and by $B_j(k)$ and

$A_j(k)$ the sets of predecessors and successors of $j$, respectively.

The main known theoretical properties are the following.

**Property 1.** *[41] Consider two jobs $i$ and $j$ with $p_i < p_j$. Then, $i$ precedes $j$ in an optimal schedule if $d_i \leq \max\{d_j, C_j\}$, else $j$ precedes $i$ in an optimal schedule if $d_i + p_i > C_j$.*

**Property 2.** *[66] Let job 1 in LPT order correspond to job $[k]$ in EDD order. Then, job 1 can be set only in positions $h \geq k$ and the jobs preceding and following job 1 are uniquely determined as $B_1(h) = \{[1], [2], \ldots, [k-1], [k+1], \ldots, [h]\}$ and $A_1(h) = \{[h+1], \ldots, [n]\}$.*

**Property 3.** *[66, 80, 87] Consider $C_1(h)$ for $h \geq k$. Job 1 cannot be set in position $h \geq k$ if:*

**(a)** $C_1(h) \geq d_{[h+1]}$, $h < n$;

**(b)** $C_1(h) < d_{[r]} + p_{[r]}$, *for some* $r = k+1, \ldots, h$.

**Property 4.** *([89]) For any pair of adjacent positions $(i, i+1)$ that can be assigned to job 1, at least one of them is eliminated by Property 3.*

In terms of complexity analysis, we recall that, if it is possible to bound above $T(n)$ by a recurrence expression of the type $T(n) \leq \sum_{i=1}^{h} T(n - r_i) + \mathcal{O}(p(n))$, then we have $\sum_{i=1}^{h} T(n - r_i) + \mathcal{O}(p(n)) = \mathcal{O}^*(\alpha(r_1, \ldots, r_h)^n)$ where $\alpha(r_1, \ldots, r_h)$ is the largest root of the function $f(x) = 1 - \sum_{i=1}^{h} x^{-r_i}$.

A basic branch and reduce algorithm TTBR1 (Total Tardiness Branch and Reduce version 1) can be designed by exploiting Property 2, which allows to decompose the problem into two smaller subproblems when the position of the longest job 1 is given. The basic idea is to iteratively branch by assigning job 1 to every possible position $(1, ..., n)$ and correspondingly decompose the problem. Each time job 1 is assigned to a certain position $i$, two different subproblems are generated, corresponding to schedule the jobs before $l$ (inducing subproblem $B_l(i)$) or after $l$ (inducing subproblem $A_l(i)$), respectively. The algorithm operates by applying to any given jobset $S$ starting at time $t$ function $TTBR1(S, t)$ that computes the corresponding optimal solution. With this notation, the original problem is indicated

by $N = \{1, ..., n\}$ and the optimal solution is reached when function $TTBR1(N,0)$ is computed.

The algorithm proceeds by solving the subproblems along the branching tree according to a depth-first strategy and runs until all the leaves of the search tree have been reached. Finally, it provides the best solution found as an output. Algorithm 5 summarizes the structure of this approach, while Proposition 1 states its worst-case complexity.

**Algorithm 5** Total Tardiness Branch and Reduce version 1 (TTBR1)

**Input:** $N = \{1, ..., n\}$ is the problem to be solved

1: **function** TTBR1$(S, t)$
2: $seqOpt \leftarrow$ a random sequence of jobs
3: $l \leftarrow$ the longest job in $N$
4: **for** $i = 1$ to n **do**
5: Branch by assigning job $l$ to position $i$
6: $seqLeft \leftarrow$ TTBR1$(B_l(i), t)$
7: $seqRight \leftarrow$ TTBR1$(A_l(i), t + \sum_{k \in B_l(i)} p_k + p_l)$
8: $seqCurrent \leftarrow$ concatenation of $seqLeft$, $l$ and $seqRight$
9: $seqOpt \leftarrow$ best solution between $seqOpt$ and $seqCurrent$
10: **end for**
11: **return** $seqOpt$
12: **end function**

**Proposition 1.** *Algorithm TTBR1 runs in $\mathcal{O}^*(3^n)$ time and polynomial space in the worst case.*

*Proof.* Whenever the longest job 1 is assigned to the first and the last position of the sequence, two subproblems of size $n-1$ are generated. For each $2 \leq i \leq n-1$, two subproblems with size $i-1$ and $n-i$ are generated. Hence, the total number of generated subproblems is $2n-2$ and the time cost related to computing the best solution of size $n$ starting from these subproblems is $\mathcal{O}(p(n))$. This induces the following recurrence for the running time $T(n)$ required by TTBR1:

$$T(n) = 2T(n-1) + 2T(n-2) + ... + 2T(2) + 2T(1) + \mathcal{O}(p(n)) \tag{4.1}$$

By replacing $n$ with $n-1$, the following expression is derived:

$$T(n-1) = 2T(n-2) + ... + 2T(2) + 2T(1) + \mathcal{O}(p(n-1)) \tag{4.2}$$

Expression 4.2 can be used to simplify the right hand side of expression 4.1 leading to:

$$T(n) = 3T(n-1) + \mathcal{O}(p(n)) \tag{4.3}$$

that induces as complexity $\mathcal{O}^*(3^n)$. The space requirement is polynomial since the branching tree is explored according to a depth-first strategy. □

An improved version of the algorithm is defined by taking into account Property 3 and Property 4, which state that for each pair of adjacent positions $(i, i+1)$, at least one of them can be discarded. The worst case occurs when the largest possible subproblems are kept. This corresponds to solving problems with size $n-1, n-3, n-5, \ldots$, that arise by branching on positions $i$ and $n-i+1$ with $i$ odd. The resulting algorithm is referred to as TTBR2 (Total Tardiness Branch and Reduce version 2). Its structure is equal to the one of TTBR1 depicted in Algorithm 5, but lines 5-9 are executed only when $l$ can be set on position $i$ according to Property 3. The complexity of the algorithm is discussed in Proposition 2.

**Proposition 2.** *Algorithm TTBR2 runs in* $\mathcal{O}^*((1+\sqrt{2})^n) = \mathcal{O}^*(2.4143^n)$ *time and polynomial space in the worst case.*

*Proof.* The proof is close to that of Proposition 1. The algorithm induces a recursion of the type:

$$T(n) = 2T(n-1) + 2T(n-3) + ... + 2T(4) + 2T(2) + \mathcal{O}(p(n)) \tag{4.4}$$

as the worst case occurs when we keep the branches that induce the largest possible subproblems. Analogously to Proposition 1, we replace $n$ with $n-2$ in the previous recurrence and we obtain:

$$T(n-2) = 2T(n-3) + 2T(n-5) + ... + 2T(4) + 2T(2) + \mathcal{O}(p(n-2)) \tag{4.5}$$

Again, we plug the latter expression into the former one and obtain the recurrence:

$$T(n) = 2T(n-1) + T(n-2) + \mathcal{O}(p(n)) \tag{4.6}$$

that induces as complexity $\mathcal{O}^*((1+\sqrt{2})^n) = \mathcal{O}^*(2.4143^n)$. The space complexity is still polynomial. □

## 4.2 A Branch and Merge Algorithm

In this section, we describe how to get an algorithm running with complexity arbitrarily close to $\mathcal{O}^*(2^n)$ in time and polynomial space by integrating a node-merging procedure into TTBR1. We recall that in TTBR1 the branching scheme is defined by assigning the longest unscheduled job to each available position and accordingly divide the problem into two subproblems. Hereafter, we focus on the scenario where the LPT sequence $(1, ..., n)$ coincides with the EDD sequence $([1], ..., [n])$, for convenience we write $LPT = EDD$. This scenario eases the description of the algorithm.

Figures 4.1 shows how an input problem $\{1, ..., n\}$ is decomposed by the branching scheme of TTBR1. Each node is labelled by the corresponding subproblem $P_j$ ($P$ denotes the input problem). Notice that from now on $P_{j_1,j_2,...,j_k}, 1 \leq k \leq n$, denotes the problem (node in the search tree) induced by the branching scheme of TTBR1 when the largest processing time job 1 is in position $j_1$, the second largest processing time job 2 is in position $j_2$ and so on till the k-th largest processing time job $k$ being placed in position $j_k$.

To roughly illustrate the guiding idea of the merging technique introduced in this section, consider Figures 4.1. Noteworthy, nodes $P_2$ and $P_{1,2}$ are identical except for the initial subsequence (21 vs 12). This fact implies, in this particular case, that the problem of scheduling jobset $\{3, ..., n\}$ at time $p_1 + p_2$ is solved twice. This kind of redundancy can however be eliminated by merging node $P_2$ with node $P_{1,2}$ and creating a single node in which the best sequence among 21 and 12 is scheduled at the beginning and the jobset $\{3, ..., n\}$, starting at time $p_1 + p_2$, remains to be branched on. Furthermore, the best subsequence (starting at time $t = 0$) between

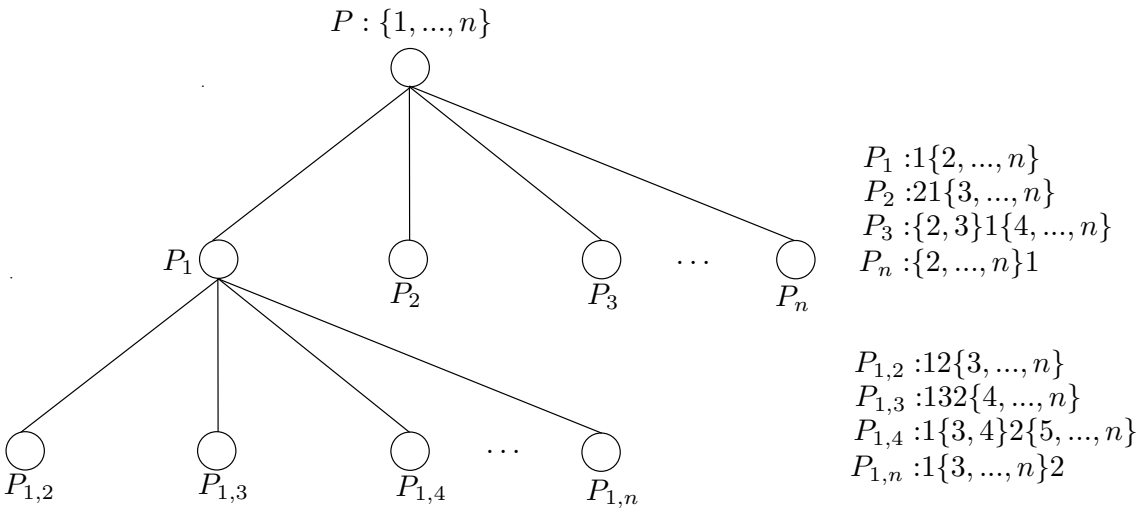



Figure 4.1: The branching scheme of TTBR1 at the root node

21 and 12 can be computed in constant time. Hence, the node created after the merging operation involves a constant time preprocessing step plus the search for the optimal solution of jobset $\{3, ..., n\}$ to be processed starting at time $p_1 + p_2$. We remark that, in the branching scheme of TTBR1, for any constant $k \geq 3$, the branches corresponding to $P_i$ and $P_{n-i+1}$, with $i = 2, ..., k$, are decomposed into two problems where one subproblem has size $n - i$ and the other problem has size $i - 1 \leq k$. Correspondingly, the merging technique presented on problems $P_2$ and $P_{1,2}$ can be generalized to all branches inducing problems of sizes less than $k$. Notice that, by means of algorithm TTBR2, any problem of size less than $k$ requires at most $\mathcal{O}^*(2.4143^k)$ time (that is constant time when $k$ is fixed). In the remainder of the chapter, for any constant $k \leq \frac{n}{2}$, we denote by left-side branches the search tree branches corresponding to problems $P_1, ..., P_k$ and by right-side branches the ones corresponding to problems $P_{n-k+1}, ..., P_n$.

In the following subsections, we show how the node-merging procedure can be systematically performed to improve the time complexity of TTBR1. Basically, two different recurrent structures hold respectively for left-side and right-side branches and allow to generate fewer subproblems at each recursion level. The node-merging mechanism is described by means of two distinct procedures, called `LEFT_MERGE` (applied to left-side branches) and `RIGHT_MERGE` (applied to right-side branches), which are discussed in Section 4.2.1 and Section 4.2.2, respectively. The final branch

and merge algorithm is described in Section 4.2.3 and embeds both procedures in the structure of TTBR1.

### 4.2.1 Merging Left-Side Branches

The first part of the section aims at illustrating the merging operations on the root node. The following proposition highlights two properties of the couples of problems $P_j$ and $P_{1,j}$ with $2 \leq j \leq k$.

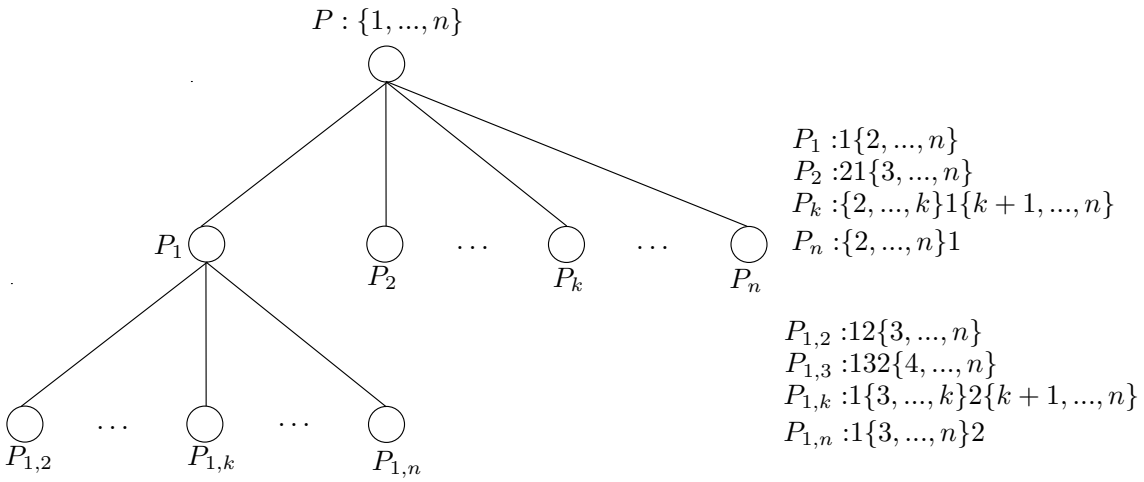
**Proposition 3.** *For a couple of problems $P_j$ and $P_{1,j}$ with $2 \leq j \leq k$, the following conditions hold:*

1. *The solution of problems $P_j$ and $P_{1,j}$ involves the solution of a common subproblem which consists in scheduling jobset $\{j+1, ..., n\}$ starting at time $t = \sum_{i=1,...,j} p_i$.*
2. *Both in $P_j$ and $P_{1,j}$, at most $k$ jobs have to be scheduled before jobset $\{j+1, ..., n\}$.*

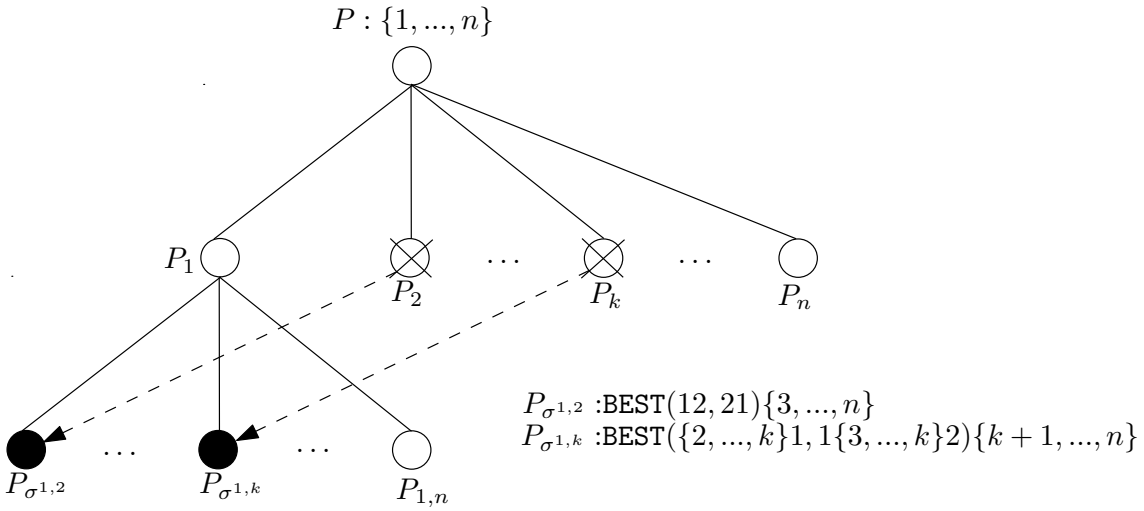
*Proof.* As problems $P_j$ and $P_{1,j}$ are respectively defined by $\{2, ..., j\}1\{j+1, ..., n\}$ and $1\{3, ..., j\}2\{j+1, ..., n\}$, the first part of the property is straightforward.
The second part can be simply established by counting the number of jobs to be scheduled before jobset $\{j+1, ..., n\}$ when $j$ is maximal, *i.e.* when $j = k$. In this case, jobset $\{k+1, ..., n\}$ has $(n-k)$ jobs which implies that $k$ jobs remain to be scheduled before that jobset. □

Each couple of problems indicated in Proposition 3 can be merged as soon as they share the same subproblem to be solved. More precisely, $(k-1)$ problems $P_j$ (with $2 \leq j \leq k$) can be merged with the corresponding problems $P_{1,j}$.

Figure 4.2 illustrates the merging operations performed for the root node on its left-side branches, by showing the branch tree before and after (Figure 4.2a and Figure 4.2b) such merging operations. For any given $2 \leq j \leq k$, problems $P_j$ and $P_{1,j}$ share the same subproblem $\{j+1, ..., n\}$ starting at time $t = \sum_{i=1}^{j} p_i$. Hence, by merging the left part of both problems which is constituted by jobset $\{1, ..., j\}$ having size $j \leq k$, we can delete node $P_j$ and replace node $P_{1,j}$ in the search tree by the node $P_{\sigma^{1,j}}$ which is defined as follows (Figure 4.2b):




(a) Left-side branches of $P$ before performing the merging operations




(b) Left-side branches of $P$ after performing the merging operations

Figure 4.2: Left-side branches merging at the root node

- Jobset $\{j+1, ..., n\}$ is the set of jobs on which it remains to branch on.

- Let $\sigma^{1,j}$ be the sequence of branching positions on which the $j$ longest jobs $1, ..., j$ are branched, that leads to the best jobs permutation between $\{2, ..., j\}1$ and $1\{3, ..., j\}2$. This involves the solutions of two problems of size at most $k-1$ (in $\mathcal{O}^*(2.4143^k)$ time by TTBR2) and the comparison of the total tardiness value of the two sequences obtained.

In the following, we describe how to apply analogous merging operations on any node of the tree. With respect to the root node, the only additional consideration is that the child nodes of a generic node may already been affected by a previous merging. As an example, let us consider Figure 4.2b. It shows that, subsequently to the merging operations performed in $P$, the first $k-1$ branches of $P_1$ may not be the subproblems induced by the usual branching scheme.

In order to define the branching scheme used with the `LEFT_MERGE` procedure, a data structure $\mathcal{L}_\sigma$ is associated with a problem $P_\sigma$. It represents a list of $k-1$ subproblems that result from a previous merging and are now the first $k-1$ child nodes of $P_\sigma$. When $P_\sigma$ is created by branching, $\mathcal{L}_\sigma = \emptyset$. When a merging operation sets the first $k-1$ child nodes of $P_\sigma$ to $P_{\sigma^1}, ..., P_{\sigma^{k-1}}$, we set $\mathcal{L}_\sigma = \{P_{\sigma^1}, ..., P_{\sigma^{k-1}}\}$. As a conclusion, the following branching scheme for a generic node of the tree holds.

**Definition 1.** *The branching scheme for a generic node $P_\sigma$ is defined as follows:*

- *If $\mathcal{L}_\sigma = \emptyset$, use the branching scheme of TTBR1;*
- *If $\mathcal{L}_\sigma \neq \emptyset$, extract problems from $\mathcal{L}_\sigma$ as the first $k-1$ branches, then branch on the longest job in the available positions from the $k$-th to the last.*

*This branching scheme, whenever necessary, will be referred to as* ***improved branching****.*

Before describing how merging operations can be applied on a generic node $P_\sigma$, we highlight its structural properties by means of Proposition 4.

**Proposition 4.** *Let $P_\sigma$ be a problem to branch on, and $\sigma$ be the permutation of positions assigned to jobs $1, \ldots, |\sigma|$, with $\sigma$ empty if no positions are assigned. The following properties hold:*

1. *$j^* = |\sigma| + 1$ is the job to branch on,*
2. *$j^*$ can occupy in the branching process, positions $\{\ell_b, \ell_b + 1, \ldots, \ell_e\}$, where*

$$\ell_b = \begin{cases} |\sigma| + 1 & \text{if } \sigma \text{ is a permutation of } 1, \ldots, |\sigma| \text{ or } \sigma \text{ is empty} \\ \rho_1 + 1 & \text{otherwise} \end{cases}$$

*with* $\rho_1 = \max\{i : i > 0,\ \textit{positions } 1, \ldots, i \textit{ are in } \sigma\}$ *and*

$$\ell_e = \begin{cases} n & \textit{if } \sigma \textit{ is a permutation of } 1, \ldots, |\sigma| \textit{ or } \sigma \textit{ is empty} \\ \rho_2 - 1 & \textit{otherwise} \end{cases}$$

*with* $\rho_2 = \min\{i :\ i > \rho_1,\ i \in \sigma\}$

*Proof.* According to the definition of the notation $P_\sigma$, $\sigma$ is a sequence of positions that are assigned to the longest $|\sigma|$ jobs. Since we always branch on the longest unscheduled job, the first part of the proposition is straightforward. The second part aims at specifying the range of positions that job $j^*$ can be branched on. Two cases are considered depending on the content of $\sigma$:

- If $\sigma$ is a permutation of $1, \ldots, |\sigma|$, it means that the longest $|\sigma|$ jobs are set on the first $|\sigma|$ positions, which implies that the job $j^*$ should be branched on positions $|\sigma| + 1$ to $n$

- If $\sigma$ is not a permutation of $\{1, \ldots, |\sigma|\}$, it means that the longest $|\sigma|$ jobs are not set on consecutive positions. As a result, the current unassigned positions may be split into several ranges. As a consequence of the decomposition property, the longest job $j^*$ should necessarily be branched on the first range of free positions, that goes from $\rho_1$ to $\rho_2$. Let us consider as an example $P_{1,9,2,8}$, whose structure is $13\{5, \ldots, 9\}42\{10, \ldots, n\}$ and the job to branch on is 5. In this case, we have: $\sigma = (1,9,2,8)$, $\ell_b = 3$, $\ell_e = 7$. It is easy to verify that 5 can only be branched on positions $\{3, \ldots, 7\}$ as a direct result of Property 2.

□

Corollary 1 emphasises the fact that even though a node may contain several ranges of free positions, only the first range is the current focus since we only branch the longest job in eligible positions.

**Corollary 1.** *Problem* $P_\sigma$ *has the following structure:*

$$\pi\{j^*, \ldots, j^* + \ell_e - \ell_b\}\Omega$$

*with $\pi$ the sequence of jobs on first $\ell_b - 1$ positions in $\sigma$ and $\Omega$ the schedule on all positions after $\ell_e$. The merging procedure is applied on jobset $\{j^*, \dots, j^* + \ell_e - \ell_b\}$ starting at time $t_\pi = \sum_{i \in \Pi} p_i$ where $\Pi$ is the jobset of $\pi$.*

The validity of merging on a general node still holds as indicated in Proposition 5, which extends the result stated in Proposition 3.

**Proposition 5.** *Let $P_\sigma$ be a generic problem and let $\pi, j^*, \ell_b, \ell_e, \Omega$ be computed relatively to $P_\sigma$ according to Corollary 1. If $\mathcal{L}_\sigma = \emptyset$ the $j$-th child node $P_{\sigma^j}$ is $P_{\sigma,\ell_b+j-1}$ for $1 \leq j \leq k$. Otherwise, the $j$-th child node $P_{\sigma^j}$ is extracted from $\mathcal{L}_\sigma$ for $1 \leq j \leq k-1$, while it is created as $P_{\sigma,\ell_b+k-1}$ for $j=k$. For any couple of problems $P_{\sigma^j}$ and $P_{\sigma^1,\ell_b+j-1}$ with $2 \leq j \leq k$, the following conditions hold:*

1. *Problems $P_{\sigma^j}$ and $P_{\sigma^1,\ell_b+j-1}$ with $2 \leq j \leq k$ have the following structure:*
   - $P_{\sigma^j}$:
   $$\begin{cases} \pi^j\{j^*+j, \dots, j^*+\ell_e-\ell_b\}\Omega & 1 \leq j \leq k-1 \text{ and } \mathcal{L}_\sigma \neq \emptyset \\ \pi\{j^*+1, \dots, j^*+j-1\}j^*\{j^*+j, \dots, j^*+\ell_e-\ell_b\}\Omega & (1 \leq j \leq k-1; \mathcal{L}_\sigma = \emptyset) \text{ or } j=k \end{cases}$$
   - $P_{\sigma^1,\ell_b+j-1}$:
   $\pi^1\{j^*+2, \dots, j^*+j-1\}(j^*+1)\{j^*+j, \dots, j^*+\ell_e-\ell_b\}\Omega$
2. *By solving all the problems of size less than $k$, that consist in scheduling the jobset $\{j^*+1, \dots, j^*+j-1\}$ between $\pi$ and $j^*$ and in scheduling $\{j^*+2, \dots, j^*+j-1\}$ between $\pi^1$ and $j^*+1$, both $P_{\sigma^j}$ and $P_{\sigma^1,\ell_b+j-1}$ consist in scheduling $\{j^*+j, ..., j^*+\ell_e-\ell_b\}\Omega$ starting at time $t_{\pi^j} = \sum_{i \in \Pi^j} p_i$ where $\Pi^j$ is the jobset of $\pi^j$.*

*Proof.* The first part of the statement follows directly from Definition 1 and simply defines the structure of the child nodes of $P_\sigma$. The problem $P_{\sigma^j}$ is the result of a merging operation with the generic problem $P_{\sigma,\ell_b+j-1}$ and it could possibly coincide with $P_{\sigma,\ell_b+j-1}$, for each $j=1, ..., k-1$. Furthermore, $P_{\sigma^j}$ is exactly $P_{\sigma,\ell_b+j-1}$ for $j=k$. The generic structure of $P_{\sigma,\ell_b+j-1}$ is $\pi\{j^*+1, \dots, j^*+j-1\}j^*\{j^*+j, \dots, j^*+\ell_e-\ell_b\}\Omega$, and the merging operations preserve the jobset to schedule after $j^*$. Thus, we

have $\Pi^j$=$\Pi\cup\{j^*, ..., j^*+j-1\}$ for each $j$=$1, ..., k-1$, and this proves the first statement. Analougosly, the structure of $P_{\sigma^1,\ell_b+j-1}$ is $\pi^1\{j^*+2, \ldots, j^*+j-1\}(j^*+1)\{j^*+j, \ldots, j^*+\ell_e-\ell_b\}\Omega$. Once the subproblem before $j^*+1$ of size less than $k$ is solved, $P_{\sigma^1,\ell_b+j-1}$ consists in scheduling the jobset $\{j^*+j, ..., j^*+\ell_e-\ell_b\}$ at time $t_{\pi^j}$=$\sum_{i\in\Pi^j} p_i$. In fact, we have that $\Pi^j$=$\Pi^1\cup\{j^*+2, \ldots, j^*+j-1\}\cup\{j^*+1\}$=$\Pi\cup\{j^*, \ldots, j^*+j-1\}$ . □

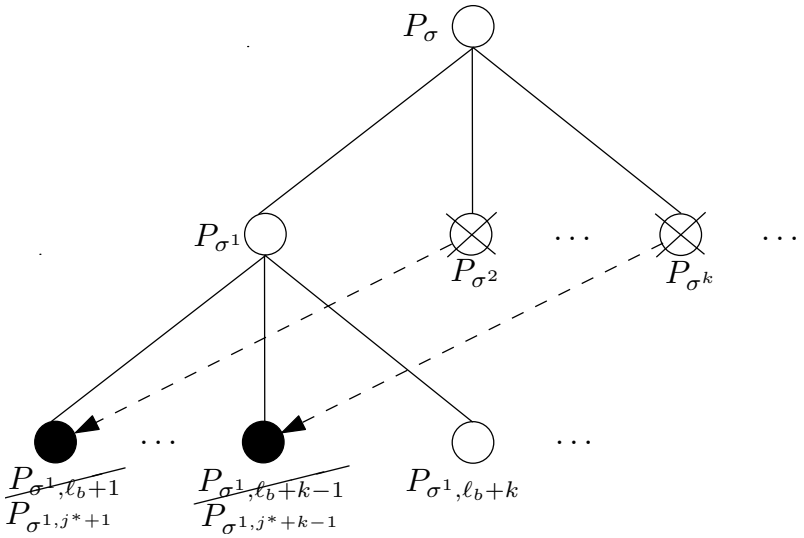



Figure 4.3: Merging for a generic left-side branch

Analogously to the root node, each couple of problems indicated in Proposition 5 can be merged. Again, $(k-1)$ problems $P_{\sigma^j}$ (with $2 \leq j \leq k$) can be merged with the corresponding problems $P_{\sigma^1,\ell_b+j-1}$. $P_{\sigma^j}$ is deleted and $P_{\sigma^1,\ell_b+j-1}$ is replaced by $P_{\sigma^{1,j^*+j-1}}$ (Figure 4.3), defined as follows:

- Jobset $\{j^*+j, ..., j^*+\ell_e-\ell_b\}\Omega$ is the set of jobs on which it remains to branch on.
- Let $\sigma^{1,j^*+j-1}$ be the sequence of branching positions on which the $j^*+j-1$ longest jobs $1, ..., j^*+j-1$ are branched, that leads to the best jobs permutation between $\pi^j$ and $\pi^1\{j^*+2, \ldots, j^*+j-1\}(j^*+1)$ for $2 \leq j \leq k-1$, and between $\pi\{j^*+1, \ldots, j^*+j-1\}j^*$ and $\pi^1\{j^*+2, \ldots, j^*+j-1\}(j^*+1)$ for $j = k$. This involves the solutions of one or two problems of size at most $k-1$ (in $\mathcal{O}^*(2.4143^k)$ time by TTBR2) and the finding of the sequence that has the

smallest total tardiness value knowing that both sequences are scheduled at time 0.

The LEFT_MERGE procedure is presented in Algorithm 6. Notice that, from a technical point of view, this algorithm takes as input one problem and produces as an output its first child node to branch on, which replaces all its $k$ left-side child nodes.

**Algorithm 6** LEFT_MERGE Procedure

**Input:** $P_\sigma$ an input problem of size $n$, with $\ell_b$, $j^*$ accordingly computed
**Output:** $Q$: a list of problems to branch on after merging

1: **function** LEFT_MERGE($P_\sigma$)
2: $Q \leftarrow \emptyset$
3: **for** $j$=1 to $k$ **do**
4: Create $P_{\sigma^j}$ ($j$-th child of $P_\sigma$) by the improved branching with the sub-problem induced by jobset $\{j^*+1, \ldots, j^*+j-1\}$ solved if $\mathcal{L}_\sigma=\emptyset$ or $j=k$
5: **end for**
6: **for** $j$=1 to $k-1$ **do**
7: Create $P_{\sigma^{1j}}$ ($j$-th child of $P_{\sigma^1}$) by the improved branching with the sub-problem induced by jobset $\{j^*+2, \ldots, j^*+j-1\}$ solved if $\mathcal{L}_{\sigma^1}=\emptyset$ or $j=k$
8: $\mathcal{L}_{\sigma^1} \leftarrow \mathcal{L}_{\sigma^1} \cup$ BEST($P_{\sigma^{j+1}}, P_{\sigma^{1j}}$)
9: **end for**
10: $Q \leftarrow Q \cup P_{\sigma^1}$
11: **return** $Q$
12: **end function**

**Lemma 1.** *The LEFT_MERGE procedure returns one node to branch on in $\mathcal{O}(n)$ time and polynomial space. The corresponding problem is of size $n-1$.*

*Proof.* The creation of problems $P_{\sigma^1,\ell_b+j-1}$, $\forall j = 2, \ldots, k$, can be done in $\mathcal{O}(n)$ time. The call of TTBR2 costs constant time. The BEST function called at line 8 consists in computing then comparing the total tardiness value of two known sequence of jobs starting at the same time instant: it runs in $\mathcal{O}(n)$ time. The overall time complexity of LEFT_MERGE procedure is then bounded by $\mathcal{O}(n)$ time as $k$ is a constant. Finally, as only the node $P_{\sigma^1}$ is returned, its size is clearly $n-1$ considering $P_\sigma$ has size $n$. □

In the final part of this section, we discuss the extension of the algorithm in the case where $LPT \neq EDD$. In this case, Property 2 is supposed to be integrated to the `LEFT_MERGE` procedure, which allows to discard subproblems associated with branching in some positions (one at least). If a problem $P$ can be discarded according to this property, then we say that $P$ does not exist and its associated node is empty.

**Lemma 2.** *Instances such that $LPT = EDD$ correspond to worst-case instances for which the `LEFT_MERGE` procedure returns one node of size $n-1$ to branch on, replacing all the $k$ left-side child nodes of its parent node.*

*Proof.* Let us consider the improved branching scheme. The following exhaustive conditions hold:

1. $1 = [1]$ and $2 = [2]$;
2. $1 = [j]$ with $j \geq 2$;
3. $1 = [1]$ and $2 = [j]$ $j \geq 3$.

In case 1, the branching scheme matches the one of Figure 4.2, hence Lemma 2 holds according to 1. In case 2, the problem $P_{\sigma^1}$ is empty if no problem has been merged to its position in the tree previously. The node associated with $P_{\sigma^1,\ell_b+\ell-1}$, $\forall \ell \leq k$, can then be considered as empty node, hence the merging can be done by simply moving the problem $P_{\sigma^\ell}$ into $P_{\sigma^1,\ell_b+\ell-1}$. As a consequence, the node returned by `LEFT_MERGE` only contains the merged nodes as children nodes, whose solution is much faster than solving a problem of size $n-1$. If $P_{\sigma^1}$ is not empty due to a previous merging operation, the merging can be performed in the ordinary way. In case 3, the nodes associated with $P_{\sigma^1,\ell_b+1}, ..., P_{\sigma^1,\ell_b+j-2}$ may or may not be empty depending on the previous merging operation concerning $P_{\sigma^1}$, in either case the merging can be done. The same reasoning holds for nodes associated with $P_{\sigma^\ell}$ and $P_{\sigma^1,\ell_b+\ell-1}$ for $\ell \geq j$.

In general, the solution of problems $P_{\sigma^\ell}$, $\forall \ell = 2, \ldots, k$, can always be avoided. In the worst case, the node associated with $P_{\sigma^1}$ contains a subproblem of size $n-1$, otherwise with the application of Property 2, it contains a problem whose certain children are set as empty. Therefore, Lemma 2 holds.

□

### 4.2.2 Merging Right-Side Branches

Due to the branching scheme, the merging of right-side branches involves a more complicated procedure than for the merging of left-side branches. In the merging of left-side branches, it is possible to merge some nodes associated with problems $P_\ell$ with child nodes of $P_1$, while for the right-side branches, it is not possible to merge some nodes $P_\ell$ with child nodes of $P_n$. We can only merge child nodes of $P_\ell$ with child nodes of $P_n$. Let us more formally introduce the right merging procedure and, again, let $k < \frac{n}{2}$ be the same constant parameter as used in the left merging.

Figure 4.4 shows an example on the structure of merging for the $k$ right-side branches with $k = 3$. The root problem $P$ consists in scheduling jobset $\{1, \ldots, n\}$. Unlike left-side merging, the right-side merging can only be done horizontally for each level. Nodes that are involved in merging are colored by a dark color, all dark nodes of the same shape and grayscale can be merged to a single one. Notice that each right-side branch of $P$ is expanded to a different depth which is actually an arbitrary decision: the expansion stops when the first child node has size $n - k - 1$ as indicated in the figure. This eases the computation of the final complexity.

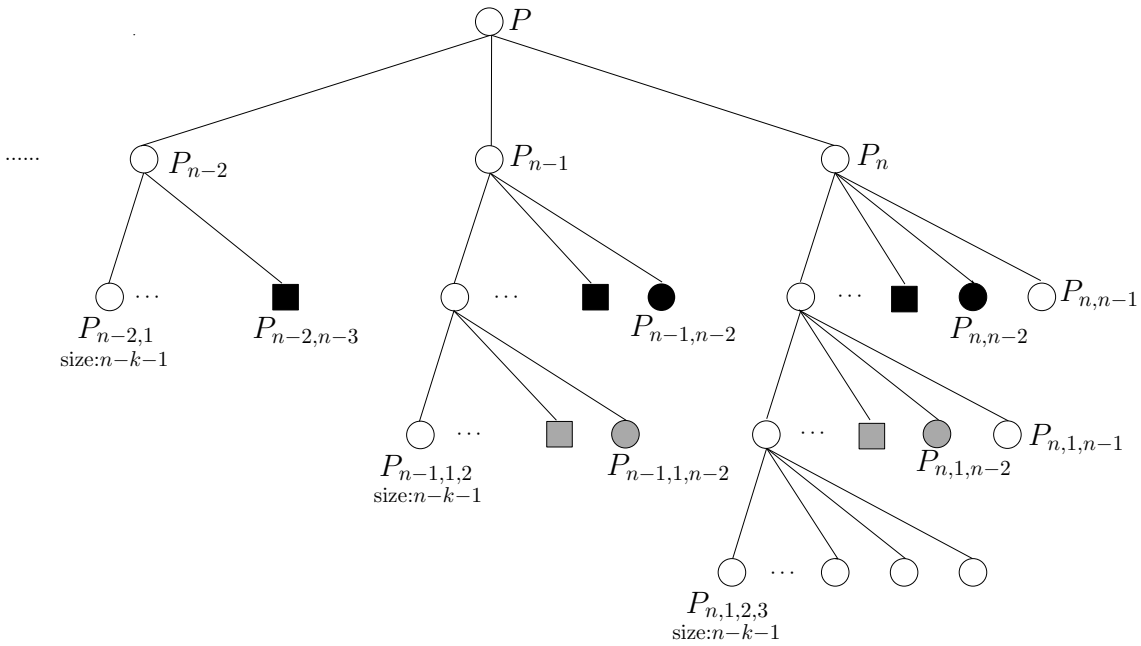
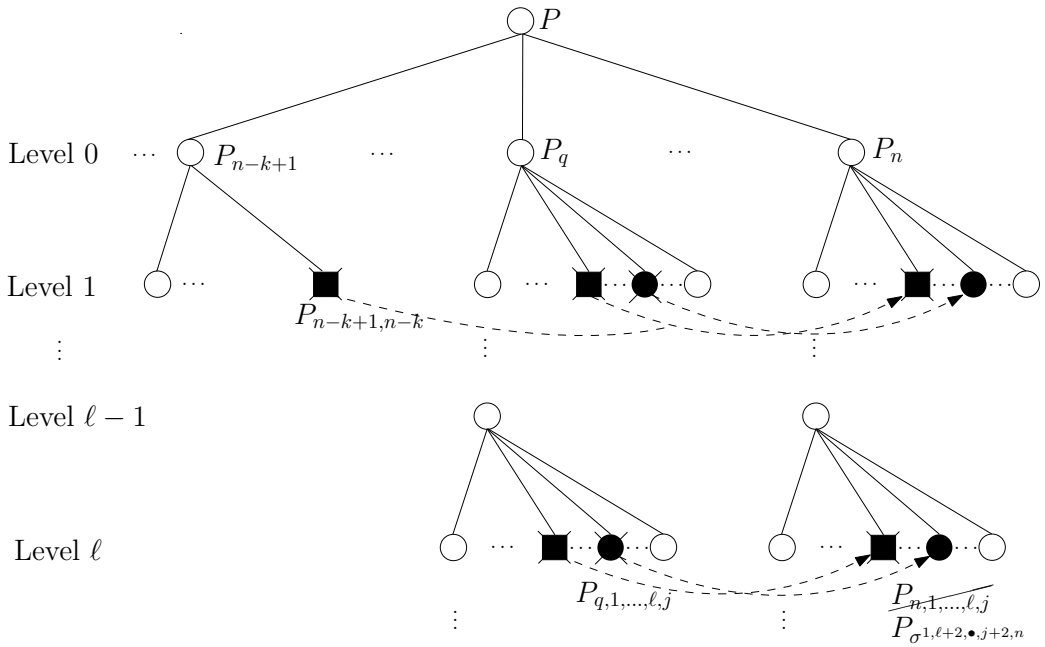



Figure 4.4: An example of right-side branches merging for $k = 3$

More generally, Figure 4.5 shows the right-side search tree and the content of the nodes involved in the merging in a generic way.




$P_{q,1,...,\ell,j} : (2,\ldots,\ell{+}1)\{\ell{+}3,...,j{+}1\}(\ell{+}2)\{j{+}2,...,q\}1\{q{+}1,...,n\}$
$P_{\sigma^{1,\ell+2,\bullet,j+2,n}} : (2,\ldots,\ell{+}1)\{\ell{+}3,...,j{+}1\}\underset{\max\{j+1,n-k+\ell+1\}\leq q\leq n}{\texttt{BEST}}((\ell{+}2)\{j{+}2,...,q\}1\{q{+}1,...,n\})$

Figure 4.5: Generic right-side merging at the root node

The rest of this section intends to describe the merging by following the same lines as for left merging. We first extend the notation $P_\sigma$ in the sense that $\sigma$ may now contain placeholders. The $i$-th element of $\sigma$ is either the position assigned to job $i$ if $i$ is fixed, or $\bullet$ if job $i$ is not yet fixed. The $\bullet$ sign is used as placeholder, with its cardinality below indicating the number of consecutive $\bullet$. As an example, the problem $\{2,\ldots,n-1\}1n$ can now be denoted by $P_{n-1,\underset{n-2}{\bullet},n}$. The cardinality of $\bullet$ may be omitted whenever it is not important for the presentation or it can be easily deduced as in the above example. Note that this adapted notation eases the presentation of right merge while it has no impact on the validity of results stated in the previous section.

**Proposition 6.** *Let $P_\sigma$ be a problem to branch on. Let $j^*, \ell_b, \ell_e, \rho_1$ and $\rho_2$ be defined as in Proposition 4. Extending Corollary 1, problem $P_\sigma$ has the following structure:*

$$\pi\{j^*, \ldots, j^* + \ell_e - \ell_b\}\gamma\Omega'$$

*where $\pi$ is defined as in Corollary 1 and $\gamma$ is the sequence of jobs on positions $\rho_2, \ldots, \rho_3$ with $\rho_3 = \max\{i : i \geq \rho_2, \text{positions } \rho_2, \ldots, i \text{ are in } \sigma\}$ and $\Omega'$ the schedule on all positions after $\rho_3$. The merging procedure is applied on jobset $\{j^*, \ldots, j^* + \ell_e - \ell_b\}$ preceded by a sequence of jobs $\pi$ and followed by $\gamma\Omega'$.*

*Proof.* The problem structure stated in Corollary 1 is refined on the part of $\Omega$. $\Omega$ is split into two parts: $\gamma$ and $\Omega'$. The motivation is that $\gamma$ will be involved in the right merging, just like the role of $\pi$ in left merging. □

Proposition 7 makes links between nodes that can be merged in the right part of the search tree of the root node.

**Proposition 7.** *For each problem in the set*

$$\mathcal{S}_{\ell,j} = \left\{ P_\sigma : \begin{array}{l} |\sigma| = \ell+2, \\ \max\{j+1, n-k+\ell+1\} \leq \sigma_1 \leq n, \\ \sigma_i = i-1, \ \forall i \in \{2, \ldots, \ell+1\}, \\ \sigma_{\ell+2} = j \end{array} \right\}^{1}$$

*with $0 \leq \ell \leq k-1$, $n-k \leq j \leq n-1$, and with $\sigma_i$ referring to the position of job $i$ in $\sigma$, we have the two following properties:*

1. *The solution of problems in $\mathcal{S}_{\ell,j}$ involves the solution of a common subproblem which consists in scheduling jobset $\{\ell+3, ..., j+1\}$ starting at time $t_\ell = \sum_{i=2}^{\ell+1} p_i$.*

2. *For any problem in $\mathcal{S}_{\ell,j}$, at most $k+1$ jobs have to be scheduled after jobset $\{\ell+3, ..., j+1\}$.*

[1] Placeholders do not count in the cardinality of $\sigma$

*Proof.* As each problem $P_\sigma$ is defined by $(2,\ldots,\ell{+}1)\{\ell{+}3,...,j{+}1\}(\ell{+}2)$ $\{j{+}2,...,\sigma_1\}1\{\sigma_1{+}1,...,n\}$, the first part of the property is straightforward. Besides, the second part can be simply established by counting the number of jobs to be scheduled after jobset $\{\ell{+}3,...,j{+}1\}$ when $j$ is minimal, *i.e.* when $j{=}n{-}k$. In this case, $(\ell{+}2)\{j{+}2,...,\sigma_1\}1\{\sigma_1{+}1,...,n\}$ contains $k{+}1$ jobs. □

The above proposition highlights the fact that some nodes can be merged as soon as they share the same initial subproblem to be solved. More precisely, at most $k{-}\ell{-}1$ nodes associated with problems $P_{q,1..\ell,j}$, $\max\{j{+}1, n{-}k{+}\ell{+}1\} \leq q \leq (n{-}1)$, can be merged with the node associated with problem $P_{n,1..\ell,j}$, $\forall j = (n{-}k),...,(n{-}1)$. The node $P_{n,1..\ell,j}$ is replaced in the search tree by the node $P_{\sigma^{1,\ell+2,\bullet,j+2,n}}$ defined as follows (Figure 4.5):

- Jobset $\{\ell+3,...,j+1\}$ is the set of jobs on which it remains to branch on.
- Let $\sigma^{1,\ell+2,\bullet,j+2,n}$ be the sequence containing positions of jobs $\{1,\ldots,\ell+2, j+2,\ldots,n\}$ and placeholders for the other jobs, that leads to the best jobs permutation among $(\ell{+}2)\{j{+}2,...,q\}1\{q{+}1,...,n\}$, $\max\{j{+}1, n{-}k{+}\ell{+}1\} \leq q \leq n$. This involves the solutions of at most $k$ problems of size at most $k+1$ (in $\mathcal{O}^*(k \times 2.4143^{k+1})$ time by TTBR2) and the finding of the sequence that has the smallest total tardiness value knowing that all these sequences are scheduled at time point $t$ which is the sum of processing time of jobs in $(2,\ldots,\ell+1)\{\ell+3,...,j+1\}$.

The merging process described above is applied from the root node, while an analogous merging can be applied from any node of the tree. With respect to the root node, the only additional consideration is that the right-side branches of a general node may have already been modified by previous mergings. As an example, let us consider Figure 4.6. It shows that, subsequently to the merging operations performed from $P$, the right-side branches of $P_n$ may not be the subproblems induced by the branching scheme. However, it can be shown in a similar way as per left-merge, that the merging can still be applied.

In order to define the branching scheme used with the `RIGHT_MERGE` procedure, a data structure $\mathcal{R}_\sigma$ is associated with a problem $P_\sigma$. It represents a list of subproblems that result from a previous merging and are now the $k$ right-side child nodes of $P_\sigma$.

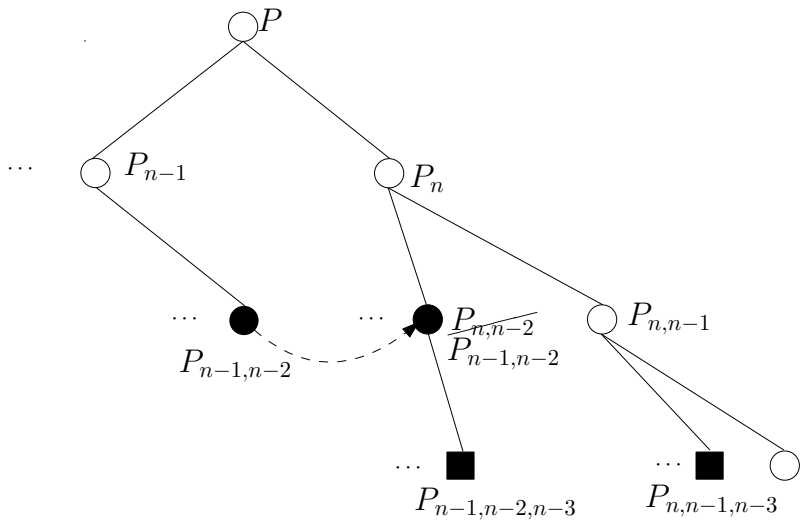



Figure 4.6: The right branches of $P_n$ have been modified when performing right-merging from $P$

When a merging operation sets the $k$ right-side child nodes of $P_\sigma$ to $P_{\sigma^{n-k+1}}, ..., P_{\sigma^n}$, we set $\mathcal{R}_\sigma = \{P_{\sigma^{n-k+1}}, ..., P_{\sigma^n}\}$, otherwise we have $\mathcal{R}_\sigma = \emptyset$. As a conclusion, the following branching scheme for a generic node of the tree is defined. It is an extension of the branching scheme defined in Definition 1.

**Definition 2.** *The branching scheme for a generic node $P_\sigma$ is defined as follows:*

- *If $\mathcal{R}_\sigma = \emptyset$, use the branching scheme defined in Definition 1;*
- *If $\mathcal{L}_\sigma = \emptyset$ and $\mathcal{R}_\sigma \neq \emptyset$, branch on the longest job in the available positions from the $1st$ to the $(n-k)$-th, then extract problems from $\mathcal{R}_\sigma$ as the last $k$ branches.*
- *If $\mathcal{L}_\sigma \neq \emptyset$ and $\mathcal{R}_\sigma \neq \emptyset$, extract problems from $\mathcal{L}_\sigma$ as the first $k-1$ branches, then branch on the longest job in the available positions from the $k$-th to the $n-k$-th, finally extract problems from $\mathcal{R}_\sigma$ as the last $k$ branches.*

*This branching scheme, whenever necessary, will be referred to as improved branching. It also replaces the one introduced in Definition 1*

Proposition 8 states the validity of merging on a general node, which extends the result in Proposition 7.

**Proposition 8.** *Let $P_\sigma$ be a generic problem and let $\pi, j^*, \ell_b, \ell_e, \gamma, \Omega'$ be computed relatively to $P_\sigma$ according to Proposition 6. If $\mathcal{R}_\sigma=\emptyset$, the right merging on $P_\sigma$ can be easily performed by considering $P_\sigma$ as a new root problem. Suppose $\mathcal{R}_\sigma\neq\emptyset$, the $q$-th child node $P_{\sigma^q}$ is extracted from $\mathcal{R}_\sigma$, $\forall n'-k+1\leq q\leq n'$, where $n'=\ell_e-\ell_b+1$ is number of child nodes of $P_\sigma$. The structure of $P_{\sigma^q}$ is $\pi\{j^*+1,...,j^*+q-1\}\gamma^q\Omega'$.*

*For $0\leq\ell\leq k-1$ and $n'-k\leq j\leq n'-1$, the following conditions hold:*

1. *Problems in $\mathcal{S}^\sigma_{\ell,j}$ have the following structure: $\pi(j^*+1,\ldots,j^*+\ell)\{j^*+\ell+2,...,j^*+j\}(j^*+\ell+1)\{j^*+j+1,...,j^*+q-1\}\gamma^q\Omega'$ with $q$ varies from $\max\{j+1, n-k+\ell+1\}$ to $n'$.*

2. *The solution of all problems in $\mathcal{S}^\sigma_{\ell,j}$ involves the scheduling of a jobset $\{j^*+j+1,...,j^*+q-1\}$, $\max\{j+1,n-k+\ell+1\}\leq q\leq n'$, which is of size less than $k$. Besides to solve problems in $\mathcal{S}^\sigma_{\ell,j}$ it is required to solve a common subproblem made of jobset $\{j^*+\ell+2,...,j^*+j\}$ starting after $\pi(j^*+1,\ldots,j^*+\ell)$ and before $(j^*+\ell+1)\{j^*+j+1,...,j^*+q-1\}\gamma^q\Omega'$.*

*Proof.* The proof is similar to the one of Proposition 5. The first part of the statement follows directly from Definition 2 and simply defines the structure of the child nodes of $P_\sigma$. For the second part, it is necessary to prove that $\{j^*+j+1,...,j^*+q-1\}\gamma^q$ consists of the same jobs for any valid value of $q$. Actually, since right-merging only merges nodes that have common jobs fixed after the unscheduled jobs, the jobs concerned in $\{j^*+j+1,...,j^*+q-1\}\gamma^q$ must be the same as in $\{j^*+j+1,...,j^*+q-1\}j^*\{j^*+q,...,j^*+n'-1\}\gamma$, $\max\{j+1,n-k+\ell+1\}\leq q\leq n'$, which proves the statement. □

Analogously to the root node, given the values of $\ell$ and $j$, all the problems in $\mathcal{S}^\sigma_{\ell,j}$ can be merged. More precisely, we rewrite $\sigma$ as $\alpha\underset{n'}{\bullet}\beta$ where $\alpha$ is the sequence of positions assigned to jobs $\{1,\ldots,j^*-1\}$, $\underset{n'}{\bullet}$ refers to the jobset to branch on and $\beta$ contains the positions assigned to the rest of jobs. At most $k-\ell-1$ nodes associated with problems $P_{\alpha,\ell_b+q-1,\ell_b..\ell_b+\ell-1,\ell_b+j-1,\bullet,\beta}$, with $\max\{j+1,n'-k+\ell+1\}\leq q\leq n'-1$, can be merged with the node associated with problem $P_{\alpha,\ell_e,\ell_b..\ell_b+\ell-1,\ell_b+j-1,\bullet,\beta}$. The node $P_{\alpha,\ell_e,\ell_b..\ell_b+\ell-1,\ell_b+j-1,\bullet,\beta}$ is replaced in the search tree by the node $P_{\alpha,\sigma^{\ell,\ell_b,j},\bullet,\beta}$ defined as follows:

- Jobset $\{j^*+\ell+2, ..., j^*+j\}$ is the set of jobs on which it remains to branch on.
- Let $\sigma^{\ell,\ell_b,j}$ be the sequence of positions among

$$\{(\ell_b+q-1, \ell_b..\ell_b+\ell-1, \ell_b+j-1) : \max\{j+1, n'-k+\ell+1\} \leq q \leq n'-1\}$$

  associated with the best job permutation on $(j^*+\ell+1)\{j^*+j+1, ..., j^*+q-1\}\gamma^q$, $\forall \max\{j+1, n'-k+\ell+1\} \leq q \leq n'$. This involves the solutions of $k$ problems of size at most $k+1$ (in $\mathcal{O}^*(k \times 2.4143^{k+1})$ time by TTBR2) and the finding of the sequence that has the smallest total tardiness value knowing that all these sequences are scheduled at time point $t$ which is the sum of processing time of jobs in $\pi(j^*+1, \ldots, j^*+\ell)\{j^*+\ell+2, ..., j^*+j\}$.

The `RIGHT_MERGE` procedure is presented in Algorithm 7. Notice that, similarly to the `LEFT_MERGE` procedure, this algorithm takes as input one problem $P_\sigma$ and provides as an output a set of nodes to branch on, which replaces all its $k$ right-side child nodes of $P_\sigma$. It is interesting to notice that the `LEFT_MERGE` procedure is also integrated.

A procedure `MERGE_RIGHT_NODES` is invoked to perform the right merging for each level $\ell = 0, ..., k-1$ in a recursive way. The initial inputs of this procedure (line 13 in `RIGHT_MERGE`) are the problem $P_\sigma$ and the list of its $k$ right-side child nodes, denoted by *rnodes*. They are created according to the improved branching (lines 4-12 of Algorithm 7). Besides, the output is a list $Q$ containing problems to branch on after merging. In the first call to `MERGE_RIGHT_NODES`, the left merging is applied to the first element of *rnodes* (line 2), all the child nodes of nodes in *rnodes* not involved in right nor left merging, are added to $Q$ (lines 3-7). This is also the case for the result of the right merging operations at the current level (lines 8-11). In Algorithm 8, the value of $r$ indicates the current size of *rnodes*. It is reduced by one at each recursive call and the value $(k-r)$ identifies the current level with respect to $P_\sigma$. As a consequence, each right merging operation consists in finding the problem with the best total tardiness value on its fixed part, among the ones in the set $\mathcal{S}^\sigma_{k-r,j}$. This is performed by the `BEST` function (line 10 of `MERGE_RIGHT_NODES`) which extends the one called in Algorithm 6 by taking at most $k$ subproblems as input and returning the dominating one.

Tthe MERGE_RIGHT_NODES procedure is then called recursively on the list containing the first child node of the 2nd to $r$-th node in *rnodes* (lines 13-17). Note that the procedure LEFT_MERGE is applied on every node in *rnodes* except the last one. In fact, for any specific level, the last node in *rnodes* belongs to the last branch of $P_\sigma$, which is $P_{\sigma,l_b+n-1,\bullet,\beta}$. Since $P_{\sigma,l_b+n-1,\bullet,\beta}$ is put into $Q$ at line 14 of RIGHT_MERGE, it means that this node will be re-processed later and LEFT_MERGE will be called on it at that moment. Since the recursive call of MERGE_RIGHT_NODES (line 18) will merge some nodes to the right-side child nodes of $P_{\alpha,\ell_b,\underset{n_r-1}{\bullet},\beta^r}$, the latter one must be added to the list $\mathcal{L}$ of $P_{\alpha,\underset{n_r}{\bullet},\beta^r}$ (line 19). In addition, since we defined $\mathcal{L}$ as a list of size either 0 or $k-1$, lines 20-24 add the other $(k-2)$ nodes to $\mathcal{L}_{\alpha,\underset{n_r}{\bullet},\beta^r}$.

It is also important to notice the fact that a node may have its $\mathcal{L}$ or $\mathcal{R}$ structures non-empty, if and only if it is the first or last child node of its parent node. A direct result is that only one node among those involved in a merging may have its $\mathcal{L}$ or $\mathcal{R}$ non-empty. In this case, these structures need to be associated with the resulting node. The reader can always refer to Figure 4.4 for a more intuitive representation.

**Lemma 3.** *The RIGHT_MERGE procedure returns a list of $\mathcal{O}(n)$ nodes in polynomial time and space.*
*The solution of the associated problems involves the solution of* 1 *subproblem of size* $(n-1)$*, of* $(k-1)$ *subproblems of size* $(n-k-1)$*, and subproblems of size* $i$ *and* $n_q-(k-r)-i-1$*,* $\forall r=2,...,k; q=1,...,r-1; i=k,...,n-k-(k-r)-2$*.*

*Proof.* The first part of the result follows directly from Algorithm 7. The only lines where nodes are added to $Q$ in RIGHT_MERGE are lines 13-14. In line 14, only one problem is added to $Q$, thus it needs to be proved that the call on MERGE_RIGHT_NODES (line 13) returns $\mathcal{O}(n)$ nodes. This can be computed by analysing the lines 2-7 of Algorithm 8. Considering all recursive calls, the total number of nodes returned by MERGE_RIGHT_NODES is $(\sum_{i=1}^{k-1}(k-i)(n-2k-i))+k-1$ which yields $\mathcal{O}(n)$. The number of all the nodes considered in right merging is bounded by a linear function on $n$. Furthermore, all the operations associated with the nodes (merging, creation, etc) have a polynomial cost. As a consequence, Algorithm 7 runs in polynomial time and space.

Regarding the sizes of the subproblems returned by RIGHT_MERGE, the node

**Algorithm 7** `RIGHT_MERGE` Procedure

**Input:** $P_\sigma = P_{\alpha,\underset{n}{\bullet},\beta}$ a problem of size $n$, with $\ell_b$, $j^*$ computed according to Proposition 4

**Output:** $Q$ : a list of problems to branch on after merging

1: **function** `RIGHT_MERGE`($P_\sigma$)
2: $Q \leftarrow \emptyset$
3: $nodes \leftarrow \emptyset$
4: **if** $\mathcal{R}_\sigma = \emptyset$ **then**
5: **for** $q = n-k+1$ to $n$ **do**
6: Create $P_{\alpha,\ell_b+q-1,\bullet,\beta}$ by branching
7: $\delta \leftarrow$ the sequence of positions of jobs $\{j^*+q, \ldots, j^*+n-1\}$ fixed by TTBR2
8: $nodes \leftarrow nodes + P_{\alpha,\ell_b+q-1,\bullet,\delta,\beta}$
9: **end for**
10: **else**
11: $nodes \leftarrow \mathcal{R}_\sigma$
12: **end if**
13: $Q \leftarrow Q \cup$ `MERGE_RIGHT_NODES`($nodes, P_\sigma$)
14: $Q \leftarrow Q \cup nodes[k]$ ▷ The last node will be re-processed
15: **return** $Q$
16: **end function**

**Algorithm 8** `MERGE_RIGHT_NODES` Procedure

**Input:** $rnodes = [P_{\alpha,\underset{n_1}{\bullet},\beta^1}, \ldots, P_{\alpha,\underset{n_r}{\bullet},\beta^r}]$, ordered list of $r$ last child nodes with $\ell_b$ defined on any node in $rnodes$. $|\alpha|+1$ is the job to branch on and $n_r = n_1+r-1$.
**Output:** $Q$, a list of problems to branch on after merging

1: **function** `MERGE_RIGHT_NODES`$(rnodes, P_\sigma)$
2: $Q \leftarrow$ `LEFT_MERGE`$(P_{\alpha,\underset{n_1}{\bullet},\beta^1})$
3: **for** $q = 1$ to $r-1$ **do**
4: **for** $j = \ell_b + k$ to $\ell_b + n_1 - 1$ **do**
5: $Q \leftarrow Q \cup P_{\alpha,j,\underset{n_q-1}{\bullet},\beta^q}$
6: **end for**
7: **end for**
8: **for** $j = \ell_b + n_1$ to $\ell_b + n_r$ **do**
9: Solve all the subproblems of size less than $k$ in $\mathcal{S}^\sigma_{k-r,j}$
10: $\mathcal{R}_{\alpha,\underset{n_r}{\bullet},\beta^r} \leftarrow \mathcal{R}_{\alpha,\underset{n_r}{\bullet},\beta^r} +$ `BEST`$(\mathcal{S}^\sigma_{k-r,j})$
11: **end for**
12: **if** $r > 2$ **then**
13: $newnodes \leftarrow \emptyset$
14: **for** $q = 2$ to $r-1$ **do**
15: $newnodes \leftarrow newnodes +$ `LEFT_MERGE`$(P_{\alpha,\underset{n_q}{\bullet},\beta^q})$
16: **end for**
17: $newnodes \leftarrow newnodes + P_{\alpha,\ell_b,\underset{n_r-1}{\bullet},\beta^r}$
18: $Q \leftarrow Q \cup$ `MERGE_RIGHT_NODES`$(newnodes, P_\sigma)$
19: $\mathcal{L}_{\alpha,\underset{n_r}{\bullet},\beta^r} \leftarrow P_{\alpha,\ell_b,\underset{n_r-1}{\bullet},\beta^r}$
20: **for** $q = 2$ to $k-1$ **do**
21: Create $P_{\alpha,\ell_b+q-1,\underset{n_r-1}{\bullet},\beta^r}$ by branching
22: $\delta \leftarrow$ the sequence of positions of jobs $\{|\alpha|+2, \ldots, |\alpha|+q\}$ fixed by TTBR2
23: $\mathcal{L}_{\alpha,\underset{n_r}{\bullet},\beta^r} \leftarrow \mathcal{L}_{\alpha,\underset{n_r}{\bullet},\beta^r} + P_{\alpha,\ell_b+q-1,\delta,\underset{n_r-1}{\bullet},\beta^r}$
24: **end for**
25: **end if**
26: **return** $Q$
27: **end function**

added in line 14 of Algorithm 7 contains one subproblem of size $n-1$, corresponding to branching the longest job on the last available position. Then, the problems added by the call to `MERGE_RIGHT_NODES` are added to $Q$. In line 2 of Algorithm 8, the size of the problem returned by `LEFT_MERGE` is one less than the input problem which is of size $n-k-(k-r)$, where $k-r$ is the current level with respect to the original node. As a consequence, the size of the resulting subproblem is $n-k-(k-r)-1$. Note that this line is executed $(k-1)$ times, $\forall r = k, \ldots, 2$, corresponding to the number of calls to `MERGE_RIGHT_NODES`. In line 5 of Algorithm 8, the list of nodes which are not involved in any merging operation are added to $Q$. This corresponds to couples of problems of size $i$ and $n_q-(k-r)-i-1$ $\forall i = k, ..., n-k-1$ and this proves the last part of the lemma. □

**Lemma 4.** *Instances such that $LPT = EDD$ correspond to worst-case instances for which the `RIGHT_MERGE` procedure returns $\mathcal{O}(n)$ nodes to branch on, whose subproblems are listed in Lemma 3, replacing all the $k$ right-side child nodes of its parent node.*

*Proof.* The proof follows similar reasoning as the one in Lemma 2. In general, if $LPT \neq EDD$ then the number of nodes in $\mathcal{S}^{\sigma}_{\ell,j}$ (defined in Proposition 8) could be less, since some nodes may not be created due to Property 2. However, all the nodes inside $\mathcal{S}^{\sigma}_{\ell,j}$ can still be merged to one except when $\mathcal{S}^{\sigma}_{\ell,j}$ is empty. In either case, we can achieve at least the same reduction as the case of $LPT = EDD$. □

### 4.2.3 Complete Algorithm and Analysis

We are now ready to define the main procedure `TTBM` (Total Tardiness Branch and Merge), stated in Algorithm 9 which is called on the initial input problem $P : \{1, ..., n\}$. The algorithm has a similar recursive structure as TTBR1. However, each time a node is opened, the sub-branches required for the merging operations are generated, the subproblems of size less than $k$ are solved and the procedures `LEFT_MERGE` and `RIGHT_MERGE` are called. Then, the algorithm proceeds recursively by extracting the next node from $Q$ with a depth-first strategy and terminates when $Q$ is empty.

**Algorithm 9** Total Tardiness Branch and Merge (TTBM)

**Input:** $P : \{1, ..., n\}$: input problem of size $n$
$\frac{n}{2} > k \geq 2$: an integer constant
**Output:** *seqOpt*: an optimal sequence of jobs

1: **function** TTBM($P$,$k$)
2: $Q \leftarrow P$
3: *seqOpt* $\leftarrow$ a random sequence of jobs
4: **while** $Q \neq \emptyset$ **do**
5: $P^* \leftarrow$ extract next problem from $Q$ (depth-first order)
6: **if** the size of $P^* < 2k$ **then**
7: Solve $P^*$ by calling TTBR2
8: **end if**
9: **if** all jobs $\{1, ..., n\}$ are fixed in $P^*$ **then**
10: *seqCurrent* $\leftarrow$ the solution defined by $P^*$
11: *seqOpt* $\leftarrow$ best solution between *seqOpt* and *seqCurrent*
12: **else**
13: $Q \leftarrow Q \cup$ `LEFT_MERGE`($P^*$) ▷ Left-side nodes
14: **for** $i = k + 1, ..., n - k$ **do**
15: Create the $i$-th child node $P_i$ by branching scheme of TTBR1
16: $Q \leftarrow Q \cup P_i$
17: **end for**
18: $Q \leftarrow$ `RIGHT_MERGE`($P^*$) ▷ Right-side nodes
19: **end if**
20: **end while**
21: **return** *seqOpt*
22: **end function**

Proposition 9 determines the time complexity of the proposed algorithm. In this regard, the complexity of the algorithm depends on the value given to $k$. The higher it is, the more subproblems can be merged and the better is the worst-case time complexity of the approach.

**Proposition 9.** *Algorithm TTBM runs in $\mathcal{O}^*((2+\epsilon)^n)$ time and polynomial space, where $\epsilon \to 0$ for large enough values of $k$.*

*Proof.* The proof is based on the analysis of the number and the size of the subproblems put in $Q$ when a single problem $P^*$ is expanded. As a consequence of Lemma 2 and Lemma 4, TTBM induces the following recursion:

$$\begin{aligned} T(n) =& 2T(n-1) + 2T(n-k-1) + ... + 2T(k) \\ &+ \sum_{r=2}^{k} \sum_{q=1}^{r-1} \sum_{i=k}^{n_1-(k-r)-2} (T(i) + T(n_q - (k-r) - i - 1)) \\ &+ (k-1)T(n_1 - 1) + \mathcal{O}(p(n)) \end{aligned}$$

First, a simple lower bound on the complexity of the algorithm can be derived by the fact that the procedures `RIGHT_MERGE` and `LEFT_MERGE` provide (among the others) two subproblems of size $n-1$, based on which the following inequality holds:

$$T(n) > 2T(n-1) \tag{4.7}$$

By solving the recurrence, we obtain that $T(n) = \omega(2^n)$. As a consequence, the following inequality holds:

$$T(n) > T(n-1) + \ldots + T(1) \tag{4.8}$$

In fact, if it does not hold, we have a contradiction on the fact $T(n) = \omega(2^n)$.

By using Equation 4.8, we can find a bound for the complexity of $T(n)$ as follows:

$$\begin{aligned} T(n) &\leq 2T(n-1) + 2T(n-k-1) + ... + 2T(k) \\ &\quad + \sum_{r=2}^{k} \sum_{q=1}^{r-1} \sum_{i=1}^{n_1-(k-r)-2} 2T(i) + (k-1)T(n_1 - 1) + \mathcal{O}(p(n)) \\ &\leq 2T(n-1) + 2T(n-k-1) + ... + 2T(k) \end{aligned}$$

$$
\begin{aligned}
&+ \sum_{r=2}^{k}(r-1)2T(n_1-(k-r)-1) + (k-1)T(n_1-1) + \mathcal{O}(p(n)) \\
&\leq 2T(n-1) + 2T(n-k-1) + ... + 2T(k) \\
&\quad + (k-1)4T(n_1-1) + (k-1)T(n_1-1) + \mathcal{O}(p(n)) \\
&\leq 2T(n-1) + 4T(n-k-1) + 5(k-1)T(n-k-1) + \mathcal{O}(p(n)) \\
&= 2T(n-1) + (5k-1)T(n-k-1) + \mathcal{O}(p(n))
\end{aligned}
$$

where $\mathcal{O}(p(n))$ includes the cost for creating all nodes for each level and the cost of all the merging operations, performed in constant time.

The recursion $T(n) = 2T(n-1) + (5k-1)T(n-k-1) + \mathcal{O}(p(n))$ is an upper limitation of the running time of TTBM. Recall that its solution is $T(n) = \mathcal{O}^*(c^n)$ where $c$ is the largest root of the function:

$$f_k(x) = 1 - \frac{2}{x} - \frac{5k-1}{x^{k+1}} \tag{4.9}$$

.

As $k$ increases, the function $f_k(x)$ converges to $1-\frac{2}{x}$, which induces a complexity of $\mathcal{O}^*(2^n)$. Table 4.1 shows the time complexity of TTBM obtained by solving Equation 4.9 for all the values of $k$ from 3 to 20. The base of the exponential is computed by solving Equation 4.9 by means of a mathematical solver and rounding up the fourth digit of the solution. The table shows that the time complexity is $\mathcal{O}^*(2.0001^n)$ for $k \geq 20$. □

| $k$ | $T(n)$ |
|---|---|
| 3 | $\mathcal{O}^*(2.5814^n)$ |
| 4 | $\mathcal{O}^*(2.4302^n)$ |
| 5 | $\mathcal{O}^*(2.3065^n)$ |
| 6 | $\mathcal{O}^*(2.2129^n)$ |
| 7 | $\mathcal{O}^*(2.1441^n)$ |
| 8 | $\mathcal{O}^*(2.0945^n)$ |
| 9 | $\mathcal{O}^*(2.0600^n)$ |
| 10 | $\mathcal{O}^*(2.0367^n)$ |
| 11 | $\mathcal{O}^*(2.0217^n)$ |
| 12 | $\mathcal{O}^*(2.0125^n)$ |
| 13 | $\mathcal{O}^*(2.0070^n)$ |
| 14 | $\mathcal{O}^*(2.0039^n)$ |
| 15 | $\mathcal{O}^*(2.0022^n)$ |
| 16 | $\mathcal{O}^*(2.0012^n)$ |
| 17 | $\mathcal{O}^*(2.0007^n)$ |
| 18 | $\mathcal{O}^*(2.0004^n)$ |
| 19 | $\mathcal{O}^*(2.0002^n)$ |
| 20 | $\mathcal{O}^*(2.0001^n)$ |

Table 4.1: The time complexity of TTBM for values of $k$ from 3 to 20

## 4.3 Conclusions

This chapter focused on the design of exact branching algorithms for the single machine total tardiness problem. By exploiting some inherent properties of the problem, we first proposed two branch and reduce algorithms, indicated with TTBR1 and TTBR2. The former runs in $\mathcal{O}^*(3^n)$, while the latter achieves a better time complexity in $\mathcal{O}^*(2.4143^n)$. The space requirement is polynomial in both cases. Furthermore, a technique called branch and merge, is presented and applied onto TTBR1 in order to improve its performance. The final achievement is a new algorithm (TTBM) with time complexity converging to $\mathcal{O}^*(2^n)$ and polynomial space. The same technique can be tediously adapted to improve the performance of TTBR2, but the resulting algorithm achieves the same asymptotic time complexity as TTBM, and thus it was omitted. To the best of authors' knowledge, TTBM is the polynomial space algorithm that has the best worst-case time complexity for

solving this problem.

Beyond the new established complexity results, the main contribution of the work is the branch and merge technique. The basic idea is very simple, and it consists of speeding up branching algorithms by avoiding to solve identical problems. The same goal is traditionally pursued by means of *Memorization* [46], where the solution of already solved subproblems are stored and then queried when an identical subproblem appears. This is at the cost of exponential space. In contrast, branch and merge discards identical subproblems but by appropriately merging, in polynomial time and space, nodes involving the solution of common subproblems. When applied systematically in the search tree, this technique enables to achieve a good worst-case time bound. On a computational side, it is interesting to notice that node merging can be relaxed to avoid solving in $\mathcal{O}^*(2.4143^k)$, with $k$ fixed, subproblems at merged nodes. This relaxation can be reduced to the comparison of active nodes with already branched nodes with the requirement of keeping use of a polynomial space. This can also be seen as memorization but with a fixed size memory used to store already explored nodes. This leads to the lost of a reduced worst-case time bound but early works [90] have shown that this can lead to substantially good practical results, at least on some scheduling problems.

As a future development of this work, our aim is twofold. First, we aim at applying the branch and merge algorithm to other combinatorial optimization problems in order to establish its potential generalization to other problems. Second, we want to explore the pratical efficiency of this algorithm on the single machine total tardiness problem and compare it with relaxed implementation where a node comparison procedure is implemented with a fixed memory space used to store already branched nodes, in a similar way than in [90].

# Part II

# Clustering Problems

# Chapter 5

# The Max-Mean Dispersion Problem

Semidefinite Programming was introduced in Section 1.2. As previously discussed, it has been adopted to design exact, heuristic and approximation algorithms for several combinatorial optimization problems, especially for those that have a quadratic formulation. Matheuristics were presented in Section 1.1. Originally, they were conceived for LP - ILP models, for which very efficient mathematical solvers are available. Recently, there has been a significant evolution in the performances on solving non-linear programming (NLP) models, in particular in specific case of the quadratic - quadratic integer programming (QP - QIP). As we will show in this chapter, they are a mature-enough instrument to design matheuristics.

In this chapter, we tackle the Max-Mean Dispersion Problem ($Max-MeanDP$), a fractional combinatorial optimization problem that received some attention in the last years. The algorithmic contribute is twofold. On the one hand, we propose an efficient matheuristic approach based on a quadratic formulation of the problem. On the other hand, we provide an exact method for the problem: an ad hoc branch and bound algorithm based on a semidefinite programming relaxation of the problem.

$Max-MeanDP$ belongs to a general category of clustering problems whose aim is to find a subset $M$ of a set $N$ which maximizes a measure of dispersion/similarity of the elements in $M$. More formally, suppose $N$ is a set of elements with cardinality $n$, and $D$ a matrix whose components $d_{i,j}$ (that may be positive, negative or null)

indicate distance/proximity between item $i \in N$ and $j \in N$. We assume that the matrix $D$ is symmetric, namely $d_{i,j} = d_{j,i}\ \forall i, j \in N$, where the values on the diagonal are equal to 0 ($d_{i,i} = 0, \forall i \in N$). When the measure of dispersion/similarity of the elements in $M$ is the sum of the $d_{i,j}$s between elements $i, j \in M$ (that is $\sum_{i,j \in M} d_{i,j}$) and the cardinality of subset $M$ is given a priori ($|M| = m$ with $m$ predefined), then we have the Maximum Diversity Problem [54, 51], which is known to be strongly NP-Hard. The Maximum Diversity Problem has been referred to with several names, which have been carefully collected in [75]. Recently, it has also been called $k - cluster$ problem [73]. Hereafter, we will refer to it as $Max - SumDP$. When the measure of dispersion/similarity of the elements in $M$ is the minimum of the $d_{i,j}$s between elements $i, j \in M$ (that is $\min_{i,j \in M} d_{i,j}$) and the cardinality of subset $M$ is given a priori, then we have the $Max - MinDP$ [31, 83], which is also known to be strongly NP-Hard. Finally, when the measure of dispersion/similarity of the elements in $M$ is the average of the distances between elements $i, j \in M$ (that is $\frac{\sum_{i,j \in M} d_{i,j}}{|M|}$), but the cardinality of subset $M$ is not given a priori, we have the $Max - MeanDP$ which is the object of this work.

This problem has a real importance in fields like architectural space planning and analysis of social networks (as claimed in [76]). Another real-world application is about web pages rank (see [62]). A different application is also shown in [84], where authors aim at selecting individuals with different abilities, in order to determine very productive non-homogeneous work teams. In such domains $d_{i,j}$s can violate the triangular inequality $d_{i,j} \leq d_{i,k} + d_{k,j} \forall i, k, j \in N$ and the non negativity condition $d_{i,j} \geq 0 \forall i, j \in N$. Finally, the considered problem can be of interest for communities mining (see [100]) where relationships between individuals and/or elements can be either positive or negative, such as like-dislike, trust-distrust. This can be useful for market surveys, pattern recognition and social network analysis and more generally when studies have to be conducted in order to derive specific (balanced) characteristic of subset of elements in a larger community. In this context, it is a common approach to tackle clustering problems with the aim of maximizing a measure of the average diversity among individuals or elements where it is important also to take into account the cluster size.

To the authors' knowledge, the state of the art literature on $Max - MeanDP$ is quite limited. In [81], $Max - MeanDP$ is shown to be strongly NP-Hard whenever

$d_{i,j}$s can take both positive and negative values. Those authors have also presented a mixed integer non-linear programming (MINLP) formulation and an equivalent ILP formulation. In [76], a randomized GRASP with path relinking is proposed for $Max-MeanDP$. The presented computational experiments dealt with a set of real world instances from a social network application.

Section 5.1 discusses the mathematical programming formulations of the $Max-MeanDP$. Section 5.2 describes the semidefinite relaxation used as a bounding method and the main features that characterize the proposed branch and bound. The matheuristic is described in Section 5.3. It is a three-phase hybrid heuristic procedure whose first phase repeatedly solves a QIP formulation of $Max-SumDP$ in order to determine an initial solutions set. The following phases enhance the quality of the initial solutions set by means of a local branching scheme and a path relinking procedure respectively. Section 5.4 provides a computational analysis that validates the efficiency of the proposed algorithms.

## 5.1 Mathematical formulations

$Max-MeanDP$ has a straightforward non-linear and fractional formulation, as introduced in [81]. Define vector $\mathbf{x} \in \{0,1\}^n$ where, for each component $x_i$, we have $x_i = 1$ if and only if element $i$ is included in the subset $M$, otherwise 0. The formulation below follows directly from the definition of $Max-MeanDP$:

$Max-MeanDP$ **Standard Formulation 1.**

$$\max \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{i,j}x_ix_j}{\sum_{i=1}^{n} x_i} \tag{5.1}$$

*subject to:*

$$\sum_{i=1}^{n} x_i \geq 2 \tag{5.2}$$

$$x_i \in \{0,1\} \forall i \in N \tag{5.3}$$

Since $D$ is symmetric, it can be written in the following vectorial form:

$Max - MeanDP$ **Standard Formulation 2.**

$$\max \frac{\boldsymbol{x}^t \boldsymbol{D} \boldsymbol{x}}{2\boldsymbol{u}^t \boldsymbol{x}} \tag{5.4}$$

*subject to:*

$$\boldsymbol{u}^t \boldsymbol{x} \geq 2 \tag{5.5}$$

$$\boldsymbol{x} \in \{0,1\}^n \tag{5.6}$$

where $\mathbf{u}^T = \overbrace{(1,1,...,1)}^{n}$, and, for convenience, the problem is converted into a minimization problem where the sign of the objective is changed. The following proposition indicates that if the integrality constraints are relaxed, then the resulting mathematical program is not convex if no other assumptions are given on the matrix $D$.

**Proposition 10.** *The function* $f(\boldsymbol{x}) : \Gamma \to \mathbb{R}$*:*

$$f(\boldsymbol{x}) = -\frac{\frac{1}{2}\boldsymbol{x}^T D \boldsymbol{x}}{\boldsymbol{u}^T \boldsymbol{x}} \tag{5.7}$$

*where* $\Gamma = \{\boldsymbol{x} \in [0,1]^n : \boldsymbol{u}^T \boldsymbol{x} \geq 2\}$*, is convex if and only if* $D \preceq 0$*.*

*Proof.* Set $\mathbf{u}^T \mathbf{x} = y$. The above function can be written as:

$$f(\mathbf{x}, y) = -\frac{\frac{1}{2}\mathbf{x}^T D \mathbf{x}}{y}$$

over the domain $\Gamma' = \{\mathbf{x} \in [0,1]^n, y \in \mathbb{R} : y = \mathbf{u}^T \mathbf{x} \geq 2\}$

The Hessian $\nabla^2 f(\mathbf{x}, y)$ is equal to:

$$\nabla^2 f(\mathbf{x}, y) = -\frac{1}{y^3} \begin{bmatrix} Dy^2 & -D\mathbf{x}y \\ -(D\mathbf{x})^T y & \mathbf{x}^t D \mathbf{x} \end{bmatrix} = -\frac{1}{y^3} D \begin{bmatrix} y \\ -\mathbf{x} \end{bmatrix} \begin{bmatrix} y \\ -\mathbf{x} \end{bmatrix}^T$$

Function $f(\mathbf{x}, y)$ is defined on the convex set $\Gamma'$ and is convex if and only if $\nabla^2 f(\mathbf{x}, y) \succeq 0$, thus if $D \preceq 0$.

□

With real world application data such as those considered in [76], the condition $D \preceq 0$ typically does not hold, hence the problem given by the continuous relaxation of the standard fractional formulation of the problem is not convex in the general case. NLP solvers can clearly be applied to this formulation (here we used XPRESS-SLP by Fair-Isaac) even though just local maxima can be guaranteed.

In addition, the following straightforward QIP model holds for $Max-SumDP$:

$Max-SumDP$ **QIP Formulation 1.**

$$\max \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{i,j} x_i x_j \tag{5.8}$$

*subject to:*

$$\sum_{i=1}^{n} x_i = m \tag{5.9}$$

$$x_i \in \{0,1\} \ \forall i \in N \tag{5.10}$$

Even if general convexity results do not hold also for the continuous relaxation of this QIP, we note that for $Max-SumDP$, the formulation corresponds to a 0/1 quadratic knapsack problem (with equality constraint) that has been much more tackled in the literature (see, e.g. [79]) and can be efficiently tackled by means of QIP solvers, such as, for instance, CPLEX. In [76], computational experiments dealt only with ILP formulations of $Max-MeanDP$ and $Max-SumDP$ showing that the iterative solution of $Max-SumDP$ was superior to the one-shot solution of $Max-MeanDP$. In our preliminary tests on $Max-SumDP$, we determined that CPLEX 12.5 was more efficient when applied to QIP Formulation 1 rather than to its standard linearization indicated in [76]. In fact, the latter required a computational time which is higher by more than an order of magnitude on instances with $n = 35$. The purpose of this work is to embed the repeated solution of the QIP formulation of $Max-SumDP$ into a heuristic framework for $Max-MeanDP$.

As a conclusion, we consider the solution approach based on the iterative solution of the QIP formulation to validate our branch and bound in the computational experiments. The pseudocode of the method, henceforth indicated by IMA, follows:

**Algorithm 10** Iterative $Max - SumDP$ Approach (IMA)

**Input:** $Max - MeanDP$ instance
$i \leftarrow 2$
**while** $i \leq n$ **do**
  Solve QIP Formulation 1 and obtain $\mathbf{x}^{(i)}$
  **if** $i = 2$ **or** $\frac{f(\mathbf{x}^{(i)})}{i} > \frac{f(\mathbf{x}^{(i*)})}{i*}$ **then**
    $\mathbf{x}^{(i*)} \leftarrow \mathbf{x}^{(i)}$
    $i_{MAX} \leftarrow i$
  **end if**
  $m \leftarrow m + 1$
**end while**
**Output:** $\mathbf{x}^{(i*)}$

## 5.2 A Semidefinite Programming Approach

In this section, the overall exact approach based on an SDP relaxation of the problem is described. Section 5.2.1 describes how the SDP relaxation is derived, while Section 5.2.2 discusses the other features of the branch and bound framework.

### 5.2.1 The Semidefinite Programming Relaxation

The proposed SDP relaxation for the $Max-MeanDP$ takes inspiration from the ones described in Section 1.2 for the Quadratic Knapsack Problem. A similar relaxation to the one presented in this section was obtained in [22] for a problem belonging to a completely different field. First, we consider the vectorial formulation of the problem ($Max-MeanDP$ Standard Formulation 2). Another equivalent form can be derived from it by defining the matrix $\mathbf{X}$ as the dyadic product $\mathbf{x}\mathbf{x}^t$. In this case, each matrix component $x_{i,j}$ represents the product $x_i x_j$.

$Max-MeanDP$ **Standard Formulation 3.**

$$\max \frac{\mathrm{Tr}(\boldsymbol{DX})}{2\,\mathrm{Tr}(\boldsymbol{IX})} \tag{5.11}$$

*subject to:*

$$\mathrm{Tr}(\boldsymbol{IX}) \geq 2 \tag{5.12}$$

$$\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^t \tag{5.13}$$

$$\boldsymbol{x} \in \{0,1\}^n \tag{5.14}$$

Since $x_{i,i} = x_i \ \forall i = 1, ..., n$, the expression $\mathbf{u}^t\mathbf{x}$ can be rewritten as $\mathrm{Tr}(\mathbf{IX})$, where $\mathbf{I}$ is the identity $n \times n$ matrix, and $\mathbf{x} = diag(\mathbf{X})$ since $x_i$s are binary. As a standard procedure to obtain a semidefinite relaxation, we discard Constraint 5.13 and Constraint 5.14 because they are non-convex, while we add the convex one $\mathbf{X} \succeq \mathbf{x}\mathbf{x}^t$. The following relaxation arises from these considerations:

$Max - MeanDP$ **Quasiconvex Relaxation (QCR).**

$$\max \frac{\mathrm{Tr}(\boldsymbol{DX})}{2\,\mathrm{Tr}(\boldsymbol{IX})} \tag{5.15}$$

*subject to:*

$$\mathrm{Tr}(\boldsymbol{IX}) \geq 2 \tag{5.16}$$

$$\begin{bmatrix} \boldsymbol{X} & \boldsymbol{x} \\ \boldsymbol{x}^t & 1 \end{bmatrix} \succeq 0 \tag{5.17}$$

Note that Constraint 5.17 is equivalent to $\mathbf{X} \succeq \mathbf{x}\mathbf{x}^t$ because of Schur complement condition for positive semidefiniteness. Constraints 5.16 and 5.17 are convex, but the objective function is not. In fact, $f(\mathbf{X}) = \frac{\mathrm{Tr}(\mathbf{DX})}{\mathrm{Tr}(\mathbf{IX})}$ is a quasiconvex function and, thus, program 5.15 - 5.17 is a quasiconvex program. Henceforth, we refer to this program as QCR. In general, quasi convex programs can be solved via bisection method [21] which is computationally expensive because a sequence of SDP feasibility problems has to be solved at each step. In analogy with [22], we prove that it is possible to compute the value of this relaxation by solving just one SDP. Let us consider the following program:

$Max - MeanDP$ **SDP Relaxation (SDPR).**

$$\max \frac{1}{2}\,\mathrm{Tr}(\boldsymbol{DZ}) \tag{5.18}$$

*subject to:*

$$\mathrm{Tr}(\boldsymbol{IZ}) = 1 \tag{5.19}$$

$$\begin{bmatrix} \boldsymbol{Z} & \boldsymbol{z} \\ \boldsymbol{z}^T & z_{n+1,n+1} \end{bmatrix} \succeq 0 \tag{5.20}$$

$$\frac{1}{n} \leq z_{n+1,n+1} \leq \frac{1}{2} \tag{5.21}$$

Here $\mathbf{Z}$ is a symmetric $n \times n$ matrix, $\mathbf{z}$ is a n-dimensional vector and $z_{n+1,n+1}$ is a scalar. This program is an SDP and can be solved in polynomial time by means of interior point methods. Let us call it SDPR. In the subsequent proposition, we state the relationship between SDPR and QCR.

**Proposition 11.** *The quasiconvex problem QCR has the same optimal value of SDPR. Furthermore, the following statements holds:*

- *an optimal solution $\boldsymbol{X}^*$ of QCR can be obtained from an optimal solution $\boldsymbol{Z}^*$ of SDPR by using the equation:*

$$\boldsymbol{X}^* = \frac{\boldsymbol{Z}^*}{z_{n+1,n+1}} \tag{5.22}$$

- *the value of $z_{n+1,n+1}$ is equal to $\frac{1}{\sum_{i=1}^{n} x_i}$.*

*Proof.* We follow the lines of [22] in the proof of their Proposition 1. First, we note that $z_{n+1,n+1} \neq 0$ because of Constraint 5.21. Then, it is easy to verify that we can always define a feasible point $\overline{\mathbf{X}}$ of QCR and $\overline{\mathbf{Z}}$ of SDPR such that $\overline{\mathbf{X}} = \frac{\overline{\mathbf{Z}}}{z_{n+1,n+1}}$. As a consequence of Constraint 5.19, we have that $\mathrm{Tr}(\mathbf{I}\overline{\mathbf{X}}) = \frac{1}{z_{n+1,n+1}} = \frac{1}{\sum_{i=1}^{n} z_i}$. By using this equality, we can see that $\overline{\mathbf{X}}$ and $\overline{\mathbf{Z}}$ has the same objective function. Finally, problems QCR and SDPR are equivalent and their optimal values $\mathbf{X}^*$ and $\mathbf{Z}^*$ are linked by the equation $\mathbf{X}^* = \frac{\mathbf{Z}^*}{z_{n+1,n+1}}$. □

In simple words, the optimum of SDPR is scaled by $\sum_{i=1}^{n} x_i$ with respect to the optimum of QCR. Thus, the logic-one value in variables $x_i$ is reflected on variables $z_i$ such that they are equal to $z_{n+1,n+1}$. A traditional approach to strengthen the quality of these relaxations is to add valid inequalities for the boolean quadric polytope. In fact, components $x_{i,j}$s model the product $x_i x_j$ in QCR and the following inequalities hold:

$$x_{i,i} + x_{j,j} \leq 1 + x_{i,j} \ \forall i, j \tag{5.23}$$

$$x_{i,k} + x_{j,k} \leq x_{k,k} + x_{i,j} \ \forall i, j, k \tag{5.24}$$

$$x_{i,j} + x_{i,k} + x_{j,k} + 1 \geq x_{i,i} + x_{j,j} + x_{k,k} \ \forall i, j, k \tag{5.25}$$

These inequalities can be reformulated by means of Equation 5.22 such that they are valid for SDPR:

$$z_{i,i} + z_{j,j} \leq z_{n+1,n+1} + z_{i,j} \ \forall i, j \tag{5.26}$$

$$z_{i,k} + z_{j,k} \leq z_{k,k} + z_{i,j} \ \forall i, j, k \tag{5.27}$$

$$z_{i,j} + z_{i,k} + z_{j,k} + z_{n+1,n+1} \geq z_{i,i} + z_{j,j} + z_{k,k} \ \forall i, j, k \tag{5.28}$$

Let $\mathcal{B}$ be the set of inequalities 5.26 - 5.28. The cardinality of set $\mathcal{B}$ is $\mathcal{O}(n^3)$, but the efficiency of the solution of SDP models is strongly affected by the number of constraints considered. Thus, it is not practical to include them all in SDPR even for instances with a few items. In this case, the common approach is to use a cutting plane algorithm in order to selects a subset of relevant inequalities which allow to achieve a good compromise between the tightness of the relaxation and the computation time to solve it.

### 5.2.2 The Branch and Bound Framework

The basic idea is to embed the relaxation SDPR of $Max - MeanDP$ into a branch and bound framework. At the root node of the branch tree, SDPR is solved for a number of iterations and at each iteration the $\theta$ most violated inequalities in $\mathcal{B}$ are added as constraints to the problem. Let us call $\overline{\mathcal{B}}$ the set of the selected inequalities within a time $T_{max}$. The result of the final relaxation provides an initial upper bound. At the other nodes, no warm start strategy is implemented, meaning that the solution of the father node is not used to reduce the computation time of the new optimum. Conversely, the SDP model is created from scratch, but the number of variables and constraints is reduced more and more as we go down in the branch tree.

### 5.2.3 Reducted SDP model

At a general point of the branch tree, we distinguish a set $\mathcal{A}$ of elements which are put in $M$ and another set $\tilde{\mathcal{A}}$ of elements which are inhibited to be in $M$. Finally, the set $N/(\mathcal{A}+\tilde{\mathcal{A}})$ is formed by the undecided elements. We impose the belonging of elements in $\mathcal{A}$ and $\tilde{\mathcal{A}}$ as linear constraints and we obtain the following problem, named SDPRA:

$Max-MeanDP$ **SDP Relaxation A (SDPRA).**

$$\max \frac{1}{2}\operatorname{Tr}(\boldsymbol{DZ}) \tag{5.29}$$

*subject to:*

$$\operatorname{Tr}(\boldsymbol{IZ}) = 1 \tag{5.30}$$

$$\begin{bmatrix} \boldsymbol{Z} & \boldsymbol{z} \\ \boldsymbol{z}^T & z_{n+1,n+1} \end{bmatrix} \succeq 0 \tag{5.31}$$

$$\frac{1}{n} \leq z_{n+1,n+1} \leq \frac{1}{2} \tag{5.32}$$

$$z_i = z_{n+1,n+1} \ \forall i \in \mathcal{A} \tag{5.33}$$

$$z_i = 0 \ \forall i \in \tilde{\mathcal{A}} \tag{5.34}$$

$$\textit{valid inequalities} \in \overline{\mathcal{B}} \tag{5.35}$$

It is possible to create a reducted version of SDPRA by substituting 5.33 and 5.34 in the rest of the model. Let us define a set $\overline{\mathcal{B}}_a \subseteq \overline{\mathcal{B}}$ of active constraints, i.e. the set of constraints in $\overline{\mathcal{B}}$ which are not redundant because of the decisions taken in the branch tree. Let us define a matrix $\mathbf{D}_a$ obtained by considering only rows and columns of $\mathbf{D}$ associated to elements in $N/(\mathcal{A}+\tilde{\mathcal{A}})$. Analogously, we define $\mathbf{Z}_a$ from $\mathbf{Z}$ by removing the same rows and columns. Both $\mathbf{D}_a$ and $\mathbf{Z}_a$ are $\hat{n} \times \hat{n}$ matrices

where $\hat{n} \leq n$ is the number of unfixed variables. The contribution of the picked elements, i.e. elements belonging to $\mathcal{A}$, in the objective is given by the expression $s = \frac{\sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{A}} d_{i,j} z_{n+1,n+1}}{2}$. Moreover, there exists a contribution brought by the insertion of an element $i \in N/(\mathcal{A}+\tilde{\mathcal{A}})$ in $M$ that is obtained as $s_i = \frac{\sum_{j\in\mathcal{A}} d_{i,j} z_{n+1,n+1}}{2}$. In the following notation, we use a function $\alpha(\cdot)$ which provides the element number $\alpha(p) \in N$ associated to a row/column index $p$. First, we define a matrix $\mathbf{U}_a$ as a diagonal matrix with the p-th component of the diagonal equal to $s_{\alpha(p)}$. Let $\mathbf{z}_a$ be a $\hat{n}$-dimensional vector whose p-th component is equal to $x_{\alpha p} z_{n+1,n+1}$. Let us consider the following $\hat{n}+1 \times \hat{n}+1$ matrices:

$$\hat{\mathbf{D}}_a = \begin{bmatrix} \mathbf{D}_a + \mathbf{U}_a & \mathbf{0} \\ \mathbf{0}^T & s \end{bmatrix} \tag{5.36}$$

$$\hat{\mathbf{Z}}_a = \begin{bmatrix} \mathbf{Z}_a & \mathbf{0} \\ \mathbf{0}^T & z_{n+1,n+1} \end{bmatrix} \tag{5.37}$$

$$\hat{\mathbf{I}} = \begin{bmatrix} \mathbf{I}_a & \mathbf{0} \\ \mathbf{0}^T & |A| \end{bmatrix} \tag{5.38}$$

Then, the following problem, called SDPRB, can be obtained:

$Max - MeanDP$ **SDP Relaxation B (SDPRB).**

$$\max \frac{1}{2} \operatorname{Tr}(\hat{\boldsymbol{D}}_a \hat{\boldsymbol{Z}}_a) \tag{5.39}$$

*subject to:*

$$\operatorname{Tr}(\hat{\boldsymbol{I}} \hat{\boldsymbol{Z}}_a) = 1 \tag{5.40}$$

$$\begin{bmatrix} \boldsymbol{Z}_a & \boldsymbol{z}_a \\ \boldsymbol{z}_a^T & z_{n+1,n+1} \end{bmatrix} \succeq 0 \tag{5.41}$$

$$\frac{1}{n} \leq z_{n+1,n+1} \leq \frac{1}{2} \tag{5.42}$$

$$valid\ inequalities \in \overline{\mathcal{B}}_a \tag{5.43}$$

It is easy to verify that SDPRA and SDPRB are equivalent problems, but SDPRB presents a lower number of variables and constraints, since unrelevant constraints and variables are not taken into consideration. Thus, SDPRB is used as a method in the branch and bound tree to determine powerful upper bounds in a reasonable amount of time.

### SDP based lower bounds

As we have seen so far, semidefinite programming is exploited in order to have powerful upper bounds. However, the semidefinite solution of the relaxation SDPRB can be used to determine good lower bounds too. The basic strategy embedded in the framework is a rounding heuristic that, starting from the fractional semidefinite programming solution $\mathbf{z}_a$, allows to reconstruct a high quality feasible solution for the $Max-MeanDP$. In an open node of the branch and bound tree, an initial feasible solution $\mathbf{x}_{\mathcal{A}}$ can be instantly obtained by fixing at 1 variables whose associated element is in $\mathcal{A}$ and at 0 all the other elements. Another initial feasible solution $\mathbf{x}_{\tilde{\mathcal{A}}}$ can be obtained in the opposite way, variables associated at elements in $\tilde{\mathcal{A}}$ are set to 0 and all the other variables are equal to 1.

The heuristic based on the solution $\mathbf{z}_a$ of SDPRB can be seen as a path relinking from solution $\mathbf{x}_{\mathcal{A}}$ to $\mathbf{x}_{\tilde{\mathcal{A}}}$. At each step of the path-relinking procedure a variable associated to an element in $N/(\mathcal{A}+\tilde{\mathcal{A}})$ is set to 1. Let us suppose that this element is $p$ and the associated variable is $x_p$. The element $p$ is choosed such that the i-th component of $\mathbf{z}_a$ with $\alpha(i) = p$ has the highest value in the semidefinite solution among the unfixed variables. Thus, the path relinking is guided by the solution of SDRB and the distance between the current solution $\mathbf{x}_{CURR}$ and $\mathbf{x}_{\tilde{\mathcal{A}}}$ decreases by one unit at each step. Note that it is not a problem if the initial solution does not satisfy Constraint 5.2, because the cardinality of set $M$ will increase along the path and feasible solutions will be found. A steepest descent three-opt local search centered in $\mathbf{x}_{CURR}$ is performed after of each iteration of the path relinking. In the end of the algorithm, the best solution found is provided as an output. The pseudocode of this heuristic approach follows.

**Algorithm 11** SDP based heuristics for $Max - MeanDP$

**Input:** $Max - MeanDP$ instance, sets $\mathcal{A}$ and $\tilde{\mathcal{A}}$, semidefinite solution $\mathbf{x}_a$
Set $\mathbf{x}_{CURR} = \mathbf{x}_{\mathcal{A}}$
**while** $\mathbf{x}_{CURR} \neq \mathbf{x}_{\tilde{\mathcal{A}}}$ **do**
    Set variable associated to the highest valued component of $\mathbf{z}_a$ at 1
    Update $\mathbf{x}_{CURR}$
    Perform steepest descent three-opt local search from $\mathbf{x}_{CURR}$
    Update best feasible solution $\mathbf{x}^*$
**end while**
**Output:** Best solution $\mathbf{x}^*$ found

**Tree expansion strategy**

Our preliminary tests revealed that the best branching rule, in order to reduce the size of the search tree, is to branch on the highest valued variable. Since the algorithm is thought to reduce the overall time to compute the optimal solution of a $Max - MeanDP$ instance, the expansion strategy is a simple depth-first search where at each step of the algorithm the node with the most promising upper bound is expanded.

## 5.3 A Three-Phase Hybrid Heuristic

This section is devoted to describe the matheuristics approach designed for the problem. In [76], it was shown that in their real world small instances the value of $m$ associated with the optimal solution lies in an interval, such that there are high quality solutions with values of $m$ inside this interval. Although these promising intervals may be disjoint in large instances, the basic conclusion is very interesting: if a set $C$ of good candidate values for $m$ is selected, the use of the pseudo-polynomial models can be restricted only to values of $m$ in $C$. In the following, an interval, i.e. the set of all possible integer values between two extreme points $m_1$ and $m_2$, where $m_1, m_2 \in \mathbb{N}$ and $2 \leq m_1 \leq m_2 \leq n$, is indicated with $[m_1, m_2]$.

The proposed hybrid algorithm for $Max-MeanDP$ is composed by three phases applied in cascade, namely

**PHASE ONE** It consists in the selection of good candidate values of $m$ to be included in set $C$, while also providing good feasible solutions;

**PHASE TWO** Given $C$, a local branching scheme is used in order to enhance the quality of the solutions provided in PHASE ONE $\forall m \in C$.

**PHASE THREE** A further solution enhancement is obtained by means of a path relinking procedure.

and the best solution found at the end of PHASE THREE is provided in output.

In the following subsections, the three phases are described in detail and for each phase the corresponding pseudocode is provided.

### 5.3.1 PHASE ONE: using the QIP solver to compute initial solutions

PHASE ONE makes use of a decision tree with $k$ branches (where $k$ is a parameter to be defined experimentally) to seek promising intervals. Starting from the initial interval $[2, n]$, it is split into $k$ intervals of equal width, where an evaluation value is computed for each interval. Then, the interval with the best evaluation is expanded at each step, resulting in a best first expansion of the decision tree. The evaluation of an interval $[m_1, m_2]$ is performed by considering the quality of the solution provided by the QIP solver after a fixed amount of time, using QIP Formulation 1 with $m = m_1$ and $m = m_2$. Thus, we evaluate a set of contiguous values of $m$ by computing two solutions at its extreme points and summing their objectives. Call $h(m)$ the result provided by the solver after $T_s$ seconds. The evaluation $\lambda(m_1, m_2) \in \mathbb{R}$, with $m_1, m_2 \in \mathbb{N}$ such that $2 \leq m_1 \leq m_2 \leq n$, of an interval $[m_1, m_2]$ can be computed as:

$$\lambda(m_1, m_2) = h(m_1) + h(m_2) \tag{5.44}$$

A set $S_1$ of feasible solutions for the $Max-MeanDP$ is computed in PHASE ONE, during the evaluations of intervals thanks to the computation of function $h$. In other words, the output of PHASE ONE is not limited to the definition of $C$, but for each value $m' \in C$ a solution $S_1$ with $m = m'$ is computed, included in $S_1$ and provided as output.

Figure 5.1 shows how the decision tree is expanded during PHASE ONE, under the assumption that gray nodes are associated to the intervals with the highest evaluation $\lambda$ at each step. Only the initial three steps of the expansion are depicted. Table 5.1 enumerates solutions computed in these steps in order to expand nodes. These solutions are indicated with the notation $\widehat{\mathbf{x}}_{m'}$ where $m'$ is the value of $m$ for the solution. In the example, PHASE ONE yields the following sets $C$ and $S_1$ after the initial three steps.

$$C = \{2{,}100{,}120{,}140{,}144{,}148{,}152{,}156{,}160{,}180{,}200{,}300{,}400{,}499\} \tag{5.45}$$

$$S = \{\widehat{\mathbf{x}}_2, \widehat{\mathbf{x}}_{100}, \widehat{\mathbf{x}}_{120}, \widehat{\mathbf{x}}_{140}, \widehat{\mathbf{x}}_{144}, \widehat{\mathbf{x}}_{148}, \widehat{\mathbf{x}}_{152}, \widehat{\mathbf{x}}_{156}, \widehat{\mathbf{x}}_{160}, \widehat{\mathbf{x}}_{180}, \widehat{\mathbf{x}}_{200}, \widehat{\mathbf{x}}_{300}, \widehat{\mathbf{x}}_{400}, \widehat{\mathbf{x}}_{500}\} \tag{5.46}$$

In general, PHASE ONE ends as soon as a specific time limit $T_1$ is achieved. We

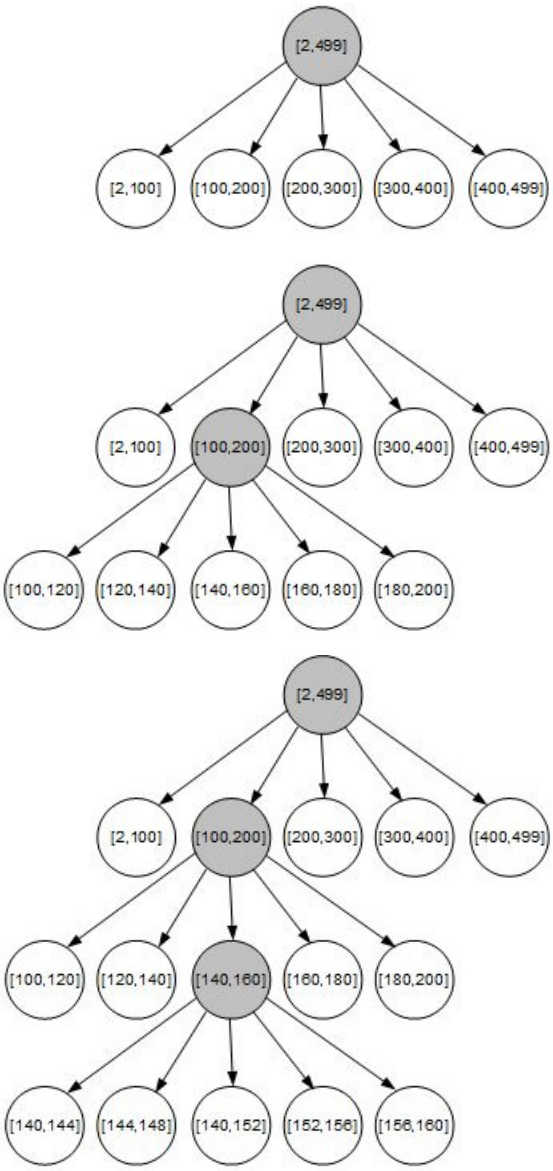



Figure 5.1: An example showing three steps of PHASE ONE

refer to the number of solutions computed in the end of PHASE ONE as $\gamma$.

Algorithm 12 depicts the pseudocode of PHASE ONE.

**Algorithm 12** PHASE ONE

**Input:** $Max - MeanDP$ instance
Compute $\widehat{\mathbf{x}}_2$ and $\widehat{\mathbf{x}}_n$ by solving to optimality $Max - SumDP$ for $m = 2$ and $m = n$
$C \leftarrow \{2, n\}$ $S_1 \leftarrow \{\widehat{\mathbf{x}}_2, \widehat{\mathbf{x}}_n\}$
Set $I \leftarrow \{[2, n]\}$
**while** time limit $T_1$ is not reached **do**
    Extract best valued interval from $I$
    Split the selected interval $[m_0, m_k]$ in $k$ intervals $[m_0, m_1] \cdots [m_{k-1}, m_k]$
    By running the QIP solver for $T_s$ seconds:
        Compute $k - 1$ solutions $\widehat{\mathbf{x}}_{m_1} \cdots \widehat{\mathbf{x}}_{m_{k-1}}$ with $m = m_1 \cdots m_{k-1}$
    $I \leftarrow I \cup \{[m_0, m_1] \cdots [m_{k-1}, m_k]\}$
    $C \leftarrow C \cup \{m_1 \cdots m_{k-1}\}$
    $S_1 \leftarrow S_1 \cup \{\widehat{\mathbf{x}}_{m_1} \cdots \widehat{\mathbf{x}}_{m_{k-1}}\}$
**end while**
**Output:** $S_1$

| First step | Second step | Third step |
|---|---|---|
| $\widehat{\mathbf{x}}_2$ | $\widehat{\mathbf{x}}_{120}$ | $\widehat{\mathbf{x}}_{144}$ |
| $\widehat{\mathbf{x}}_{100}$ | $\widehat{\mathbf{x}}_{140}$ | $\widehat{\mathbf{x}}_{148}$ |
| $\widehat{\mathbf{x}}_{200}$ | $\widehat{\mathbf{x}}_{160}$ | $\widehat{\mathbf{x}}_{152}$ |
| $\widehat{\mathbf{x}}_{300}$ | $\widehat{\mathbf{x}}_{180}$ | $\widehat{\mathbf{x}}_{156}$ |
| $\widehat{\mathbf{x}}_{400}$ | | |
| $\widehat{\mathbf{x}}_{500}$ | | |

Table 5.1: Solutions computed in PHASE ONE

### 5.3.2 PHASE TWO: local branching

PHASE TWO applies a local branching scheme [43] to each solution in $S_1$ and provides in output a set $S_2$ of new solutions. A general description of the method is provided in Section 1.1.1. For each solution $\widehat{\mathbf{x}}_{m'}$, a local search is performed, considering as a neighborhood of $\widehat{\mathbf{x}}_{m'}$ the set $\Psi(\widehat{\mathbf{x}}_{m'})$ of solutions whose Hamming distance from $\widehat{\mathbf{x}}_{m'}$ is less than or equal to a parameter $\delta$, while the overall number of selected items is kept constant. This local search is carried out by running the QIP solver on an extension of QIP formulation 1 for $T_{LS}$ seconds. Recall that in QIP formulation 1 the value of $m$ is fixed a priori, here $m = m'$. The following constraint is added to the formulation, in order to take into account the Hamming distance constraint:

$$2H(\mathbf{x}, \widehat{\mathbf{x}}) = \sum_{i=1}^{n} (x_i(1 - \widehat{x}_i) + \widehat{x}_i(1 - x_i)) \leq 2\delta \tag{5.47}$$

where $H(\mathbf{x}, \widehat{\mathbf{x}})$ is the Hamming distance between $\mathbf{x}$ and $\widehat{\mathbf{x}}$.

At this point, the QIP solver can provide two different results:

- an improving solution $\widehat{\mathbf{x}}^*_{m'} \in \Psi(\widehat{\mathbf{x}}_{m'})$ is found and can be used as a starting point for the subsequent local search;
- no improving solution is found in $\Psi(\widehat{\mathbf{x}}_{m'})$.

In the first case, PHASE TWO continues by discarding Constraint 5.47 and forcing the exploration of the remaining solution space. To this end, the following constraint

is kept:

$$\sum_{i=1}^{n}(x_i(1-\widehat{x}_i)+\widehat{x}_i(1-x_i)) \geq 2\delta \tag{5.48}$$

Accordingly, a new constraint that indicates the maximum Hamming distance from $\widehat{\mathbf{x}}^*_{m'}$ is added, the solver is run on the resulting model and the process is iterated.

In the second case, Constraint 5.47 is replaced by:

$$2\delta < \sum_{i=1}^{n}(x_i(1-\widehat{x}_i)+\widehat{x}_i(1-x_i)) \leq 4\delta \tag{5.49}$$

and the solver is run again on the resulting model. If an improved solution is found, PHASE TWO proceeds as in the first case. Otherwise, it moves to the next initial solution in $S_1$. Each solution provided by the local branching step is included in the set $S_2$. PHASE TWO stops when a time limit $T_2$ is reached, or when the whole set $S_1$ is scanned.

Algorithm 13 depicts the pseudocode of PHASE TWO.

**Algorithm 13** PHASE TWO

**Input:** $Max-MeanDP$ instance, $S_1$
$S_2 \leftarrow \emptyset$
**while** $S_1 \neq \emptyset$ **and** time limit $T_2$ is not reached **do**
  Extract best solution $\widehat{\mathbf{x}}$ in $S_1$
  Compute $\widehat{\mathbf{x}}'$ with a Local Branching procedure starting from $\widehat{\mathbf{x}}$
  $S_2 \leftarrow S_2 \cup \{\widehat{\mathbf{x}}'\}$
**end while**
**Output:** $S_2$

### 5.3.3 PHASE THREE: path relinking

Path relinking [52] is a technique that has been proved to be computationally effective for many combinatorial optimization problems. It takes as input an initial set of feasible solutions, called the elite set $\mathcal{E}$. We build such a set by scanning $S_2$ in non-increasing order and adding the current solution $\widehat{\mathbf{x}}$ to $\mathcal{E}$ if there are no solutions in $\mathcal{E}$

whose Hamming distance from $\widehat{\mathbf{x}}$ is less than or equal to 1. This step is iterated until all solutions in $S_2$ are considered. Recall that solutions provided by PHASE TWO have different values of $m$, thus $\mathcal{E}$ is constructed by considering solutions which are sufficiently different from one to another. Correspondingly, we generate a vector $\mathcal{C}_\mathcal{E}$ containing all couples of solutions ($\widehat{\mathbf{x}}$,$\widehat{\mathbf{y}}$) (with $\widehat{\mathbf{x}}, \widehat{\mathbf{y}} \in \mathcal{E}$ and $\widehat{\mathbf{x}} \neq \widehat{\mathbf{y}}$) sorted in nonincreasing order of $f(\widehat{\mathbf{x}}) + f(\widehat{\mathbf{y}})$. Then, we scan the sorted vector $\mathcal{C}_\mathcal{E}$ considering one at a time (until all couples are considered or a time limit $T_3$ is met) each couple ($\widehat{\mathbf{x}}$,$\widehat{\mathbf{y}}$) and we determine a sequence of moves that leads from $\widehat{\mathbf{x}}$ to $\widehat{\mathbf{y}}$ and viceversa. In this context, a move is determined by an item deletion or insertion such that a decrease in the Hamming distance between the solutions occurs. At each step, the move leading to the best solution (ties broken at random) is picked and the process is iterated until the destination is reached. The final solution returned by PHASE THREE is the best of those in $\mathcal{E}$ and those computed along the paths.

Algorithm 14 depicts the pseudocode of PHASE THREE.

**Algorithm 14** PHASE THREE

**Input:** $Max - MeanDP$ instance, $S_2$
**while** $S_2 \neq \emptyset$ **and** $T_3$ is not reached **do**
  Extract best solution $\widehat{\mathbf{x}}$ in $S_2$
  **if** $\nexists \widehat{\mathbf{e}} \in \mathcal{E} : H(\widehat{\mathbf{e}}, \widehat{\mathbf{x}}) \leq 1$ **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \widehat{\mathbf{x}}$
  **end if**
**end while**
Create set $\mathcal{C}_\mathcal{E}$ and sort it in increasing order of $f(\widehat{\mathbf{x}}) + f(\widehat{\mathbf{y}})$
**for all** sorted couples ($\widehat{\mathbf{x}}$,$\widehat{\mathbf{y}}$) $\in \mathcal{C}_\mathcal{E}$ **do**
  Perform path relinking from $\widehat{\mathbf{x}}$ to $\widehat{\mathbf{y}}$
**end for**
**Output:** Best solution found

## 5.4 Computational Experiments

The aim of this section is to show the practical efficiency of the branch and bound algorithm and of the matheuristic described so far. The instances used for the tests have different sizes, we distinguish small instances ($n$ = 20,25,30,35), medium

size instances ($n$ = 50,75,100) and large size instances ($n$ = 150,500). Small instances and large instances are the ones introduced in [76], while the medium size instances are randomly generated following the same lines. The instances belong to two classes:

**Type I** distances $d_{i,j}$s are uniformly distributed in $[-10,10]$;

**Type II** distances $d_{i,j}$s are uniformly distributed in $[-10,-5] \cup [5,10]$.

Since 10 instances are considered for every different type and size, the whole dataset that we adopted is formed by 180 different instances. All the proposed approaches were tested on an Intel Core i5-3550 3.30GHz with 4GB of RAM, and they were implemented in C++. Cplex 12.5 was used as a QIP solver. In the following, the tests performed for the exact method will regard only small and medium instances. Contrarily, the heuristic is only tested on large instances, for which the exact method is not practical.

### 5.4.1 Tests for the Branch and Bound Algorithm

Let us discuss the parametrization of the proposed method, hereafter indicated with SDP B&B. A key point for the effectiveness of the method is the fast solution of the problem SDPRB. In the implementation, we solve SDPRB via CSDP 6.1.1 [19], which is a robust and efficient SDP solver. It implements a primal dual interior point method that is a variant of the algorithm described in [57]. Let us start with the settings of the cutting plane algorithm executed in the root node. We set $\theta$ to 5, meaning that the 5 most violated inequalities among the ones in 5.26 - 5.28. The time $T_{max}$ is set to a value which depends on the size of the instance. These values, reported in Table 5.2, are choosed experimentally in order to obtain a good upper bound in the root node by adding a sufficiently small set of constraints $\overline{\mathcal{B}}$. In fact, we also want that the solution of SDPRB in the other nodes of the tree is not too costly.

We did not set a time limit for SDP B&B. However, the time limit of 10 hours used for IMA was reached only in two instances of size 100 (MDPI5_100 and MDPII5_100). The results of the tests are reported in Table 5.3 and Table 5.4. In these tables, the first three columns are relative to the name of the instance, the

| $n$ | $T_{max}$ |
|---|---|
| 20,25 | 0.5 seconds |
| 30,35 | 1 second |
| 50 | 20 seconds |
| 75 | 5 minutes |
| 100 | 20 minutes |

Table 5.2: Values of $T_{max}$ used in the tests

value of the cardinality $m$ of the selected subset in the optimal solution and the value of the optimal objective. The columns labeled with $\mathbf{T}(s)$ indicate the time in seconds required by the approach (IMA or SDP B&B) to solve the instances to optimality. Relatively to the SDP method, the column named "Nodes" indicate the number of nodes opened in proving the optimality of the solution. Table 5.3 shows that for small instances ($n$ = 20,25,30,35) the two approaches are quite equivalent, and the number of nodes opened by SDP B&B is low. This scenario changes when we consider medium size instances ($n$ = 50,75,100). In the case of $n = 50$, the proposed method is able to solve the instances considered roughly in a half of the time required by IMA. As the value of $n$ increases, the dominance of SDP B&B with respect to IMA is stronger. In fact, IMA can not manage instances with $n = 75$ in most of the cases, while SDP B&B was capable of solving each of them within 32 minutes. For $n = 100$, SDP B&B solved 18 over 20 instances within a time limit of 10 hours, with a high variance (for instance $MDPI4\_100$ was solved in approximately 38 minutes). Table 5.5 summarizes the results obtained in all the tests by grouping them according to the instance size. In this table, $\mathbf{T}_{avg}(s)$ is the average computational time and **Average Nodes** is the average number of nodes opened. As a conclusion, the proposed approach allows to solve instances with up to 100 elements, while previously even instances with size 75 could be hardly managed. Further developments of this work could regard the implementation of a more efficient interior point algorithm which exploits some properties of the relaxation.

| | Optimum | | IMA | SDP B&B | | | Optimum | | IMA | SDP B&B | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Instance** | **m** | **value** | **T**(s) | **Nodes** | **T**(s) | **Instance** | **m** | **value** | **T**(s) | **Nodes** | **T**(s) |
| **MDPI1_20** | 7 | 13,880 | 1 | 6 | 0 | **MDPI1_30** | 8 | 19,861 | 2 | 9 | 1 |
| **MDPI2_20** | 5 | 13,608 | 0 | 5 | 0 | **MDPI2_30** | 9 | 18,813 | 1 | 13 | 1 |
| **MDPI3_20** | 7 | 11,796 | 1 | 8 | 0 | **MDPI3_30** | 9 | 15,249 | 2 | 30 | 2 |
| **MDPI4_20** | 8 | 17,540 | 1 | 3 | 0 | **MDPI4_30** | 15 | 22,717 | 2 | 16 | 1 |
| **MDPI5_20** | 8 | 16,006 | 1 | 1 | 0 | **MDPI5_30** | 9 | 17,237 | 3 | 43 | 3 |
| **MDPI6_20** | 11 | 14,606 | 0 | 3 | 0 | **MDPI6_30** | 11 | 18,376 | 2 | 20 | 2 |
| **MDPI7_20** | 9 | 14,882 | 0 | 4 | 0 | **MDPI7_30** | 8 | 15,293 | 3 | 17 | 1 |
| **MDPI8_20** | 7 | 14,461 | 0 | 10 | 0 | **MDPI8_30** | 11 | 19,247 | 2 | 18 | 2 |
| **MDPI9_20** | 6 | 14,035 | 0 | 5 | 0 | **MDPI9_30** | 14 | 22,004 | 2 | 9 | 1 |
| **MDPI10_20** | 6 | 13,443 | 0 | 7 | 0 | **MDPI10_30** | 13 | 18,699 | 2 | 17 | 2 |
| **MDPII1_20** | 8 | 18,855 | 0 | 5 | 0 | **MDPII1_30** | 14 | 22,272 | 2 | 39 | 3 |
| **MDPII2_20** | 7 | 17,830 | 0 | 6 | 0 | **MDPII2_30** | 13 | 26,914 | 3 | 9 | 1 |
| **MDPII3_20** | 7 | 18,110 | 0 | 3 | 0 | **MDPII3_30** | 11 | 21,897 | 5 | 40 | 3 |
| **MDPII4_20** | 10 | 17,842 | 0 | 9 | 0 | **MDPII4_30** | 8 | 20,538 | 3 | 32 | 2 |
| **MDPII5_20** | 5 | 16,344 | 0 | 9 | 0 | **MDPII5_30** | 11 | 22,790 | 3 | 21 | 2 |
| **MDPII6_20** | 6 | 17,610 | 0 | 3 | 0 | **MDPII6_30** | 10 | 20,351 | 3 | 31 | 2 |
| **MDPII7_20** | 6 | 18,938 | 0 | 9 | 0 | **MDPII7_30** | 10 | 27,655 | 2 | 5 | 1 |
| **MDPII8_20** | 8 | 21,880 | 0 | 9 | 0 | **MDPII8_30** | 12 | 26,884 | 2 | 20 | 1 |
| **MDPII9_20** | 8 | 19,785 | 0 | 18 | 0 | **MDPII9_30** | 9 | 24,177 | 2 | 31 | 2 |
| **MDPII10_20** | 10 | 22,599 | 0 | 10 | 0 | **MDPII10_30** | 10 | 24,800 | 2 | 17 | 1 |
| **MDPI1_25** | 12 | 17,271 | 1 | 13 | 0 | **MDPI1_35** | 12 | 19,183 | 5 | 24 | 4 |
| **MDPI2_25** | 7 | 15,121 | 1 | 10 | 0 | **MDPI2_35** | 10 | 17,168 | 4 | 45 | 5 |
| **MDPI3_25** | 6 | 14,182 | 1 | 8 | 0 | **MDPI3_35** | 13 | 17,075 | 4 | 50 | 5 |
| **MDPI4_25** | 12 | 19,857 | 1 | 6 | 0 | **MDPI4_35** | 15 | 23,350 | 4 | 33 | 4 |
| **MDPI5_25** | 7 | 17,537 | 1 | 12 | 0 | **MDPI5_35** | 13 | 19,018 | 4 | 41 | 5 |
| **MDPI6_25** | 9 | 17,967 | 1 | 9 | 0 | **MDPI6_35** | 12 | 19,445 | 8 | 37 | 5 |
| **MDPI7_25** | 10 | 16,207 | 1 | 13 | 0 | **MDPI7_35** | 14 | 19,497 | 5 | 30 | 4 |
| **MDPI8_25** | 12 | 18,137 | 1 | 11 | 0 | **MDPI8_35** | 15 | 21,231 | 7 | 58 | 6 |
| **MDPI9_25** | 9 | 17,478 | 1 | 10 | 0 | **MDPI9_35** | 9 | 20,980 | 5 | 17 | 3 |
| **MDPI10_25** | 12 | 19,459 | 1 | 9 | 0 | **MDPI10_35** | 9 | 16,938 | 6 | 39 | 5 |
| **MDPII1_25** | 10 | 21,810 | 1 | 10 | 0 | **MDPII1_35** | 12 | 25,968 | 8 | 38 | 5 |
| **MDPII2_25** | 8 | 22,185 | 1 | 9 | 0 | **MDPII2_35** | 11 | 26,136 | 6 | 50 | 5 |
| **MDPII3_25** | 9 | 23,564 | 1 | 8 | 0 | **MDPII3_35** | 13 | 24,159 | 6 | 57 | 6 |
| **MDPII4_25** | 8 | 19,740 | 1 | 15 | 0 | **MDPII4_35** | 14 | 24,415 | 6 | 40 | 5 |
| **MDPII5_25** | 10 | 20,790 | 1 | 15 | 0 | **MDPII5_35** | 12 | 23,858 | 9 | 121 | 9 |
| **MDPII6_25** | 9 | 20,174 | 1 | 19 | 0 | **MDPII6_35** | 13 | 24,673 | 8 | 27 | 4 |
| **MDPII7_25** | 10 | 19,947 | 1 | 27 | 0 | **MDPII7_35** | 13 | 29,394 | 5 | 23 | 3 |
| **MDPII8_25** | 10 | 23,921 | 1 | 8 | 0 | **MDPII8_35** | 11 | 25,217 | 5 | 24 | 4 |
| **MDPII9_25** | 10 | 25,016 | 1 | 8 | 0 | **MDPII9_35** | 12 | 27,435 | 7 | 8 | 2 |
| **MDPII10_25** | 12 | 23,575 | 1 | 10 | 0 | **MDPII10_35** | 12 | 25,713 | 10 | 56 | 6 |

Table 5.3: Results on [76]'s small instances

| | Optimum | | IMA | SDP B&B | | | Optimum | | IMA | SDP B&B | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Instance** | **m** | **value** | **T**(s) | **Nodes** | **T**(s) | **Instance** | **m** | **value** | **T**(s) | **Nodes** | **T**(s) |
| **MDPI1_50** | 16 | 25,742 | 124 | 36 | 31 | **MDPII1_75** | 20 | 38,511 | >36000 | 365 | 721 |
| **MDPI2_50** | 16 | 24,362 | 75 | 27 | 29 | **MDPII2_75** | 26 | 40,982 | >36000 | 1775 | 1180 |
| **MDPI3_50** | 15 | 23,298 | 236 | 65 | 41 | **MDPII3_75** | 25 | 36,843 | >36000 | 1483 | 1899 |
| **MDPI4_50** | 13 | 21,321 | 216 | 103 | 47 | **MDPII4_75** | 22 | 38,359 | >36000 | 1504 | 1749 |
| **MDPI5_50** | 13 | 21,291 | 213 | 174 | 64 | **MDPII5_75** | 22 | 38,576 | >36000 | 273 | 866 |
| **MDPI6_50** | 17 | 26,314 | 68 | 23 | 26 | **MDPII6_75** | 32 | 44,099 | >36000 | 123 | 437 |
| **MDPI7_50** | 13 | 22,639 | 150 | 144 | 58 | **MDPII7_75** | 24 | 39,403 | >36000 | 301 | 673 |
| **MDPI8_50** | 15 | 23,386 | 307 | 230 | 71 | **MDPII8_75** | 29 | 42,424 | >36000 | 272 | 593 |
| **MDPI9_50** | 17 | 24,594 | 109 | 37 | 32 | **MDPII9_75** | 28 | 38,673 | 35493 | 543 | 993 |
| **MDPI10_50** | 13 | 21,039 | 232 | 137 | 54 | **MDPII10_75** | 28 | 42,302 | 12524 | 120 | 465 |
| **MDPII1_50** | 14 | 28,587 | 245 | 105 | 48 | **MDPI1_100** | 29 | 34,710 | >36000 | 213 | 2893 |
| **MDPII2_50** | 18 | 31,283 | 118 | 57 | 36 | **MDPI2_100** | 28 | 35,246 | >36000 | 3029 | 11954 |
| **MDPII3_50** | 17 | 30,701 | 126 | 119 | 47 | **MDPI3_100** | 30 | 36,172 | >36000 | 749 | 5237 |
| **MDPII4_50** | 14 | 29,892 | 162 | 37 | 31 | **MDPI4_100** | 29 | 37,226 | >36000 | 138 | 2285 |
| **MDPII5_50** | 19 | 33,934 | 401 | 204 | 61 | **MDPI5_100** | 34 | 34,651 | >36000 | 16993 | 63686 |
| **MDPII6_50** | 15 | 31,219 | 240 | 174 | 56 | **MDPI6_100** | 36 | 30,691 | >36000 | 5799 | 32580 |
| **MDPII7_50** | 15 | 33,394 | 116 | 37 | 31 | **MDPI7_100** | 27 | 35,040 | >36000 | 7788 | 28681 |
| **MDPII8_50** | 16 | 35,934 | 87 | 22 | 26 | **MDPI8_100** | 25 | 33,390 | >36000 | 2088 | 10570 |
| **MDPII9_50** | 19 | 30,410 | 358 | 241 | 72 | **MDPI9_100** | 32 | 35,725 | >36000 | 2377 | 11266 |
| **MDPII10_50** | 17 | 29,811 | 65 | 108 | 49 | **MDPI10_100** | 34 | 35,167 | >36000 | 1060 | 6325 |
| **MDPI1_75** | 30 | 34,113 | >36000 | 121 | 439 | **MDPII1_100** | 26 | 44,731 | >36000 | 1978 | 10015 |
| **MDPI2_75** | 26 | 31,045 | >36000 | 255 | 592 | **MDPII2_100** | 34 | 43,142 | >36000 | 3026 | 13235 |
| **MDPI3_75** | 23 | 27,750 | >36000 | 148 | 509 | **MDPII3_100** | 24 | 48,827 | >36000 | 2082 | 8375 |
| **MDPI4_75** | 22 | 32,885 | 23830 | 72 | 384 | **MDPII4_100** | 38 | 50,972 | >36000 | 506 | 3456 |
| **MDPI5_75** | 27 | 28,536 | >36000 | 318 | 655 | **MDPII5_100** | 35 | 44,749 | >36000 | 30130 | 65076 |
| **MDPI6_75** | 26 | 27,743 | >36000 | 451 | 909 | **MDPII6_100** | 36 | 46,135 | >36000 | 2271 | 16017 |
| **MDPI7_75** | 25 | 29,445 | >36000 | 408 | 759 | **MDPII7_100** | 28 | 45,411 | >36000 | 928 | 5743 |
| **MDPI8_75** | 36 | 37,173 | >36000 | 100 | 392 | **MDPII8_100** | 32 | 43,664 | >36000 | 1329 | 9613 |
| **MDPI9_75** | 30 | 29,395 | >36000 | 343 | 714 | **MDPII9_100** | 32 | 50,435 | >36000 | 830 | 7841 |
| **MDPI10_75** | 26 | 30,393 | 34894 | 76 | 395 | **MDPII10_100** | 33 | 49,252 | >36000 | 3389 | 12423 |

Table 5.4: Results on [76]'s medium instances

| | IMA | SDP B&B | |
|---|---|---|---|
| **n** | $\mathbf{T}_{avg}(s)$ | **Average Nodes** | $\mathbf{T}_{avg}(s)$ |
| **20** | $\leq$1 | 6,65 | $\leq$1 |
| **25** | $\leq$1 | 11,5 | $\leq$1 |
| **30** | $\approx$ 2 | 21,85 | $\approx$ 2 |
| **35** | $\approx$ 6 | 40,9 | $\approx$ 5 |
| **50** | $\approx$ 182 | 45,5 | $\approx$ 104 |
| **75** | $\geq$36000 | 766,2 | $\approx$ 453 |
| **100** | $\geq$36000 | 16363,55 | $\approx$ 4335 |

Table 5.5: Results grouped by instance size

### 5.4.2 Tests for the Hybrid Heuristic

Preliminary tests were conducted with respect to the parameter settings of the hybrid heuristic. These tests were performed on the instances of [76] for $n = 500$. The CPU time limit for executing the three phases in cascade was set to 600 seconds. The tests indicated that the best results were obtained by setting $k = 5$ for phase 1, $\delta = 16$ for phase 2 and CPU time split within the phases assigning 240 seconds to phase 1, 240 seconds to phase 2 and 120 seconds to phase 3 (from now denoted as time distribution 1), that is 2/5 of the time available assigned to phase 1, 2/5 assigned to phase 2 and 1/5 assigned to phase 3.

To assess the validity of these settings, we checked the performances of the algorithm by perturbing the parameters of one phase only at a time. We do not provide the results of these experiments on phase 1 as, for $2 \leq k \leq 10$, the results were substantially the same while for larger $k$ the results were progressively worse. This can be explained by the fact that, for different $k \leq 10$, even if the exploration of the search tree is different, the best $m$ of this phase is sooner or later encountered. We sticked then to $k = 5$ for the rest of the experimentation.

Table 5.6 provides the output of phase 2 for different values of $\delta$ (with $k = 5$ in phase 1 and 240 seconds allowed to phase 1 and phase 2, respectively) in the range $2 \leq \delta \leq 64$. It is shown that the results are quite similar for $2 \leq \delta \leq 32$ while a degradation occurs for larger values. The best value is obtained for $\delta = 16$.

Table 5.7 and Table 5.8 provide the output of the three phases for different distributions of time among the phases with $k = 5$, $\delta = 16$ and total CPU time

| Instances | $\delta = 2$ | $\delta = 4$ | $\delta = 8$ | $\delta = 16$ | $\delta = 32$ | $\delta = 64$ |
|---|---|---|---|---|---|---|
| **MDPI1_500** | 81,21 | 81,25 | 81,28 | 81,25 | 81,05 | 79,73 |
| **MDPI2_500** | 77,31 | 77,71 | 77,74 | 77,74 | 77,71 | 76,68 |
| **MDPI3_500** | 75,11 | 75,34 | 75,74 | 76,26 | 76,09 | 74,53 |
| **MDPI4_500** | 82,06 | 82,23 | 82,24 | 82,22 | 82,17 | 81,11 |
| **MDPI5_500** | 80,01 | 80,02 | 80,04 | 80,01 | 79,98 | 79,09 |
| **MDPI6_500** | 81,10 | 81,25 | 81,25 | 81,20 | 81,17 | 80,47 |
| **MDPI7_500** | 78,03 | 77,99 | 78,16 | 78,16 | 77,87 | 76,70 |
| **MDPI8_500** | 78,87 | 78,84 | 78,99 | 79,03 | 78,84 | 78,37 |
| **MDPI9_500** | 76,99 | 77,05 | 77,16 | 77,13 | 76,98 | 76,35 |
| **MDPI10_500** | 81,08 | 81,23 | 81,24 | 81,24 | 81,02 | 80,61 |
| **MDPII1_500** | 109,17 | 109,21 | 109,26 | 109,24 | 109,38 | 108,49 |
| **MDPII2_500** | 105,03 | 105,03 | 105,09 | 105,19 | 104,96 | 103,45 |
| **MDPII3_500** | 107,59 | 107,64 | 107,77 | 107,76 | 107,33 | 106,24 |
| **MDPII4_500** | 105,08 | 105,28 | 105,42 | 105,61 | 105,93 | 104,38 |
| **MDPII5_500** | 106,33 | 106,33 | 106,55 | 106,48 | 106,27 | 105,87 |
| **MDPII6_500** | 105,10 | 105,52 | 105,52 | 105,76 | 105,74 | 104,08 |
| **MDPII7_500** | 106,65 | 106,36 | 106,77 | 106,90 | 106,67 | 104,82 |
| **MDPII8_500** | 103,43 | 103,43 | 103,41 | 103,66 | 103,28 | 103,28 |
| **MDPII9_500** | 106,17 | 106,17 | 106,24 | 106,24 | 105,98 | 105,30 |
| **MDPII10_500** | 103,80 | 104,13 | 104,15 | 104,06 | 103,70 | 102,97 |
| **Total** | 1850,13 | 1852,01 | 1854,00 | 1855,10 | 1852,12 | 1832,51 |

Table 5.6: Results of phase 2 on the instances of [76] with $n = 500$ for different values of $\delta$ with $k = 5$ and time distribution 1

| | Time distribution 1 | | | Time distribution 2 | | |
|---|---|---|---|---|---|---|
| | PHASE1 | PHASE2 | PHASE3 | PHASE1 | PHASE2 | PHASE3 |
| **Instances** | 240,00 | 240,00 | 120,00 | 120,00 | 240,00 | 240,00 |
| **MDPI1_500** | 79,73 | 81,25 | 81,25 | 79,73 | 81,25 | 81,25 |
| **MDPI2_500** | 76,59 | 77,74 | 77,77 | 76,59 | 77,52 | 77,64 |
| **MDPI3_500** | 74,53 | 76,26 | 76,30 | 74,53 | 75,74 | 76,07 |
| **MDPI4_500** | 81,11 | 82,22 | 82,33 | 81,10 | 82,18 | 82,27 |
| **MDPI5_500** | 79,09 | 80,01 | 80,08 | 79,09 | 80,01 | 80,08 |
| **MDPI6_500** | 80,46 | 81,20 | 81,25 | 80,46 | 81,20 | 81,25 |
| **MDPI7_500** | 76,70 | 78,16 | 78,16 | 76,70 | 77,80 | 78,16 |
| **MDPI8_500** | 78,37 | 79,03 | 79,06 | 78,37 | 78,99 | 79,03 |
| **MDPI9_500** | 76,35 | 77,13 | 77,30 | 76,35 | 77,13 | 77,28 |
| **MDPI10_500** | 80,61 | 81,24 | 81,25 | 80,61 | 81,24 | 81,25 |
| **MDPII1_500** | 108,49 | 109,24 | 109,30 | 108,49 | 109,23 | 109,30 |
| **MDPII2_500** | 103,45 | 105,19 | 105,30 | 103,45 | 105,00 | 105,12 |
| **MDPII3_500** | 106,24 | 107,76 | 107,79 | 106,24 | 107,76 | 107,79 |
| **MDPII4_500** | 103,63 | 105,61 | 106,10 | 103,63 | 105,71 | 106,10 |
| **MDPII5_500** | 105,87 | 106,48 | 106,54 | 105,87 | 106,48 | 106,54 |
| **MDPII6_500** | 104,08 | 105,76 | 105,77 | 104,08 | 105,61 | 105,61 |
| **MDPII7_500** | 104,82 | 106,90 | 106,92 | 104,82 | 106,74 | 107,06 |
| **MDPII8_500** | 103,28 | 103,66 | 103,78 | 103,28 | 103,66 | 103,78 |
| **MDPII9_500** | 105,30 | 106,21 | 106,24 | 105,30 | 106,21 | 106,24 |
| **MDPII10_500** | 102,88 | 104,06 | 104,10 | 102,88 | 104,04 | 104,10 |
| **Total** | 1831,57 | 1855,08 | 1856,60 | 1831,56 | 1853,46 | 1855,92 |

Table 5.7: Results on the instances of [76] of Type II with $n = 500$ for different distributions

| | Time distribution 3 | | | Time distribution 4 | |
|---|---|---|---|---|---|
| | PHASE1 | PHASE2 | PHASE3 | PHASE1 | PHASE3 |
| **Instances** | 360,00 | 120,00 | 120,00 | 240,00 | 360 |
| **MDPI1_500** | 79,73 | 81,25 | 81,25 | 79,73 | 80,27 |
| **MDPI2_500** | 76,59 | 77,28 | 77,28 | 76,59 | 77,01 |
| **MDPI3_500** | 74,53 | 76,26 | 76,26 | 74,53 | 74,93 |
| **MDPI4_500** | 81,11 | 82,18 | 82,18 | 81,11 | 81,81 |
| **MDPI5_500** | 79,09 | 80,01 | 80,01 | 79,09 | 79,86 |
| **MDPI6_500** | 80,46 | 81,20 | 81,20 | 80,46 | 80,78 |
| **MDPI7_500** | 76,70 | 78,16 | 78,16 | 76,70 | 77,53 |
| **MDPI8_500** | 78,37 | 78,99 | 78,99 | 78,37 | 78,73 |
| **MDPI9_500** | 76,35 | 77,13 | 77,13 | 76,35 | 76,81 |
| **MDPI10_500** | 80,61 | 81,24 | 81,24 | 80,61 | 81,07 |
| **MDPII1_500** | 108,49 | 109,24 | 109,24 | 108,49 | 108,89 |
| **MDPII2_500** | 103,45 | 105,05 | 105,05 | 103,45 | 104,52 |
| **MDPII3_500** | 106,24 | 107,76 | 107,76 | 106,24 | 107,17 |
| **MDPII4_500** | 103,63 | 105,71 | 105,71 | 103,63 | 104,53 |
| **MDPII5_500** | 105,87 | 106,48 | 106,48 | 105,87 | 106,37 |
| **MDPII6_500** | 104,08 | 105,61 | 105,61 | 104,08 | 104,96 |
| **MDPII7_500** | 104,82 | 106,90 | 106,90 | 104,82 | 105,80 |
| **MDPII8_500** | 103,28 | 103,43 | 103,43 | 103,28 | 103,43 |
| **MDPII9_500** | 105,30 | 106,21 | 106,21 | 105,30 | 105,92 |
| **MDPII10_500** | 102,88 | 104,04 | 104,04 | 102,88 | 103,37 |
| **Total** | 1831,57 | 1854,09 | 1854,09 | 1831,57 | 1843,75 |

Table 5.8: Results on the instances of [76] of Type II with $n = 500$ for different distributions

equal to 600 s. For any time distribution tested, time limits for each phase are reported. Best results were reached with time distribution 1. Notice that deleting phase 2 (time distribution 4) or phase 3 (results not presented here) leads to worse results. The final parameters setting for the hybrid heuristic was $k = 5$, $\delta = 16$ and time distribution 1. We tested then the behaviour of the hybrid heuristic on the whole dataset of [76] with $n \geq 150$ for different CPU time limits, namely 15, 60 and 120 seconds for $n = 150$ and 150, 600 and 1200 seconds for $n = 500$. We denote as TOTCPU-1 the case where the CPU time limit is 15 for the instances with $n = 150$ and 150 for the instances with $n = 500$. Then, we denote as TOTCPU-2 the case where the CPU time limit is 60 for the instances with $n = 150$ and 600 for the instances with $n = 500$. Finally, we denote as TOTCPU-3 the case where the CPU time limit is 120 for the instances with $n = 150$ and 1200 for the instances with $n = 500$.

Table 5.9 provides the related results. It shows a very minor improvements in quality from TOTCPU-1 to TOTCPU-3 even though the times are 8 eight times larger in the latter case. From the table, we notice that PHASE TWO leads to

| | TOTCPU-1 | | | TOTCPU-2 | | | TOTCPU-3 | | |
|---|---|---|---|---|---|---|---|---|---|
| **Instances** | PHASE 1 | PHASE 2 | PHASE 3 | PHASE 1 | PHASE 2 | PHASE 3 | PHASE 1 | PHASE 2 | PHASE 3 |
| **MDPI1_150** | 45,89 | 45,89 | 45,92 | 45,89 | 45,89 | 45,92 | 45,89 | 45,92 | 45,92 |
| **MDPI2_150** | 43,39 | 43,39 | 43,39 | 43,39 | 43,39 | 43,39 | 43,39 | 43,39 | 43,39 |
| **MDPI3_150** | 39,89 | 39,91 | 40,00 | 39,89 | 39,91 | 40,04 | 39,89 | 39,91 | 40,04 |
| **MDPI4_150** | 43,88 | 43,98 | 44,04 | 43,88 | 44,04 | 44,04 | 43,88 | 44,04 | 44,04 |
| **MDPI5_150** | 42,48 | 42,48 | 42,48 | 42,48 | 42,48 | 42,48 | 42,48 | 42,48 | 42,48 |
| **MDPI6_150** | 43,55 | 43,57 | 43,72 | 43,55 | 43,64 | 43,72 | 43,55 | 43,64 | 43,72 |
| **MDPI7_150** | 45,99 | 45,99 | 46,08 | 45,99 | 45,99 | 46,08 | 45,99 | 45,99 | 46,08 |
| **MDPI8_150** | 42,14 | 42,26 | 42,45 | 42,29 | 42,35 | 42,45 | 42,29 | 42,35 | 42,43 |
| **MDPI9_150** | 41,78 | 41,78 | 41,82 | 41,82 | 41,82 | 41,82 | 41,82 | 41,82 | 41,82 |
| **MDPI10_150** | 41,76 | 41,76 | 41,80 | 41,80 | 41,80 | 41,80 | 41,80 | 41,80 | 41,80 |
| **MDPII1_150** | 57,48 | 57,48 | 57,48 | 57,48 | 57,48 | 57,48 | 57,48 | 57,48 | 57,48 |
| **MDPII2_150** | 57,34 | 57,46 | 57,82 | 57,47 | 57,62 | 57,82 | 57,47 | 57,64 | 57,80 |
| **MDPII3_150** | 57,42 | 58,02 | 58,42 | 58,42 | 58,42 | 58,42 | 58,42 | 58,42 | 58,42 |
| **MDPII4_150** | 56,11 | 56,28 | 57,38 | 56,51 | 57,06 | 57,22 | 56,51 | 57,06 | 57,38 |
| **MDPII5_150** | 54,09 | 54,09 | 54,14 | 54,09 | 54,09 | 54,14 | 54,09 | 54,09 | 54,23 |
| **MDPII6_150** | 56,44 | 56,44 | 56,44 | 56,44 | 56,44 | 56,44 | 56,44 | 56,44 | 56,44 |
| **MDPII7_150** | 58,43 | 58,43 | 58,60 | 58,47 | 58,47 | 58,60 | 58,47 | 58,47 | 58,77 |
| **MDPII8_150** | 57,97 | 57,97 | 57,97 | 57,97 | 57,97 | 57,97 | 57,97 | 57,97 | 57,97 |
| **MDPII9_150** | 58,06 | 58,14 | 58,30 | 58,06 | 58,14 | 58,30 | 58,06 | 58,14 | 58,30 |
| **MDPII10_150** | 56,18 | 56,55 | 57,06 | 56,18 | 56,84 | 57,18 | 56,18 | 56,84 | 57,18 |
| **MDPI1_500** | 79,73 | 81,18 | 81,25 | 79,73 | 81,25 | 81,25 | 79,73 | 81,25 | 81,25 |
| **MDPI2_500** | 76,59 | 77,34 | 77,56 | 76,59 | 77,74 | 77,77 | 76,59 | 77,74 | 77,79 |
| **MDPI3_500** | 74,40 | 75,57 | 75,99 | 74,53 | 76,26 | 76,30 | 74,53 | 76,16 | 76,30 |
| **MDPI4_500** | 81,05 | 82,15 | 82,27 | 81,11 | 82,22 | 82,33 | 81,11 | 82,22 | 82,33 |
| **MDPI5_500** | 79,09 | 79,91 | 80,02 | 79,09 | 80,01 | 80,08 | 79,09 | 80,01 | 80,08 |
| **MDPI6_500** | 80,46 | 81,20 | 81,25 | 80,46 | 81,20 | 81,25 | 80,46 | 81,20 | 81,25 |
| **MDPI7_500** | 76,68 | 77,78 | 78,16 | 76,70 | 78,16 | 78,16 | 76,70 | 78,16 | 78,16 |
| **MDPI8_500** | 78,37 | 78,93 | 78,99 | 78,37 | 79,03 | 79,06 | 78,37 | 79,03 | 79,06 |
| **MDPI9_500** | 76,35 | 76,95 | 77,04 | 76,35 | 77,13 | 77,30 | 76,35 | 77,32 | 77,36 |
| **MDPI10_500** | 80,54 | 80,99 | 81,05 | 80,61 | 81,24 | 81,25 | 80,61 | 81,24 | 81,25 |
| **MDPII1_500** | 108,49 | 109,03 | 109,23 | 108,49 | 109,24 | 109,30 | 108,49 | 109,24 | 109,30 |
| **MDPII2_500** | 103,45 | 104,83 | 105,12 | 103,45 | 105,19 | 105,30 | 103,45 | 105,19 | 105,33 |
| **MDPII3_500** | 105,96 | 107,62 | 107,79 | 106,24 | 107,76 | 107,79 | 106,24 | 107,76 | 107,79 |
| **MDPII4_500** | 103,47 | 105,25 | 105,49 | 103,63 | 105,61 | 106,10 | 103,63 | 105,71 | 106,10 |
| **MDPII5_500** | 105,87 | 106,32 | 106,37 | 105,87 | 106,48 | 106,54 | 105,87 | 106,45 | 106,54 |
| **MDPII6_500** | 104,08 | 105,51 | 105,61 | 104,08 | 105,76 | 105,77 | 104,08 | 105,76 | 105,77 |
| **MDPII7_500** | 104,82 | 106,90 | 106,92 | 104,82 | 106,90 | 106,92 | 104,82 | 106,90 | 107,06 |
| **MDPII8_500** | 103,28 | 103,43 | 103,48 | 103,28 | 103,66 | 103,78 | 103,28 | 103,66 | 103,78 |
| **MDPII9_500** | 105,30 | 106,14 | 106,24 | 105,30 | 106,21 | 106,24 | 105,30 | 106,21 | 106,24 |
| **MDPII10_500** | 102,24 | 103,80 | 104,02 | 102,88 | 104,06 | 104,10 | 102,88 | 104,06 | 104,10 |

Table 5.9: Results on the large instances for different CPU time limits

increased solution quality with respect to PHASE ONE. Also, PHASE THREE is capable of further improving the solutions of PHASE TWO.

We compared then the hybrid three-phase heuristic with parameters setting $k = 5$, $\delta = 16$, time distribution 1 and TOTCPU-1 as CPU time limit to the GRASP with path relinking proposed in [76] and the XPRESS-SLP non-linear solver applied to the Standard Formulation 1. The results are presented in Table 5.10. GRASP with path relinking was tested on an Intel Core Solo 1.4GHz with 3GB of RAM, while we recall that the other approaches were tested on an Intel Core i5-3550 3.30GHz with 4GB of RAM. Even though an exact comparison between the two processors is not

possible, we can evince from [4] that the latter machine is somewhat in between 3 and 4 times faster than the first one. XPRESS-SLP was applied in its default version with no time limits where the results obtained correspond to a local maximum reached by the solver.

| | GRASP [76] | | | XPRESS-SLP | | | Hybrid algorithm | | |
|---|---|---|---|---|---|---|---|---|---|
| **Instances** | **best value** | **m** | **time (s)** | **best value** | **m** | **time (s)** | **best value** | **m** | **time (s)** |
| **MDPI1_150** | **45,92** | 53 | 83 | 45,55 | 52 | 5 | **45,92** | 52 | 15 |
| **MDPI2_150** | 43,33 | 41 | 72 | 43,03 | 59 | 12 | **43,39** | 42 | 15 |
| **MDPI3_150** | 39,64 | 43 | 60 | 38,38 | 58 | 13 | **40,00** | 49 | 15 |
| **MDPI4_150** | 43,70 | 57 | 76 | 43,76 | 58 | 12 | **44,04** | 58 | 15 |
| **MDPI5_150** | **42,48** | 49 | 71 | 40,63 | 55 | 11 | **42,48** | 49 | 15 |
| **MDPI6_150** | 43,67 | 40 | 73 | 43,27 | 53 | 10 | **43,72** | 44 | 15 |
| **MDPI7_150** | **46,08** | 53 | 60 | 45,44 | 52 | 8 | **46,08** | 52 | 15 |
| **MDPI8_150** | 42,39 | 45 | 61 | 41,71 | 53 | 12 | **42,45** | 44 | 15 |
| **MDPI9_150** | **42,14** | 42 | 64 | 41,49 | 53 | 22 | 41,82 | 46 | 15 |
| **MDPI10_150** | **41,80** | 41 | 55 | **41,80** | 41 | 23 | **41,80** | 41 | 15 |
| **MDPII1_150** | 56,72 | 49 | 62 | 55,95 | 59 | 5 | **57,48** | 50 | 15 |
| **MDPII2_150** | 57,80 | 47 | 61 | 56,18 | 54 | 14 | **57,82** | 46 | 15 |
| **MDPII3_150** | 58,28 | 45 | 59 | 57,49 | 50 | 15 | **58,42** | 44 | 15 |
| **MDPII4_150** | **57,38** | 47 | 59 | 55,21 | 60 | 10 | **57,38** | 47 | 15 |
| **MDPII5_150** | **54,23** | 42 | 48 | 51,05 | 54 | 7 | 54,14 | 35 | 15 |
| **MDPII6_150** | **56,44** | 49 | 58 | 55,10 | 55 | 9 | **56,44** | 49 | 15 |
| **MDPII7_150** | **58,89** | 48 | 61 | 57,62 | 54 | 10 | 58,60 | 40 | 15 |
| **MDPII8_150** | **57,97** | 54 | 66 | **57,97** | 54 | 14 | **57,97** | 54 | 15 |
| **MDPII9_150** | **58,30** | 42 | 56 | 55,84 | 52 | 9 | **58,30** | 42 | 15 |
| **MDPII10_150** | 56,92 | 39 | 55 | 54,25 | 52 | 20 | **57,06** | 41 | 15 |
| **MDPI1_500** | 78,60 | 152 | 716 | 77,23 | 175 | 135 | **81,25** | 158 | 150 |
| **MDPI2_500** | 76,87 | 150 | 682 | 75,17 | 177 | 262 | **77,56** | 170 | 150 |
| **MDPI3_500** | 75,69 | 128 | 668 | 73,86 | 172 | 182 | **75,99** | 136 | 150 |
| **MDPI4_500** | 81,81 | 166 | 647 | 79,89 | 185 | 277 | **82,27** | 144 | 150 |
| **MDPI5_500** | 78,57 | 140 | 683 | 78,75 | 170 | 339 | **80,02** | 148 | 150 |
| **MDPI6_500** | 79,64 | 156 | 732 | 77,40 | 169 | 246 | **81,25** | 152 | 150 |
| **MDPI7_500** | 75,50 | 146 | 607 | 74,18 | 182 | 221 | **78,16** | 133 | 150 |
| **MDPI8_500** | 76,98 | 151 | 666 | 78,22 | 176 | 294 | **78,99** | 176 | 150 |
| **MDPI9_500** | 75,72 | 128 | 635 | 75,45 | 175 | 317 | **77,04** | 159 | 150 |
| **MDPI10_500** | 80,38 | 137 | 849 | 80,09 | 159 | 370 | **81,05** | 149 | 150 |
| **MDPII1_500** | 108,15 | 165 | 766 | 107,78 | 174 | 234 | **109,23** | 163 | 150 |
| **MDPII2_500** | 103,29 | 121 | 656 | 103,46 | 169 | 275 | **105,12** | 125 | 150 |
| **MDPII3_500** | 106,30 | 140 | 710 | 101,74 | 206 | 267 | **107,79** | 161 | 150 |
| **MDPII4_500** | 104,62 | 154 | 725 | 101,01 | 197 | 111 | **105,49** | 158 | 150 |
| **MDPII5_500** | 103,61 | 149 | 707 | 103,85 | 182 | 253 | **106,37** | 161 | 150 |
| **MDPII6_500** | 104,81 | 158 | 713 | 102,57 | 169 | 229 | **105,61** | 167 | 150 |
| **MDPII7_500** | 104,50 | 148 | 626 | 103,26 | 181 | 335 | **106,92** | 149 | 150 |
| **MDPII8_500** | 100,02 | 135 | 609 | 99,28 | 178 | 245 | **103,48** | 153 | 150 |
| **MDPII9_500** | 104,93 | 130 | 636 | 104,58 | 177 | 239 | **106,24** | 141 | 150 |
| **MDPII10_500** | 103,50 | 144 | 649 | 102,13 | 176 | 428 | **104,02** | 160 | 150 |

Table 5.10: Comparing GRASP, XPRESS SLP and the hybrid algorithm on the large instances

The solutions provided by XPRESS-SLP had a low quality, thing which indicates that the solver generally gets stuck in low quality local maxima. The GRASP approach proposed in [76] was globally dominated by our heuristic both for $n = 150$ and $n = 500$ (taking into account the different performances between the machine used by GRASP and the one used by the hybrid algorithm, the CPU times were roughly the same). Also, Table 5.10 shows that the dominance increased as the instance size grows. Indeed, for $n = 150$ the GRASP approach outperformed our heuristic on 3 instances while it was outperformed in 9 instances and reached the same result in 8 instances. For $n = 500$ the GRASP approach was outperformed by the hybrid heuristic on all instances. Finally, from the tests we evince that the instance type did not affect the performances of the three approaches.

As a further validation of the quality of the solutions provided by the three-phase heuristic, for each instance we checked the robustness of the best solution found after PHASE THREE by performing a local search based on a 3-opt neighborhood. We defined this 3-opt neighborhood of a solution $\widehat{\mathbf{x}}^*$ as the set given by the solutions that can be obtained by means of one, two or three moves of insertion/deletion of an item. While this neighborhood has a large size (i.e. $\mathcal{O}(n^3)$), our tests revealed that in one case only (instance MDPI1_500) the 3-opt neighborhood search was able to obtain an improved solution (81.2751).

## 5.5 Conclusions

This chapter proposed an SDP based branch and bound algorithm and a three-phase hybrid heuristic for the $Max - MeanDP$.

The results showed that the proposed exact method outperforms the best performing exact method known for the problem. The proposed method was able to solve instances up to 100 elements, that were not solvable by previous methods. In addition, the presented branch and bound method can be extended to other fractional problems that have a similar formulation, for which a similar SDP relaxation can be derived. Future research will be devoted to find other successful applications of the proposed method for other combinatorial problems.

The matheuristic embeds three main ingredients: the use of a QIP solver, a local branching scheme and a path relinking enhancement procedure. The first

phase selects a set of promising values for the cardinality of an unknown subset and generates a pool of initial solutions by means of the QIP solver. The second and third phases enhance the quality of the solutions provided by the first phase by using local branching and path relinking. As shown above, this hybrid heuristic globally outperforms the best state of the art heuristic. In our opinion, the performance of the proposed approach indicates that QIP models and related solvers can also be taken into consideration in the design of hybrid heuristics for other combinatorial optimization problems that can be modeled by means of a QIP formulation.

# Chapter 6

# The Multi-Meter Covering Problem

This chapter deals with a problem that arises in a smart city context, where concentrators and meters are spread along the city area. These devices communicate through a point-to-multi-point connection where each concentrator is capable of receiving data from a heterogeneous set of meters. Currently, the number of meters in the towns is going to increase and all meters require a connection. The goal is to minimize the global cost, that is, to minimize the weighted sum of activated concentrators, where each weight corresponds to the activation cost of the concentrator.

Such devices may work at different frequencies (169MHz, 868MHz, 2.4GHz) with distinct propagation features which induce different coverage areas. The concentrators can be placed in two different types of locations, already established along the city area. We refer to such locations as cabinets and base stations. As a consequence of their technological properties, we observe that the same concentrator placed in a different location (cabinet or base station) is capable of covering a significantly different area. In general, the information on the expected coverage area is derived from a simplified model of propagation, validated by Telecom Italia with field measurements.

The aim of this work is to formulate the problem in a formal way and to design an approach which provides solutions directly usable in real contexts. In particular, the focus is on two Italian cities (Asti and Torino) for which real data will be considered

and an estimate of the overall cost will be derived. The proposed approach is able to determine both solutions where the maximal coverage is attained and solutions where the coverage is very close to the maximum.

In Section 6.1, the problem is formally described. The mathematical formulation and the literature review are then discussed in Section 6.2 and Section 6.3. The proposed algorithm is carefully described in Section 6.4 and computational results are presented in Section 6.5 where the obtained results are compared with the ones provided by a built in-house approach implemented by Telecom staff.

## 6.1 Problem Description

Consider a set $N$ of $n$ concentrators, each concentrator $c \in N$ is placed in the geographical coordinates $\alpha_c$ and $\beta_c$ which respectively represent the longitude and the latitude where the concentrator is placed. Each concentrator has a coverage radius $r_c$, meaning that it can cover a sphere of radius $r_c$ in first approximation. This radius is determined as a function of the location and of the frequency. These concentrators cannot be moved, so their geographical coordinates are constant data inputs. As we mentioned, the final aim of the company is to attain the maximal covered area by minimizing the costs, thus the number of concentrators used.

Some assumptions are needed in order to define and formulate the problem in a mathematical way. First of all, the input coordinates represent points on the Earth's surface which are hard to be handled. For this reason, we use Haversine [6] formula to obtain a good projection of the points with a negligible approximation error according to the specifications provided by Telecom staff (see in Algorithm 15 the relevant MATLAB code by Telecom). The points are thus projected on a plane, so as to formulate the whole problem in two dimensions. In this context, the area covered by a concentrator is a circle whose center is given by the projected coordinates of the point, while the radius is the same mentioned before. We indicate as $x_c$ and $y_c$ the projection of respectively $\alpha_c$ and $\beta_c$ for any $c \in N$ (all input data provided by TELECOM already include $x_c$ and $y_c$ projection coordinates) Another important assumption is about the number of connections that a single concentrators can handle. In fact, multiple meters can use the same concentrator and the technology allows to use a limited (although high) number of connections. However,

the maximum number of connections is sufficiently large in practice and thus we do not consider this kind of limitation.

**Algorithm 15** MATLAB code for the projection of the geographical coordinates

```
function [x, y] = geo2km(latlon)
    minlat = min(latlon(:, 1));
    minlon = min(latlon(:, 2));
    maxlat = max(latlon(:, 1));
    maxlon = max(latlon(:, 2));
    dy = lat2m(maxlat - minlat, mean(latlon(:, 2)));
    dx = lon2m(maxlon - minlon, mean(latlon(:, 1)));
    dlon = maxlon - minlon;
    dlat = maxlat - minlat;
    y = 1e-3 * dy * (latlon(:, 1) - minlat) / dlat;
    x = 1e-3 * dx * (latlon(:, 2) - minlon) / dlon;

    function dy = lat2m(dlat,alat)
        rlat = alat * pi/180;
        m = 111132.09 + zeros(size(rlat)) - ...
            566.05 * cos(2 * rlat) + 1.2 * cos(4 * rlat);
        dy = dlat .* m;
    end

    function dx = lon2m(dlon, alat)
        % dx = lon_to_m(dlon, alat)
        % dx   = longitude difference in meters
        % dlon = longitude difference in degrees
        % alat = average latitude between the two fixes
        rlat = alat * pi/180;
        p = 111415.13 * cos(rlat) - 94.55 * cos(3 * rlat);
        dx = dlon .* p;
    end
```

## 6.2 Mathematical Formulation

In the following, we indicate with $A_c$ the set of points that can be covered by a concentrator $c \in N$ and with $A = \cup_{c \in N} A_c$ the overall area that can be covered. The problem can be formulated as a covering problem where we want to minimize the weighted sum of the used concentrators (each weight $w_c$ refers to the activation cost for the concentrator) and we impose that any point in $A$ is covered by at least one concentrator. Given a point $p \in A$, we indicate with $C_p$ the set of concentrators that can cover the point $p$, i.e. the set formed by all the concentrators for which the distance between $(x_c, y_c)$ and $p$ is less than or equal to the concentrator radius $r_c$. The only variables that we are going to use are the binary variables $x_c \; \forall c \in N$, each variable $x_c$ is equal to 1 if and only if the concentrator is activated, 0 otherwise. The mathematical formulation we obtain is the following:

$$\min w_c x_c \tag{6.1}$$

$$\sum_{c \in C_p} x_c \geq k \; \forall p \in A \tag{6.2}$$

$$x_c \in \{0,1\} \; \forall c \in N \tag{6.3}$$

The parameter $k$ refers to the degree of coverage that we want to achieve for the area. In fact, we may want to cover each single point for a multiple amount of times, in order to guarantee the robustness of the coverage. For example, we know that if we solve the problem with $k = 2$ and afterwards a concentrator $c$ is out of service, each point of the area $A_c$ is still covered by a concentrator. Hereafter, we refer to the case where $k$ is set to 1, but all the considerations still apply for a general value of $k$.

The formulation above cannot be used in practice because of the infinite number of constraints. However, if the set $A$ is finite, the formulation corresponds to a classical NP-hard combinatorial optimization problem called Weighted Set Cover Problem [47], which has been intensively studied through the years. A simple relaxation of model 6.1 6.3 can be obtained if we consider a set of points $\overline{A} \subset A$:

$$\min w_c x_c \tag{6.4}$$

$$\sum_{c \in C_p} x_c \geq k \ \forall p \in \overline{A} \subset A \tag{6.5}$$

$$x_c \in \{0,1\} \ \forall c \in N \tag{6.6}$$

The optimal solution of this model yields a lower bound on the optimal solution of 6.1 - 6.3. The quality of this lower bound strongly depends on the choice of the set $\overline{A}$, thus on how the points belonging to $A$ are chosen. Roughly speaking, we can say that the more representative of $A$ the points are, the higher is the quality of the lower bound obtained.

## 6.3 Literature

We formulated the problem via a Weighted Set Cover model. Many exact algorithms and heuristics have been designed for this problem through the years. We refer to [101] for a recent contribution on this problem. In general, it is possible to handle instances up to some thousands of variables and constraints. However, the description of the problem has some common insights with two well studied NP-hard problems, the Circle Packing Problem [86] and its dual, the Circle Covering Problem [13]. Basically, we deal with circles as in circle packing and circle covering problems. However, the circles area cannot overlap in circle packing. In simple words, our problem consists of covering the maximal amount of area with the minimum amount of circles. This is the same goal of circle covering, the important difference is about the fact that there is a limited amount of circles (i.e. locations for concentrators) among which we can choose. In general, when we deal with circle covering problems the position where we can place the circles is intended to be variable, and they are usually no constraints on it. We can conclude by saying that, to the best of our knowledge, the problem we deal with has not already been studied in the literature, at least in its current form.

## 6.4 The Proposed Algorithm

The first issue that we encountered in the design of an exact algorithm for this real world problem is the evaluation of the maximal area. In fact, it is not trivial to compute the area of a figure obtained by overlapping circles, as the number of the circles considered increases. There are some deterministic numerical methods which allow to solve this task with a good accuracy (see [9]). The method we adopted is a scan-line method [9]: it is able to solve problems with thousands of circles within several seconds. The first step of the algorithm is the computation of the maximal area $\mathcal{A}_{max}$ that the concentrators can cover. We did this by using a procedure that takes as an input the projected coordinates $(x_c, y_c)$ of each concentrator $c \in N$, and gives as an output the value of the area covered by them. This function is indicated with `DOUBLE AREA(SET OF CONCENTRATORS)`.

One simple property of the problem gives us the chance to reduce the size of the instances. In fact, a concentrator is necessarily put in the optimal solution if it is able to cover at least one point that is not covered by any other concentrator. We indicate the set of such concentrators as $N_{necessary}$. How can we select such concentrators in an efficient way? The answer is relatively simple, we are going to use the `AREA` function. We scan the set of the concentrators $N$ in an arbitrary order and we evaluate the value provided by the call `AREA(`$N/\ c$`)` for each concentrator $c \in N$. If the value obtained is less than $\mathcal{A}_{max}$, then the concentrator $c$ covers a portion of the area which is not covered by any other concentrator in $N$. In this step, all the variables associated with the concentrators included in $N_{necessary}$ are set to 1 without loss of optimality. Since the concentrators are scanned one by one, this operation can be performed in $\mathcal{O}(n)$. Note that we already have an initial lower bound to the optimal solution, given by the sum of the weights of the concentrators in $N_{necessary}$. The computation of the necessary concentrators is done by the function `SET OF CONCENTRATORS GET_NECESSARY(SET OF CONCENTRATORS)`. Generally speaking, it takes as an input a set of concentrators and it provides only the necessary ones to cover the whole area as an output.

Hence, we start a loop, where a set of $\lambda$ points is generated at each iteration. These points are generated via simulation by means of a routine `SET OF POINTS NEW_POINTS(SET OF CONCENTRATORS, SET OF CONCENTRATORS, INT)`. This routine

receives as an input two set of concentrators and the number of points to be generated. It provides as an output a set of $\lambda$ points that are covered by the first set of concentrators, but not by the second one. We define the set $P$ as the set formed by all the points generated at the current iteration. We indicate with $N_{current}$ the set of concentrators which are currently selected, while $\mathcal{A}_{current}$ is the value of the area covered by concentrators in $N_{current}$.

The points are generated in the area which is not covered by any concentrator in $N_{current}$, in order to add as a constraint the coverage of points in $A$ that were not covered at the previous iterations. At this point, we solve the Weighted Set Cover model 6.4 - 6.6 by setting $\overline{A} = P$ and fixing all the variables associated with the necessary concentrators to 1. The model can be solved by means of any ILP solver, this is performed in function `FLOAT SOLVE_SET_COVER(SET OF CONCENTRATORS, SET OF CONCENTRATORS, SET OF POINTS)`. The inputs are the set of necessary concentrators, the set of the concentrators to use and the numbers of points to cover. The output is a floating point value which represents a lower bound for the original problem. A drawback in the use of the ILP model is that the time required for the solution can be too long if the number of variables and constraints of the model is too high. For this reason, we set a time limit to the solver and thus we may not solve model 6.4 - 6.6 to optimality. However, the solver will provide a feasible solution and a lower bound for model 6.4 - 6.6. The lower bound obtained in this way is also a lower bound for the original problem represented by formulation 6.1 - 6.3. On the contrary, the solution provided may not be feasible for the original problem and we need to modify it to make it feasible.

This goal can be reached by using different type of heuristics. Our purpose is "to fill the holes" in the overall area by using the minimal number of concentrators, so we keep all the concentrators in $N_{current}$ and we try to add new concentrators in order to achieve the maximal area as soon as possible. Here it is important to note that local search strategies can hardly be applied to search for better solutions. In fact, the computational cost due to the evaluation of the area is too high if we need $\mathcal{O}(n^2)$ computations. For this reason, we had to design a basic constructive heuristic that is able to obtain in $\mathcal{O}(n)$ a feasible solution. We adopted a greedy approach where at each iteration, only the concentrators which allow an increase of the coverage greater than or equal to a specific threshold, are added. This threshold is halved after each

iteration and the procedure ends as soon as the maximal coverage is achieved. This heuristic is implemented in the function `SET OF CONCENTRATORS COMPUTE_UB(SET OF CONCENTRATORS, SET OF CONCENTRATORS)` which takes as an input the initial set of concentrators and the set of concentrators which are currently fixed. It gives as an output the set of concentrators which determines a feasible solution covering all the area. Figure 6.1 depicts the flowchart of the procedure.

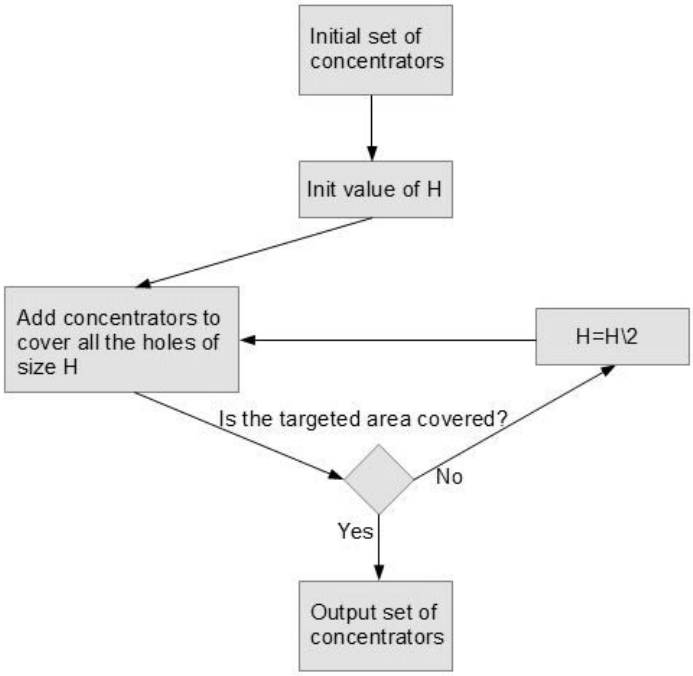



Figure 6.1: Flowchart of the procedure `COMPUTE_UB`

The same heuristic can be applied to determine solutions that reach a partial coverage of the overall area. The only step that we need to change in the heuristic is the stopping criterion. In fact, we can stop adding new concentrators when a specific amount of area, for example a percentage of the maximal area $\mathcal{A}_{max}$, is covered.

The function used to obtain partial coverage solutions is `SETS OF CONCENTRATORS COMPUTE_PARTIAL_SOLUTIONS(SET OF CONCENTRATOR, SET OF CONCENTRATOR, VECTOR OF DOUBLE)`. Its inputs are the set of concentrators that we can use, the set of selected concentrators and a vector which represents the area values that we aim to cover. The output is formed by a sequence of concentrator sets, each set represents a solution which covers one of the input area values.

The algorithm keeps generating points and solving Set Covering Problems of increasing size. The termination criteria are two: we stop the algorithm when we find a solution which is optimal with tolerance up to $\epsilon$ or when an overall time limit is reached. The pseudo-code of the algorithm is depicted in Algorithm 16, where all the functions defined above are used in order to clarify the structure of this approach.

**Algorithm 16** Exact algorithm for the Multi-meter Covering Problem

**Input:** A set $N$ of concentrators
$\mathcal{A}_{max} \leftarrow$ `AREA`$(N)$
$N_{necessary} \leftarrow$ `GET_NECESSARY_CONCENTRATORS`$(N)$
$P \leftarrow \emptyset$
$N_{current} \leftarrow \emptyset$
**while** $\mathcal{A}_{max} - \mathcal{A}_{current} < \epsilon$ **and** a time limit is not reached **do**
  $P \leftarrow P \cup$ `NEW_POINTS`$(N, N_{current}, \lambda)$
  $N_{current} \leftarrow$ `SOLVE_SET_COVER`$(N_{necessary}, N, P)$
  `COMPUTE_UB`$(N, N_{current})$
  `COMPUTE_PARTIAL_SOLUTIONS`$(N, N_{current},$ coverage values $)$
  $\mathcal{A}_{current} \leftarrow$ `AREA`$(N_{current})$
**end while**
**Output:** The best upper/lower bounds, partial solutions

## 6.5 Computational Experiments

The proposed approach was implemented in C++ using CPLEX 12.5 as a MIP solver. It was tested on two real world instances which correspond to the Torino area and to the Asti area running on an Intel Core i5 at 3GHz with 4GB of RAM. We used the exact positions where concentrators can be placed in these Italian cities. These two instances are completely different due to the different structure of the cities. For the smallest one, Asti, 1841 possible locations were considered, while for the largest, Torino, the number of the possible locations was 9631. We estimated the area that can be covered by putting in this locations concentrators of radius around 230 $m$, thus we considered concentrators of a single type. The maximal area coverable was approximately 64.726 $km^2$ in Asti and 114.192 $km^2$ in Torino, respectively.

We managed to solve to optimality Asti instance by means of our algorithm, while we derived upper and lower bounds with a deviation of approximately 15% for Torino. Figure 6.2 depicts a map of Asti area where all the concentrators available are activated. The area covered by them is represented by green circles. Analogously, Figure 6.3 takes into consideration the same city, but the circles drawn are relative to concentrators activated in the optimal solution. Notice that some concentrators are necessarily included in the optimal solutions because they are the only ones that can cover a very limited sub-area.

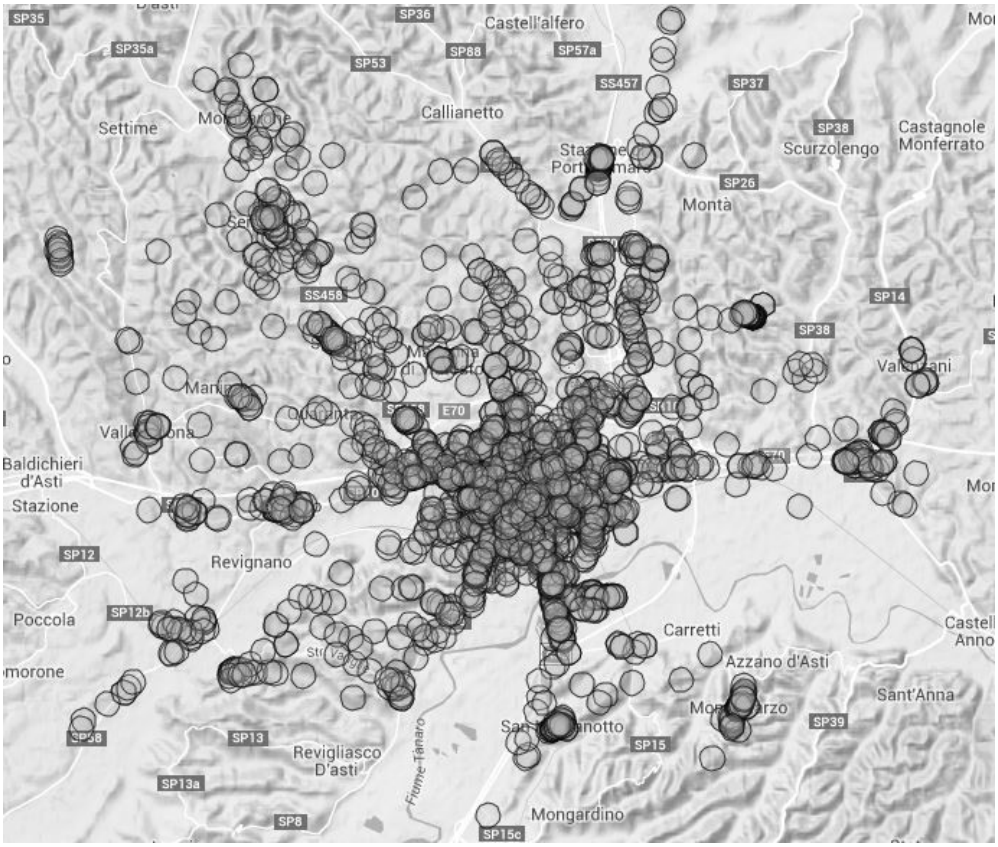



Figure 6.2: Asti area with all the concentrators

Table 6.3 and Table 6.4 depict in details the results obtained for Asti area and Torino area respectively. In such tables, we grouped the columns according to the percentage of the coverage given by the considered solution. For each given coverage, the columns denoted with "**Area** " indicate the specific area (in $km^2$) covered by the solution, while with "**Obj** " we indicate the number of concentrators used. Each row

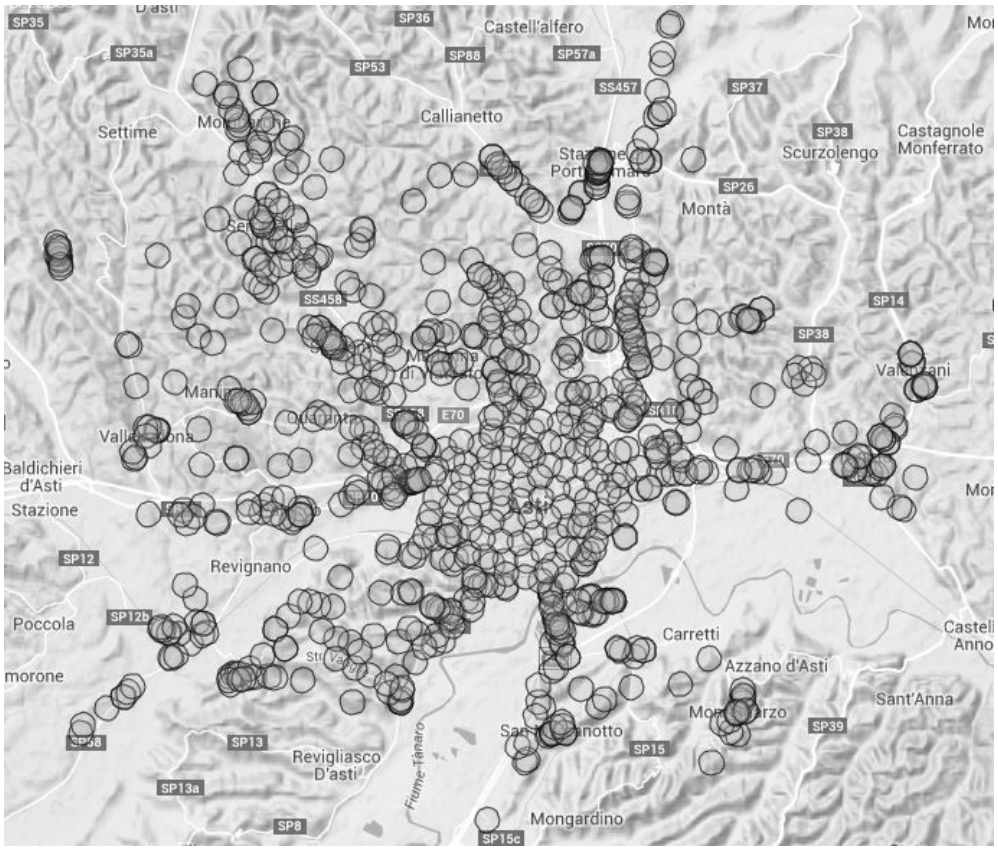



Figure 6.3: Asti area with the concentrators included in the optimal solution

refers to a different iteration of the algorithm, as long as we go on with the iterations, new points are considered and then new constraints are added to the ILP model. The last column depicts the cumulative CPU time in seconds. Note that the column about the lower bound obtained, denoted with "**LB**", contains only non decreasing values because we never discard the old constraints as the algorithm proceeds. This consideration cannot be done for the upper bounds, in fact they are obtained by means of heuristic algorithms which compute completely different solution when they start from different lower bound. As the quality of the lower bound increases, the quality of the related upper bounds does not necessarily increase. This is due to the fact that the upper bounds are obtained by means of a heuristic approach.

Tables 6.1 and 6.2 compare the results obtained by the proposed method (denoted as Polito) with the ones already available by Telecom on Asti area and Torino area, respectively. In each Table are provided different percentages of coverage and

corresponding number of activated concentrators for the two methods. It is possible to see that for high coverage values, the proposed approach clearly outperforms the results provided by Telecom. Same comparison is graphically expressed in Figures 6.4 and 6.5.

| Telecom | | Our approach | |
|---|---|---|---|
| **Area** | **Obj** | **Area** | **Obj** |
| 64,7266 | 859 | 64,7266 | 835 |
| 64,6427 | 838 | 64,7266 | 835 |
| 62,959 | 523 | 63,4558 | 507 |
| 62,9462 | 521 | 63,4558 | 507 |
| 62,3509 | 499 | 62,812 | 478 |
| 61,7228 | 482 | 62,1649 | 458 |
| 61,105 | 461 | 61,5033 | 443 |
| 60,4916 | 445 | 60,8551 | 428 |
| 59,7973 | 432 | 60,2394 | 417 |
| 59,2846 | 423 | 59,5769 | 405 |
| 58,2315 | 403 | 58,2585 | 391 |
| 57,1253 | 386 | 57,6424 | 388 |
| 56,1872 | 371 | 56,3522 | 371 |
| 55,26 | 358 | 55,6957 | 364 |
| 54,1325 | 346 | 54,414 | 354 |
| 52,9593 | 335 | 53,1333 | 343 |
| 51,7364 | 324 | 51,8457 | 331 |

Table 6.1: Telecom vs Polito solutions for Asti instance

| Telecom | | Our approach | |
|---|---|---|---|
| **Area** | **Obj** | **Area** | **Obj** |
| 114,1835 | 1778 | 114,191 | 1456 |
| 112,9996 | 1165 | 113,056 | 1010 |
| 112,1108 | 1087 | 113,056 | 1010 |
| 111,3432 | 1020 | 111,923 | 932 |
| 110,0519 | 969 | 110,787 | 889 |
| 109,2742 | 951 | 109,639 | 851 |
| 108,5957 | 916 | 109,639 | 851 |
| 107,9039 | 894 | 108,522 | 828 |
| 107,1488 | 876 | 107,4 | 803 |
| 106,498 | 838 | 107,4 | 803 |
| 105,8068 | 821 | 106,245 | 781 |
| 105,1608 | 806 | 106,245 | 781 |
| 104,6451 | 797 | 105,084 | 762 |
| 104,0757 | 783 | 105,084 | 762 |
| 103,4006 | 768 | 103,995 | 739 |
| 102,6439 | 753 | 102,857 | 719 |
| 102,0062 | 743 | 102,857 | 719 |
| 101,3197 | 734 | 101,746 | 705 |
| 100,8395 | 726 | 101,746 | 705 |
| 100,2156 | 718 | 100,57 | 689 |

Table 6.2: Telecom vs Polito solutions for Torino instance

| | | | 100% | | 99.80% | | 99.40% | | 99% | | 96% | | 93% | | 90% | | CPU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **It.** | **LB** | **Area** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **(s)** |
| 1 | 767 | 59,8021 | 64,7266 | 854 | 64,4695 | 606 | 64,212 | 565 | 62,8066 | 482 | 60,8839 | 431 | 58,9582 | 403 | 58,3003 | 396 | 494 |
| 2 | 790 | 62,7103 | 64,7266 | 858 | 64,4716 | 604 | 64,2135 | 563 | 62,8066 | 482 | 60,8839 | 431 | 58,9162 | 401 | 58,2983 | 395 | 777 |
| 3 | 801 | 63,6339 | 64,7266 | 860 | 64,4716 | 604 | 64,2135 | 563 | 62,8066 | 482 | 60,8839 | 431 | 58,9162 | 401 | 58,2983 | 395 | 1057 |
| 4 | 805 | 63,9539 | 64,7266 | 861 | 64,4716 | 604 | 64,2135 | 563 | 62,8049 | 481 | 60,8983 | 430 | 58,9162 | 401 | 58,2911 | 394 | 1339 |
| 5 | 810 | 64,1755 | 64,7266 | 863 | 64,4716 | 604 | 64,2135 | 563 | 62,8049 | 481 | 60,8983 | 430 | 58,9162 | 401 | 58,2911 | 394 | 1623 |
| 6 | 813 | 64,3946 | 64,7266 | 860 | 64,4716 | 604 | 64,2135 | 563 | 62,8049 | 481 | 60,8551 | 428 | 58,9129 | 399 | 58,2911 | 394 | 1912 |
| 7 | 815 | 64,3863 | 64,7266 | 862 | 64,4716 | 604 | 64,2135 | 563 | 62,8049 | 481 | 60,8551 | 428 | 58,9129 | 399 | 58,2911 | 394 | 2200 |
| 8 | 816 | 64,4548 | 64,7266 | 858 | 64,4728 | 603 | 64,2167 | 562 | 62,8049 | 481 | 60,8551 | 428 | 58,9129 | 399 | 58,2911 | 394 | 2489 |
| 9 | 819 | 64,5007 | 64,7266 | 859 | 64,4728 | 603 | 64,2167 | 562 | 62,8049 | 481 | 60,8551 | 428 | 58,9129 | 399 | 58,2911 | 394 | 2778 |
| 10 | 821 | 64,5396 | 64,7266 | 856 | 64,4702 | 601 | 64,2134 | 561 | 62,7918 | 480 | 60,8551 | 428 | 58,9129 | 399 | 58,2911 | 394 | 3068 |
| 11 | 822 | 64,6057 | 64,7266 | 858 | 64,4704 | 599 | 64,213 | 559 | 62,812 | 478 | 60,8551 | 428 | 58,9129 | 399 | 58,2997 | 392 | 3357 |
| 12 | 823 | 64,6322 | 64,7266 | 851 | 64,4704 | 599 | 64,213 | 559 | 62,812 | 478 | 60,8551 | 428 | 58,9129 | 399 | 58,2997 | 392 | 3647 |
| 13 | 825 | 64,6613 | 64,7266 | 859 | 64,4715 | 597 | 64,213 | 559 | 62,812 | 478 | 60,8551 | 428 | 58,9129 | 399 | 58,2997 | 392 | 3941 |
| 14 | 826 | 64,6352 | 64,7266 | 854 | 64,4715 | 597 | 64,213 | 559 | 62,812 | 478 | 60,8551 | 428 | 58,9129 | 399 | 58,2997 | 392 | 4262 |
| 15 | 827 | 64,6643 | 64,7266 | 852 | 64,4715 | 597 | 64,2189 | 557 | 62,812 | 478 | 60,8551 | 428 | 58,9129 | 399 | 58,2997 | 392 | 4559 |
| 16 | 828 | 64,6552 | 64,7266 | 856 | 64,4715 | 597 | 64,2189 | 557 | 62,812 | 478 | 60,8551 | 428 | 58,9348 | 398 | 58,2997 | 392 | 4942 |
| 17 | 828 | 64,6734 | 64,7266 | 853 | 64,4715 | 597 | 64,2189 | 557 | 62,812 | 478 | 60,8551 | 428 | 58,9348 | 398 | 58,2997 | 392 | 5317 |
| 18 | 828 | 64,6939 | 64,7266 | 854 | 64,4715 | 597 | 64,2144 | 556 | 62,812 | 478 | 60,8551 | 428 | 58,9348 | 398 | 58,2997 | 392 | 5635 |
| 19 | 828 | 64,6684 | 64,7266 | 854 | 64,4715 | 597 | 64,2144 | 556 | 62,812 | 478 | 60,8551 | 428 | 58,9348 | 398 | 58,2997 | 392 | 6002 |
| 20 | 829 | 64,6894 | 64,7266 | 857 | 64,4715 | 597 | 64,2144 | 556 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2997 | 392 | 6372 |
| 21 | 830 | 64,6823 | 64,7266 | 852 | 64,4715 | 597 | 64,2144 | 556 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2997 | 392 | 6695 |
| 22 | 830 | 64,7028 | 64,7266 | 855 | 64,4715 | 597 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 7019 |
| 23 | 831 | 64,6971 | 64,7266 | 852 | 64,4715 | 597 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 7392 |
| 24 | 832 | 64,6797 | 64,7266 | 848 | 64,4715 | 597 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 7777 |
| 25 | 832 | 64,7025 | 64,7266 | 851 | 64,4715 | 597 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 8122 |
| 26 | 832 | 64,7039 | 64,7266 | 853 | 64,4715 | 597 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 8463 |
| 27 | 832 | 64,701 | 64,7266 | 850 | 64,4715 | 597 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 8820 |
| 28 | 832 | 64,6992 | 64,7266 | 846 | 64,4715 | 597 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 9124 |
| 29 | 833 | 64,7205 | 64,7266 | 848 | 64,4712 | 596 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 9529 |
| 30 | 833 | 64,7176 | 64,7266 | 844 | 64,4712 | 596 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 9972 |
| 31 | 833 | 64,7002 | 64,7266 | 850 | 64,4712 | 596 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 10391 |
| 32 | 833 | 64,7187 | 64,7266 | 841 | 64,472 | 594 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 10749 |
| 33 | 833 | 64,7146 | 64,7266 | 847 | 64,472 | 594 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 11135 |
| 34 | 833 | 64,7199 | 64,7266 | 844 | 64,472 | 594 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 11531 |
| 35 | 833 | 64,7189 | 64,7266 | 840 | 64,472 | 594 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 11937 |
| 36 | 833 | 64,7193 | 64,7266 | 846 | 64,472 | 594 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 12331 |
| 37 | 833 | 64,6976 | 64,7266 | 848 | 64,472 | 594 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 12760 |
| 38 | 833 | 64,7227 | 64,7266 | 843 | 64,472 | 594 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 13285 |
| 39 | 833 | 64,7184 | 64,7266 | 844 | 64,472 | 594 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 13811 |
| 40 | 833 | 64,7253 | 64,7266 | 839 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 14299 |
| 41 | 833 | 64,7203 | 64,7266 | 843 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 15220 |
| 42 | 833 | 64,7178 | 64,7266 | 844 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 16076 |
| 43 | 833 | 64,7139 | 64,7266 | 841 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 17491 |
| 44 | 834 | 64,7184 | 64,7266 | 848 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 18569 |
| 45 | 834 | 64,7108 | 64,7266 | 847 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 19516 |
| 46 | 834 | 64,7224 | 64,7266 | 845 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 20623 |
| 47 | 834 | 64,7247 | 64,7266 | 840 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 22068 |
| 48 | 834 | 64,724 | 64,7266 | 841 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 23460 |
| 49 | 834 | 64,719 | 64,7266 | 845 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 24778 |
| 50 | 834 | 64,7203 | 64,7266 | 844 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 25763 |
| 51 | 834 | 64,7235 | 64,7266 | 843 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 26909 |
| 52 | 834 | 64,7263 | 64,7266 | 841 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 27900 |
| 53 | 834 | 64,7257 | 64,7266 | 839 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 30298 |
| 54 | 834 | 64,7171 | 64,7266 | 840 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 31616 |
| 55 | 834 | 64,7263 | 64,7266 | 839 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 32335 |
| 56 | 834 | 64,7242 | 64,7266 | 845 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 34666 |
| 57 | 834 | 64,726 | 64,7266 | 836 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 35887 |
| 58 | 834 | 64,7261 | 64,7266 | 839 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 37500 |
| 59 | 834 | 64,7238 | 64,7266 | 842 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 39266 |
| 60 | 834 | 64,7238 | 64,7266 | 839 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 40080 |
| 70 | 835 | 64,7236 | 64,7266 | 840 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 51633 |
| 80 | 835 | 64,7266 | 64,7266 | 837 | 64,4718 | 593 | 64,2158 | 555 | 62,812 | 478 | 60,8551 | 428 | 58,9382 | 397 | 58,2585 | 391 | 64992 |
| **Best** | 835 | | | 836 | | 593 | | 555 | | 478 | | 428 | | 397 | | 391 | - |

Table 6.3: Detailed Polito results for Asti instance

| | | | 100% | | 99.80% | | 99.40% | | 99% | | 96% | | 93% | | 90% | | CPU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **It.** | **LB** | **Area** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **Area** | **Obj** | **(s)** |
| 1 | 992 | 82,0394 | 114,192 | 1539 | 113,966 | 1184 | 113,513 | 1079 | 113,058 | 1028 | 109,65 | 871 | 106,235 | 785 | 102,861 | 720 | 12101 |
| 2 | 1054 | 88,9706 | 114,192 | 1547 | 113,968 | 1178 | 113,513 | 1070 | 113,054 | 1017 | 109,64 | 863 | 106,249 | 786 | 102,832 | 724 | 14898 |
| 3 | 1087 | 93,4377 | 114,192 | 1543 | 113,966 | 1179 | 113,51 | 1074 | 113,054 | 1023 | 109,655 | 859 | 106,26 | 782 | 102,882 | 721 | 17767 |
| 4 | 1115 | 96,7064 | 114,192 | 1547 | 113,965 | 1183 | 113,512 | 1074 | 113,055 | 1022 | 109,662 | 865 | 106,255 | 786 | 102,792 | 720 | 20649 |
| 5 | 1137 | 99,9811 | 114,192 | 1546 | 113,966 | 1182 | 113,51 | 1078 | 113,055 | 1024 | 109,649 | 867 | 106,214 | 787 | 102,83 | 717 | 23462 |
| 6 | 1151 | 101,942 | 114,192 | 1541 | 113,965 | 1175 | 113,513 | 1072 | 113,061 | 1024 | 109,636 | 866 | 106,251 | 790 | 102,838 | 726 | 26248 |
| 7 | 1167 | 103,364 | 114,192 | 1538 | 113,965 | 1178 | 113,512 | 1072 | 113,058 | 1026 | 109,656 | 867 | 106,257 | 784 | 102,877 | 725 | 29021 |
| 8 | 1180 | 104,248 | 114,192 | 1546 | 113,966 | 1190 | 113,514 | 1085 | 113,052 | 1033 | 109,657 | 872 | 106,243 | 789 | 102,876 | 721 | 31801 |
| 9 | 1191 | 105,766 | 114,192 | 1554 | 113,967 | 1190 | 113,511 | 1084 | 113,061 | 1032 | 109,666 | 867 | 106,223 | 786 | 102,801 | 717 | 34573 |
| 10 | 1203 | 106,799 | 114,192 | 1564 | 113,968 | 1181 | 113,513 | 1075 | 113,057 | 1026 | 109,644 | 867 | 106,244 | 791 | 102,805 | 726 | 37352 |
| 11 | 1212 | 107,429 | 114,192 | 1548 | 113,966 | 1182 | 113,512 | 1076 | 113,059 | 1026 | 109,635 | 864 | 106,27 | 784 | 102,791 | 721 | 40098 |
| 12 | 1219 | 107,897 | 114,192 | 1554 | 113,967 | 1185 | 113,512 | 1080 | 113,053 | 1026 | 109,668 | 869 | 106,24 | 789 | 102,814 | 724 | 42924 |
| 13 | 1226 | 108,559 | 114,192 | 1551 | 113,966 | 1176 | 113,51 | 1075 | 113,06 | 1027 | 109,636 | 868 | 106,271 | 789 | 102,796 | 723 | 45815 |
| 14 | 1233 | 109,297 | 114,192 | 1556 | 113,965 | 1195 | 113,513 | 1088 | 113,053 | 1028 | 109,656 | 866 | 106,27 | 790 | 102,808 | 723 | 48714 |
| 15 | 1240 | 109,678 | 114,192 | 1551 | 113,966 | 1178 | 113,511 | 1073 | 113,059 | 1022 | 109,652 | 864 | 106,234 | 784 | 102,827 | 722 | 52684 |
| 16 | 1248 | 109,972 | 114,192 | 1562 | 113,965 | 1177 | 113,509 | 1069 | 113,059 | 1018 | 109,639 | 858 | 106,266 | 785 | 102,831 | 720 | 57774 |
| 17 | 1252 | 110,454 | 114,192 | 1552 | 113,966 | 1174 | 113,51 | 1068 | 113,059 | 1021 | 109,652 | 857 | 106,207 | 782 | 102,798 | 718 | 66441 |
| 18 | 1259 | 110,582 | 114,192 | 1562 | 113,967 | 1182 | 113,51 | 1076 | 113,056 | 1023 | 109,653 | 860 | 106,278 | 784 | 102,812 | 717 | 73869 |
| 19 | 1264 | 110,831 | 114,192 | 1560 | 113,966 | 1170 | 113,512 | 1062 | 113,051 | 1014 | 109,668 | 862 | 106,27 | 781 | 102,879 | 719 | 80965 |
| 20 | 1270 | 111,076 | 114,192 | 1541 | 113,966 | 1173 | 113,511 | 1070 | 113,061 | 1017 | 109,64 | 862 | 106,235 | 783 | 102,797 | 718 | 88196 |
| 21 | 1274 | 111,477 | 114,192 | 1544 | 113,967 | 1165 | 113,51 | 1062 | 113,055 | 1010 | 109,648 | 862 | 106,27 | 781 | 102,884 | 721 | 95303 |
| 22 | 1280 | 111,681 | 114,192 | 1551 | 113,966 | 1173 | 113,513 | 1064 | 113,061 | 1014 | 109,656 | 864 | 106,271 | 782 | 102,826 | 717 | 102501 |
| 23 | 1284 | 111,811 | 114,192 | 1545 | 113,967 | 1177 | 113,512 | 1069 | 113,055 | 1012 | 109,649 | 861 | 106,212 | 786 | 102,813 | 720 | 109908 |
| 24 | 1288 | 111,922 | 114,192 | 1537 | 113,968 | 1172 | 113,512 | 1065 | 113,057 | 1012 | 109,645 | 859 | 106,276 | 782 | 102,806 | 720 | 117316 |
| 25 | 1288 | 111,938 | 114,192 | 1545 | 113,968 | 1177 | 113,511 | 1070 | 113,057 | 1018 | 109,668 | 859 | 106,219 | 785 | 102,811 | 724 | 124217 |
| 26 | 1292 | 112,066 | 114,192 | 1535 | 113,968 | 1161 | 113,514 | 1059 | 113,056 | 1011 | 109,654 | 863 | 106,257 | 784 | 102,866 | 720 | 131539 |
| 27 | 1294 | 112,352 | 114,192 | 1549 | 113,968 | 1170 | 113,51 | 1059 | 113,06 | 1010 | 109,667 | 859 | 106,243 | 784 | 102,85 | 719 | 140507 |
| 28 | 1298 | 112,388 | 114,192 | 1548 | 113,967 | 1171 | 113,51 | 1063 | 113,056 | 1010 | 109,639 | 851 | 106,245 | 781 | 102,857 | 719 | 147930 |
| **Best** | 1298 | | | 1535 | | 1161 | | 1059 | | 1010 | | 851 | | 781 | | 717 | - |

Table 6.4: Detailed Polito results for Torino instance

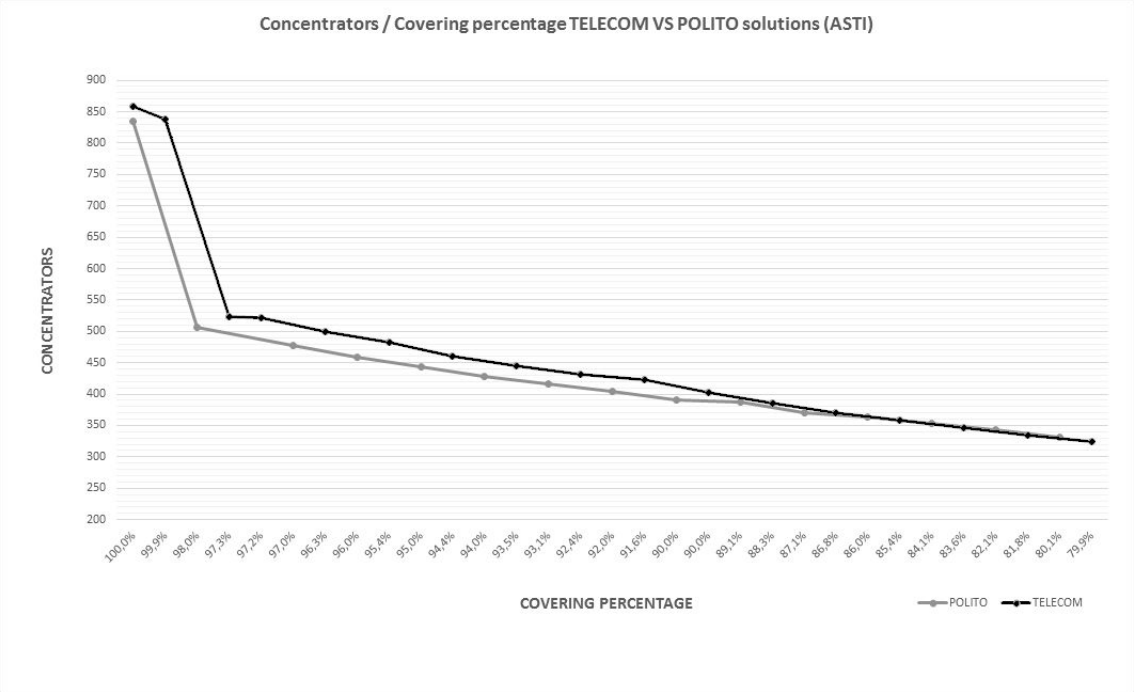



Figure 6.4: Comparing Telecom - Polito solutions on Asti instance

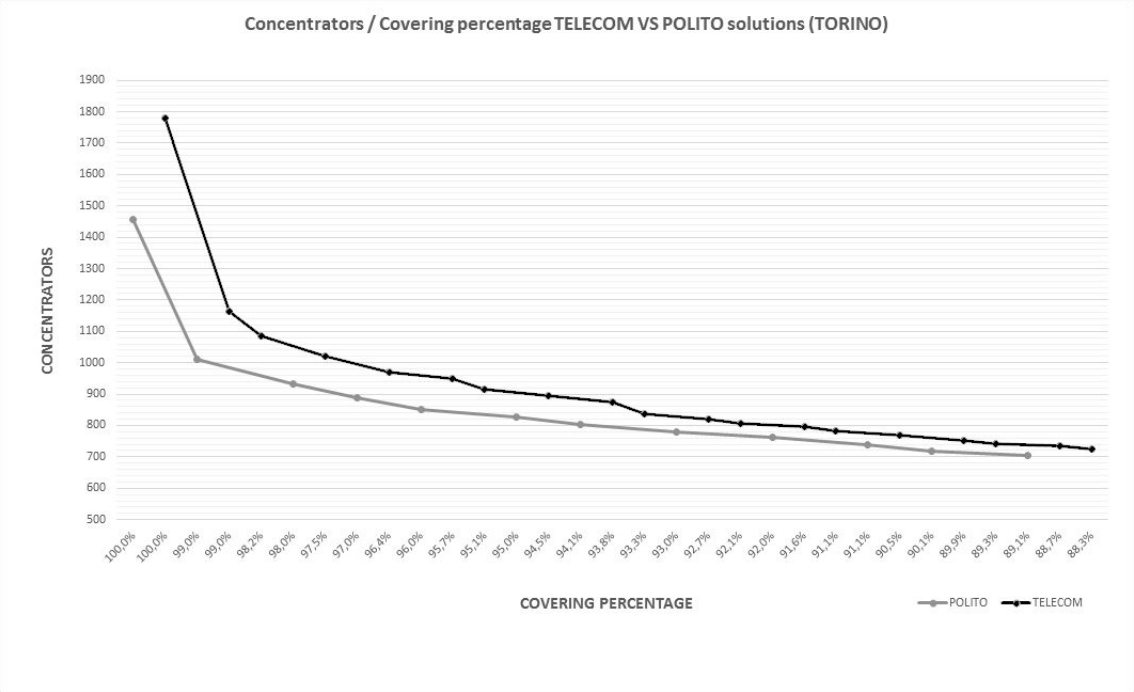



Figure 6.5: Comparing Telecom - Polito solutions on Torino instance

## 6.6 Conclusions

This chapter presented an exact approach for a real world clustering problem. The aim of the problem is to select the concentrators cluster of minimal size that covers all the area. The exact approach is based on a constraint generation approach, that iteratively adds constraints which refer to the coverage of some points inside the area, generated via simulation. The approach is enhanced by a heuristic method that computes feasible solutions at each step.

It is interesting to note that a mathematical solver is used to determine both a lower bound and an upper bound at each iteration. Then, the resulting method is a typical hybrid method that uses mathematical programming techniques to solve a combinatorial problem. The computational results showed the effectiveness of the proposed approach.

# Chapter 7

# Conclusions and Future Developments

This thesis dealt with the design of exact and heuristic algorithms for scheduling and clustering combinatorial optimization problems. All the works are linked by the fact that all the presented methods are basically hybrid algorithms, that mix techniques used in the world of combinatorial optimization. Section 2 presented practical solution algorithms based on an ILP model for an energy scheduling combinatorial problem that arises in a smart buildings. Section 3 presented a new cutting stock problem and introduced a heuristic solution approach based on a heuristic column generation scheme. Section 4 provided an exact exponential algorithm, whose importance is only theoretical so far, for a classical scheduling problem: the total tardiness problem. Section 5 provided an exact approach based on semidefinite programming and a heuristic approach based on a quadratic solver for a fractional clustering combinatorial optimization problem. Section 6 proposed practical solution methods for a real world clustering problem arising in a smart city context.

An important part of the thesis is the study conducted in Section 4 for the design of an exact exponential algorithm for the total tardiness problem. The proposed algorithm used a new technique, called branch and merge, that avoids the solution of repeated equivalent subproblems in a branching tree context. The merging operations are based on comparing the two prefixed/suffixed sequences of the two subproblems and then keeping the dominating one. This required solutions of

small subproblems (of size up to $k$), and a comparison between the two fixed sequences for determining the dominating one. A future development of this work is to make the proposed method practical from a computational point of view. This may regard the design of a fast method to merge the nodes of the branch tree, which may involve considerations on the lower bounds of the small subproblems considered. Such method may then use typical mathematical programming techniques to establish the dominating sequence in a very short time in the average case. Since we are trying to find other applications of the branch and merge technique, these computational considerations may also involve other problems.

Another possible development of this research may involve the definition of exact exponential algorithms based on the use of convex bounding techniques that have a worst case quality guarantee. This type of study would be very innovative for the field of exact exponential algorithms, but at the same time, the worst case quality guarantee is very difficult to be achieved.

In Section 5, a branch and bound method based on a semidefinite programming relaxation is proposed for a fractional combinatorial optimization problem and it is likely to be extended to other problems with a similar structure. Apart from the generalization of the method, it would be interesting to find applications where semidefinite programming allows to derive well-performing matheuristic for hard combinatorial problems. As an example, this may regard the use of a semidefinite programming based quadratic integer programming solver, such as BiqCrunch, or some other schemes based on the rounding of the result of the semidefinite relaxation.

Another possible development is the integration of semidefinite programming in a column generation framework, for instance to perform the pricing operation. This research line has already been investigated in works like [64].

# Bibliography


[1] BiqCrunch. http:/lipn.univ-paris13.fr/BiqCrunch/.

[2] Coin-or Branch and Cut (CBC). https:/projects.coin-or.org/cbc.

[3] Coin-or Symphony. https:/projects.coin-or.org/symphony.

[4] CPU Benchmark. https:/www.cpubenchmark.net/.

[5] FICO Xpress Optimization Suite. http:/www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx.

[6] Haversine formula. https:/en.wikipedia.org/wiki/Haversine_formula.

[7] ILOG Cplex. http:/www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/.

[8] INTRePID. http:/www.fp7-intrepid.eu.

[9] Total circles area. http:/rosettacode.org/wiki/Total_circles_area.

[10] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5:13–51, 1993.

[11] M. Armbruster. A solution procedure for a pattern sequencing problem as part of a one-dimensional cutting stock problem in the steel industry. *European Journal of Operational Research*, 141:328–340, 2002.

[12] M. O. Ball. Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16:21–38, 2010.

[13] B. Bánhelyi, E. Palatinus, and B. L. Lévay. Optimal circle covering problems and their applications. *Central European Journal of Operations Research*, 23:815–832, 2015.

[14] A. Barbato, A. Capone, G. Carello, M. Delfanti, M. Merlo, and A. Zaminga. House energy demand optimization in single and multi-user scenarios. In *2nd*

*IEEE International Conference on Smart Grid Communications (SmartGridComm 2011) Brussels, Belgium*, 2011.

[15] T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega: The International Journal of Management Science*, 34:209–219, 2006.

[16] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9:61–63, 1962.

[17] H. Ben Amor, J. Desrosiers, and J.M. Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. *Operations Research*, 54:454–463, 2006.

[18] H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos. On exact algorithms for treewidth. *ACM Transactions on Algorithms*, 9:article n. 12, 23 pages, 2012.

[19] B. Borchers. Csdp, a c library for semidefinite programming. *Optimization Methods and Software*, 11:613–Â623, 1999.

[20] C. Borean and E. Grasso. Qpsol: *Quantum Particle Swarm Optimization with Levy's Flight: Optimization of appliance scheduling for smart residential energy grids*, booktitle = The Ninth International Multi-Conference on Computing in the Global Information Technology (ICCGI 2014) Seville, Spain, year = 2014.

[21] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[22] T. H. Chang, C. W. Hsin, W.K. Ma, and C.Y. Chi. A linear fractional semidefinite relaxation approach to maximum-likelihood detection of higher-order qam ostbc in unknown channels. *IEEE Transactions On Signal Processing*, 58:2315 – 2326, 2010.

[23] R.K. Congram. *Polynomially searchable exponential neighbourhoods for sequencing problems in combinatorial optimisation*. PhD thesis, University of Southampton, 2000.

[24] T.H. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, second edition*. The MIT Press, 2001.

[25] Y. Cui and Y. Yang. A heuristic for the one-dimensional cutting stock problem with usable leftover, volume = 204, year = 2010. *European Journal of*

*Operational Research*, pages 245–250.

[26] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, Ma. Pilipczuk, Mi. Pilipczuk, and S. Saurabh. *Parametrized Algorithms*. Springer, 2015.

[27] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[28] F. Della Croce and M. Garraffa. The selective fixing algorithm for the closest string problem. *Computers & Operations Research*, 41:24–30, 2014.

[29] F. Della Croce, M. Garraffa, and F. Salassa. A hybrid three-phase approach for the max-mean dispersion problem. *Computers & Operations Research*, 2016 forthcoming.

[30] F. Della Croce and A. Grosso. Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem. *Computers & Operations Research*, 39:27–31, 2012.

[31] F. Della Croce, A. Grosso, and M. Locatelli. A heuristic approach for the max-min diversity problem based on max-clique. *Computers & Operations Research*, 36:2429–2433, 2009.

[32] F. Della Croce, A. Grosso, and F. Salassa. Matheuristics: embedding milp solvers into heuristic algorithms for combinatorial optimization problems. *Heuristics: theory and applications*, 2012.

[33] F. Della Croce, A. Grosso, and F. Salassa. A matheuristic approach for the two–machine total completion time flow shop problem. *Annals of Operations Research*, 213:67–78, 2014.

[34] F. Della Croce and F. Salassa. Improved lp-based algorithms for the closest string problem. *Computers & Operations Research*, 39:746–749, 2012.

[35] F. Della Croce, F. Salassa, and V. T'Kindt. A hybrid heuristic approach for single machine scheduling with release times. *Computers & Operations Research*, 45:7–11, 2014.

[36] F. Della Croce, R. Tadei, B. Baracco, and A. Grosso. A new decomposition approach for the single machine total tardiness scheduling problem. *Journal of the Operational Research Society*, 49:1101–1106, 1998.

[37] J. Desrosiers and Lübbecke M. E. A primer in column generation. *Column Generation*, pages 1–32, 2005.

[38] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[39] J. Du and J. Y. T. Leung. Minimizing total tardiness on one machine is np-hard. *Mathematics of Operations Research*, 15:483–495, 1990.

[40] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[41] H. Emmons. One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17:701–715, 1969.

[42] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005.

[43] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

[44] M. Fischetti, G. Sartor, and A. Zanette. A mip-and-refine matheuristic for smart grid energy management. *International Transactions in Operational Research*, 22:49–59, 2015.

[45] K. Fleszar and K. S. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29:821–839, 2002.

[46] F. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Springer, 2010.

[47] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.

[48] M. Garraffa, F. Salassa, W. Vancroonenburg, G. Vanden Berghe, and T. Wauters. The one-dimensional cutting stock problem with sequence-dependent cut losses. *International Transactions in Operational Research*, 23:5–24, 2016.

[49] M. Gendreau and J. Potvin. *Handbook of Metaheuristics*. Springer, 2010.

[50] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem-part ii. *Operations Research*, 11:864–888, 1963.

[51] F. Glover, C.-C. Kuo, and K.S. Dhir. Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19:109–132, 1998.

[52] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

[53] M.X. Goemans and D.P. Williamson. Semidefinite programming relaxations for the quadratic assignment problem. *Journal of the ACM*, 42:1115–1145, 1995.

[54] J.B. Gosh. Computational aspects of the maximum diversity problem. *Operations Research Letters*, 19:175–181, 1996.

[55] Y. Gurevich and S. Shelah. Expected computation time for hamiltonian path problem. *SIAM Journal on Computing*, 16:486â€“502, 1987.

[56] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10:673–696, 2006.

[57] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6:342–361, 1996.

[58] C. Helmberg, F. Rendl, and R. Weismantel. A semidefinite programming approach to the quadratic knapsack problem. *Journal of Combinatorial Optimization*, 4:197–215, 2000.

[59] O. Holthaus. Decomposition approaches for solving the integer one-dimensional cutting stock problem with different types of standard lengths. *European Journal of Operational Research*, 141:295–312, 2002.

[60] E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

[61] C. Joncour, S. Michel, R. Sadykov, D. Sverdlov, and F. Vanderbeck. Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics*, 36:695–702, 2010.

[62] C. Kerchove and P.V. Dooren. The page trust algorithm: how to rank web pages when negative links are allowed? In *SIAM International Conference on Data Mining*, pages 346–352, 2008.

[63] C. Koulamas. The single-machine total tardiness scheduling problem: review and extensions. *European Journal of Operational Research*, 202:1–7, 2010.

[64] K. Krishnan and J. E. Mitchell. A semidefinite programming based polyhedral cut and price approach for the maxcut problem. *Computational Optimization and Applications*, 33:51–71, 2006.

[65] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[66] E.L. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize

total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

[67] C. Lenté, M. Liedloff, A. Soukhal, and V. T'Kindt. On an extension of the sort & search method with application to scheduling theory. *Theoretical Computer Science*, 511:13–22, 2013.

[68] C. Lenté, M. Liedloff, A. Soukhal, and V. T'Kindt. Exponential algorithms for scheduling problems. Technical report, 2014. https:/hal.archives-ouvertes.fr/hal-00944382.

[69] R. Lewis, X. Song, K. Dowsland, and J. Thompson. An investigation into two bin packing problems with ordering and orientation implications. *European Journal of Operational Research*, 213:52–65, 2011.

[70] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[71] M. J. P. Lopes and J. V. de Carvalho. A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research*, 176:1508–1527, 2007.

[72] M.E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53:1007–1023, 2005.

[73] J. Malick and F. Roupin. Solving k-cluster problems to optimality with semidefinite programming. *Mathematical Programming*, 136:279–300, 2012.

[74] V. Maniezzo, T. Stützle, and Stefan Voß. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*. 2009.

[75] R. Martí, M. Gallego, A. Duarte, and E. G. Pardo. Heuristics and metaheuristics for the maximum diversity problem. *Journal of Heuristics*, 19:591 – 615, 2013.

[76] R. Martí and F. Sandoya. Grasp and path relinking for the equitable dispersion problem. *Computers & Operations Research*, 40:3091 – 3099, 2013.

[77] A. Mobasher and A. Ekici. Solution approaches for the cutting stock problem with setup cost. *Computers & Operations Research*, 40:225–235, 2013.

[78] V. T. Paschos. When polynomial approximation meets exact computation. *4OR*, 13:227–245, 2015.

[79] D. Pisinger. The quadratic knapsack problem - a survey. *Discrete Applied Mathematics*, 155:623–648, 2007.

[80] C. N. Potts and L. N. Van Wassenhove. A decomposition algorithm for the

single machine total tardiness problem. *Operations Research Letters*, 5:177–181, 1982.

[81] O. A. Prokopyev, N. Kong, and D.L. Martinez-Torres. The equitable dispersion problem. *Discrete Applied Mathematics*, 197:59 – 67, 2009.

[82] W. Rei, M. Gendreau, and P. Soriano. A hybrid monte carlo local branching algorithm for the single vehicle routing problem with stochastic demands. *Transportation Science*, 44:136–146, 2010.

[83] M.G.C. Resende, R. Martí, M. Gallego, and A. Duarte. Grasp and path relinking for the maxâ€“min diversity problem. *Computers & Operations Research*, 37:498 – 508, 2010.

[84] F. Sandoya and R. Aceves. Grasp and path relinking to solve the problem of selecting efficient work teams. In Javier Del Ser, editor, *Recent Advances on Meta-Heuristics and Their Application to Real Scenarios*, chapter 2. 2013.

[85] K.C. Sou, J. Weimer, H. Sandberg, and K.H. Johansson. Scheduling smart home appliances using mixed integer linear programming. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC) Orlando, USA*, pages 5144–5149, 2011.

[86] K. Stephenson. Circle packing: a mathematical tale. *Notices Amer. Math. Soc*, 50:1376–1388, 2003.

[87] W. Szwarc. Single machine total tardiness problem revisited. *Creative and Innovative Approaches to the Science of Management*, pages 407–419, 1993.

[88] W. Szwarc, A. Grosso, and F. Della Croce. Algorithmic paradoxes of the single machine total tardiness problem. *Journal of Scheduling*, 4:93–104, 2001.

[89] W. Szwarc and S. Mukhopadhyay. Decomposition of the single machine total tardiness problem. *Operations Research Letters*, 19:243–250, 1996.

[90] V. T'kindt, F. Della Croce, and C. Esswein. Revisiting branch and bound search strategies for machine scheduling problems. *Journal of Scheduling*, 7:429–440, 2004.

[91] P. Trkman and M. Gradisar. One-dimensional cutting stock optimization in consecutive time periods. *European Journal of Operational Research*, 179:291–301, 2007.

[92] W. Vancroonenburg, P. De Causmaecker, F. Spieksma, and G. Vanden Berghe. Column generation for the 1D-CSP with SDCL, and the orienteering problem

with multiplicity. Technical report, KU Leuven, Department of Computer Science, 2014. http:/allserv.kahosl.be/~wimvc/cg-1D-csp-with-sdcl.pdf.

[93] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[94] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109–1130, 2007.

[95] D. P. Williamson and D. B. Shmovs. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[96] G.J. Woeginger. Exact algorithms for np-hard problems: a survey. *Lecture Notes in Computer Science*, 2570:185–207, 2003.

[97] H. Wolkowicz, R. Saigal, and L. Vandenberghe. *Handbook of Semidefinite Programming*. Springer, 2000.

[98] Leung J. Y-T. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[99] H. H. Yanasse and M. S. Limeira. A hybrid heuristic to reduce the number of different patterns in cutting stock problems. *Computers & Operations Research*, 33:2744–2756, 2007.

[100] B. Yang, W. Cheung, and J. Liu. Community mining from signed social networks. *IEEE Transactions on Knowledge and Data Engineering*, 19:1333–Â1348, 2007.

[101] B. Yelbay, S. Birbil, and K. Bulbul. The set covering problem revisited: an empirical study of the value of dual information. Technical report, 2012. Optimization Online.

[102] C. Yu, D. Zhang, and H.Y.K. Lau. Mip-based heuristics for solving robust gate assignment problems. *Computers & Industrial Engineering*, 98:23–47, 2016.

[103] D. Zhang, L.G. Papageorgiou, N.J. Samsatli, and N. Shah. Optimal scheduling of smart homes energy consumption with microgrid. In *The First International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies (ENERGY 2011) Venice/Mestre, Italy*, pages 70–75, 2011.

[104] Q. Zhao, S. Karisch, F. Rendl, and H. Wolkowicz. Semidefinite programming relaxations for the quadratic assignment problem. *Journal of Combinatorial*

*Optimization*, 2:71–109, 1998.