Alma Mater Studiorum – Università di Bologna

Dottorato di Ricerca in
Automatica e Ricerca Operativa

XXI Ciclo

MAT/09

Enhanced Mixed Integer Programming Techniques and Routing Problems

Andrea Tramontani

Il Coordinatore
Prof. Claudio Melchiorri

I Relatori
Prof. Paolo Toth
Prof. Andrea Lodi

A.A. 2005–2008

Keywords

Mixed integer programming

Disjunctive cuts

Two-row cuts

Vehicle routing problems

Traveling salesman problem with time windows

Local search

Branch-and-cut

A Monica

Contents

Acknowledgments

First of all, my thanks go to my advisers, Andrea Lodi and Paolo Toth, strong researchers and teachers, for giving me the great opportunity of working with them. They followed me during all my PhD with useful suggestions, ideas and remarks. Simply, without them this work would not have been possible.

I want also to thank all the members of the Operations Research group of Bologna, for their professional help and their friendship.

Many thanks go to Matteo Fischetti, who always answers all my questions. Every day I go to Padova, I know in advance that I will learn something.

Thanks to Jon Lee, for giving me the opportunity of spending three wonderful months at IBM. And many thanks go to Sanjeeb Dash and Oktay Günlük, real gentlemen. They followed me every day during my period at IBM and are still helping me in my research activity.

Thanks also to Santanu Dey and Laurence Wolsey, who introduced me in the two-row world.

I was honored to work with all these fantastic people and I am in debt with all of them. I really hope to have the opportunity to work with them again and again in the future.

Finally, a special thank goes to Monica. Thank you for your patience, for encouraging and trusting me in all my choices. You are in me every day of my life and this thesis is also your thesis.

Bologna, March 15, 2009

Andrea Tramontani

List of Figures

List of Tables

Preface

Mixed integer programming is up today one of the most widely used techniques for dealing with hard optimization problems. On the one side, many practical optimization problems arising from real-world applications (such as, e.g., scheduling, project planning, transportation, telecommunications, economics and finance, timetabling, etc) can be easily and effectively formulated as Mixed Integer linear Programs (MIPs). On the other hand, 50 and more years of intensive research has dramatically improved on the capability of the current generation of MIP solvers to tackle hard problems in practice. However, many questions are still open and not fully understood, and the mixed integer programming community is still more than active in trying to answer some of these questions. As a consequence, a huge number of papers are continuously developed and new intriguing questions arise every year.

When dealing with MIPs, we have to distinguish between two different scenarios. The first one happens when we are asked to handle a general MIP and we cannot assume any special structure for the given problem. In this case, a Linear Programming (LP) relaxation and some integrality requirements are all we have for tackling the problem, and we are "forced" to use some general purpose techniques. The second one happens when mixed integer programming is used to address a somehow structured problem. In this context, polyhedral analysis and other theoretical and practical considerations are typically exploited to devise some special purpose techniques.

Interestingly, these two different scenarios are indeed strongly related, as the general algorithmic approach used in both cases is typically the same: i.e., branch-and-cut. In particular, it is worth noting that several among the main ingredients which are actually embedded in the most effective general purpose MIP solvers (e.g., cutting planes) were first devised and proven to be effective for specific combinatorial optimization problems, and only afterwards were successfully extended to the general MIP context. The history of the branch-and-cut framework itself is strongly related to the history of the Traveling Salesman Problem (TSP), probably one of the most widely studied combinatorial optimization problems. On the other side, general purpose MIP solvers are now able to solve a large variety of hard optimization problems arising from many different contexts, because most of the tools embedded in general MIP solvers (e.g., again cutting planes) are effective in practice. The connections between these two different situations are more important every day. Having a magic black-box which allows one to solve any problem is clearly not possible, but the current trend when addressing specific problems by means of specific approaches is to keep them as much general as possible, in the spirit of possibly extend such approaches to cover other related problems.

This thesis tries to give some insights in both the above mentioned situations. The first part of the work is focused on general purpose cutting planes, which are probably the key ingredient behind the success of the current generation of MIP solvers. Chapter 1 presents a quick overview of the main ingredients of a branch-and-cut algorithm, while Chapter 2 recalls some results from the literature in the context of disjunctive cuts and their connections with Gomory mixed integer cuts. Chapter 3 presents a theoretical and computational investigation of disjunctive cuts. In particular, we analyze the connections between different normalization conditions (i.e., conditions to truncate the cone associated with disjunctive cutting planes) and other crucial aspects as cut rank, cut density and cut strength. We give a theoretical characterization of weak rays of the disjunctive cone that lead to dominated cuts, and propose a practical method to possibly strengthen those cuts arising from such weak extremal solution. Further, we point out how redundant constraints can affect the quality of the generated disjunctive cuts, and discuss possible ways to cope with them. Finally, Chapter 4 presents some preliminary ideas in the context of multiple-row cuts. Very recently, a series of papers have brought the attention to the possibility of generating cuts using more than one row of the simplex tableau at a time. Several interesting theoretical results have been presented in this direction, often revisiting and recalling other important results discovered more than 40 years ago. However, is not clear at all how these results can be exploited in practice. As stated, the chapter is a still work-in-progress and simply presents a possible way for generating two-row cuts from the simplex tableau arising from lattice-free triangles and some preliminary computational results.

The second part of the thesis is instead focused on the heuristic and exact exploitation of integer programming techniques for hard combinatorial optimization problems in the context of routing applications. Chapters 5 and 6 present an integer linear programming local search algorithm for Vehicle Routing Problems (VRPs). The overall procedure follows a general destroy-and-repair paradigm (i.e., the current solution is first randomly destroyed and then repaired in the attempt of finding a new improved solution) where a class of exponential neighborhoods are iteratively explored by heuristically solving an integer programming formulation through a general purpose MIP solver. Chapters 7 and 8 deal with exact branch-and-cut methods. Chapter 7 presents an extended formulation for the Traveling Salesman Problem with Time Windows (TSPTW), a generalization of the well known TSP where each node must be visited within a given time window. The polyhedral approaches proposed for this problem in the literature typically follow the one which has been proven to be extremely effective in the classical TSP context. Here we present an overall (quite) general idea which is based on a relaxed discretization of time windows. Such an idea leads to a stronger formulation and to stronger valid inequalities which are then separated within the classical branch-and-cut framework. Finally, Chapter 8 addresses the branch-and-cut in the context of Generalized Minimum Spanning Tree Problems (GMSTPs) (i.e., a class of $\mathcal{NP}$-hard generalizations of the classical minimum spanning tree problem). In this chapter, we show how some basic ideas (and, in particular, the usage of general purpose cutting planes) can be useful to improve on branch-and-cut methods proposed in the literature.

Chapter 1

An overview of Mixed Integer Programming

1.1 Introduction

A general *Mixed Integer linear Program* (MIP) can be defined as a "non linear" optimization problem, where all the non linearities can be expressed by imposing integrality restrictions on a subset of variables: i.e., any general MIP can be expressed in the form

$$\min\{cx : Ax \geq b, x \geq 0,\ x_j \in \mathbb{Z}\ \forall j \in J\}, \tag{1.1}$$

where $c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$ are the given objective function and constraint matrix, while $J \subseteq \{1, \ldots, n\}$ denotes the set of variables constrained to be integer. When all the variables are integer-constrained (i.e., $J = \{1, \ldots, n\}$), problem (1.1) is referred as a *pure Integer linear Program* (IP).

Integer programming and mixed integer programming emerged in the late 1950's and early 1960's, when researchers realized that the ability to solve mixed integer programming models would had great impact for practical applications (see, e.g., Dantzig [63]). Up today, mixed integer programming is one of the most widely used techniques for handling hard combinatorial optimization problems. On the one side, many combinatorial optimization problems arising from practical applications (such as, e.g., scheduling, project planning, transportation, telecommunications, economics and finance, timetabling) can be easily formulated as MIPs. On the other hand, several academic and commercial MIP solvers now available on the market can solve hard MIPs in practice.

The basic approach for tackling a MIP is the well known *branch-and-bound* algorithm [110], which relies on the iterative solution of the Linear Programming (LP) relaxation of (1.1),

$$\min\{cx : Ax \geq b, x \geq 0\}, \tag{1.2}$$

where all the integrality requirements on the x variables are dropped. The reason for dropping such constraints is that MIP is $\mathcal{NP}$-hard while LP is polynomially solvable and general-purpose techniques for its solution are efficient in practice.

When dealing with MIPs and MIP solvers, we have to distinguish between two different situations. The first one happens when we have no knowledge on the given problem (i.e., when we cannot assume any special structure for the given matrix A in (1.1)). In this case, we are "forced" to use a *general purpose* algorithmic approach which is typically based on the above branch-and-bound scheme. The second one happens when the addressed formulation arises from a particular optimization problem. In this case, polyhedral analysis and other theoretical and practical considerations can be exploited to devise some *special purpose* techniques for handling the problem and/or to improve on the performance of a general purpose MIP solver. Interestingly, several among the main ingredients which are actually embedded in the most effective general purpose MIP solvers (e.g., cutting planes) were first devised and implemented for tackling specific combinatorial optimization problems and specific MIP formulations. On the other hand, general purpose MIP solvers can now be easily customized in order to deal with the specific structure of MIPs arising from hard combinatorial optimization problems.

This chapter[1] presents an overview of mixed integer programming techniques. In particular, Section 1.2 recalls the branch-and-bound and the branch-and-cut frameworks, Section 1.3 provides a historical overview of general purpose MIP solvers, while Section 1.4 describes some of the main ingredients of a branch-and-cut based MIP solver. The attention is focused on the current generation of general purpose MIP solvers. However, when dealing with a particular optimization problem, the main ingredients which turn out to be of crucial importance for an effective branch-and-cut algorithm are typically the same. Obviously, each one of them can be customized by exploiting the specific structure of the addressed formulation.

1.2 Branch-and-bound, cutting-plane and branch-and-cut

Roughly speaking, MIP solvers integrate the *branch-and-bound* and the *cutting-plane* algorithms through variations of the general *branch-and-cut* scheme proposed by Padberg & Rinaldi [134] in the context of the *Traveling Salesman Problem* (TSP)[2].

The branch-and-bound algorithm [110]. In its basic version the branch-and-bound algorithm iteratively partitions the solution space into sub-MIPs (the children nodes) which have the same theoretical complexity of the originating MIP (the father node, or the root node if it is the initial MIP). Given the optimal solution x^* of the LP relaxation of the current sub-MIP (initially, the optimal solution of the LP relaxation of the root node), the branching usually creates two children by choosing one integer-constrained variable, say x_j, having a fractional value x_j^* in the current x^* solution, and imposing the *disjunctive* condition

$$x_j \leq \lfloor x_j^* \rfloor \quad \text{OR} \quad x_j \geq \lceil x_j^* \rceil$$

[1] Part of this chapter has been taken from: A. Lodi, "MIP computation and beyond", to appear in M. Jünger, T. Liebling, D. Naddef, W. Pulleyblank, G. Reinelt, G. Rinaldi, L. Wolsey, editors, *50 Years of Integer Programming 1958–2008*, Springer-Verlag, 2008 [117].

[2] The history of the branch-and-cut framework is tightly connected to the history of the TSP itself. We refer the reader to Cook [52] for a detailed account on this topic.

on the two created nodes (i.e., the two nodes are created by imposing, respectively, the constraint $x_j \leq \lfloor x_j^* \rfloor$ on the first one and the constraint $x_j \geq \lceil x_j^* \rceil$ on the latter). On each of the sub-MIPs the integrality requirement on the variables $x_j, \forall j \in J$ is relaxed and the LP relaxation is solved again. Despite the theoretical complexity, the sub-MIPs become smaller and smaller due to the partition mechanism (basically some of the decisions are taken) and eventually the LP relaxation is directly integral for all the variables in J. In addition, whenever the optimal solution value cx^* of the LP relaxation turns out to be not smaller than the best feasible solution encountered so far, called *incumbent*, the node can safely be fathomed without further partitioning its corresponding sub-MIP, because none of its children will yield a better solution than the incumbent. Finally, a node is also fathomed if its LP relaxation is infeasible.

The cutting-plane algorithm [98]. Any MIP can be solved without branching by "simply" finding a linear programming description of the *convex hull* of its feasible solutions, i.e., a linear programming description of the set

$$P := conv\{x \in \mathbb{R}^n : Ax \geq b, x \geq 0,\ x_j \in \mathbb{Z}\ \forall j \in J\}.$$

In order to do that, one can iteratively solve the so called *separation problem*:

> (Separation problem). Given a feasible solution x^* of the LP relaxation (1.2) which is not feasible for the MIP (1.1), find a linear inequality $\gamma x \geq \gamma_0$ which is valid for (1.1), i.e., satisfied by all feasible solutions $\bar{x}$ of the system (1.1), while it is violated by x^*, i.e., $\gamma x^* < \gamma_0$.

Any inequality solving the separation problem is called *cutting plane* (or *cut*, for short) and has the effect of tightening the LP relaxation to better approximate the convex hull.

Gomory [98] has given an algorithm that converges in a finite number of iterations for pure *integer linear programs* with integer data, thus implicitly proving that IPs can be solved by pure cutting-plane methods. Such an algorithm solves the separation problem above in an efficient and elegant manner in the special case in which x^* is an optimal basis of the LP relaxation. No algorithm of this kind is known for MIPs, that being one of the most intriguing open questions in the area (see, e.g., Cook, Kannan & Schrijver [54]).

The branch-and-cut algorithm incorporates the cutting-plane algorithm in the branch-and-bound scheme, by separating cuts at each[3] node of the search tree. The idea behind integrating the two algorithms above is that LP relaxations (1.2) do not naturally well approximate, in general, the convex hull of mixed-integer solutions of MIPs (1.1), thus some extra work to devise a better approximation by tightening any relaxation with additional linear inequalities (cutting planes) increases the chances that fewer nodes in the search tree are needed. On the other hand, pure cutting plane algorithms show, in general, a slow convergence and the addition of too many cuts can lead to very large

[3]Clearly, several heuristic criteria are typically used to decide, at each node of the search tree, if cut separation may be useful in practice to improve on the current LP relaxation or not.

LPs which in turn present numerical difficulties for the solvers[4]. As already mentioned, branch-and-cut has been initially proven to be very effective for combinatorial optimization problems (like TSP, see Padberg & Rinaldi [134, 135]) with special-purpose cuts based on a polyhedral analysis of the addressed formulation. Later on, such a framework has been successfully extended to the general MIP context.

1.3 MIP Evolution

The early general-purpose MIP solvers were mainly concerned with developing a fast and reliable LP machinery used within good branch-and-bound schemes.

There are at least two remarkable exceptions to this trend. The first is a paper of Crowder, Johnson & Padberg [61] which describes the implementation of a general-purpose code for pure 0–1 IPs, called PIPX, that used the IBM linear programming system MPSX/370 and the IBM integer programming system MIP/370 as building blocks. The authors were able to solve ten instances obtained from real-world industrial applications. Those instances were very large for that time, up to 2,756 binary variables and 755 constraints. Some cutting planes were integrated in a good branch-and-bound framework (with some preprocessing to reduce variables' coefficients) and in particular *cover inequalities* which will be discussed in Section 1.4.2 below.

The second exception is a paper by Van Roy & Wolsey [170] in which the authors describe a system called MPSARX that solved mixed 0–1 industrial instances arising from applications like production planning, material requirement planning, multilevel distribution planning, etc. Also in this case, the instances were very large for the time, up to 2,800 variables and 1,500 constraints. The system acted in two phases: in phase one some preprocessing was performed and, more importantly, cutting planes were generated. Such a phase was called reformulation in [170] and performed by a system called ARX. The second phase was the solution of the new model by the mathematical programming system called SCICONIC/VM.

After the success of the two early MIP solvers [61, 170] and, mainly, the formalization of the branch-and-cut algorithm given by Padberg and Rinaldi in the TSP context [135], a major step was required to prove that cutting plane generation in conjunction with branching could work in general, i.e., without exploiting the structure of the underlying polyhedron, in particular in the cutting plane separation phase.

The proof of concept came through two papers both published in 1996. In [20], Balas, Ceria and Cornuéjols showed that a branch-and-cut algorithm based on general-purpose *disjunctive* cuts (see, Balas [16]) could lead to very effective results for 0–1 MIPs. More precisely, the cutting planes used are the so called *lift-and-project* cuts in which the separation problem amounts to solving an LP in an extended space[5].

[4]Note, however, that very recently Zanette, Fischetti & Balas [177] have shown that the original cutting plane algorithm of Gomory [98] can be surprisingly effective, thus raising to many open questions related to an effective cutting planes exploitation.

[5]Disjunctive cuts are recalled more in detail in Chapter 2 and are then investigated in Chapter 3.

The computational results in [20] showed a very competitive and stable behavior. In addition, those results showed that the quality and the speed of LP solvers in the late nineties allowed the solution of LPs as an additional building block of the overall algorithm. Such an outcome was not granted and LP rapid evolution showed to have reached a very stable state.

In [21], Balas, Ceria, Cornuéjols and Natraj revisited the classical Gomory mixed integer cuts [99] by overruling the common sense that these cuts had a purely theoretical interest. In fact, the results in [21] showed that, embedded in a branch-and-cut framework, Gomory mixed-integer cuts were a fundamental tool for the solution of 0–1 MIPs. Such an improvement was mainly obtained by separating and adding groups (denoted as "rounds") of cuts[6] instead of one cut at a time.

The two papers above have played a central role in the move to the current generation of MIP solvers and the fact that the use of cutting planes, and in particular within a branch-and-cut framework, has been a breakthrough is shown by the following very meaningful experiment due to Achterberg & Bixby and reported by Lodi [117]. On a testbed of 1,734 MIP instances all Cplex [105] versions beginning with Cplex 1.2, the first having MIP capability, have been extensively compared. The results of this experiment are shown in Tables 1.1 and 1.2. Specifically, Table 1.1 reports the evolution of Cplex by comparing each version with the most current one, version 11.0. The first column of the table indicates the version while the second recalls the year in which the version has been released. The third and fourth columns count the number of times each version is better and worse with respect to Cplex 11.0, respectively: computing times within 10% are considered equivalent, i.e., counted neither better nor worse. Finally, the last column gives the computing time as geometric mean again normalized with respect to Cplex 11.0. A time limit of 30,000 CPU seconds for each instance was provided on a cluster of Intel Xeon machines 3 GHz. Besides the comparison with the current Cplex version, the most interesting way of interpreting the results reported in the table is looking at the trend the columns show: together with a very stable decreasing of the computing time from version 1.2 up, it is clear that the biggest step forward in a version-to-version scale has been made with Cplex 6.5 which is the first having full cutting plane capability, and in particular Gomory mixed-integer cuts. Indeed, the geometric mean of the computing times drops from 21.30% to 7.47% going from version 6.0 to 6.5, by far the biggest decrease.

The trend is confirmed by the numbers reported in Table 1.2. On a slightly larger set of 1,852 MIPs (including some models in which older versions encountered numerical troubles), the table highlights the version-to-version improvement in the number of solved problems. Besides the first two columns which report the same information as Table 1.1, the third and fourth column report the number and percentage of problems solved to proven optimality, respectively. Finally, the last column gives precisely the version-to-version improvement in the percentage of problems optimally solved.

[6] In principle, one Gomory mixed-integer cut can be separated from each tableau row where an integer variable is non-zero and fractional.

Table 1.1: Computing times for 12 CPLEX versions: normalization with respect to CPLEX 11.0.

CPLEX version	year	better	worse	time
11.0	2007	–	–	1.00
10.0	2005	201	650	1.91
9.0	2003	142	793	2.73
8.0	2002	117	856	3.56
7.1	2001	63	930	4.59
6.5	1999	71	997	**7.47**
6.0	1998	55	1060	21.30
5.0	1997	45	1069	22.57
4.0	1995	37	1089	26.29
3.0	1994	34	1107	34.63
2.1	1993	13	1137	56.16
1.2	1991	17	1132	67.90

Table 1.2: Version-to-version comparison on 12 CPLEX versions with respect to the number of solved problems.

CPLEX version	year	# optimal	% optimal	v-to-v % improvement
11.0	2007	1,243	67.1%	7.8%
10.0	2005	1,099	59.3%	3.5%
9.0	2003	1,035	55.9%	2.6%
8.0	2002	987	53.3%	2.5%
7.1	2001	941	50.8%	4.3%
6.5	1999	861	46.5%	**13.4%**
6.0	1998	613	33.1%	1.0%
5.0	1997	595	32.1%	1.8%
4.0	1995	561	30.3%	4.4%
3.0	1994	479	25.9%	6.2%
2.1	1993	365	19.7%	4.7%
1.2	1991	278	15.0%	—

1.4 Main ingredients of MIP solvers

This section provides an overview of the main ingredients of a branch-and-cut based MIP solver. For many more details on the arguments of this section we refer the reader to Part II of the PhD dissertation of Achterberg [3].

1.4.1 Presolving

In the presolving (often called preprocessing) phase the solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process. This is done without changing the set of optimal solutions of the problem at hand and it affects two main situations.

On the one side, it is often the case that MIP models, in particular those originating from real-world applications and created by using modeling languages, contain some "garbage", i.e., irrelevant or redundant information that tend to slow down the solution process forcing the solver to perform useless operations. More precisely, there are two types of sources of inefficiency: first, the models are unnecessary large and thus harder to manage. This is the case in which there are redundant constraints or variables which are already fixed and nevertheless appear in the model as additional constraints. Second, the variable bounds can be unnecessary large or the constraints could have been written in a loose way, for example with coefficients weaker than they could possibly be.

Thus, modern MIP solvers have the capability of "cleaning up" the models so as to create a presolved instance associated with the original one on which the MIP technology is then applied. With respect to the two issues above, MIP presolve is able to reduce the size of the problems by removing such redundancies and it generally provides tools that, exploiting the information on the integrality of the variables in set J, strengthen variables' bound and constraints' coefficients. If the first improvement has only the effect of making each LP relaxation smaller and then quicker to be solved, the second one has the, sometimes crucial, effect of making such relaxations stronger, i.e., better approximating the convex hull of mixed-integer solutions.

On the other hand, more sophisticated presolve mechanisms are also able to discover important implications and sub-structures that might be of fundamental importance later on in the computation for both branching purposes and cutting plane generation. As an example, the presolve phase determines the clique table or conflict graph, i.e., groups of binary variables such that no more than one can be non-zero at the same time. The conflict graph is then fundamental to separate *clique inequalities* (see, Johnson and Padberg [107]) which are written as

$$\sum_{j \in Q} x_j \leq 1 \tag{1.3}$$

where Q denotes a subset of (indices of) binary variables with the property, stated above, that at most one of them can be non-zero.

The most fundamental work about presolving is the one of Savelsbergh [157] to which the interested reader is referred to.

At first glance, the presolving phase seems to be relevant only when dealing with "badly-written" MIP formulations, which typically contain many redundancies and weak constraints. However, this is definitely not the case, as experienced with the *Traveling Salesman Problem with Time Windows* (TSPTW) (see Chapter 7). For this problem, a strong MIP formulation is proposed and a branch-and-cut algorithm is investigated in this thesis. In this context, a *special-purpose* preprocessing which exploits the

specific structure of the problem turns out to be of crucial importance for all the above mentioned reasons. First, it is fundamental for reducing the size of the formulation and for strengthening the corresponding LP relaxation. Second, it allows to discover important connections among the variables which are then used to derive strong valid inequalities, separated during the branch-and-cut algorithm. We refer the reader to Chapter 7 for a detailed account on this work.

1.4.2 Cutting plane generation

As shown in Section 1.3, cutting plane generation has been a key step for the success of MIP solvers and their capability of being effective for a wide variety of problems. Several groups of cuts are actually available in most academic and commercial MIP solvers (as, e.g., Cbc [44], Cplex [105] and Xpress [65]). These groups are strongly related each other, and include Chvátal-Gomory cuts [98, 49], Gomory mixed-integer cuts [99], mixed-integer rounding cuts [129, 67], $\{0, \frac{1}{2}\}$ cuts [41], lift-and-project cuts [16, 20] and split cuts [54] (also known as disjunctive cuts). Essentially, all these inequalities are obtained by applying a disjunctive argument on a mixed-integer set of a single constraint only, which is often derived by aggregating many others. Gomory mixed-integer cuts and disjunctive cuts are addressed more in detail in Chapter 2 of this thesis. For a brilliant and unified presentation of all these family of cuts we refer the reader to Cornuéjols [57].

Besides the groups above and clique inequalities already discussed in the previous section, two more classes of very useful cuts are generally used within a MIP solver, namely *cover inequalities* and *flow cover inequalities*, which are briefly discussed in the following.

Cover cuts. Somehow similarly to the clique inequalities, cover constraints define a property on a set of binary variables. More precisely, given a knapsack kind constraint in the form $\gamma x \leq \gamma_0$ where we assume that $\gamma \in \mathbb{Z}_+^{|V|}$, $\gamma_0 \in \mathbb{Z}_+$ and V is a subset of (indices of) binary variables, a set $Q \subseteq V$ is called a *cover* if $\sum_{j \in Q} \gamma_j > \gamma_0$. The cover is said to be minimal if $\sum_{j \in Q \setminus \{\ell\}} \gamma_j \leq \gamma_0$ for all $\ell \in Q$. In other words, Q is a set of binary variables which cannot be all together non-zero at the same time. In the light of the definition above, the simplest version of a cover inequality is

$$\sum_{j \in Q} x_j \leq |Q| - 1. \tag{1.4}$$

The amount of work devoted to cover cuts is huge starting with the seminal work of Balas [15] and Wolsey [174]. In particular, cover cuts do not define facets, i.e., polyhedral faces of dimension one less of the dimension of the associated polyhedron[7], but can be lifted (see, Padberg [133]) to become facet defining.

Flow cover cuts. The polyhedral investigation of the so called *0–1 single node flow problem* is at the base of the definition of flow cover cuts by Padberg, Van Roy & Wolsey

[7]For a detailed presentation of polyhedral basic concepts as dimensions, faces, facets, etc. the reader is referred for example to Papadimitriou & Steiglitz [137].

[136]. Essentially, this is a fixed charge problem with arcs incoming and outgoing to a single node: to each of these arcs is associated a continuous variables measuring the flow on the arc and upper bounded by the arc capacity, if the flow over the arc is non-zero a fixed cost must be paid and this is triggered by a binary variable associated with the same arc. A flow balance on the node must also be satisfied. Flow cover cuts can then be devised as mixed-integer rounding inequalities (see, Marchand [122]) and then strengthened by lifting (see, Gu, Nemhauser & Savelsbergh [101]). We also refer the to Louveaux & Wolsey [118] for a general discussion on these mixed-integer sets.

It is worth noting that despite the origin of the flow cover cuts is a specific polyhedral context, they are useful and applied in general within MIP solvers. The reason is that it is easy to aggregate variables (and constraints) of a MIP in order to derive a mixed-integer set like the 0–1 single node flow set (see, e.g., [122]).

The separation of all mentioned cuts (including cliques) but lift-and-project cuts is $\mathcal{NP}$-hard for a general x^*. Note that this is not in contrast with the fact that one can separate, e.g., Gomory mixed-integer cuts by a polynomial (very cheap) procedure [98, 99] once the fractional solution x^* to be cut off is a vertex of the continuous relaxation.

Cutting planes have been widely studied in the literature and the arsenal of separation algorithms has been continuously enlarged over the years. However, there are still several fundamental questions about the use of cutting planes which are probably not fully answered, thus reducing what we could really gain from cuts. Some of these questions are discussed in the following and are probably strongly related each other.

Accuracy. Accuracy checks are needed in basically all parts of a MIP solver but maybe the most crucial part is cutting plane generation. Sophisticated cutting plane procedures challenge the floating-point arithmetic of the solvers heavily, the danger being the generation of invalid cuts, i.e., linear inequalities cutting off mixed-integer feasible solutions, eventually the optimal one. The common reaction is defining confidence threshold values for a cut to be safe. This is only a heuristic action, probably very conservative too. The accuracy issue has been recently investigated in two papers. Margot [123] studied a methodology for testing accuracy and strength of cut generators based on random dives of the search tree by recording a well-chosen set of indicators. Cook, Dash, Fukasawa & Goycoolea [53] proposed a method that slightly weakens the Gomory mixed-integer cuts but makes them "safe" with respect to accumulation of floating-point errors.

Both papers above investigate the accuracy issue of cutting plane generation mainly from a research viewpoint. At the moment, the standard method of dealing with inaccuracy of cuts in the solvers is discarding "suspicious" or "dangerous" cuts without doing extra work to either test their correctness or separating them in a proven careful way. Among other indicators, cutting planes with high rank[8] are considered suspicious

[8]The rank of the inequalities in the original formulation is 0 while every inequality obtained as combination of two or more inequalities of rank 0 has rank 1. An inequality has rank k ($k \geq 2$) if it can

in the sense that they might have accumulated relevant floating-point errors while dense inequalities are said to be dangerous since they generally slow down the LP machinery.

Cut selection and interaction. The number of existing procedures for generating cuts is relevant and that is the number of cuts we could generate at any round. Thus, one of the most relevant issues is which cuts should be selected and added to the current LP relaxation. In other words, cuts should be able to interact in a profitable way together so as to have an overall "cooperative behavior". Results in this direction have been reported by Andreello, Caprara & Fischetti [8].

Cut strength. The strength of a cut is generally measured by its violation. However, scaling, normalization conditions and other factors can make such a measure highly misleading. If this is the case, cut selection is very hard and, for example, separating over a larger class of cuts is not necessarily a good idea with respect to preferring a smaller (thus, theoretically weaker) sub-class which is known to produce effective cuts in practice. An example of such a counter-intuitive behavior is the separation of Chvátal-Gomory cuts instead of their stronger mixed integer rounding version as experienced by Dash, Günlük & Lodi [67]: separating over the sub-class as a heuristic mixed integer rounding procedure helped accelerating the cutting plane convergence dramatically[9].

The role of the rank and other aspects such as normalization conditions, (i.e., conditions to truncate the cone associated with disjunctive cutting planes) are not fully understood and much more can be done in this direction. Some new results on the connections among normalization conditions, cut rank, cut density and cut strength are presented in Chapter 3 of this thesis and also appear in Fischetti, Lodi & Tramontani [85].

We close this section by mentioning one of the most exciting and potentially helpful challenges in the cutting plane context. A recent series of papers [7, 29, 39, 60, 71, 75, 176] has brought the attention of the community to the possibility of generating cuts using more than one row of the simplex tableau at a time. These *multiple-row cuts* are based on the characterization of lattice-free triangles (and lattice-free bodies in general) instead of simply split disjunctions as in the case of the single row tableau cuts discussed in this section. Some new ideas for generating *two-row cuts* from the simplex tableau by exploiting lattice-free triangles and some computational results in this direction are presented in Chapter 4 of this thesis.

be obtained as combination of two or more rows of rank $\leq k-1$ but not as a combination of inequalities of rank $\leq k-2$ only. The reader is referred to Chvátal [49] for a more formal definition of the rank of an inequality and its implications for IP.

[9]Note that such an outcome might be also related to the stable behavior of Chvátal-Gomory cuts with respect to Gomory mixed-integer or mixed integer rounding cuts due to their numerical accuracy [83].

1.4.3 Branching strategies

The branching mechanism introduced in Section 1.1 requires to take two independent and important decisions at any step: node selection and variable selection. We will analyze them separately in the following by putting more attention on the latter.

Node Selection. One extreme is the so called *best-bound first* strategy in which one always considers the most promising node, i.e., the one with the smallest LP value, while the other extreme is *depth first* where one goes deeper and deeper in the tree and starts backtracking only once a node is fathomed, i.e., it is either mixed-integer feasible, or LP infeasible or it has a dual bound not better (smaller) than the incumbent. The main advantages and drawbacks of each strategy are well known: the former explores less nodes but generally maintains a larger tree in terms of memory while the latter can explode in terms of nodes and it can, in the case some bad decisions are taken at the beginning, explore useless portions of the tree itself. All other techniques, more or less sophisticated, are basically hybrids around these two ideas, like interleaving best-bound and diving[10] in an appropriate way.

Variable Selection. The variable selection problem is the one of deciding how to partition the current node, i.e., on which variable one has to branch on in order to create the two children. For a long time, a classical choice has been branching on the most fractional variable, i.e., in the 0–1 case the closest to 0.5. That rule has been computationally shown by Achterberg, Koch & Martin [5] to be worse than a complete random choice. However, it is of course very easy to evaluate. In order to devise stronger criteria one has to do much more work. The extreme is the so called *strong branching* technique (see, e.g., Applegate, Bixby, Chvátal & Cook [9] and Linderoth & Savelsbergh [116]). In its full version, at any node one has to tentatively branch on each candidate fractional variable and select the one on which the increase in the bound on the left branch times the one on the right branch is the maximum. Of course, this is generally unpractical but its computational effort can be easily limited in two ways: on the one side, one can define a much smaller candidate set of variables to branch on and, on the other hand, can limit to a fixed (small) amount the number of Simplex pivots to be performed in the variable evaluation. Another sophisticated technique is *pseudocost branching* which goes back to Benichou, Gauthier, Girodet & Hentges [31] and keeps a history of the success (in terms of the change in the LP relaxation value) of the branchings already performed on each variable as an indication of the quality of the variable itself. The most recent effective and sophisticated method is called *reliability branching* [5] and it integrates strong and pseudocost branchings. The idea is to define a reliability threshold, i.e., a level below which the information of the pseudocosts is not considered accurate enough and some strong branching is performed. Such a threshold is mainly associated with the number of previous branching decisions that involved the variable.

More sophisticated branching mechanisms are clearly possible, an example being

[10] A *dive* in the tree is a sequence of branchings without backtracking.

the so called *Special Order Sets* (SOS) branching (Beale & Tomlin [30]). When, for instance, the formulation at hand contains a constraint of the form

$$\sum_{j \in B} x_j = 1,$$

where B is a set of binary variables, one can branch by imposing the disjunctive condition

$$\sum_{j \in B_1} x_j = 1 \quad \text{OR} \quad \sum_{j \in B_2} x_j = 1$$

on the children nodes, with $B_1 \cup B_2 = B$ and $B_1 \cap B_2 = \emptyset$. This idea is detailed, e.g., in Williams [173].

Karamanov & Cornuéjols [108] highlighted the possibility of using some cutting plane theory in the branching context. They experimented with branching on Gomory mixed-integer disjunctions obtained by the optimal basis of the LP relaxation. More precisely, from each tableau row i in which an integer variable x_ℓ is basic at the fractional value x^*_ℓ, say $\bar{a}_i x = x^*_\ell$, instead of deriving the associated Gomory mixed-integer cut, they consider the corresponding disjunction

$$\pi x \leq \lfloor x^*_\ell \rfloor \quad \text{OR} \quad \pi x \geq \lceil x^*_\ell \rceil,$$

where

$$\pi_j = \begin{cases} \lfloor \bar{a}_{ij} \rfloor & \text{if } j \in J,\ j \neq \ell,\ \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor \leq x^*_\ell - \lfloor x^*_\ell \rfloor \\ \lceil \bar{a}_{ij} \rceil & \text{if } j \in J,\ j \neq \ell,\ \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor > x^*_\ell - \lfloor x^*_\ell \rfloor \\ 1 & \text{if } j = \ell \\ 0 & \text{otherwise.} \end{cases}$$

The best disjunction according to a heuristic measure is selected for branching.

In the line of the work in [108], Cornuéjols, Liberti & Nannicini [59] try to improve the disjunction to be used for branching by row manipulations in the spirit of the reduce-and-split approach for cuts [6].

Branching on general disjunctions has been also used in context less related to cutting plane theory. A pioneering work in this direction is the one of Owen & Mehrotra [132] where the general disjunctions to branch on are generated by a search heuristic in the neighborhood containing all disjunctions with coefficients in $\{-1, 0, 1\}$ on the integer variables with fractional values at the current node. Moreover, branching on appropriate disjunctions has recently been proposed in the context of highly symmetric MIPs by Ostrowsky, Linderoth, Rossi & Smriglio [131]. Finally, Bertacco [33] has experimented with branching on general disjunctions either associated with "thin" directions of the polyhedron or resulting in the maximization of the corresponding lower bound. A very similar approach has been followed recently by Mahajan & Ralphs [121].

As discussed above, until now the methods proposed in research papers and implemented by MIP solvers to do enumeration have been rather structured, very relying on the tree paradigm which has proven to be stable and effective. Few attempts of seriously revisiting the tree framework have been made, one notable exception being

the work of Chvátal called *resolution search* [50]. In that paper, the idea is to detect conflicts, i.e., sequences of variable fixings that yield an infeasible subproblem, making each sequence minimal and use the information to construct an enumeration strategy. A similar concept of conflicts has been used by Achterberg [2, 3] to do propagation in MIP and it has been known in the Artificial Intelligence and SATisfiability (SAT) communities with the name of "no good recording" (see, e.g., Stallman & Sussman [159]) since the seventies.

1.4.4 Primal heuristics

Primal heuristics, going from easy to complex, are generally used within any effective branch-and-cut algorithm. These heuristics typically start from the fractional solution of the continuous relaxation of each sub-MIP of the search tree, trying to produce a feasible mixed-integer solution. In the last years, the continuous hybridization between rounding and diving techniques with local search has dramatically improved the capability of the current generation of MIP solvers to quickly find in the search tree very high quality solutions.

Following the structure of Chapter 9 of [3], we distinguish among the following types of heuristics.

Rounding heuristics. Starting from a feasible solution of the continuous relaxation (generally the optimal one) one rounds up or down the fractional values of the variables in J by trying to produce a mixed-integer solution still satisfying the linear constraints. This can be done in a straightforward and very fast manner or in a more and more complex way, i.e., by allowing some backtracking mechanism in the case a (partial) mixed-integer assignment becomes infeasible.

Diving heuristics. With respect to a rounding heuristic, the diving is performed in an iterative way: after rounding one or more variables the LP relaxation is solved again and the process is iterated. In this category, Achterberg [3] distinguishes between "hard" rounding in which the variables are really fixed to a specified value and "soft" rounding in which the effect is obtained implicitly by changing the objective function coefficient of the variables to be "rounded" in an appropriate way. In the latter category falls the feasibility pump heuristic [81, 34, 4]. The idea of the algorithm is to alternatively satisfy either the linear constraints (by solving LP relaxations) or integer constraints (by applying rounding). The LP relaxations are obtained by replacing the original objective function with one taking into account the distance with respect to the current mixed-integer rounded solution. If the trajectory of the rounded solutions intersects the one of the LP (neighborhood) relaxations, then a feasible solution with respect to both the linear constraints and the integer ones has been found. Such an algorithmic framework has been recently extended to convex mixed-integer non-linear programming [38].

Improving heuristics. The heuristics in this category are designed to improve the quality of the current incumbent solution, i.e., they exploit that solution or a group of

solutions so as to get a better one. These heuristics are usually local search methods and we have recently seen a variety of algorithms which solve sub-MIPs for exploring the neighborhood of the incumbent or of a set of solutions. When these sub-MIPs are solved in a general-purpose fashion, i.e., through a nested call to a MIP solver, this is referred to as MIPping [82] and will be the content of Section 1.4.5 below. Improvement heuristics of this type have connections to the so called *metaheuristic* techniques [95] which have proven very successful on hard and large combinatorial problems in the nineties. Techniques from metaheuristics have been now incorporated in the MIP solvers, see e.g., [82, 152]. As a result, MIP solvers have now improved their ability of quickly finding very good feasible solutions and can be seen as competitive heuristic techniques if used in a truncated way, i.e., with either a time or node limit.

It is interesting to note that Achterberg [3] has shown that the impact of heuristics is not dramatic in terms of ability of the MIP solver to prove optimality in a (much) quicker way. However, in many cases mixed integer programming techniques are used to derive effective heuristics for hard optimization problems. In this context, the goal is to find high-quality feasible solutions, with a minor attention to the possibility of also prove the optimality. Clearly, when the MIP solver is embedded in an overall heuristic procedure, the capability of the solver itself to produce high-quality feasible solutions very quickly is of crucial importance to improve on the performance of the overall algorithm.

1.4.5 MIPping

The MIP computation has reached such an effective and stable quality to allow the solution of sub-MIPs during the solution of a MIP itself, i.e., the *MIPping* approach [84]. In other words, optimization sub-problems are formulated as general-purpose MIPs and solved, often heuristically, by using MIP solvers, i.e., without taking advantage of any special structure. The novelty is not in solving optimization sub-problems having different levels of complexity, but in using a general-purpose MIP solver for handling them. In some sense such a technique, originally proposed by Fischetti & Lodi [82] for modeling large neighborhoods of 0–1 solutions, shows the high quality of the current generation of MIP solvers like the solution of large LPs in the cutting plane generation phase [20] showed more than ten years ago the effectiveness of the LP phase.

After the original paper on the primal heuristic side, the MIPping technique has been successfully applied in the cutting plane generation phase [83], for accelerating Benders decomposition [148], and again to devise very high quality primal solutions [62], just to cite a few applications.

In Chapters 5 and 6 of this thesis, the MIPping approach is investigated to derive an effective local search algorithm in the context of the *Vehicle Routing Problem* (VRP). The overall local search algorithm is based on a class of exponential neighborhoods which are iteratively explored, in a heuristic way, by solving an IP through a general purpose MIP solver. The results of Chapters 5 and 6 also appear, respectively, in Toth & Tramontani [169] and Salari, Toth & Tramontani [153].

Chapter 2

Disjunctive cuts and Gomory Mixed Integer cuts

2.1 Introduction

Disjunctive cuts for mixed integer linear programs have been introduced by Egon Balas [16] in the late 70's, and successfully exploited in practice since the late 90's (see, e.g., Balas, Ceria and Cornuéjols [20]). This chapter recalls some theoretical results from the literature related to the separation of *disjunctive cuts*, and the main connections between disjunctive cuts and *Gomory Mixed Integer (GMI) cuts* [99]. A new investigation and some new results on this topic are developed in Chapter 3.

In the following we consider a mixed integer set S of the form $S = \{x \in \mathbb{R}^n : Ax \geq b, x \geq 0,\ x_j \in \mathbb{Z}\ \forall j \in J\}$, where $A \in \mathbb{R}^{m \times n}$ is the given constraint matrix and $J \subseteq \{1, \ldots, n\}$ denotes the set of variables constrained to be integer. We assume for the sake of simplicity the set S to be nonempty and we denote with $P = \{x \in \mathbb{R}^n : Ax \geq b, x \geq 0\}$ the underlying polyhedron obtained by dropping all the integral requirements from S.

2.2 Gomory Mixed Integer cuts

Let $\alpha x \geq \alpha_0$ be any valid inequality for P (recall that, by Farkas lemma, the valid inequalities for P are of the form $(\lambda A + \mu)x \geq \lambda b - \delta$, with $\lambda \in \mathbb{R}^m_+$, $\mu \in \mathbb{R}^n_+$, $\delta \in \mathbb{R}_+$), and add a nonnegative surplus variable s to $\alpha x \geq \alpha_0$, thus obtaining the valid equality $\alpha x - s = \alpha_0$. Define $f_0 = \alpha_0 - \lfloor \alpha_0 \rfloor$, $f_j = \alpha_j - \lfloor \alpha_j \rfloor$ for any $j \in J$, and assume $f_0 > 0$. Then, the following equality is valid for P,

$$\sum_{j \in J : f_j \leq f_0} f_j x_j + \sum_{j \in J : f_j > f_0} (f_j - 1) x_j + \sum_{j \notin J} \alpha_j x_j - s = k + f_0,$$

where k is some integer. Since $k \leq -1$ or $k \geq 0$, any $x \in S$ satisfies the disjunction

$$\sum_{j \in J : f_j \leq f_0} \frac{-f_j}{1 - f_0} x_j + \sum_{j \in J : f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j + \sum_{j \notin J} \frac{-\alpha_j}{1 - f_0} x_j + \frac{s}{1 - f_0} \geq 1$$

OR

$$\sum_{j\in J:f_j\le f_0}\frac{f_j}{f_0}x_j+\sum_{j\in J:f_j>f_0}\frac{f_j-1}{f_0}x_j+\sum_{j\notin J}\frac{\alpha_j}{f_0}x_j-\frac{s}{f_0}\ge 1.$$

The above disjunction is of the form $a^1z \ge 1$ or $a^2z \ge 1$ which naturally implies $\sum_j \max\{a^1_j, a^2_j\}z_j \ge 1$ for any $z \ge 0$. For each j, one of the coefficient in the disjunction is positive and the other is negative. Hence, the following inequality is valid for S:

$$\sum_{j\in J:f_j\le f_0}\frac{f_j}{f_0}x_j+\sum_{j\in J:f_j>f_0}\frac{1-f_j}{1-f_0}x_j+\sum_{j\notin J:\alpha_j>0}\frac{\alpha_j}{f_0}x_j+\sum_{j\notin J:\alpha_j<0}\frac{-\alpha_j}{1-f_0}x_j+\frac{s}{1-f_0}\ge 1. \tag{2.1}$$

The inequality (2.1) is the well known *Gomory Mixed Integer* inequality [99]. Clearly, the GMI (2.1) can be expressed in the x space by rewriting the surplus variable s in terms of the x variables (i.e., by rewriting $s = \alpha x - \alpha_0$). The *GMI closure* is obtained from P by adding all the GMI inequalities for S.

As shown by Caprara and Letchford [42] and, independently, by Cornuéjols and Li [58], it is $\mathcal{NP}$-hard to optimize a linear function over the GMI closure relative to a polyhedron P. Equivalently, given a point $\bar{x} \in P \setminus S$, it is $\mathcal{NP}$-hard to find a GMI which separates $\bar{x}$ or show that none exists. Note, however, that this is not in contrast with the problem of finding a GMI cut that cuts off a basic solution x^* of the polyhedron P. Indeed, given a basic solution x^* of P and its corresponding tableau, each row of the tableau corresponding to a basic variable x_j, with $j \in J$ and x^*_j fractional, can be used to derive a violated GMI cut by following the simple procedure described above.

2.3 Disjunctive cuts

For any given disjunction of the form

$$\pi x \le \pi_0 \quad \text{OR} \quad \pi x \ge \pi_0 + 1 \tag{2.2}$$

such that $(\pi, \pi_0) \in \mathbb{Z}^{n+1}$, $\pi_j = 0\ \forall j \notin J$, let P_0 (respectively, P_1) be the polyhedron obtained from P by imposing the additional restriction $\pi x \le \pi_0$ (resp., $\pi x \ge \pi_0 + 1$). A *disjunctive inequality* is an inequality $\gamma x \ge \gamma_0$ valid both for P_0 and P_1, and then also for $conv(P_0 \cup P_1)$ and hence for $conv(S)$. Disjunctive cuts which can be derived by imposing a single disjunction such as (2.2) on a polyhedron P are also known as *split cuts*; see Cook, Kannan, and Schrijver [54]. The intersection of all split inequalities (for all the possible disjunctions of the form (2.2)) is called the *split closure* relative to P.

By Farkas lemma, the validity of $\gamma x \ge \gamma_0$ for P_0 and for P_1 can always be certified by means of nonnegative multipliers (u, u_0, v, v_0) associated with the inequalities defining P_0 and P_1 according to the following scheme:

	P_0				P_1		
(u)	Ax	$\ge$	b	(v)	Ax	$\ge$	b
(u_0)	$-\pi x$	$\ge$	$-\pi_0$	(v_0)	πx	$\ge$	$\pi_0 + 1$

Given a fractional solution x^* of P, a most-violated disjunctive cut can therefore be found by solving the following problem that determines the disjunction (π, π_0) and the Farkas multipliers so as to maximize the violation of the resulting cut with respect to the given point x^*:

$$\begin{array}{lll} \min & \gamma x^* - \gamma_0 & \\ \gamma & \geq uA - u_0\pi & \\ \gamma & \geq vA + v_0\pi & \\ \gamma_0 & \leq ub - u_0\pi_0 & \\ \gamma_0 & \leq vb + v_0(\pi_0 + 1) & \\ & u, v, u_0, v_0 \geq 0 & \\ & \pi_0 \in \mathbb{Z} & \\ & \pi_j \in \mathbb{Z}, \quad j \in J & \\ & \pi_j = 0, \quad j \notin J. & \end{array} \tag{2.3}$$

By construction, any feasible solution with negative objective function value in (2.3) corresponds to a violated disjunctive cut. However, when solving the separation problem (2.3), there are two different aspects to be considered. On the one side, (2.3) is a mixed integer nonlinear program involving products of integer and continuous variables. On the other hand, even if the disjunction is fixed a priori, the resulting Linear Program (LP) is defined on a cone and needs to be truncated so as to produce a bounded LP in case a violated cut exists. From a theoretical point of view, Caprara and Letchford [42] have formulated the problem of optimizing over the split closure as a mixed integer nonlinear program and have shown that the separation problem for split cuts is strongly $\mathcal{NP}$-hard. On a practical side, Balas and Saxena [26] addressed the problem by exploiting the normalization condition $u_0 + v_0 = 1$, which allows one to restate (2.3) and to solve it as a parametric MIP with a single parameter. A similar approach was also proposed by Dash, Günlük and Lodi [66, 67] in the context of the Mixed Integer Rounding (MIR) closure[1].

A typical approach for separating disjunctive cuts in a branch-and-cut framework is the one arising from Balas, Ceria and Cornuéjols [20], which showed the effectiveness of *lift-and-project* cuts in the context of 0–1 MIPs[2]. This approach can be easily extended to general MIPs (i.e., MIPs with general integer-constrained variables) and can be described as follows. Given the current fractional solution x^* of the current LP relaxation, a violated disjunction of the form (2.2) is fixed, with $\pi x^* - \pi_0 \in\ ]0, 1[$, and a disjunctive cut is separated by solving the so-called Cut Generating Linear Program (CGLP):

$$\begin{array}{lll} \text{(CGLP)} \quad \min & \gamma x^* - \gamma_0 & \\ & \gamma \geq uA - u_0\pi & \\ & \gamma \geq vA + v_0\pi & \\ & \gamma_0 \leq ub - u_0\pi_0 & \\ & \gamma_0 \leq vb + v_0(\pi_0 + 1) & \\ & u, v, u_0, v_0 \geq 0. & \end{array} \tag{2.4}$$

[1] Note that the MIR closure and the Split closure define the same polyhedron; see e.g., [57, 66, 67].

[2] Lift-and-project cuts are a particular class of disjunctive cuts for 0–1 MIPs arising from elementary disjunctions of the form $x_j \leq 0$ OR $x_j \geq 1$; see, e.g., [57].

As already stated, CGLP is defined on a cone and needs to be truncated with a suitable normalization condition so as to produce a bounded LP in case a violated cut exists. Different normalization conditions lead in general to very different results. The connections among normalization conditions, cut strength and other related issues are investigated in Chapter 3 of this thesis.

Usually, the CGLP is projected onto the support of x^* (see, e.g., [20]). Given a variable x_k such that $x_k^* = 0$, the value of the cut coefficient γ_k does not affect the cut violation. Hence, one can avoid considering the CGLP constraints associated with γ_k. The resulting (reduced) CGLP is then solved and the cut coefficient γ_k is derived afterwards as

$$\gamma_k = \max\{uA_k - u_0\pi_k, vA_k + v_0\pi_k\}, \tag{2.5}$$

where all the Farkas multipliers are fixed as in the optimal solution of the reduced CGLP.

In practice, the disjunction selected for solving CGLP is typically *elementary*, i.e., it involves only one integer variable, thus reading $x_j \leq \lfloor x_j^* \rfloor$ OR $x_j \leq \lceil x_j^* \rceil$ $(j \in J)$. As such, the disjunctive cut only exploits the integrality requirement on a single variable and can therefore be easily improved by an *a posteriori cut strengthening* procedure as the one proposed by Balas and Jeroslow [23].

Theorem 2.1 (Balas and Jeroslow [23]). *Let $(\gamma, \gamma_0, u, v, u_0, v_0)$ be an optimal solution of CGLP with respect to a certain disjunction (π, π_0). Define $m_j = \frac{uA_j - vA_j}{u_0 + v_0}$, and*

$$\tilde{\gamma}_j = \begin{cases} \min\{uA_j - u_0\lfloor m_j \rfloor, vA_j + v_0\lceil m_j \rceil\} & \text{if } j \in J, \\ \max\{uA_j, vA_j\} & \text{if } j \notin J. \end{cases}$$

Then the inequality $\tilde{\gamma}x \geq \gamma_0$ is valid for $conv(S)$ and dominates $\gamma x \geq \gamma_0$ over the set $\{x \in \mathbb{R}^n : x \geq 0\}$.

The strengthening described in the above theorem can be interpreted as finding the best disjunction for the given set of multipliers, by fixing $\pi_j = \lfloor m_j \rfloor$ or $\pi_j = \lceil m_j \rceil$ for any $j \in J$.

2.4 Some connections

GMI cuts and disjunctive cuts are strongly related. On the one side, Nemhauser and Wolsey [128, 129] have shown that the GMI closure and the split closure relative to a polyhedron P are identical. On the other hand, Balas and Perregaard [24, 25] discovered several connections between GMIs from the tableau and basic solutions of CGLP in the context of 0–1 MIPs[3].

In particular, Balas and Perregaard [25] addressed the CGLP truncated with normalization

$$\sum_{i=1}^{m} u_i + \sum_{i=1}^{m} v_i + u_0 + v_0 = 1$$

[3]All the results from [24, 25] were presented in the context of 0–1 MIPs. The results reported in this section also apply to the general-integer case.

and considered elementary disjunctions of the form $x_j \leq 0$ or $x_j \geq 1$. Given a basic solution x^* of the polyhedron P and a row of the corresponding simplex tableau (lifted in the space (x, s) by including also the nonnegative surplus variables $s_i = b_i - a_i x$) associated with a basic variable x_j ($j \in J$, x_j^* fractional), they first showed that the *simple* disjunctive cut which can be read from the given tableau row[4] corresponds to a basic (and, generally, nonoptimal) solution of the CGLP obtained by considering the elementary disjunction on variable x_j. Further, they also showed that, by applying the Balas-Jeroslow [23] strengthening procedure to the above basic solution, one gets precisely the GMI cut associated with the same tableau row. Finally, they discovered a precise correspondence between the (possibly infeasible) bases of the simplex tableau of the polyhedron P in the space (x, s) and the bases of the CGLP. This allowed them to develop an elegant and efficient way of solving the CGLP by making pivot operations in the "natural" tableau involving the original x variables only (plus surplus variables). Such a method represents a crucial speed-up in the implementation of a disjunctive cut separation procedure.

Roughly speaking, the GMI cut from the tableau is a basic solution, generally nonoptimal, of CGLP. Hence, solving the CGLP, in the extended space which involves the Farkas multipliers u, v or in the original space through the procedure developed in [25], can be interpreted as a way of strengthening the GMI cut. A new investigation on this topic is presented in Chapter 3.

[4]Recall that the simple disjunctive cut associated with a tableau row $\tilde{a}_i x + \tilde{g}_i s = x_j^*$ is the cut which can be obtained by applying the disjunction $x_j \leq \lfloor x_j^* \rfloor$ or $x_j \geq \lceil x_j^* \rceil$ to the system $\tilde{a}_i x + \tilde{g}_i s = x_j^*$, $x \geq 0$, $s \geq 0$. See, e.g., [25].

Chapter 3

On the Separation of Disjunctive Cuts

3.1 Introduction

As discussed in Chapter 1, cutting planes (and, in particular, *Gomory mixed integer cuts*) are probably the main ingredient behind the success of the current generation of general purpose MIP solvers. Cutting planes have been widely studied in the literature and the arsenal of separation algorithms has been continuously enlarged over the years. However, there are still several fundamental questions about the use of cutting planes which are probably not fully answered, thus reducing what we could really gain from cuts.

This chapter[1] presents an investigation of the main aspects related to the separation of *disjunctive cuts*, which, as recalled in Chapter 2, are known to be strictly related to Gomory mixed integer cuts.

In the following we consider the MIP

$$\min\{cx : Ax \geq b,\ x_j \in \mathbb{Z}\ \forall j \in J\} \tag{3.1}$$

with bounds on x (if any) included in $Ax \geq b$, where $c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$ are the given objective function and constraint matrix, while $J \subseteq \{1, \ldots, n\}$ denotes the set of variables constrained to be integer. For technical reasons, we assume w.l.o.g. that the system $Ax \geq b$ implies (or contains explicitly) the trivial inequality $0x \geq -1$, in the sense that this latter inequality can be obtained as a nonnegative combination of the rows of $Ax \geq b$[2].

Let x^* denote an optimal solution of the continuous relaxation $\min\{cx : x \in P\}$

[1] The results of this chapter appear in: M. Fischetti, A. Lodi and A. Tramontani, "On the separation of disjunctive cuts", Technical Report OR-08-2, DEIS, University of Bologna, 2008 [85].

[2] For problems with at least one bounded variable, the trivial inequality can always be obtained by adding the bound constraints on a single variable, say $x_j \geq LB_j$ and $-x_j \geq -UB_j$, and dividing the resulting inequality by $UB_j - LB_j > 0$.

where

$$P := \{x \in \mathbb{R}^n : Ax \geq b\}. \tag{3.2}$$

We are given a disjunction of the form

$$\pi x \leq \pi_0 \quad \text{OR} \quad \pi x \geq \pi_0 + 1 \tag{3.3}$$

such that (π, π_0) is integer, $\pi_j = 0, \forall j \notin J$ and $\pi x^* - \pi_0 = \eta^*$, with $\eta^* \in \ ]0, 1[$.

In this chapter[3] we are interested is deriving the "strongest" (in some sense to be discussed later) disjunctive cut $\gamma x \geq \gamma_0$ violated by x^*, according to the classical approach of Balas [16]. (Disjunctive cuts which can be derived by imposing a single disjunction such as (3.3) on a polyhedron P are also known as *split* cuts; see Cook, Kannan and Schrijver [54].) To this end, let us denote by P_0 (respectively, P_1) the polyhedron obtained from P by imposing the additional restriction $\pi x \leq \pi_0$ (resp., $\pi x \geq \pi_0 + 1$). By Farkas lemma, the validity of $\gamma x \geq \gamma_0$ for P_0 and for P_1, and hence for $conv(P_0 \cup P_1)$, can always be certified by means of nonnegative multipliers (u, u_0, v, v_0) associated with the inequalities defining P_0 and P_1 according to the following scheme:

		P_0					P_1		
(u)	Ax	$\geq$	b		(v)	Ax	$\geq$	b
(u_0)	$-\pi x$	$\geq$	$-\pi_0$		(v_0)	πx	$\geq$	$\pi_0 + 1$

A most-violated disjunctive cut can therefore be found by solving the following *Cut Generating Linear Program* (CGLP) that determines the Farkas multipliers so as to maximize the violation of the resulting cut with respect to the given point x^*:

$$\begin{aligned}
(\text{CGLP}) \quad \min \quad & \gamma x^* - \gamma_0 && (3.4)\\
\gamma \quad & = uA - u_0\pi && (3.5)\\
\gamma \quad & = vA + v_0\pi && (3.6)\\
\gamma_0 \quad & = ub - u_0\pi_0 && (3.7)\\
\gamma_0 \quad & = vb + v_0(\pi_0 + 1). && (3.8)\\
& u, v, u_0, v_0 \geq 0 && (3.9)
\end{aligned}$$

Note that, according to Farkas lemma, the two equations (3.7) and (3.8) defining γ_0 should be relaxed into $\leq$ inequalities. However it is not difficult to see that, due to the (possibly implicit) presence of the trivial inequality $0x \geq -1$, one can always require that equality holds in both cases.

By construction, any feasible CGLP solution with negative objective function value corresponds to a violated disjunctive cut. However, as stated, the feasible CGLP set is

[3]The main steps related to a typical disjunctive cut separation procedure (i.e., projection onto the support, subsequent lifting, *a posteriori cut strengthening* by changing the disjunction) have been already discussed in Chapter 2. To avoid confusion, these steps are briefly recalled even in this chapter, in the context of the required different notation, in which the bounds on x variables are eventually included in the system $Ax \geq b$.

a cone and needs to be truncated so as to produce a bounded LP in case a violated cut exists. This crucial step will be addressed in the next section.

Usually, the CGLP is projected onto the support of x^*. Given a variable x_k restricted to be nonnegative and such that $x_k^* = 0$[4], it is well known [20] that one can project x_k away. More precisely, one can avoid considering the CGLP constraints associated with γ_k and neglect constraint $x_k \geq 0$ in both P_0 and P_1. The resulting (reduced) CGLP is then solved and the cut coefficient γ_k is derived afterwards by solving the trivial lifting problem

$$\min\{\gamma_k : \gamma_k = uA_k - u_0\pi_k = vA_k + v_0\pi_k, \quad u, v \geq 0\}, \tag{3.10}$$

where the Farkas multipliers u and v are fixed as in the optimal solution of the reduced CGLP but those related to the previously neglected bound constraint $x_k \geq 0$.

In practice, disjunction (3.3) is typically *elementary*, i.e., it involves only one integer variable and it reads $x_j \leq \lfloor x_j^* \rfloor$ OR $x_j \geq \lceil x_j^* \rceil$, with $j \in J$ and x_j^* fractional. As such, the disjunctive cut only exploits the integrality requirement on a single variable and can therefore be improved easily by an *a posteriori cut strengthening* procedure as the one proposed by Balas and Jeroslow [23]. As already discussed in Chapter 2, such a strengthening can be also interpreted as finding the best disjunction for the given set of multipliers.

Recently, Balas and Perregaard [25] developed an elegant and efficient way of solving the CGLP by making pivot operations in the "natural" tableau involving the original x variables only (plus surplus variables), which represents a crucial speed-up in the implementation of the method.

In this chapter we investigate computationally the main ingredients of a disjunctive cut separation procedure, and analyze their impact on the overall performance at the root node of the branching tree. To be more specific, we consider a testbed of MIPs taken from MIPLIB library [36]. For each instance, we solve the root-node LP relaxation and generate 10 rounds of disjunctive cuts computed according to alternative strategies. In each round, a violated disjunctive cut is generated for each fractional LP components x_j^*, by exploiting the disjunction $x_j \leq \lfloor x_j^* \rfloor$ OR $x_j \geq \lceil x_j^* \rceil$. In order to limit possible side effects, no *a posteriori* cut strengthening procedure is applied, unless otherwise stated.

The chapter is organized as follows. In Section 3.2 we compare classical normalization conditions used to truncate the CGLP cone, and try to better understand their role. In Section 3.3 we characterize weak rays/vertices of the CGLP leading to dominated cuts and we propose a practical heuristic method to strengthen them. In Section 3.4 we show that using redundant constraints in the CGLP can lead to very weak cuts, and we analyze such an issue with respect to the normalization used. In Section 3.5 we introduce a new normalization which is particularly suited for set-covering type problems and we analyze its theoretical properties and computational behavior. Finally, some conclusions are drawn in Section 3.6.

[4]Of course, variables with nonzero lower bound can be shifted, while variables at the upper bound can be complemented.

3.2 The role of normalization

In order to truncate the CGLP cone one can introduce a suitable cut normalization condition expressed as a linear (in)equality. A possible normalization, called *trivial* in the sequel, is as follows:

$$u_0 + v_0 = 1. \tag{3.11}$$

One of the most widely-used (and effective) truncation condition, called the *Standard Normalization Condition* (SNC) in the following, reads instead:

$$\sum_{i=1}^{m} u_i + \sum_{i=1}^{m} v_i + u_0 + v_0 = 1. \tag{3.12}$$

This latter condition was proposed in Balas [17] and investigated by Ceria and Soares [45] and by Balas and Perregaard [24, 25]. (Obviously, (3.11) and (3.12) are just two among the possible normalization conditions for truncating the CGLP cone, probably the simplest and the most widely used, respectively. For a detailed account on other normalization conditions we refer the reader to [45, 24].)

The choice of the normalization condition turns out to be crucial for an effective selection of a "strong" disjunctive cut in that it affects heavily the choice of the optimal CGLP solution. To see this it is enough to observe that, since the CGLP feasible set is a cone and assuming a violated cut exists, one can always swap the role of the objective function and of the normalization condition. In other words, one could equivalently fix the objective function to a given negative value (say, -1) so as to only allow for violated cuts, and use the left-hand side of the normalization condition as the objective function to be minimized. Hence, the actual CGLP "optimal" cut depends heavily on the normalization condition.

Balas and Perregaard [25] showed that the well-known Gomory Mixed-Integer (GMI) cut [99] is a basic solution of the CGLP when either the SNC or the trivial normalization is applied. Our first result is to prove that this solution is indeed optimal when the trivial normalization (3.11) is used. We start with a useful lemma.

Lemma 3.1 *Let $x^* \in P$ and let $(\gamma, \gamma_0, u, v, u_0, v_0)$ be a feasible solution of the CGLP (3.4)-(3.9). Then valid upper bounds on the cut violation can be computed as follows:*

$$\begin{aligned}
&\textit{UB1:} \quad && \gamma_0 - \gamma x^* \leq u_0 \eta^* \\
&\textit{UB2:} \quad && \gamma_0 - \gamma x^* \leq v_0 (1 - \eta^*) \\
&\textit{UB3:} \quad && \gamma_0 - \gamma x^* \leq (u_0 + v_0)\,(1 - \eta^*)\,\eta^*.
\end{aligned}$$

Proof. Because of (3.5) and (3.7), $\gamma x^* - \gamma_0 = u(Ax^* - b) - u_0(\pi x^* - \pi_0) \geq -u_0\eta^*$. Analogously, from (3.6) and (3.8) we obtain $\gamma x^* - \gamma_0 = v(Ax^* - b) + v_0(\pi x^* - \pi_0 - 1) \geq -v_0(1 - \eta^*)$. Adding up the two inequalities above weighed by $1 - \eta^*$ and η^*, respectively, one gets the claimed UB3 bound. □

Given a vertex x^* of P and the associated basis, the next theorem shows how to compute a solution of the CGLP whose violation is equal to bound UB3 above—for any given disjunction (3.3). Moreover, as shown in [24, 25], for an appropriate choice of a non-elementary disjunction this CGLP solution yields precisely a GMI cut associated with the optimal LP tableau (see Chapter 2, Section 2.4). As a consequence of Lemma 3.1, this easily-computable cut has a violation that is optimal among the cuts with constant $u_0 + v_0$, i.e., when the trivial normalization (3.11) is imposed. Note however that this is not necessarily the case when a different normalization (in particular, the SNC one) is applied.

For any vector v, let operator $[v]_+$ takes the maximum between the argument and zero (componentwise); by definition, $v \equiv [v]_+ - [-v]_+$ with $[v]_+ \geq 0$ and $[-v]_+ \geq 0$.

Theorem 3.1 *Assume w.l.o.g. $rank(A) = n$. Given a vertex x^* of P, let system $Ax \geq b$ be partitioned into $Bx \geq b_B$ and $Nx \geq b_N$, where $Bx^* = b_B$ and B is an $n \times n$ nonsingular matrix. Let (u_B, v_B) and (u_N, v_N) denote the Farkas multipliers associated with the rows of B and N, respectively. For a given disjunction (3.3) with $\eta^* = \pi x^* - \pi_0 \in [0, 1]$, let $u_0^* = 1 - \eta^*$, $v_0^* = \eta^*$, $u_N^* = v_N^* = 0$, $u_B^* = [\pi B^{-1}]_+$ and $v_B^* = [-\pi B^{-1}]_+$, while γ^* and γ_0^* are defined through equations (3.5) and (3.7), respectively. Then $(\gamma^*, \gamma_0^*, u^*, v^*, u_0^*, v_0^*)$ is an optimal CGLP solution w.r.t. the trivial normalization (3.11).*

Proof. We first prove feasibility. Consistency between (3.5) and (3.6) requires $u^*A - u_0^*\pi = v^*A + v_0^*\pi$, i.e., $u_B^* - v_B^* = (u_0^* + v_0^*)\pi B^{-1} = \pi B^{-1}$, a condition that follows directly from the definition of u_B^* and v_B^*. Analogously, consistency between (3.7) and (3.8) requires $(u_B^* - v_B^*)b_B = (u_0^* + v_0^*)\pi_0 + v_0^*$, i.e., $\pi B^{-1} b_B = \pi_0 + v_0^*$. This latter equation is indeed satisfied because $B^{-1}b_B = x^*$ and $v_0^* = \eta^* = \pi x^* - \pi_0$. As to optimality, we first observe that $u_0^* + v_0^* = 1$ holds by definition. Because of (3.5) and (3.7), $\gamma x^* - \gamma_0 = u^*(Ax^* - b) - u_0^*(\pi x^* - \pi_0) = u_B^*(Bx^* - b_B) + u_N^*(Nx^* - b_N) - u_0^*\eta^* = 0 + 0 - (1 - \eta^*)\eta^*$, hence the cut violation attains bound UB3 of Lemma 3.1. □

The theorem above shows that, in case the trivial normalization is adopted, the CGLP can be solved in a closed form for any vertex x^*. Moreover, with this normalization, in all optimal CGLP solutions the slack constraints receive a null Farkas multiplier, i.e., only tight constraints play a role in the cut derivation. This is an unnecessary restriction that can actually lead to weak cuts, as computationally shown in the sequel.

The first set of experiments we designed was aimed at evaluating the actual practical impact of different normalization conditions. In particular, we compared the SNC normalization (3.12) with the alternative trivial normalization (3.11) by warm starting each CGLP with the basic feasible solution from Theorem 3.1[5]. Moreover, unless explicitly stated, CGLP is projected onto the support of x^*, possibly after complementing and shifting variables at their bound.

[5]The Balas and Perregaard [25] technique working on the original tableau and the solution of CGLP by using the GMI as a warm start, are equivalent procedures.

The outcome of our experiments is given in Table 3.1. As already mentioned, we applied 10 rounds of cuts. At each round, a cut was generated from each fractional variable. No a-posteriori cut strengthening was applied. As usual, the CGLP is solved projected on the x^* support. Instances denoted as "*" are neglected in the average computations. The table reports (i) the number of separated cuts, (ii) the quality of the lower bound (i.e., percentage gap closed at the root node) and (iii) the average cardinality of the support of vector $u + v$, denoted as $S(u, v) := \{i \in \{1, \ldots, m\} : u_i + v_i > 0\}$ ($|S|$ for short), i.e., how many constraints are actually used, on average, to generate a cut.

Table 3.1: Trivial vs. SNC normalization.

	Trivial normalization (GMI)			SNC normalization		
Instance	# cuts	%gap	$\|S\|$	# cuts	%gap	$\|S\|$
bell3a	137	70.74	59.49	71	70.74	43.72
bell5	202	28.18	31.20	178	94.29	11.75
blend2	156	28.73	11.70	192	30.51	8.10
flugpl	93	15.15	7.57	92	18.36	5.85
gt2	191	98.71	14.52	196	93.46	10.28
lseu	152	32.94	14.34	196	41.33	9.17
*markshare1	68	0.00	1.00	74	0.00	1.39
mod008	104	12.09	10.40	139	17.05	12.41
p0033	103	58.33	5.72	113	67.86	4.81
p0201	574	18.58	56.03	767	93.82	13.43
rout	445	8.52	135.39	434	24.26	68.07
*stein27	235	0.00	19.74	252	0.00	6.53
vpm1	255	36.95	9.03	263	55.84	5.39
vpm2	424	42.08	71.72	403	74.96	17.27
avg.	236.333	37.583	35.593	253.667	56.873	17.521

Table 3.1 shows clearly that normalizations (3.12) and (3.11) yield quite different results. As a matter of fact, the dual support of cuts separated with (3.12) is much sparser (i.e., less constraints are used in the cut derivation) and the quality of final bound is significantly improved. To get more insights on the different behaviors of (3.12) and (3.11), for instance `p0201` we provide a full picture of the main differences between the separated inequalities.

Figures 3.1–3.3 report, for each iteration, the dual bound reached after adding the cuts, the average density of the cuts (i.e., the number of nonzero coefficients), and the average cardinality of $S(u, v)$. Figure 3.4 reports, for each $k = 1, \ldots, 10$, the number of separated cuts having "rank" k. Here, we use a relaxed definition of rank, namely we compute the rank $rnk(\gamma, \gamma_0)$ of a cut $\gamma x \geq \gamma_0$ as

$$rnk(\gamma, \gamma_0) := 1 + \max_{i \in S(u,v)} rnk(a_i, b_i),$$

where $rnk(a_i, b_i)$ is the rank of constraint $a_i x \geq b_i$ (constraints in the original formula-

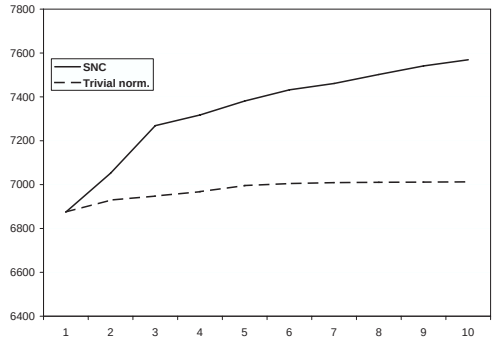

Figure 3.1: SNC vs. GMI: dual bound for instance $p0201$.

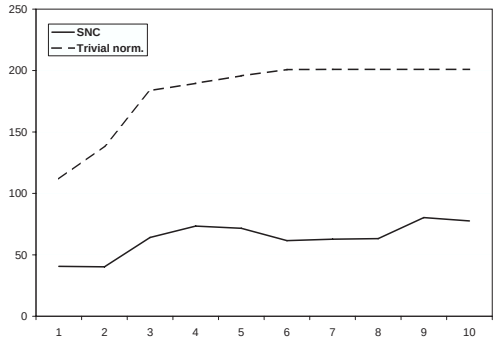

Figure 3.2: SNC vs. GMI: avg. cut density for instance $p0201$.

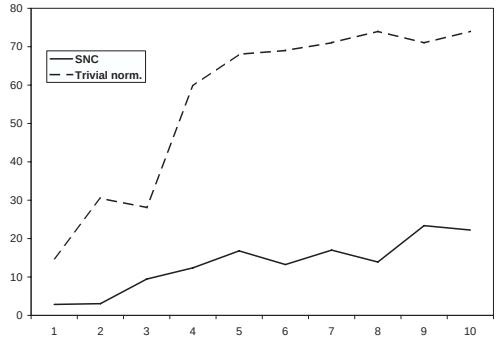

Figure 3.3: SNC vs. GMI: avg. cardinality of $S(u, v)$ for instance $p0201$.

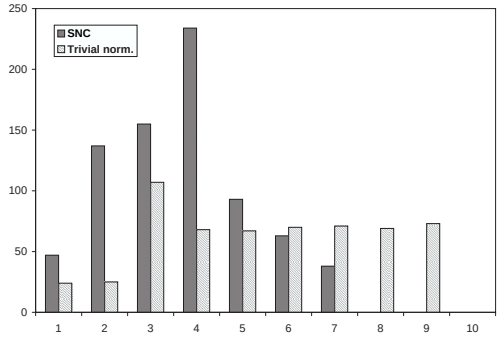

Figure 3.4: SNC vs. GMI: cut rank for instance $p0201$.

tion are defined to be of zero rank)[6].

3.2.1 Why does SNC normalization work so well?

A careful analysis of the computational results in Table 3.1 and Figures 3.1–3.4 reveal a very (tricky but) important feature of the SNC scheme that improves significantly its performance. Indeed, it turns out that the use of the SNC normalization (3.12) enforces the following very nice properties:

1. The norm of the separated cuts tends to become smaller and smaller as a result of the small multipliers used for the newly generated cuts (that is, in turn, a consequence of having limited the multiplier sum to 1). This means that the separated cuts inserted in the LP are automatically scaled so as to have "small coefficients". Therefore, in the subsequent iterations these cuts would need big Farkas multipliers to become relevant, a situation that is however penalized by the normalization condition itself. As a consequence, the normalization penalizes

[6]Note that this way of computing the rank provides just an upper bound on the classical definition of Chvátal rank [49].

implicitly the rank of the cuts to be generated, because high-rank cuts will be "expensive" in terms of multiplier sum, hence low-rank cuts tend to be separated at each step.

2. Since low-rank cuts are preferred and since the original (rank-0) inequalities are generally sparse, the separated cuts tend to remain sparse; this is also a consequence of the fact that the SNC normalization tends to reduce the sum of the components of the Farkas multiplier vector and hence it increases the sparsity of its support, so a small number of constraints are typically used in the disjunctive cut derivation.

Trivial normalization (3.11), instead, takes care only of the Farkas multipliers u_0 and v_0 associated with the disjunction. Indeed, as shown in Section 3.2, only constraints which are tight at x^* are used in the cut derivation, thus the rank of the cuts increases very quickly, basically at each iteration. Moreover, all other constraint multipliers are not penalized, hence (i) several constraints are used in the cut derivation, thus cuts increase their density, and (ii) Farkas multipliers can assume huge values, thus the subsequent cut lifting procedure may produce very weak coefficients for the variables outside the support of x^*.

In the SNC normalization case the coefficient lifting is not an issue. Indeed, since all the constraint multipliers in the SNC normalization are penalized and each multiplier tends to be small, the coefficient lifting of the variables outside the support of x^* – to be performed afterwards – is "safe", i.e., also the coefficients of these variables remain under control.

3.2.2 Nothing is perfect!

Although it produced good results in the experiments reported in Table 3.1, there are cases where normalization (3.12) may lead to very weak disjunctive cuts.

Bad Scaling.

A bad feature of the SNC normalization is its dependency on the relative scaling of the constraints, in the sense that the relative size of the Farkas multipliers (whose sum is fixed to 1) depends on the relative size of the coefficients of the corresponding constraints. Indeed, it is easy to see that the multiplication by a positive factor ϕ of the i-th constraint in the system $Ax \geq b$ implies that the corresponding u_i and v_i multipliers are divided by ϕ, which in turn is equivalent to use a coefficient $1/\phi$ (instead of 1) in the normalization condition (3.12). Thus, the scaled constraint is "cheaper" if one interprets the right hand side of (3.12) as a resource.

The following experiment clearly demonstrates this unstable behavior: we ran the CGLP code with the classical SNC normalization condition, as in Table 3.1, but we just multiplied by 1,000 each disjunctive cut before its addition to the current LP. At first glance, one could guess that this "innocent change" would not have any impact on the overall performance, but the actual results reported in Table 3.2 show that this is definitely not the case.

Table 3.2: "Classical" SNC approach vs. "Bad scaled" SNC approach.

	"Classical" SNC			"Bad scaled" SNC		
Instance	# cuts	%gap	$\|S\|$	# cuts	%gap	$\|S\|$
bell3a	71	70.74	43.72	69	70.74	44.32
bell5	178	94.29	11.75	214	88.83	17.47
blend2	192	30.51	8.10	166	28.91	11.71
flugpl	92	18.36	5.85	90	15.40	7.40
gt2	196	93.46	10.28	184	93.42	17.22
lseu	196	41.33	9.17	137	38.58	10.88
*markshare1	74	0.00	1.39	206	0.00	14.60
mod008	139	17.05	12.41	104	3.90	10.21
p0033	113	67.86	4.81	94	57.09	6.40
p0201	767	93.82	13.43	610	49.91	45.72
rout	434	24.26	68.07	435	13.03	152.66
*stein27	252	0.00	6.53	248	0.00	22.39
vpm1	263	55.84	5.39	244	47.59	8.50
vpm2	403	74.96	17.27	420	54.39	22.27
avg.	253.667	56.873	17.521	230.583	46.816	29.563

As explained, multiplying by 1,000 the generated cuts is equivalent to dividing by 1,000 the coefficient of the corresponding Farkas multipliers u_i and v_i in the normalization condition, so we actually weaken the penalty on the choice $u_i + v_i > 0$ that leads to low-rank sparse cuts. In other words, the scaling operation interferes with the nice SNC tendency of producing low-rank cuts, and the overall performance deteriorates significantly, as shown in detail for problem `p0201` in Figures 3.5–3.8. (Incidentally, the above discussion shows the importance of "small implementation details" when evaluating the performance of a method–two apparently equivalent implementations of precisely the same idea lead to very different outcomes.)

Bad Examples.

Even for toy instances, the CGLP can have hard time in finding a good disjunctive cut. This is illustrated by the following two simple 2-dimensional cases, where the optimal CGLP solution may correspond to very weak cuts.

Example 3.1 *Consider the simple ILP whose continuous relaxation, depicted in Figure*

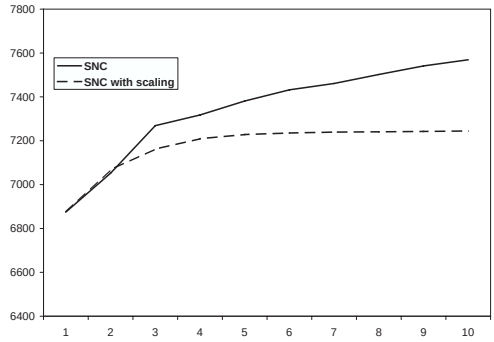

Figure 3.5: "Classical" SNC vs. "Bad scaled" SNC: dual bound for instance $p0201$.

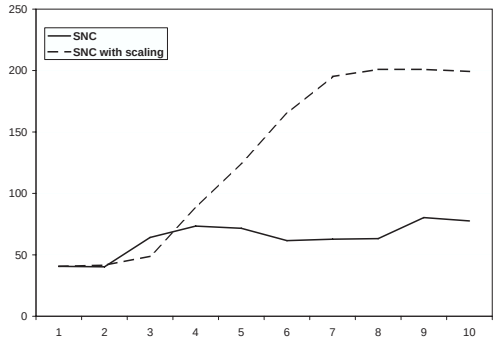

Figure 3.6: "Classical" SNC vs. "Bad scaled" SNC: avg. cut density for instance $p0201$.

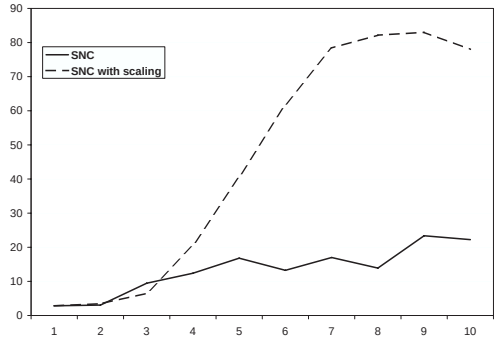

Figure 3.7: "Classical" SNC vs. "Bad scaled" SNC: avg. cardinality of $S(u, v)$ for instance $p0201$.

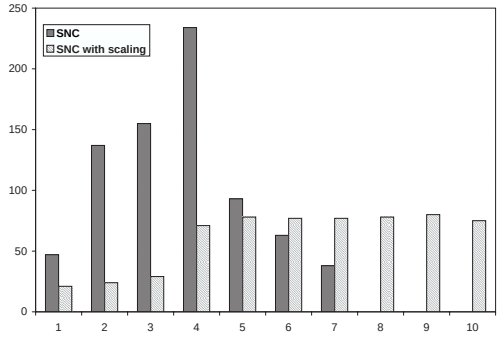

Figure 3.8: "Classical" SNC vs. "Bad scaled" SNC: cut rank for instance $p0201$.

3.9, has one of the constraints, namely (a5), scaled by a parameter $k > 0$:

$$\begin{array}{lrrcr} \min & -x_1 & -2x_2 & & \\ (a1) & 4x_1 & -4x_2 & \geq & -2 \\ (a2) & -2x_1 & -2x_2 & \geq & -3 \\ (a3) & 8x_1 & -4x_2 & \geq & -1 \\ (a4) & -x_1 & & \geq & -1 \\ (a5) & & -kx_2 & \geq & -k \\ (a6) & x_1 & & \geq & 0 \\ (a7) & & x_2 & \geq & 0 \end{array}$$

The optimal solution of the LP relaxation is $x^* = (\frac{1}{2}, 1)$ and three cuts can be derived from disjunction $x_1 \leq 0$ OR $x_1 \geq 1$, namely:

(c1) $2x_2 \leq 1$, *corresponding to the basic solution of the CGLP (u_1, v_2, u_0, v_0), of value $z_1 = -\frac{2}{11}$, optimal for $k \leq 8$;*

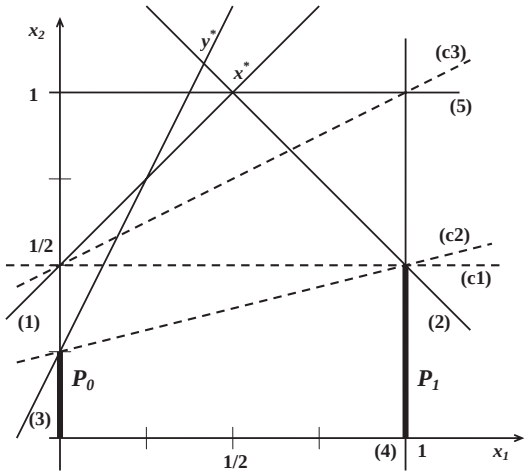

Figure 3.9: Example 3.1 depicted.

(c2) $-x_1 + 4x_2 \leq 1$, *corresponding to the basic solution of the CGLP* (u_3, v_2, u_0, v_0), *of value* $z_2 = -\frac{1}{6}$, *never optimal.*

(c3) $-x_1 + 2x_2 \leq 1$, *corresponding to the basic solution of the CGLP* (u_1, v_5, u_0, v_0), *of value* $z_3 = -\frac{k}{4+5k}$, *optimal for* $k \geq 8$.

So, depending on the value of k, *the optimal CGLP solution corresponds to weak cuts, either (c1) or (c3), whereas the facet-defining cut (c2) will never be selected.* □

Note that the redundant (with respect to P) constraint (a5) is only used in the above example to show dependency on scaling. In fact, such a constraint can be removed without making cut (c1) optimal—the constraint is indeed removed in the slightly modified example that follows.

Example 3.2 *For the ILP whose continuous relaxation is depicted in Figure 3.10:*

$$\begin{array}{lrrcr} \min & -x_1 & -2x_2 & & \\ (b1) & 2x_1 & -2x_2 & \geq & -1 \\ (b2) & -2x_1 & -2x_2 & \geq & -3 \\ (b3) & 4x_1 & +4x_2 & \geq & 3 \\ (b4) & -x_1 & & \geq & -1 \\ (b5) & x_1 & & \geq & 0 \\ (b6) & & x_2 & \geq & 0 \end{array}$$

the optimal solution of the continuous relaxation is again $x^* = (\frac{1}{2}, 1)$ *and three cuts can be derived from disjunction* $x_1 \leq 0$ *OR* $x_1 \geq 1$ *(yielding* $P_0 = \emptyset$*), namely:*

(c1) $2x_2 \leq 1$, *corresponding to the basic solution of the CGLP* (u_1, v_2, u_0, v_0), *of value* $z_1 = -\frac{1}{6}$ *(optimal);*

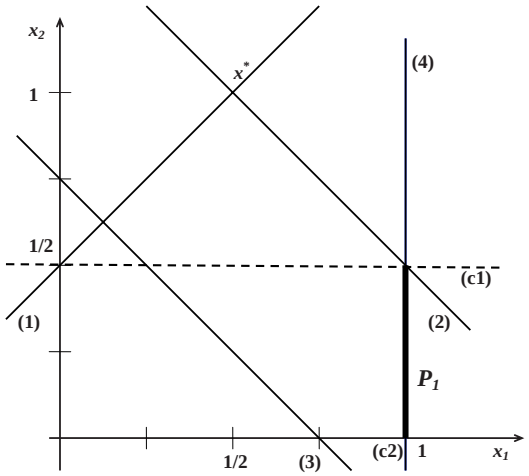

Figure 3.10: Example 3.2 depicted.

(c2) $x_1 \geq 1$, *corresponding to the basic solution of the CGLP* (u_1, u_3, u_0, v_0), *of value* $z_2 = -\frac{1}{22}$ *(nonoptimal).*

Since (c2) is not optimal for the associated CGLP, the only facet-defining cut (c2) will not be selected. □

3.2.3 Comments

The examples above show clearly the following fact: even if the solution of the CGLP is a vertex, the corresponding disjunctive cut can be very weak. At first glance, this may be seen as a counter-intuitive result as one would expect that CGLP vertices correspond to facets of $conv(P_0 \cup P_1)$. This is however not the case, as discussed e.g. in Balas and Perregaard [24], since the CGLP is not defined in the "natural" reverse polar space (γ, γ_0) but in an enlarged space involving the Farkas variables explicitly. As a matter of fact, in the extended space $(\gamma, \gamma_0, u, v, u_0, v_0)$ there are several rays/vertices whose projection in the (γ, γ_0) space is nonextremal, therefore the corresponding cut can be obtained as the sum of other valid cuts and hence is dominated. By using software PORTA [46] we can get a clear picture of the situation in Example 3.1. In the natural polar space (γ, γ_0), the projected CGLP cone has only 4 extreme rays that correspond to the facets of $conv(P_0 \cup P_1)$. In space $(\gamma, \gamma_0, u, v, u_0, v_0)$, instead, the CGLP cone has 117 extreme rays that correspond to 117 vertices once normalization (3.12) is applied. Only 6 of these vertices correspond to violated constraints, and 3 of them correspond to the cuts depicted in Figure 3.9. So, most CGLP vertices in the $(\gamma, \gamma_0, u, v, u_0, v_0)$ space correspond to very weak cuts, and the cut separation procedure can be in trouble in returning a facet-defining cut even in this toy example. As mentioned above, this is essentially due to the fact that the cut is separated in the extended space $(\gamma, \gamma_0, u, v, u_0, v_0)$, where a dominated cut could turn out not to be dominated in terms of the multipliers used for its generation. For instance, 3 extreme rays of the CGLP cone for Example 3.1 are

reported below.

	γ_1	γ_2	γ_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	v_1	v_2	v_3	v_4	v_5	v_6	v_7	u_0	v_0
(r_1)	1	-4	-1	0	0	1	0	0	0	0	0	2	0	0	0	0	0	7	5
(r_2)	-1	0	-1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
(r_3)	0	-4	-2	1	0	0	0	0	0	0	0	2	0	0	0	0	0	4	4

In the (γ, γ_0) space, the third constraint is clearly dominated as it is just the sum of the previous ones, but there is no way to obtain ray r_3 as conic combination of rays r_1 and r_2 in the extended space, due to the presence of the Farkas components. The above drawback is even more evident in Example 3.2 where $P_0 = \emptyset$, hence $x_1 \geq 1$ itself is a valid cut (c2), but not the best one for the CGLP.

3.3 Weak CGLP rays/vertices and dominated cuts

The examples in the previous section show that some rays/vertices of the CGLP lead to weak cuts and should not be used. In the next section we formally characterize those rays/vertices which correspond to cuts that are trivially dominated by other cuts associated with solutions of the same CGLP (Section 3.3.1). In Section 3.3.2 we propose a heuristic procedure to strengthen disjunctive cuts associated with dominated rays/vertices, whose practical effect is computationally investigated in Section 3.3.3.

3.3.1 Characterization

The first step to characterize weak rays/vertices is the following definition.

Definition 3.1 *(Strictly dominated cuts) Let $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ be a cut valid for $conv(P_0 \cup P_1)$ but not for P. If there exists another cut $\overline{\gamma}x \geq \overline{\gamma}_0$ valid for $conv(P_0 \cup P_1)$ such that $\{x \in P : \overline{\gamma}x \geq \overline{\gamma}_0\} \subsetneq \{x \in P : \widetilde{\gamma}x \geq \widetilde{\gamma}_0\}$, then the cut $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ is said to be strictly dominated w.r.t. P.*

Note that, in the above definition, the domination of cut $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ only depends on a single other cut ($\overline{\gamma}x \geq \overline{\gamma}_0$).

Lemma 3.2 *Let $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ be a valid cut for $conv(P_0 \cup P_1)$ such that $\widetilde{P} := \{x \in P : \widetilde{\gamma}x \geq \widetilde{\gamma}_0\} \subsetneq P$, and assume $\widetilde{P}$ full dimensional. If there exists another cut $\overline{\gamma}x \geq \overline{\gamma}_0$ valid for $conv(P_0 \cup P_1)$ and such that $\widetilde{\gamma} = \overline{\gamma} + \mu A$, $\widetilde{\gamma}_0 = \overline{\gamma}_0 + \mu b$ for a certain $\mu \in \mathbb{R}^m_+ \setminus \{0\}$, then $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ is strictly dominated w.r.t. P.*

Proof. Define $\overline{P} := \{x \in P : \overline{\gamma}x \geq \overline{\gamma}_0\}$. By definition, $x \in P$ and $\overline{\gamma}x \geq \overline{\gamma}_0$ imply $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$, hence $\overline{P} \subseteq \widetilde{P}$. We need to show that the above inclusion is always strict. Indeed, let $\widetilde{F} := \{x \in P : \widetilde{\gamma}x = \widetilde{\gamma}_0\}$ denote the face of $\widetilde{P}$ induced by $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$, and consider any given $h \in \{1, \ldots, m\}$ such that $\mu_h > 0$. Since $\widetilde{P}$ is full dimensional, there exists $\hat{x} \in \widetilde{F}$ such that $a_h\hat{x} > b_h$ (otherwise $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ would be a positive multiple of $a_hx \geq b_h$, impossible since we are assuming $\widetilde{P} \subsetneq P$). Hence $\overline{\gamma}\hat{x} - \overline{\gamma}_0 = (\widetilde{\gamma}\hat{x} - \widetilde{\gamma}_0) - \mu(A\hat{x} - b) \leq -\mu_h(a_h\hat{x} - b_h) < 0$, i.e., $\hat{x} \in \widetilde{P} \setminus \overline{P}$. □

For any feasible solution $(\gamma, \gamma_0, u, v, u_0, v_0)$ of (3.5)–(3.9), we denote by $S(u) := \{i \in \{1, \ldots, m\} : u_i > 0\}$ and $S(v) := \{i \in \{1, \ldots, m\} : v_i > 0\}$ the support of vectors u and v, respectively. It is not difficult to show that in any extreme ray of (3.5)–(3.9) yielding a cut nonvalid for P, both u_0 and v_0 are strictly positive, while $S(u)$ and $S(v)$ are disjoint. This property is also inherited by the vertices of the CGLP with normalization (3.12) (see, Balas and Perregaard [25]). We next give a characterization of the extreme rays/vertices of the CGLP that lead to strictly dominated cuts according to Definition 3.1.

Theorem 3.2 *Assume $conv(P_0 \cup P_1)$ full dimensional. Let $(\widetilde{\gamma}, \widetilde{\gamma}_0, \widetilde{u}, \widetilde{v}, \widetilde{u}_0, \widetilde{v}_0)$ be an extreme ray of the CGLP cone (3.5)–(3.9) corresponding to a cut $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ nonvalid for P. Then $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ is strictly dominated w.r.t. P if and only if there exists a feasible solution $(\widetilde{\gamma}, \widetilde{\gamma}_0, \hat{u}, \hat{v}, \hat{u}_0, \hat{v}_0)$ of (3.5)–(3.9) such that $S(\hat{u}) \cap S(\hat{v}) \neq \emptyset$.*

Proof. We first prove the if condition. Given a feasible solution $(\widetilde{\gamma}, \widetilde{\gamma}_0, \hat{u}, \hat{v}, \hat{u}_0, \hat{v}_0)$ of (3.5)–(3.9) such that $S(\hat{u}) \cap S(\hat{v}) \neq \emptyset$, define $\mu = \min\{\hat{u}, \hat{v}\}$ (componentwise) and note that $\mu_i > 0$ for any $i \in S(\hat{u}) \cap S(\hat{v})$. Then, define $\overline{u} = \hat{u} - \mu \geq 0$, $\overline{v} = \hat{v} - \mu \geq 0$, $\overline{\gamma} = \widetilde{\gamma} - \mu A$, $\overline{\gamma}_0 = \widetilde{\gamma}_0 - \mu b$. Since $(\overline{\gamma}, \overline{\gamma}_0, \overline{u}, \overline{v}, \hat{u}_0, \hat{v}_0)$ is a feasible solution of (3.5)–(3.9), the cut $\overline{\gamma}x \geq \overline{\gamma}_0$ is valid for $conv(P_0 \cup P_1)$ and dominates $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ w.r.t. P from Lemma 3.2. Concerning the only if condition, assume $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ to be strictly dominated w.r.t. P by $\overline{\gamma}x \geq \overline{\gamma}_0$, and let $(\overline{\gamma}, \overline{\gamma}_0, \overline{u}, \overline{v}, \overline{u}_0, \overline{v}_0)$ be a feasible solution of (3.5)–(3.9) yielding the dominating cut. Then, there exist $\mu \in \mathbb{R}^m_+ \setminus \{0\}$ and $\mu_0 > 0$ such that $\widetilde{\gamma} = \mu A + \mu_0 \overline{\gamma}$, $\widetilde{\gamma}_0 = \mu b + \mu_0 \overline{\gamma}_0$. Hence $(\widetilde{\gamma}, \widetilde{\gamma}_0, \hat{u}, \hat{v}, \hat{u}_0, \hat{v}_0)$ is a feasible solution of (3.5)–(3.9) yielding the dominated cut, where $\hat{u} = \mu + \mu_0 \overline{u}$, $\hat{v} = \mu + \mu_0 \overline{v}$, $\hat{u}_0 = \mu_0 \overline{u}_0$, $\hat{v}_0 = \mu_0 \overline{v}_0$ and $S(\hat{u}) \cap S(\hat{v}) \neq \emptyset$. □

Corollary 3.1 *Let $(\gamma, \gamma_0, u, v, u_0, v_0)$ be an optimal solution of the CGLP with normalization (3.12), yielding a cut violated by x^* (i.e., $\gamma x^* - \gamma_0 < 0$). Then $S(u) \cap S(v) = \emptyset$.*

Note that the above corollary holds even if the CGLP cone is truncated with a more general normalization than (3.12), e.g., the one to be discussed in Section 3.5.

3.3.2 Strengthening

Theorem 3.2 above suggests a way to strengthen disjunctive cuts arising from weak rays/vertices of the CGLP. Let us assume to be given a vertex $(\widetilde{\gamma}, \widetilde{\gamma}_0, \widetilde{u}, \widetilde{v}, \widetilde{u}_0, \widetilde{v}_0)$ of the CGLP associated with a valid disjunction (3.3) and truncated by any normalization, e.g., (3.11) or (3.12). Consider the following LP:

$$\begin{aligned}
\max \quad & 1^T \mu && (3.13)\\
\widetilde{\gamma} \quad & = (u + \mu)A - u_0 \pi && (3.14)\\
\widetilde{\gamma} \quad & = (v + \mu)A + v_0 \pi && (3.15)\\
\widetilde{\gamma}_0 \quad & = (u + \mu)b - u_0 \pi_0 && (3.16)\\
\widetilde{\gamma}_0 \quad & = (v + \mu)b + v_0(\pi_0 + 1) && (3.17)\\
& u, v, \mu, u_0, v_0 \geq 0, && (3.18)
\end{aligned}$$

where $(\widetilde{\gamma}, \widetilde{\gamma}_0)$ is fixed. Assuming $conv(P_0 \cup P_1)$ to be full dimensional, it is not difficult to see that the above LP is always bounded and the optimal solution value is greater than 0 if and only if $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ is strictly dominated w.r.t. P. Moreover, in this case any optimal solution $(\overline{u}, \overline{v}, \overline{u}_0, \overline{v}_0, \overline{\mu})$ of (3.13)–(3.18) yields a valid cut $\overline{\gamma}x \geq \overline{\gamma}_0$ for $conv(P_0 \cup P_1)$, computed as $\overline{\gamma} = \overline{u}A - \overline{u}_0\pi = \overline{v}A + \overline{v}_0\pi$, $\overline{\gamma}_0 = \overline{u}b - \overline{u}_0\pi_0 = \overline{v}b + \overline{v}_0(\pi_0 + 1)$, which strictly dominates $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ w.r.t P. However, the LP (3.13)–(3.18) involves three sets of Farkas multipliers and might be quite time consuming in practice.

A practical heuristic way to look for a dominating cut is based on the following *Cut Dominating LP* (CDLP), which uses only two sets of Farkas multipliers as in the CGLP:

$$
\begin{aligned}
\text{(CDLP)} \quad \max \quad & z = \sum_{i \in S(\widetilde{v})} u_i + \sum_{i \in S(\widetilde{u})} v_i && (3.19)\\
& \widetilde{\gamma} = uA - u_0\pi && (3.20)\\
& \widetilde{\gamma} = vA + v_0\pi && (3.21)\\
& \widetilde{\gamma}_0 = ub - u_0\pi_0 && (3.22)\\
& \widetilde{\gamma}_0 = vb + v_0(\pi_0 + 1) && (3.23)\\
& u, v, u_0, v_0 \geq 0. && (3.24)
\end{aligned}
$$

Let us assume the above CDLP to be bounded and consider an optimal solution (u^*, v^*, u_0^*, v_0^*) of value z^*, yielding the same cut $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ as $(\widetilde{u}, \widetilde{v}, \widetilde{u}_0, \widetilde{v}_0)$. For any $\alpha \in [0, 1]$, the convex combination of (u^*, v^*, u_0^*, v_0^*) and $(\widetilde{u}, \widetilde{v}, \widetilde{u}_0, \widetilde{v}_0)$ computed as

$$
\begin{array}{llll}
\hat{u} = \alpha\widetilde{u} + (1-\alpha)u^*, & \hat{v} = \alpha\widetilde{v} + (1-\alpha)v^*, \\
\hat{u}_0 = \alpha\widetilde{u} + (1-\alpha)u_0^*, & \hat{v}_0 = \alpha\widetilde{v}_0 + (1-\alpha)v_0^*,
\end{array} \qquad (3.25)
$$

still yields cut $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$. However, in case $z^* > 0$ we have $S(\hat{u}) \cap S(\hat{v}) \neq \emptyset$, i.e., we have obtained the same cut from two sets of nondisjoint multipliers. Hence, a valid disjunctive cut $\overline{\gamma}x \geq \overline{\gamma}_0$ which dominates $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ w.r.t. P can be computed through Theorem 3.2 as:

$$
\begin{array}{llllll}
\overline{\mu} = \min\{\hat{u}, \hat{v}\}, & \overline{u} = \hat{u} - \overline{\mu}, & \overline{v} = \hat{v} - \overline{\mu}, \\
\overline{u}_0 = \hat{u}_0, & \overline{v}_0 = \hat{v}_0, \\
\overline{\gamma} = \overline{u}A - \overline{u}_0\pi, & \overline{\gamma} = \overline{v}A + \overline{v}_0\pi, \\
\overline{\gamma}_0 = \overline{u}b - \overline{u}_0\pi_0, & \overline{\gamma}_0, = \overline{v}b + \overline{v}_0(\pi_0 + 1).
\end{array} \qquad (3.26)
$$

Obviously, the dominance might be not strict if $conv(P_0 \cup P_1)$ is not full dimensional.

3.3.3 Empirical Analysis

In order to understand how much we can improve on the disjunctive cuts obtained by solving the CGLP with SNC, we performed the following experiment. For any violated cut $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ separated by solving the CGLP with SNC normalization (3.12), we try to strengthen it by solving the corresponding CDLP (3.19)–(3.24). If $z^* > 0$, then we compute the new dominating cut $\overline{\gamma}x \geq \overline{\gamma}_0$ by using (3.25), with $\alpha = 1/2$, and (3.26), and we replace the original cut $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$ by the dominating one. Otherwise, if either $z^* = 0$ or CDLP turns out to be unbounded, we keep the original cut $\widetilde{\gamma}x \geq \widetilde{\gamma}_0$.

The computational results reported in Tables 3.3 and 3.4 compare the cuts obtained by using the SNC normalization with those strengthened through the additional solution of the CDLP. In order to get a better understanding of the impact of the strengthening, we first solved both CGLP and CDLP *without* the projection onto the support of x^* (Table 3.3). Indeed, solving the problem on the complete model guarantees that the strengthened cut dominates the original one, domination being guaranteed to be strict in the full dimensional case. Since the computing time in the complete variable space is not negligible, we also solved the problem in the projected space (Table 3.4). However, a domination on the support might not correspond to a dominated cut once the cut is lifted outside the support. In fact, a cut which is stronger in the support might turn out to be weaker overall. Hence, to limit such a phenomenon, in the experiments in Table 3.4 we strengthen the cut by the Balas-Jeroslow procedure [23] before defining and solving the CDLP. Of course, both the original and the dominating cuts are also strengthened afterwards.

Table 3.3: SNC Normalization vs. SNC Normalization + CDLP. No projection (and no Balas-Jeroslow strengthening).

	SNC Normalization			SNC Normalization + CDLP				
Instance	# cuts	%gap	sep. time	# cuts	# dom	# unb	%gap	sep. time
bell3a	71	70.74	0.1	71	19	14	70.74	0.4
bell5	188	94.12	0.5	186	11	125	94.31	1.2
blend2	197	30.49	4.7	210	54	2	32.72	12.0
flugpl	93	18.34	0.1	91	26	0	18.36	0.1
gt2	218	94.13	1.2	200	141	1	94.49	2.8
lseu	171	42.46	0.5	191	68	4	42.51	0.7
*markshare1	77	0.00	0.1	75	0	75	0.00	0.2
mod008	107	15.46	3.9	112	27	0	15.84	6.0
p0033	116	57.25	0.1	106	55	8	57.30	0.2
p0201	692	92.53	46.4	750	38	622	98.97	70.3
rout	349	29.46	80.5	351	159	142	30.93	118.1
*stein27	251	0.00	0.6	248	21	0	0.00	1.3
vpm1	267	50.62	2.3	275	8	115	59.91	5.9
vpm2	390	74.73	7.5	397	84	130	75.71	14.5
avg.	238.250	55.861	12.317	245.000			57.649	19.350

Tables 3.3 and 3.4 report the number of separated cuts, the percentage gap closed (within ten rounds) and the computing time spent on separation. In addition, for the strengthened version of the cuts we also report how many times the CDLP returns $z^* > 0$ (column '# dom') and the number of times in which CDLP was instead unbounded (column '# unb'). Separation time of the strengthened version includes CGLP solution time.

Both Tables 3.3 and 3.4 show that the CDLP is indeed effective to change the disjunctive cuts obtained using the SNC normalization. In general, the procedure is computationally rather cheap and allows an improvement in the %gap closed which is sometimes non-negligible.

Of course, the same CDLP can be constructed and solved to strengthen a disjunctive cut obtained by using any normalization, e.g., the trivial one (3.11). Indeed, we also

Table 3.4: SNC Normalization vs. SNC Normalization + CDLP. CGLP and CDLP solved projected onto the x^* support. Balas-Jeroslow strengthening applied before and after CDLP.

	SNC Normalization			SNC Normalization + CDLP				
Instance	# cuts	%gap	sep. time	# cuts	# dom	# unb	%gap	sep. time
bell3a	71	70.74	0.1	69	11	23	70.74	0.2
bell5	172	96.16	0.3	179	38	63	96.16	0.7
blend2	215	33.45	0.7	225	77	30	33.66	5.1
flugpl	92	18.36	0.1	90	29	0	18.59	0.1
gt2	151	96.19	0.2	157	50	55	96.19	0.7
lseu	179	81.09	0.2	171	9	135	86.04	0.4
*markshare1	80	0.00	0.0	80	3	46	0.00	0.1
mod008	100	31.46	0.1	98	46	9	36.33	0.2
p0033	104	70.98	0.1	113	14	71	75.85	0.2
p0201	669	100.00	10.9	674	1	663	100.00	17.0
rout	603	47.91	38.0	613	6	600	49.50	55.7
*stein27	251	0.00	0.5	252	14	0	0.00	1.1
vpm1	298	57.88	1.2	255	25	23	58.97	2.1
vpm2	400	75.11	4.0	401	133	29	75.76	6.0
avg.	254.500	64.944	4.658	253.75			66.483	7.367

tested it to strengthen classical GMIs. The results are reported in Table 3.5.

Table 3.5: GMI vs. GMI + CDLP. CDLP solved projected onto the x^* support. Balas-Jeroslow strengthening applied before and after CDLP.

	GMI			GMI + CDLP				
Instance	# cuts	%gap	sep. time	# cuts	# dom	# unb	%gap	sep. time
bell3a	128	70.74	0.1	124	2	103	70.74	0.4
bell5	208	42.23	0.0	200	49	116	67.31	1.0
blend2	125	25.44	0.1	147	56	45	26.26	2.6
flugpl	93	15.15	0.0	91	69	0	15.44	0.1
gt2	146	99.12	0.1	180	35	88	99.99	0.3
lseu	134	53.67	0.1	134	24	106	53.71	0.2
*markshare1	69	0.00	0.0	69	0	33	0.00	0.0
mod008	77	29.86	0.1	77	0	10	29.86	0.1
p0033	114	71.37	0.0	120	8	103	71.37	0.1
p0201	490	62.29	0.7	490	0	490	62.29	9.0
rout	424	7.60	1.4	424	0	424	7.60	19.6
*stein27	238	0.00	0.1	245	220	0	0.00	0.8
vpm1	259	36.69	0.1	280	116	100	43.35	2.3
vpm2	389	43.40	0.3	402	74	252	45.22	18.3
avg.	215.583	46.463	0.250	222.417			49.428	4.500

The table has the same structure as Table 3.4 and compares the GMI cuts obtained from the tableau (i.e., obtained by considering a strengthened non-elementary disjunction; see Chapter 2, Section 2.4) with those obtained through the additional solution of the CDLP in the projected space. In particular, the CDLP is solved by considering an elementary disjunction (note that, the non-elementary disjunction associated with GMI is indeed elementary in the x^* support) and the cuts are afterwards strengthened through the Balas-Jeroslow procedure [23].

Overall, if the "starting point" is the GMI cut, CDLP is not competitive with the optimal solution of CGLP with SNC normalization, 49.428% average gap w.r.t. 64.944%. However, it can be viewed as a general tool to improve on the optimal solution of CGLP, whenever such a solution corresponds to a weak ray/vertex, according to Definition 3.1.

3.4 Redundancy hurts

Loosely speaking, a *redundant* constraint for a polyhedron is a constraint whose removal does not enlarge the polyhedron itself. By Farkas lemma, a constraint $a_i x \geq b_i$ is said to be *redundant* for $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ if there exist $\delta \geq 0$ and $\lambda_I \in \mathbb{R}^{m-1}_+$ such that $a_i = \lambda_I A_I$ and $\lambda_I b_I = b_i + \delta$, where A_I (resp., b_I) denotes the submatrix of A (resp., subvector of b) induced by the row index set $I = \{1, \ldots, m\} \setminus \{i\}$. Redundancy is *strict* if $\delta > 0$.

In the attempt to find a way to get rid of the "weak vertices" in the CGLP, we looked for more combinatorial properties. A more careful analysis of Example 3.1 reveals a more general property that allows one to classify as "bad" certain constraints. Indeed, consider the role of constraint (a1) with respect to the left-branch polytope P_0. This constraint is clearly redundant for P_0 (note that this is not the case if the original P is considered). However, if constraint (a1) participates with a positive multiplier to the definition of the disjunctive cut whereas constraint (a3) does not (i.e., if $u_1 > 0$ and $u_3 = 0$), then the cut itself has to be valid for the point $x_1 = 0$, $x_2 = 1/2$ and cannot be "pushed" any further inside P_0. This is precisely what happens for the weak cuts (c3) and (c1), that cannot be supporting for P_0 precisely because of the bad choice $u_1 > 0$.

The role of redundancy is formally stated as follows.

Proposition 3.1 *If a constraint that is strictly redundant for P_0 (resp. P_1) is used in the cut derivation with a nonzero multiplier, then the resulting disjunctive cut is nonsupporting in P_0 (resp. P_1).*

Proof. Let $(\widetilde{\gamma}, \widetilde{\gamma}_0, \widetilde{u}, \widetilde{u}_0, \widetilde{v}, \widetilde{v}_0)$ be a feasible solution of the CGLP, with $\widetilde{u}_i > 0$, and assume that constraint i is strictly redundant for P_0. Then there exists $(\lambda, \lambda_0, \delta) \in \mathbb{R}^{m+1}_+$, with $\delta > 0$ such that $a_i = \lambda_I A_I - \lambda_0 \pi$ and $b_i = \lambda_I b_I - \lambda_0 \pi_0 - \delta$. By using equations (3.5) and (3.7), we get

$$\begin{array}{rcrcrcrcrcrcl} \widetilde{\gamma} & = & \widetilde{u}_I A_I & + & \widetilde{u}_i a_i & - & \widetilde{u}_0 \pi & = & (\widetilde{u}_I + \widetilde{u}_i \lambda_I) A_I & - & (\widetilde{u}_0 + \widetilde{u}_i \lambda_0) \pi & & \\ \widetilde{\gamma}_0 & = & \widetilde{u}_I b_I & + & \widetilde{u}_i b_i & - & \widetilde{u}_0 \pi_0 & = & (\widetilde{u}_I + \widetilde{u}_i \lambda_I) b_I & - & (\widetilde{u}_0 + \widetilde{u}_i \lambda_0) \pi_0 & - & \widetilde{u}_i \delta \end{array}$$

Thus, for each $x \in P_0$ we have $\widetilde{\gamma} x - \widetilde{\gamma}_0 = (\widetilde{u}_I + \widetilde{u}_i \lambda_I)(A_I x - b_I) - (\widetilde{u}_0 + \widetilde{u}_i \lambda_0)(\pi x - \pi_0) + \widetilde{u}_i \delta \geq \widetilde{u}_i \delta > 0$, and this shows that cut $\widetilde{\gamma} x \geq \widetilde{\gamma}_0$ is nonsupporting in P_0. In the same way it can be shown that if $v_h > 0$ for a constraint h strictly redundant for P_1, then the cut $\widetilde{\gamma} x \geq \widetilde{\gamma}_0$ does not support P_1. □

By definition, a redundant constraint for P_0 or P_1 can be obtained as a conic combination of other constraints. If the sum of the multipliers in the conic combination is greater than 1, then using a redundant constraint is cheaper (with respect to normalization (3.12)) than using the constraints that generate it, hence a redundant constraint can in fact be preferred by the CGLP. This is formally proved by the following theorem dealing with redundancy for P_0 (the case dealing with P_1 being perfectly analogous).

Theorem 3.3 *Assume that constraint $a_i x \geq b_i$ is redundant for P_0 as conic combination of $A_I x \geq b_I, -\pi x \geq -\pi_0$ with multipliers $(\lambda_I, \lambda_0) \in \mathbb{R}^m_+$, and let $(\overline{\gamma}, \overline{\gamma}_0, \overline{u}, \overline{u}_0, \overline{v}, \overline{v}_0)$ be a feasible solution of the CGLP with normalization (3.12), such that $\overline{\gamma} x^* < \overline{\gamma}_0$ and $\overline{u}_i > 0$. Then there exist $\theta > 0$ and a feasible solution $(\widetilde{\gamma}, \widetilde{\gamma}_0, \widetilde{u}, \widetilde{u}_0, \widetilde{v}, \widetilde{v}_0)$ of the CGLP with normalization (3.12) such that $\widetilde{u}_i = 0$, $\widetilde{\gamma} := \overline{\gamma}/\theta$, $\widetilde{\gamma}_0 = \overline{\gamma}_0/\theta$, $\overline{\gamma} x^* - \overline{\gamma}_0 = \theta(\widetilde{\gamma} x^* - \widetilde{\gamma}_0)$, and $\theta > 1$ if and only if $1\lambda_I + \lambda_0 > 1$.*

Proof. Since $(\overline{\gamma}, \overline{\gamma}_0, \overline{u}, \overline{u}_0, \overline{v}, \overline{v}_0)$ is feasible for the CGLP with normalization (3.12), writing $a_i x \geq b_i$ in terms of the multipliers (λ_I, λ_0) one gets

$$\begin{array}{rclcrclcl} \overline{\gamma} & = & (\overline{u}_I + \overline{u}_i\lambda_I)A_I & - & (\overline{u}_0 + \overline{u}_i\lambda_0)\pi & = & \overline{v}A & + & \overline{v}_0\pi \\ \overline{\gamma}_0 & = & (\overline{u}_I + \overline{u}_i\lambda_I)b_I & - & (\overline{u}_0 + \overline{u}_i\lambda_0)\pi_0 & = & \overline{v}b & + & \overline{v}_0(\pi_0 + 1) \end{array}$$

while from the normalization condition $1\overline{u} + 1\overline{v} + \overline{u}_0 + \overline{v}_0 = 1$ one obtains

$$\theta := 1(\overline{u}_I + \overline{u}_i\lambda_I) + (\overline{u}_0 + \overline{u}_i\lambda_0) + 1\overline{v} + \overline{v}_0,$$

which is then simplified as

$$\theta = 1 + \overline{u}_i(1\lambda_I + \lambda_0 - 1).$$

Since cut $\overline{\gamma} x \geq \overline{\gamma}_0$ is violated, one must have $\overline{u}_0 + \overline{v}_0 > 0$, hence $\theta > 0$ holds. Therefore, one can define the nonnegative quantities $\widetilde{u}_I := (\overline{u}_I + \overline{u}_i\lambda_I)/\theta$, $\widetilde{u}_i = 0$, $\widetilde{u}_0 := (\overline{u}_0 + \overline{u}_i\lambda_0)/\theta$, $\widetilde{v} = \overline{v}/\theta$, $\widetilde{v}_0 = \overline{v}_0/\theta$, $\widetilde{\gamma} := \overline{\gamma}/\theta$, $\widetilde{\gamma}_0 = \overline{\gamma}_0/\theta$, thus getting a feasible solution of the CGLP that satisfies normalization (3.12) and such that $\overline{\gamma} x^* - \overline{\gamma}_0 = \theta(\widetilde{\gamma} x^* - \widetilde{\gamma}_0)$. Moreover, being $\overline{u}_i > 0$, one has $\theta > 1$ if and only if $1\lambda + \lambda_0 > 1$. □

The above theorem shows that redundant constraints do not introduce new cuts, but just scaled copies of already-existing cuts that may have a better objective function (violation). Loosely speaking, redundant constraints can "trick" normalization (3.12), in the sense that they can create vertices of the CGLP corresponding to scaled copies of cuts that are strictly dominated but more attractive (i.e., with a better objective function value) than the dominating ones.

A very natural way to cope with redundancy is to just eliminate the redundant constraints from the CGLP, or equivalently to fix their Farkas multipliers to zero. In Example 3.1, the CGLP without redundant constraints has only 9 extreme rays and 9 vertices (instead of 117), and only one of them corresponds to a violated constraint – namely, the facet-defining cut (c2).

At a first glance, this example seems to suggest that only *strictly redundant* (i.e., nonsupporting) constraints should be avoided in the cut generation. However, redundant constraints should be avoided even in case they are supporting, as shown by the example reported below.

Example 3.3 *For the simple ILP*

$$\begin{array}{lrrrcr} \min & -x_1 & -2x_2 & +10x_3 & & \\ (a1) & 4x_1 & -4x_2 & & \geq & -2 \\ (a2) & -2x_1 & -2x_2 & -x_3 & \geq & -3 \\ (a3) & 3x_1 & -2x_2 & -x_3 & \geq & -1 \\ (a4) & x_1 & & & \geq & 0 \\ (a5) & & x_2 & & \geq & 0 \\ (a6) & & & x_3 & \geq & 0 \end{array}$$

only two cuts violated by the optimal solution of the LP relaxation $x^* = (\frac{1}{2}, 1, 0)$ *can be derived from disjunction* $x_1 \leq 0$ *OR* $x_1 \geq 1$*, namely:*

(c1) $2x_2 \leq 1$*, corresponding to the basic solution of the CGLP* $(u_1, v_2, v_6, u_0, v_0)$*, of value* $z_1 = -\frac{2}{13}$ *(optimal);*

(c2) $2x_2 + x_3 \leq 1$*, corresponding to the basic solution of the CGLP* $(u_3, v_2, v_6, u_0, v_0)$*, of value* $z_2 = -\frac{1}{7}$ *(nonoptimal).*

$Conv(P_0 \cup P_1)$ *has 6 vertices, namely* $V_1 = (0, 0, 0)$*,* $V_2 = (0, \frac{1}{2}, 0)$*,* $V_3 = (0, 0, 1)$*,* $V_4 = (\frac{3}{2}, 0, 0)$*,* $V_5 = (1, \frac{1}{2}, 0)$*, and* $V_6 = (1, 0, 1)$*. In the reverse polar space* (γ, γ_0)*, the projected CGLP cone has only 5 extreme rays that correspond to the facets of* $conv(P_0 \cup P_1)$*. In space* $(\gamma, \gamma_0, u, v, u_0, v_0)$*, instead, the CGLP cone has 33 extreme rays that correspond to 33 vertices once normalization (3.12) is applied. In the optimal basis, constraint (a1) (which is redundant but supporting for* P_0*) is used with* $u_1 > 0$*, and the corresponding cut (c1) supports both* P_0 *and* P_1 *in* V_2 *and* V_5*, respectively, but it is not facet-defining. The CGLP without redundant constraints (in particular without (a1)) has only 10 extreme rays and 10 vertices (instead of 33), and only one of them corresponds to a violated constraint—namely, the facet-defining cut (c2). Note that cut (c2) dominates cut (c1) and is facet-defining since it supports* $conv(P_0 \cup P_1)$ *in the 3 affinely independent vertices* V_2, V_3*, and* V_5*. For illustration purposes, 3 extreme rays* r_1–r_3 *and a nonextremal direction* α *of the CGLP cone are reported below.*

	γ_1	γ_2	γ_3	γ_0	u_1	u_2	u_3	u_4	u_5	u_6	v_1	v_2	v_3	v_4	v_5	v_6	u_0	v_0
(r_1)	0	−2	−1	−1	0	0	1	0	0	0	0	1	0	0	0	0	3	2
(r_2)	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
(α)	0	−2	0	−1	0	0	1	0	0	1	0	1	0	0	0	1	3	2
(r_3)	0	−2	0	−1	1/2	0	0	0	0	0	0	1	0	0	0	1	2	2

The weak cut (c1) is strictly dominated w.r.t. P by (c2), as the vector α *is just the sum of the extreme rays* r_1 *and* r_2*, the latter corresponding to the original constraint* $x_3 \geq 0$*. However, the redundant constraint (a1) creates an extremal copy of the weak cut – the extreme ray* r_3 *– which turns out to be the optimal vertex once normalization (3.12) is applied.* □

The previous discussion shows that extreme rays of the CGLP cone in the extended space $(\gamma, \gamma_0, u, v, u_0, v_0)$ may be nonextremal when projected onto the (γ, γ_0) space, and redundant constraints can add to the cone several extreme-rays corresponding to very weak cuts. The SNC normalization (3.12) simply maps extreme rays to vertices, and

creates a possibly "wrong" ranking among the vertices. Unfortunately, as far as we know no normalization equation is able to truncate the CGLP cone so as to guarantee that an optimal CGLP vertex in the extended space remains a vertex when projected in the (γ, γ_0) space. For instance, consider normalizations of the form

$$\gamma(q - x^*) = 1, \tag{3.27}$$

which have been deeply investigated in Bonami [37]. Balas and Perregaard [24] proved that, if $q \in conv(P_0 \cup P_1)$, then the CGLP truncated with (3.27) has a finite optimum and that there *exists* an optimal vertex of the resulting polyhedron in the extended space whose projection in the natural reverse polar space (γ, γ_0) remains extremal. However, this does not imply that any optimal vertex in the extended space is a vertex in the projected space – hence even normalization (3.27) could not help in finding a facet-defining cut.

Example 3.4 *(Example 3.3 continued)*
Consider again the simple ILP discussed in Example 3.3. If the corresponding CGLP cone is truncated with normalization (3.27), with $q = (0, 0, 0)$, the resulting polyhedron has 60 extreme rays and 20 vertices. As before, only two vertices correspond to violated cuts, namely:

i) the basic solution $(u_1, v_2, v_6, u_0, v_0)$, of value $z_1 = -\frac{1}{2}$ (optimal), corresponding to the weak cut (c1);

ii) the basic solution $(u_3, v_2, v_6, u_0, v_0)$, of value $z_2 = -\frac{1}{2}$ (optimal), corresponding to the facet-defining cut (c2).

So, even in this case, the separation procedure could select the weak cut (c1), since the choice of q makes (c1) and (c2) completely equivalent in terms of objective function. □

3.4.1 Empirical Analysis

In our fourth set of experiments we eliminated redundant constraints in a trivial way (i.e., by solving LPs) before solving the CGLP. To get a clearer picture, we did not project the separation problem onto the support of x^* since such a projection makes the definition of what is redundant and what is not less clear.

The results are reported in Table 3.6 and show that removing redundant constraints is indeed very useful. Besides an average improvement in the percentage gap closed of around 2.5%, only for two problems, namely `bell5` and `gt2`, the "Classical" SNC is slightly better than the "No redundancy" SNC version, while for some single problems the improvement is quite substantial, up to 13% for instance `p0033`. Concerning the average cardinality of the dual support of the cut there is a slight increase in the "No redundancy" version which however does not seem relevant.

3.4.2 Working on the Support

Projecting the separation problem onto the support of x^* has of course the advantage of dealing with a problem of smaller size. However, according to our experience the

Table 3.6: "Classical" SNC vs. "No redundancy" SNC with no projection.

	"Classical" SNC			"No redundancy" SNC		
Instance	# cuts	%gap	$\|S\|$	# cuts	%gap	$\|S\|$
bell3a	71	70.74	64.65	54	70.74	66.19
bell5	188	94.12	16.83	189	93.54	15.80
blend2	197	30.49	71.42	212	30.63	119.90
flugpl	93	18.34	6.45	90	18.83	6.48
gt2	218	94.13	58.11	167	93.68	63.16
lseu	171	42.46	23.86	184	45.10	30.96
*markshare1	77	0.00	55.99	77	0.00	56.00
mod008	107	15.46	304.18	107	15.48	304.19
p0033	116	57.25	8.75	126	70.32	10.99
p0201	692	92.53	23.40	757	98.31	37.44
rout	349	29.46	189.07	384	31.93	202.18
*stein27	251	0.00	7.29	249	0.00	6.46
vpm1	267	50.62	11.13	282	54.55	11.10
vpm2	390	74.73	24.23	376	76.47	22.82
avg.	238.250	55.861	66.840	244.000	58.298	74.267

projection can enlarge the set of redundant constraints in a way that decreases the positive effects associated with their removal. A possible explanation of this behavior is that projection may hide the redundancy of some bound constraints, hence weakening the final disjunctive cut. Indeed, consider a variable x_k restricted to being nonnegative and such that $x^*_k = 0$. If x_k is projected away with the aim of computing coefficient γ_k afterwards through (3.10), then we lose any control on the Farkas variables associated with the constraint $x_k \geq 0$, say $u_{i(k)}$ and $v_{i(k)}$. In fact, if it happens that constraint $x_k \geq 0$ is redundant, it is very useful to keep explicitly constraints $\gamma_k = uA_k - u_0\pi_k = vA_k + v_0\pi_k$ in the CGLP and to impose the additional requirement $u_{i(k)} = 0$ and/or $v_{i(k)} = 0$.

As the above property seems to be crucial for the variable bounds, we defined an *extended support* of x^* by avoiding projecting away any variable whose bound condition is (tight in x^* and) redundant. The results when using the extended support are reported in Table 3.7, where $\%supp$ indicates the average percentage of the x variables which are kept in the (extended) support.

The first part of Table 3.7 reports the same figures as Table 3.1, plus a column which gives the percentage size (w.r.t. the nominal size) of the CGLP projected on the support. By comparing the first and second part of the table, we note that the gain shown by Table 3.6 due to the redundancy removal is lost here (the average gap closed of 56.873% deteriorates to 54.269%), thus confirming our intuition about the smaller precision of the redundancy test in such a case. However, the situation is totally recovered using the extended support as defined above. Indeed, the percentage value 56.873 improves to 58.793, and the average size of the support does not increase much (from 50.268% to 53.529%). The only large increase in the size of the CGLP arises for problem `rout`, which is in fact a very instructive case: the "Classical" SNC

Table 3.7: "Classical" SNC vs. "No redundancy" SNC with cuts separated projected on the support.

	"Classical" SNC				"No redundancy" support				"No redundancy" ext. support			
Instance	# cuts	%gap	%supp	$\|S\|$	# cuts	%gap	%supp	$\|S\|$	# cuts	%gap	%supp	$\|S\|$
bell3a	71	70.74	69.25	43.72	88	70.74	69.32	44.82	54	70.74	65.61	44.60
bell5	178	94.29	72.69	11.75	207	94.62	72.88	13.32	180	94.29	71.64	11.99
blend2	192	30.51	53.06	8.10	200	30.99	53.54	10.84	193	30.53	53.99	8.34
flugpl	92	18.36	86.11	5.85	93	18.94	86.11	5.89	93	18.86	86.29	5.95
gt2	196	93.46	18.30	10.28	191	94.13	18.14	10.58	187	93.88	20.00	13.10
lseu	196	41.33	29.44	9.17	191	40.16	27.08	12.28	178	43.45	29.41	9.08
*markshare1	74	0.00	11.94	1.39	130	0.00	13.39	2.56	77	0.00	12.59	1.69
mod008	139	17.05	4.51	12.41	136	17.70	4.42	12.17	157	19.13	5.85	14.43
p0033	113	67.86	55.76	4.81	106	70.32	55.76	5.74	146	70.29	58.84	5.89
p0201	767	93.82	45.02	13.43	873	81.59	43.43	25.83	769	100.00	48.93	13.39
rout	434	24.26	42.19	68.07	355	6.56	38.11	58.23	353	30.88	69.46	140.29
*stein27	252	0.00	93.70	6.53	252	0.00	93.70	6.68	251	0.00	93.61	7.13
vpm1	263	55.84	62.14	5.39	275	50.18	62.25	6.30	259	57.63	65.18	6.60
vpm2	403	74.96	64.74	17.27	377	75.30	65.08	18.10	373	75.84	67.15	17.71
avg.	253.667	56.873	50.268	17.521	257.667	54.269	49.677	18.675	245.167	58.793	53.529	24.281

closed 24.26% of the gap, the "No redundancy" SNC version on the support closes only 6.56%, while in the extended support the situation is totally recovered (and improved) with 30.88% gap closed. For such a particular instance the extended support size is also substantially different from the support size, namely 69.46% with respect to 42.19%. Our interpretation is the following: in order to forbid the use of some variable bounds in the derivation of the cut we have to enlarge the support considerably (half of the projected variables are re-inserted) with the overall effect of generating much stronger cuts.

3.4.3 A practical perspective

Of course, a more efficient approach to check redundancy would be to do it "on the fly", by only considering those constraints that have a nonzero Farkas multiplier in the optimal CGLP solution. This approach would require to extend the support also "on the fly", i.e., any time a bound constraint is discovered to be redundant, its corresponding variable must be added to the support so as to be able to fix to zero the Farkas multiplier associated with the constraint, as explained in the previous section.

It is possible that working on the support together with such an on-the-fly redundancy check might become practical in terms of computational effort. On the other hand, another way to decrease the computational effort on the redundancy check is to use heuristics. Both ideas are currently under investigation.

3.5 An effective (and fast) normalization for the set covering

As shown in the previous sections, the standard normalization has the main advantage of generating low-rank inequalities, which is in general a desirable property. As a matter of fact, it has been recently showed that rank-1 inequalities on general disjunctions are able to close a large portion of the integrality gap (see, e.g., Fischetti and Lodi [83], Balas and Saxena [26], Dash, Günlük and Lodi [66, 67]). When normalization (3.12) is applied,

the norm of the separated cuts tends to be smaller with respect to the constraints used for their generation, and small-norm constraints are implicitly penalized by the normalization itself. Thus, high-rank constraints are selected in the cut derivation only if needed, hence generating weak cuts does not hurt the overall separation procedure in that these cuts are less likely to be used in the next iterations. However, as stated in Section 3.4, the standard normalization creates a ranking among the CGLP vertices which depends on the scaling of the constraints, i.e., the overall separation procedure is heavily affected by the scaling of the constraints in the original formulation. To overcome the latter drawback, one can replace the standard normalization with the following *Euclidean Normalization* (EN):

$$\sum_{i=1}^{m} \|a_i\| u_i + \sum_{i=1}^{m} \|a_i\| v_i + \|\pi\| u_0 + \|\pi\| v_0 = 1, \tag{3.28}$$

where $\|t\|$ denotes the Euclidean norm of vector t.

Lemma 3.3 *Let $\widetilde{A}x \geq \widetilde{b}$ be a scaled copy of system $Ax \geq b$ where, for all $i \in \{1, \ldots, m\}$, $\widetilde{a}_i := a_i/K_i$ and $\widetilde{b}_i := b_i/K_i$, with $K_i > 0$. For any solution of the CGLP with normalization (3.28) corresponding to a cut $\gamma x \geq \gamma_0$, there exists a solution of the CGLP associated with the system $\widetilde{A}x \geq \widetilde{b}$, still with normalization (3.28), corresponding to the same cut.*

Proof. Let $(\gamma, \gamma_0, u, v, u_0, v_0)$ be a solution of the CGLP with normalization (3.28), and for all $i \in \{1, \ldots, m\}$ define $\widetilde{u}_i = K_i u_i$ and $\widetilde{v}_i = K_i v_i$. From equations (3.5) we obtain

$$\gamma = uA - u_0\pi = \sum_{i=1}^{m} u_i a_i - u_0\pi = \sum_{i=1}^{m} (K_i u_i)(a_i/K_i) - u_0\pi = \widetilde{u}\widetilde{A} - u_0\pi.$$

Analogously, from (3.6) we get $\gamma = \widetilde{v}\widetilde{A} + v_0\pi$ and from (3.7)–(3.8) we have $\gamma_0 = \widetilde{u}\widetilde{b} - u_0\pi_0 = \widetilde{v}\widetilde{b} + v_0(\pi_0 + 1)$. Hence $(\gamma, \gamma_0, \widetilde{u}, \widetilde{v}, u_0, v_0)$ is a feasible solution of the CGLP associated with system $\widetilde{A}x \geq \widetilde{b}$ yielding the same cut as $(\gamma, \gamma_0, u, v, u_0, v_0)$. Since $\|\widetilde{a}_i\|\widetilde{u}_i + \|\widetilde{a}_i\|\widetilde{v}_i = \|a_i\|u_i + \|a_i\|v_i \ \forall\ i \in \{1, \ldots, m\}$, then $(\gamma, \gamma_0, \widetilde{u}, \widetilde{v}, u_0, v_0)$ fulfills normalization (3.28) as well. □

The above lemma shows that the CGLP with Euclidean normalization is not affected by scaling issues. Moreover, the CGLP with (3.28) is the same as the CGLP with the standard normalization for a system $\widetilde{A}x \geq \widetilde{b}$ where all the constraints have been scaled in order to have Euclidean norm equal to 1 (i.e., $\|\widetilde{a}_i\| = 1 \ \forall\ i \in \{1, \ldots, m\}$). By replacing normalization (3.12) with (3.28) we are losing the implicit penalization of high-rank inequalities hidden in the standard normalization. However, the Euclidean normalization associates penalties with the Farkas multipliers which are in some way related to the structure of the corresponding constraints instead of being all equal to 1 without any distinction.

Unfortunately, normalization (3.28) is not likely to work well in all cases. For example, consider a constraint $a_i x \geq b_i$ having both positive and negative coefficients, say

$a_{ij} > 0$ and $a_{ik} < 0$, and assume the nonnegative variables x_j and x_k are nonbasic and hence cut coefficients γ_j and γ_k do not affect violation. In this case, the effect of the Farkas multipliers u_i and v_i on the strength of the cut coefficients γ_j and γ_k is not univocal—a large value of the Farkas multipliers u_i or v_i would lead to a weak cut coefficient γ_j but to a strong cut coefficient γ_k. Hence, associating a penalty $||a_i||$ with constraint a_i in the normalization might not be the right choice.

On the other hand, for constraints with nonnegative coefficients only, the Euclidean norm of a constraint is likely to give a reliable measure on how bad is to use its associated Farkas multipliers u_i and v_i. In particular, the Euclidean Normalization seems to be particularly suited for Set Covering problems. Indeed, set covering constraints have nonnegative coefficients only, and this property is known to be inherited by nontrivial valid inequalities, including the disjunctive cuts we can separate through our procedure. This implies that weighing a constraint by means of its norm has a more direct impact on the cut density, and gives a sensible indication for nonbasic variables.

Finally, note that it is not necessary to solve the CGLP in the lifted space to use normalization EN. This has been recently shown by Balas and Bonami [19] who implemented our normalization on the original tableau by adapting the approach of Balas and Perregaard [25], thus proving that EN might be used within a fast implementation.

We performed a set of computational experiments on a test-bed of Set Covering instances taken from the OR–Library [35], and the results are reported in Table 3.8. The improvement in the percentage gap closed by EN w.r.t. SNC is quite substantial

Table 3.8: SNC normalization vs. Euclidean normalization on SCP instances.

	SNC normalization			Euclidean normalization		
Instance	# cuts	%gap	$\|S\|$	# cuts	%gap	$\|S\|$
scpnre1	904	13.67	89.22	951	17.35	93.62
scpnre2	963	9.38	95.42	997	12.51	98.14
scpnre3	923	15.14	91.41	944	18.13	92.82
scpnre4	878	13.25	85.99	897	15.70	87.82
scpnre5	889	16.84	87.77	935	21.03	91.16
scpnrf1	678	10.23	67.75	682	12.62	67.77
scpnrf2	655	9.62	65.42	689	12.90	68.50
scpnrf3	586	12.08	58.34	617	15.58	60.93
scpnrf4	664	10.21	66.35	692	12.59	68.91
scpnrf5	661	8.63	66.05	700	11.85	69.70
avg.	780.100	11.905	77.372	810.400	15.026	79.937

as it ranged from 2.38% to 4.19%, with an average of 3.12%.

Figures 3.12–3.14 describe the behavior of the two normalizations on the particular instance `scpnre5`. One can observe that the considerably higher rank of the cuts generated using the Euclidean normalization with respect to those obtained through SNC (see Figure 3.14) does not correspond at all to denser cuts. Indeed, as shown in Figure 3.12, the former cuts are consistently sparser than the latter. (Cuts generated using SNC are fully dense: the number of variables of the instance is 5,000 and the number

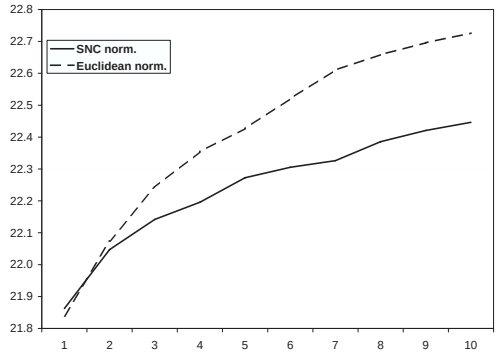

Figure 3.11: SNC vs. Euclidean normalization: dual bound for instance *scpnre*5.

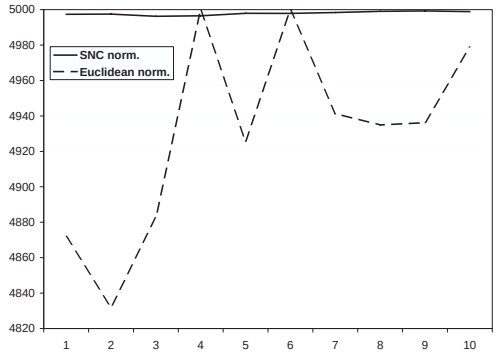

Figure 3.12: SNC vs. Euclidean normalization: avg. cut density for instance *scpnre*5.

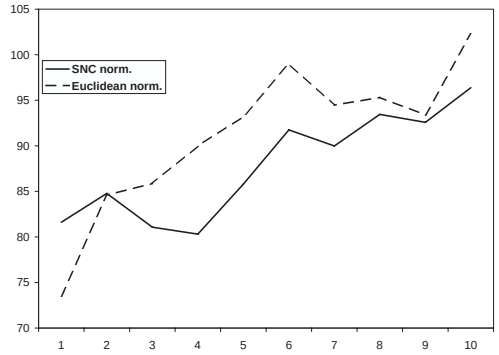

Figure 3.13: SNC vs. Euclidean normalization: avg. cardinality of $S(u, v)$ for instance *scpnre*5.

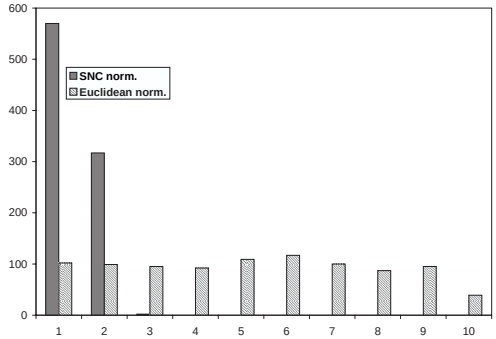

Figure 3.14: SNC vs. Euclidean normalization: cut rank for instance *scpnre*5.

of nonzero coefficients is almost always very close to 5,000 too.)

A natural question concerns the effectiveness of the Euclidean Normalization (3.28) on the MIPLIB instances [36] used in the previous sections. The answer is empirically given by the computational results in Table 3.9 which compares the standard and the Euclidean normalization on the MIPLIB instances. As expected, the table shows that the Euclidean normalization does not provide a relevant improvement on general MIP problems. In particular, for instance `rout` the gap closed decreases from 24.26% to 7.92% (note, in any case, that without such an instance the Euclidean normalization would exhibit an average improvement of 1.7%). However, most of the negative results are recovered by removing redundant constraints as shown in Table 3.10. The table compares the "classical" SNC and the Euclidean normalization after removal of redundant constraints and by working in the extended support.

Table 3.9: SNC normalization vs. Euclidean normalization on MIPLIB instances.

	SNC normalization			Euclidean normalization		
Instance	# cuts	%gap	$\|S\|$	# cuts	%gap	$\|S\|$
bell3a	71	70.74	43.72	71	70.74	34.95
bell5	178	94.29	11.75	213	93.01	13.78
blend2	192	30.51	8.10	182	31.80	9.04
flugpl	92	18.36	5.85	89	18.75	6.41
gt2	196	93.46	10.28	166	93.68	12.03
lseu	196	41.33	9.17	177	42.35	10.40
*markshare1	74	0.00	1.39	75	0.00	1.77
mod008	139	17.05	12.41	135	18.27	11.98
p0033	113	67.86	4.81	97	70.32	4.28
p0201	767	93.82	13.43	741	99.07	28.72
rout	434	24.26	68.07	412	7.92	71.53
*stein27	252	0.00	6.53	254	0.00	12.39
vpm1	263	55.84	5.39	256	67.17	5.59
vpm2	403	74.96	17.27	379	71.68	12.34
avg.	253.667	56.873	17.521	243.167	57.063	18.421

Table 3.10: SNC vs. Euclidean normalization on MIPLIB instances. Redundant constraints removed in both versions and projection on the extended support of x^*.

	"No redundancy" SNC			"No redundancy" EN		
Instance	# cuts	%gap	$\|S\|$	# cuts	%gap	$\|S\|$
bell3a	54	70.74	44.60	50	70.74	44.21
bell5	180	94.29	11.99	194	94.07	15.39
blend2	193	30.53	8.34	181	32.83	9.99
flugpl	93	18.86	5.95	92	19.20	6.02
gt2	187	93.88	13.10	215	94.71	13.54
lseu	178	43.45	9.08	165	45.00	8.86
*markshare1	77	0.00	1.69	75	0.00	1.77
mod008	157	19.13	14.43	135	18.36	11.98
p0033	146	70.29	5.89	99	70.32	4.47
p0201	769	100.00	13.39	765	100.00	32.44
rout	353	30.88	140.29	351	29.42	193.85
*stein27	251	0.00	7.13	251	0.00	13.17
vpm1	259	57.63	6.60	272	68.91	6.44
vpm2	373	75.84	17.71	361	76.61	14.38
avg.	245.167	58.793	24.281	240.000	60.014	30.131

Indeed, once the redundant constraints are removed, the average gap closed increases from 58.793% ("No redundancy" SNC) to 60.014% ("No redundancy" EN) and the negative behavior of EN on instance `rout` is almost fully recovered.

3.6 Conclusions and future work

Disjunctive cuts are a well-known (and often quite effective) class of cutting planes for mixed-integer linear programs. It is well known that, if the disjunction is fixed, then one can search for violated disjunctive cuts by solving an auxiliary LP, the so-called "Cut Generating Linear Program". Unfortunately, the solution of such an LP can yield weak cuts in some cases.

In this chapter we explored possible causes of this problem and possible ways to solve it. The main ingredients of a disjunctive cut separation procedure were critically analyzed, both from a theoretical and a computational point of view, often with the help of simple illustrating examples.

In particular, we compared classical normalization conditions used to truncate the CGLP cone, tried to better understand their role, and addressed a new normalization based on Euclidean norms.

We gave a theoretical characterization of weak rays/vertices of the CGLP cone that leads to dominated cuts, and proposed a practical heuristic method to strengthen them by solving a simple "Cut Dominating Linear Program".

We also pointed out, for the first time, the negative effects of redundancy in the quality of the generated disjunctive cuts, and discussed possible ways to cope with them.

Future work should address the practical implementation of the proposed strengthening procedures and of the new normalization condition within an efficient solution framework, possibly in conjunction with the open-source Balas and Bonami [19] implementation of the Balas-Perregaard pivoting scheme.

Chapter 4

Two-row cuts from the simplex tableau: a preliminary investigation

4.1 Introduction

Very recently, a series of papers have brought the attention to the possibility of generating cuts using more than one row of the simplex tableau at a time. Several interesting theoretical results have been presented in this direction, often revisiting and recalling other important results discovered more than 40 years ago. This chapter[1] presents a possible way for generating two-row cuts from the simplex tableau arising from lattice-free triangles and some preliminary computational results on a large set of mixed integer binary knapsack problems with two constraints. Interestingly, the results show that triangles are useful in practice to improve on the dual bound yielded by the classical GMI cuts.

The chapter is organized as follows. The remainder of this section introduces the required notation and discusses some of the main results from the literature. A first idea to derive maximal lattice-free triangles from two rows of the simplex tableau is presented in Section 4.2, while Section 4.3 reports the computational results, comparing triangle cuts with classical GMI cuts. Some preliminary conclusions and some open questions are highlighted in Section 4.4.

4.1.1 Multiple-row cuts

Let us assume to be given a mixed integer set

$$S = \{x \in \mathbb{R}^n : Ax = b, x \geq 0,\ x_j \in \mathbb{Z}\ \forall j \in J\}, \tag{4.1}$$

with $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, and denote the continuous relaxation of S as $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$. We assume for the sake of simplicity the set S to be nonempty.

[1] The results of this chapter are part of a joint work-in-progress with S. Dey, A. Lodi and L.A. Wolsey.

Given a basis $B \subset \{1, \ldots, n\}$ corresponding to the vertex $x^* \in P \setminus S$ of the polyhedron P, the set S can be rewritten as

$$\begin{array}{rcl} x_B & = & x^*_B + \sum_{j \in N} r^j x_j, \\ x & \geq & 0, \\ x_j & \in & \mathbb{Z},\ j \in J, \end{array} \tag{4.2}$$

where N denotes the set of nonbasic variables. A valid relaxation of S can be obtained by dropping the nonnegative restrictions on all the basic variables and considering a subset Q (with $|Q| = q$) of rows of (4.2) associated with basic integer-constrained variables (i.e., a subset of variables x_i with $i \in B \cap J$), thus getting

$$(S_Q) \qquad \begin{array}{rcl} x_i & = & f_i + \sum_{j \in N} r^j x_j,\ i \in Q \\ x_j & \geq & 0,\ j \in N \\ x_j & \in & \mathbb{Z},\ j \in J \end{array} \tag{4.3}$$

with $f_i = x^*_i - \lfloor x^*_i \rfloor$ for any $i \in Q$ and $f_i > 0$ for some $i \in Q$. In the following we assume w.l.o.g. $Q = \{1, \ldots, q\}$.

Set S_Q can be further relaxed by dropping all the integrality requirements on the nonbasic variables, thus getting a system of the form

$$\begin{array}{rcl} x & = & f + \sum_{j=1}^{k} r^j s_j, \\ x & \in & \mathbb{Z}^q, \\ s & \in & \mathbb{R}^k_+, \end{array} \tag{4.4}$$

where all the continuous variables have been renamed as s and $|N| = k$. Recall that $f, r^1, \ldots, r^k \in \mathbb{Q}^q$ and $f \notin \mathbb{Z}^q$, and denote as $R_f(r^1, \ldots, r^k)$ the convex hull of all vectors $s \in \mathbb{R}^k$ for which there exists $x \in \mathbb{R}^q$ such that (x, s) satisfies (4.4). Since S in non empty and all the data are rational entries, $R_f(r^1, \ldots, r^k)$ is a rational full-dimensional polyhedron: i.e., it is a rational polyhedron and its recession cone is $\mathbb{R}^k_+$.

Borozan and Cornuéjols [39] considered relaxing the k-dimensional space of variables $s = (s_1, \ldots, s_k)$ to an infinite-dimensional space, where the variables s_r are defined for any $r \in \mathbb{Q}^q$, thus getting the following *semi-infinite relaxation*, related to Gomory and Johnson's infinite group problem [100]:

$$\begin{array}{rcl} x & = & f + \sum_{r \in \mathbb{Q}^q} r s_r, \\ x & \in & \mathbb{Z}^q, \\ s & \geq & 0 \text{ with finite support} \end{array} \tag{4.5}$$

where $s = (s_r)_{r \in \mathbb{Q}^q}$ is said to have a finite support if it has a finite numbers of nonzero components. Let now R_f be the convex hull of all $s \in \mathbb{R}^{\mathbb{Q}^q}$ for which there exists $x \in \mathbb{R}^q$ such that (x, s) satisfies (4.5). In this general context, they showed that any valid inequality for R_f that cuts off the infeasible solution $s = 0$ can be written in the form

$$\sum_{r \in \mathbb{Q}^q} \psi(r) s_r \geq 1 \tag{4.6}$$

where $\psi : \mathbb{Q}^q \longrightarrow \mathbb{Q} \cup \{+\infty\}$ is said to be a *valid function* if the corresponding inequality (4.6) is valid for R_f. Then, they provided a strong correspondence between minimal valid inequalities and maximal lattice-free convex sets[2] as follows. Let define a *minimal* valid function as a valid function ψ such that there is no other valid function ψ' with $\psi'(r) \leq \psi(r)$ for all $r \in \mathbb{Q}^q$ and $\psi'(r) < \psi(r)$ for some $r \in \mathbb{Q}^q$. For any valid function ψ, consider the set $B_\psi = \{x \in \mathbb{Q}^q : \psi(x - f) \leq 1\}$ associated with ψ and denote as $\text{cl}(B_\psi)$ the topological closure of B_ψ in $\mathbb{R}^q$. The following theorem holds:

Theorem 4.1 (Borozan and Cornuéjols [39]). *Let $f \in \mathbb{Q}^q \setminus \mathbb{Z}^q$. A minimal valid function ψ for R_f is nonnegative, piecewise linear, positively homogeneous and convex. Furthermore the set $cl(B_\psi)$ is a full-dimensional maximal lattice-free convex set containing f. Conversely, for any full-dimensional maximal lattice-free convex set $B \subset \mathbb{R}^q$ containing f, there exists a minimal valid function ψ for R_f such that $cl(B_\psi) = B$, and when f is in the interior of B, this function is unique.*

In practice, the above theorem shows that any minimal valid inequality for R_f arises from a maximal lattice-free convex set B containing f. Further, any maximal lattice-free convex set B containing f in the interior lead to a *unique* minimal valid inequality. However, the latter results does not hold if f is on the boundary of B; i.e., any maximal lattice-free convex set containing f on the boundary leads, in the general case, to several minimal valid inequalities arising from different minimal valid functions ψ. Let denote any maximal lattice-free convex set B containing f on the boundary as a *degenerate set*. With the same meaning, let denote any minimal valid function ψ associated with a degenerate set as a *degenerate function*. The following question naturally arises from Theorem 4.1: should one consider degenerate sets/functions in practice?

The answer has been provided by Zambelli [176] for the finite dimensional case (i.e., for the set $R_f(r^1, \ldots, r^k)$):

Theorem 4.2 (Zambelli [176]). *Given a minimal valid inequality $\sum_{j=1}^{k} \alpha_j s_j \geq 1$ for $R_f(r^1, \ldots, r^k)$, there exists a nondegenerate minimal valid function ψ such that $\psi(r^j) = \alpha_j$ for all $j = 1, \ldots, k$.*

Thus, from the practical point of view of generating cuts for the finite dimensional set $R_f(r^1, \ldots, r^k)$, one does not need to be concerned with the complications arising from degenerate functions, since any minimal valid inequality can be obtained from a nondegenerate minimal valid function.

A first computational investigation on the possibility of generating cuts from a set of multiple rows has been provided by Espinoza [75], who considered relaxing the simplex tableau of a general MIP as a set of the form (4.4). This is obtained by selecting a subset Q of rows associated with basic integer-constrained variables and by dropping nonnegative requirements on all the basic variables and integrality requirements on all the nonbasic variables. The computational results reported in [75] on a large set of MIPLIB [36] instances showed that cutting planes from multiple-row sets are effective

[2] A lattice-free convex set is a convex set with no integer point in the interior.

in practice to improve on the lower bound at the root node yielded by the "classical" single-row cuts.

4.1.2 Two-row cuts

All the results reported above of course apply to the particular case of a two-row system of the form 4.4 (i.e. a system as 4.4 with $q = 2$). In this particular case, several authors provided even stronger results, thus giving a precise characterization of the facets of $R_f(r^1, \dots, r^k)$ for $q = 2$.

A first interesting result has been provided by Andersen, Louveaux, Weismantel and Wolsey [7], which formally proved that all the facets of $R_f(r^1, \dots, r^k)$ are *intersection cuts* (Balas [14]) arising from two-dimensional lattice-free convex sets. These set are triangles, quadrilaterals and split bodies, thus implying that $R_f(r^1, \dots, r^k)$ can be obtained as intersection of the *split closure* $S_f(r^1, \dots, r^k)$, the *triangle closure* $T_f(r^1, \dots, r^k)$ and the *quadrilateral closure* $Q_f(r^1, \dots, r^k)$[3]. Interestingly, they also provided a characterization of facets which cannot obtained as split cuts.

More recently, Cornuéjols and Margot [60] provided a precise description of which split bodies, triangles and quadrilaterals are facet inducing for $R_f(r^1, \dots, r^k)$, and also pointed out that degenerate sets are not required to obtain a full description of $R_f(r^1, \dots, r^k)$, thus giving an alternative proof of Theorem 4.2 for the case $q = 2$.

Finally, Basu, Bonami, Cornuéjols and Margot [29] compared the strength of the above mentioned closure from a theoretical point of view. First, they proved that the triangle closure and the quadrilateral closure are at least as strong as the split closure: i.e., $T_f(r^1, \dots, r^k) \subseteq S_f(r^1, \dots, r^k)$ and $Q_f(r^1, \dots, r^k) \subseteq S_f(r^1, \dots, r^k)$. Then, they showed how the triangle and the quadrilateral closures always provide a good approximation of $R_f(r^1, \dots, r^k)$ in a well defined sense (i.e., they always close at least half of the integrality gap), while the approximation provided by the split closure can be arbitrarily bad.

4.1.3 Lifting triangle inequalities

The typical approach to derive a valid inequality for a MIP defined over the set S is to consider only one row of (4.3) (i.e., by considering $Q = \{h\}$ for any $h \in B \cap J$ with $f_h > 0$). This leads to the well known GMI cut [99]. As discussed in Chapter 2, GMI cut can be obtained by lifting (i.e., strengthening) the intersection cut arising from the set $\{x_h \in \mathbb{R} : 0 \leq x_h \leq 1\}$ through the Balas-Jeroslow [23] procedure[4].

Roughly speaking, a GMI cut is in practice a valid inequality which is first obtained as a "trivial" intersection cut and it is afterwards strengthened by lifting the cut coefficients of nonbasic variables $j \in N \cap J$ (i.e., by exploiting the integrality requirements

[3]The split closure is defined as the intersection of all the valid inequalities arising from lattice-free split bodies. In a similar way, the triangle and the quadrilateral closure are defined as the intersection of all the valid inequalities arising, respectively, from lattice-free triangles and lattice-free quadrilaterals.

[4]The intersection cut from the set $\{x_h \in \mathbb{R} : 0 \leq x_h \leq 1\}$ was also denoted in Chapter 2 as the *simple* disjunctive cut associated with the row $x_h = x_h^* + \sum_{j \in N} r_h^j x_j$ and the split disjunction $x_h \leq \lfloor x_h^* \rfloor$ or $x_h \geq \lceil x_h^* \rceil$ (see Section 2.3).

on the nonbasic variables.)[5]. Dey and Wolsey [71] addressed the set S_Q for $|Q| = 2$ and the problem of lifting intersection cuts arising from maximal lattice-free triangles.

Let consider two rows of the system (4.3). Assume w.l.o.g. $Q = \{1, 2\}$ and rewrite S_Q in the more convenient form

$$\begin{array}{lrcl} (S_2) & x_1 & = & f_1 + \sum_{j \in N} r_1^j x_j \\ & x_2 & = & f_2 + \sum_{j \in N} r_2^j x_j \\ & x_j & \geq & 0,\ j \in N \\ & x_j & \in & \mathbb{Z},\ j \in J \end{array} \tag{4.7}$$

Given any maximal lattice-free convex set Π in $\mathbb{R}^2$ containing f in the interior, define the following function $\pi : \mathbb{Q}^2 \longrightarrow \mathbb{Q}_+$:

$$\pi(\omega) = \begin{cases} 0, & \text{if } \omega = (0,0) \\ \lambda, & \text{if } \omega \neq (0,0) \text{ and } \dfrac{\omega}{\lambda} \in Boundary(\Pi). \end{cases} \tag{4.8}$$

For all the already mentioned reasons, the inequality

$$\sum_{j \in N} \pi(r_j) x_j \geq 1 \tag{4.9}$$

is clearly valid for S_2. However, in the general case it is not minimal for $conv(S_2)$.

Let $f(v) = (f(v_1), f(v_2))$ denote the fractional part of any vector $v = (v_1, v_2) \in \mathbb{Q}^2$ (i.e., $f(v_k) = v_k - \lfloor v_k \rfloor, k = 1, 2$). Dey and Wolsey [71] considered the Johnson's mixed integer infinite group problem [106] to show that the inequality

$$\sum_{j \in N \setminus J} \pi(r_j) x_j + \sum_{j \in N \cap J} \phi^{\overline{0}}(f(r^j)) \geq 1 \tag{4.10}$$

is valid for S_2, where $\phi^{\overline{0}} : [0,1[^2 \longrightarrow [0,1]$ is the so-called *"trivial" fill-in* function (see [71]) defined as

$$\phi^{\overline{0}}(u) = \min_{n \in \mathbb{Z}^2} (\pi(u + n)). \tag{4.11}$$

Clearly, for a given set Π, inequality (4.9) dominates (4.10), since $\phi^{\overline{0}}(f(r^j)) \leq \pi(r^j)$ for any $j \in N \cap J$. Furthermore, addressing the special case in which Π is a triangle, they provided the following strong characterization of maximal lattice-free point triangles:

Proposition 4.1 (Dey and Wolsey [71]). *If* Π *is a maximal lattice-free triangle in* $\mathbb{R}^2$*, then exactly one of the following is true:*

1. *One side of* Π *contains more than one integral point in its relative interior.*
2. *All the vertices are integral and each side contains one integral point in its relative interior.*
3. *The vertices are non-integral and each side contains one integral point in its relative interior.*

[5]The reader is referred to Nemhauser and Wolsey [128] for a complete overview on lifting.

Then, they proved that function (π, ϕ) is minimal for the Johnson's mixed integer infinite group problem if Π is a triangle of Type 1 or Type 2. Finally, they provided a more general fill-in procedure yielding minimal inequalities even for triangle of Type 3 under certain sufficient conditions.

4.2 A practical idea to generate triangle inequalities

Generating cuts from a single tableau row poses no problem. When moving to the two-row case, instead, natural questions concern the choice of the row pairs and, for any fixed pair, the choice of one or more maximal-lattice free sets. In this chapter we decided to investigate the "potential" of lattice-free triangles of type 1 (i.e., according to Proposition 4.1, triangles with one edge containing more than one integral point in its relative interior) for which the lifting described in the previous section has been proven to be the best possible by Dey and Wolsey [71]. Hence, we consider all the row pairs whose corresponding basic variables x_{B_1} and x_{B_2} (say) are integer-constrained and *strictly* fractional (i.e., $f_{B_1} > 0$ and $f_{B_2} > 0$)[6], and for each pair we generate a bunch of maximal-lattice free triangle of type 1 by looking at the directions provided by the rays associated with *continuous* variables in the selected rows.

Rewrite the set S_2 of the form (4.7) associated with x_{B_1} and x_{B_2} by renaming $f = (f_{B_1}, f_{B_2})$ as $f = (f_1, f_2)$, the nonbasic integer variables as $x_1, \ldots, x_{n_1}$ and the nonbasic continuous variables as $y_1, \ldots, y_{n_2}$. We get then the system

$$\begin{aligned} x_{B_1} &= f_1 + \sum_{i=1}^{n_1} a_1^i x_i + \sum_{j=1}^{n_2} b_1^j y_j \\ x_{B_2} &= f_2 + \sum_{i=1}^{n_1} a_2^i x_i + \sum_{j=1}^{n_2} b_2^j y_j \\ & x \in \mathbb{Z}_+^{n_1} \quad y \in \mathbb{R}_+^{n_2} \end{aligned}$$

For any pair of rows defining the system above (with $f_1 > 0$ and $f_2 > 0$) we generate a bunch of cuts according to the following scheme:

1. Select some sets of three continuous variables $(y_{j_1}, y_{j_2}, y_{j_3})$ such that the positive combination of b^{j_1}, b^{j_1}, and b^{j_3} spans $\mathbb{R}^2$ according to the following strategy:

   (a) Consider, in turn, all pair of variables (y_{j_1},y_{j_2}) with $b^{j_1} \neq (0,0)$ and $b^{j_2} \neq (0,0)$.

   (b) For each selected pair, let $c := (c_1, c_2)$ be the direction bisecting the angle made by $-b^{j_1}$ and $-b^{j_2}$ (with c belonging to the cone spanned by $-b^{j_1}$ and $-b^{j_2}$), and look for the third variable y_{j_3} whose associated direction b^{j_3} is as close as possible to c. (If no variable can be found in the interior of the cone spanned by $-b^{j_1}$ and $-b^{j_2}$ we skip the pair (y_{j_1},y_{j_2}).)

[6] Generating cuts from two rows with $f_{B_1} = 0$ or $f_{B_2} = 0$ is clearly possible. However, we decide to leave these "special" cases to a future investigation.

2. For each set of three selected variables $(y_{j_1}, y_{j_2}, y_{j_3})$, look for a maximal lattice-free triangle Π of type 1, with $f \in interior(\Pi)$, in the attempt of finding an inequality

$$\pi(b^{j_1})y_{j_1} + \pi(b^{j_2})y_{j_2} + \pi(b^{j_3})y_{j_3} \geq 1$$

extreme for the problem

$$x_B = f + b^{j_1}y_{j_1} + b^{j_2}y_{j_2} + b^{j_3}y_{j_3}, \quad x_B \in \mathbb{Z}^2,\ y_{j_1}, y_{j_2}, y_{j_2} \geq 0$$

where $\pi : \mathbb{R}^2 \to \mathbb{R}_+$ is the function defined in (4.8). This step is discussed in detail in Section 4.2.1.

3. For any triangle found in the previous step, compute the corresponding valid inequality by means of the functions π and $\phi^{\overline{0}}$ (from (4.11)), thus getting the cut

$$\sum_{i=1}^{n_1} \phi^{\overline{0}}(f(a^i))x_i + \sum_{j=1}^{n_2} \pi(b^j)y_j \geq 1.$$

4.2.1 Strategy to generate triangles with multiple integer points on one side

The procedure described in the previous section generates (for any pair of rows) at most one cut for any pair of nonbasic continuous variables. Given an ordered set of three variables $(y_{j_1}, y_{j_2}, y_{j_3})$ and the corresponding three directions $(b^{j_1}, b^{j_2}, b^{j_3})$ in the two tableau rows, we next outline a potential strategy to find maximal lattice-free triangles of type 1 when the exact rational representation of b^j is not known. The procedure works as follows: i) find a facet for a problem with two continuous variables (this step is equivalent to finding one side of the triangle which contains multiple integer points) and then ii) lift the third continuous variable. In the following we rewrite $j_1 = 1$, $j_2 = 2$ and $j_3 = 3$ to simplify the notation.

- Step1: Finding a facet with two variables.

  Choose the first two variables y_1, y_2 and consider the set Y_2:

  $$(Y_2) \qquad x = f + b^1y_1 + b^2y_2, \quad x \in \mathbb{Z}^2,\ y_1, y_2 \geq 0.$$

  The goal here is to generate one of the facets of $conv(Y_2)$ other than the "unbounded facets", i.e., a facet that is "most different" from a Gomory mixed integer cut. We divide this step into two sub steps: (a) finding the first integer point, and (b) finding the second integer point such that line segment passing through the two integer points forms a side of the triangle or proving that no such integer point exists.

  Step (a): Choose any vector $c = (c_1, c_2)$ in the interior of the cone formed by b^1 and b^2. Solve the problem

  $$\begin{array}{ll} \min & cx \\ \text{s.t.} & x = f + \lambda_1 b^1 + \lambda_2 b^2 \\ & \lambda_1, \lambda_2 \geq 0, x \in \mathbb{Z}^2. \end{array}$$

Let the optimal solution of this problem be $\overline{x}^1$. $\overline{x}^1$ is going to be one of the integer points in the interior of the side of the triangle. Note that $\overline{x}^1$ gives a valid cut of the form $c_1x_1 + c_2x_2 \geq c\overline{x}^1$. Re-written in the y-space, we will obtain a cut of the form $\nu_1y_1 + \nu_2y_2 \geq 1$ for some $\nu_1, \nu_2 > 0$. However, this cut is not necessarily facet defining for $conv(Y_2)$.

Step (b): Define the vector $v = \overline{x}^1 - f$ and the angles $\alpha = (b^1, v)$ and $\beta = (b^2, v)$. By following the same approach described in Step (a), we look for an integer point $\overline{x}^2 \neq \overline{x}^1$ in $cone(v, b^1)$ if $\alpha > \beta$, or in $cone(v, b^2)$ otherwise (the choice of the cone based on the angles α and β is done just to limit possible numerical troubles). Suppose w.l.o.g. that $\beta > \alpha$. Then we define $b' = v + \theta b^2$, with $\theta > 0$, we choose any vector $c = (c_1, c_2)$ in the interior of the cone formed by b^2 and b', we solve the problem

$$\begin{array}{ll} \min & cx \\ \text{s.t.} & x = f + \lambda_1 b^2 + \lambda_2 b' \\ & \lambda_1, \lambda_2 \geq 0, x \in \mathbb{Z}^2 \end{array}$$

and we denote as $\overline{x}^2$ its optimal solution.

The line segment passing through $\overline{x}^2$ and $\overline{x}^1$ may not give a valid cut. Hence we repeat the following step to obtain a valid cut.

1. Let $e \in \mathbb{R}^2$ such that $e\overline{x}^2 = e\overline{x}^1 = 1$ and $ef < 1$.
2. Solve the problem:

$$\begin{array}{ll} \min & ex \\ \text{s.t.} & x = f + \lambda_1 b^1 + \lambda_2 b^2 \\ & \lambda_1, \lambda_2 \geq 0, x \in \mathbb{Z}^2. \end{array}$$

   Let the optimal solution of this problem be $\tilde{x}$. If $e\tilde{x} \geq 1$, then the inequality $ex \geq 1$ is a valid cut and can be rewritten in the y-space. Otherwise, if $e\tilde{x} < 1$, then set $\overline{x}^2 \leftarrow \tilde{x}$ and repeat from Step 1.

One final check is needed to verify that the final $\overline{x}^2$ is a vertex of $conv(Y_2)$. This check becomes relevant when $conv(Y_2)$ has one vertex only (i.e., $\overline{x}^1$) and hence the line passing through $\overline{x}^2$ and $\overline{x}^1$ does not intercept b^1 and/or b^2 (in such a case, we skip the triangle). In order to perform this check, one can simply verify that $(\overline{x}^1 - \overline{x}^2)$ and $(\overline{x}^2 - \overline{x}^1)$ do not belong to the cone formed by b^1 and b^2.

- Step 2: Lifting the third continuous variable.

  From the previous step we have two integer points $\overline{x}^1$ and $\overline{x}^2$ that are tight at the inequality. Let v^1 (resp. v^2) be the vertex obtained by extending the line segment passing through $\overline{x}^1$ and $\overline{x}^2$ until meeting the ray $f + \mu b^1$, $\mu \geq 0$ (resp. $f + \mu b^2$, $\mu \geq 0$). v^1 and v^2 are two vertices of the triangle and the line segment connecting v^1 and v^2 is the edge of the triangle containing multiple integer points in its interior (e.g., $\overline{x}^1$ and $\overline{x}^2$).

We need now to identify the other two integer points which lie in the interior of the other two sides of the triangle, and then we can complete the triangle. This is done through the following steps.

Step (a): Let r_1 be the line passing through $\overline{x}^2$ and $\overline{x}^1$, defined as

$$(r_1) \qquad x = \overline{x}^1 + \lambda w,$$

with $w = \overline{x}^2 - \overline{x}^1$. Shift r_1 in the direction provided by the third ray b^3 until meeting some integer point, thus getting a line r_2 parallel to r_1. r_1 and r_2 describe a maximal lattice-free split body, and the line r_2 can be expressed as

$$(r_2) \qquad x = \tilde{x} + \lambda w,$$

with $\tilde{x} \in \mathbb{Z}^2$.

Step (b): From the previous step we know that the other two integer points which lie on the boundary of the triangle must belong to the line r_2. They can be obtained as $\overline{x}^{13} = \tilde{x} + kw$ and $\overline{x}^{23} = \tilde{x} + (k+1)w$ for some integer k, by considering two different situations. If f lies in the interior of the split described by r_1 and r_2, then the choice of k depends on b^3. In this case we solve the system

$$\begin{aligned} \tilde{x} + \lambda w &= f + \mu b^3 \\ \mu &\geq 0. \end{aligned}$$

Otherwise, if f it is not in the interior of the split described by r_1 and r_2, then the choice of k does not depend on b^3. In this case we solve the system

$$\begin{aligned} \tilde{x} + \lambda w &= f + \mu(\overline{x}^1 - f) \\ \mu &\geq 0. \end{aligned}$$

In both cases we set $k = \lfloor \lambda \rfloor$.

Step (c): Once we know the integer points $\overline{x}^{13}$ and $\overline{x}^{23}$ which lie in the interior of the other two sides of the triangle, we can easily complete the triangle by extending the rays $v^1 + \mu(\overline{x}^{13} - v^1), \mu \geq 0$ and $v^2 + \mu(\overline{x}^{23} - v^2), \mu \geq 0$ until they meet, thus getting the third vertex v^3.

Example 4.1 (Finding the first side of triangle) *Suppose that* $b^1 = (1, 0)$, $b^2 = (-1, -2)$ *and* $f = (0.5, 0.5)$.

1. *Let us choose* $c = (1, -2)$. *Therefore we solve the problem,*

$$\begin{aligned} \textit{min} \quad & x_1 - 2x_2 \\ \textit{s.t.} \quad & x_1 = 0.5 + \lambda_1 - \lambda_2 \\ & x_2 = 0.5 - 2\lambda_2 \\ & \lambda_1, \lambda_2 \geq 0, x \in \mathbb{Z}^2 \end{aligned}$$

The optimal solution is $\overline{x}^1 = (1, 0)$ *and* $c\overline{x}^1 = 1$. *This gives us the cut* $x_1 - 2x_2 \geq 1$. *Re-written in the y-space, we obtain the cut,* $\frac{2}{3}y_1 + 2y_2 \geq 1$. *This cut is not facet-defining for* $conv(Y_2)$ *as illustrated in Figure 4.1.*

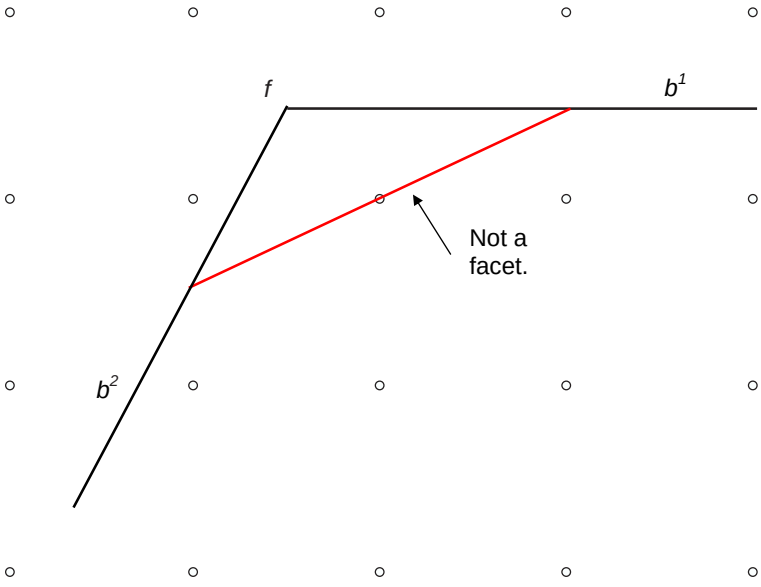

Figure 4.1: Step 1(a): finding the first integer point.

Next we would like to improve this cut. Since in this case $\beta > \alpha$*, then we look for an integer point* $\overline{x}^2 \neq \overline{x}^1$ *in the cone spanned by* $v = \overline{x}^1 - f = (0.5, -0.5)$ *and by* b^2*. Let* $\theta = 1$*. So* $b' = (0.5, -0.5) + (-1, -2) = (-0.5, -2.5)$*. Let choose* $c := (-1, 0)$*. Therefore, we solve*

$$\begin{array}{ll} min & -x_1 \\ s.t. & x_1 = 0.5 - \lambda_1 - 0.5\lambda_2 \\ s.t. & x_2 = 0.5 - 2\lambda_1 - 2.5\lambda_2 \\ & \lambda_1, \lambda_2 \geq 0, x \in \mathbb{Z}^2. \end{array}$$

The optimal solution is $\overline{x}^2 = (0, -1)$*. Then we take* $e = (1, -1)$ *(since* $e\overline{x}^1 = e\overline{x}^2 = 1$ *and* $ef < 1$*) and we solve the problem*

$$\begin{array}{ll} min & x_1 - x_2 \\ s.t. & x = (0.5, 0.5) + \lambda_1 b^1 + \lambda_2 b^2 \\ & \lambda_1, \lambda_2 \geq 0, x \in \mathbb{Z}^2 \end{array}$$

The optimal objective function is 1*. Therefore the inequality* $x_1 - x_2 \geq 1$ *is a valid cut and the two integer point in the interior of the first edge are precisely* $\overline{x}^1 = (1, 0)$ *and* $\overline{x}^2 = (0, -1)$*, as depicted in Figure 4.2. Rewritten in the y-space, the above inequality reads as* $y_1 + y_2 \geq 1$*.* □

Example 4.2 (Completing the triangle) *In the previous example we obtained the inequality:* $y_1 + y_2 \geq 1$ *for the system with* $b^1 = (1, 0)$*,* $b^2 = (-1, -2)$ *and* $f = (0.5, 0.5)$*.*

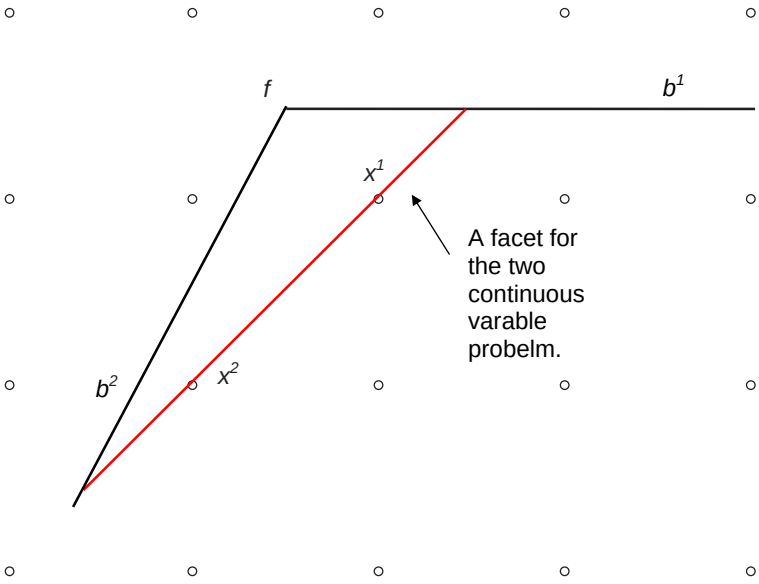

Figure 4.2: Step 1(b): finding the second integer point.

The first edge of the triangle is passing through the integer points $\overline{x}^1 = (1, 0)$ *and* $\overline{x}^2 = (0, -1)$*. Then we consider the line* r_1 *defined as*

$$(r_1) \qquad x = (1, 0) + \lambda(-1, -1).$$

Suppose we want to lift the continuous variable y_3 *corresponding to* $b^3 = (-0.5, 1)$*. By shifting* r_1 *in the direction provided by* b^3 *we find the line* r_2 *defined as*

$$(r_2) \qquad x = (-1, -1) + \lambda(-1, -1).$$

Since in this case f *belongs to* r_2*, then the other two integer points do not depend on the choice of* b^3*. By solving the system*

$$(-1, -1) + \lambda(-1, -1) = (0.5, 0.5) + \mu(0.5, -0.5), \quad \mu \geq 0$$

we obtain $\lambda = -1.5$ *and hence we get* $k = -2$*. Then we compute* $\overline{x}^{13} = (-1, -1) + k(-1, -1) = (1, 1)$*,* $\overline{x}^{23} = (-1, -1) + (k + 1)(-1, -1) = (0, 0)$ *and we complete the triangle as depicted in Figure 4.3.* □

4.3 Preliminary computational results

This section presents some preliminary computational experiments aimed at providing some insights on the practical impact of two-row cuts arising from maximal lattice-free triangles of Type 1. When moving to two-row cuts, the first natural question concerns how much we can gain, with respect to the classical single-row cuts, on two-row problems. Hence, as a first testbed, we considered a large set of 0–1 mixed knapsack

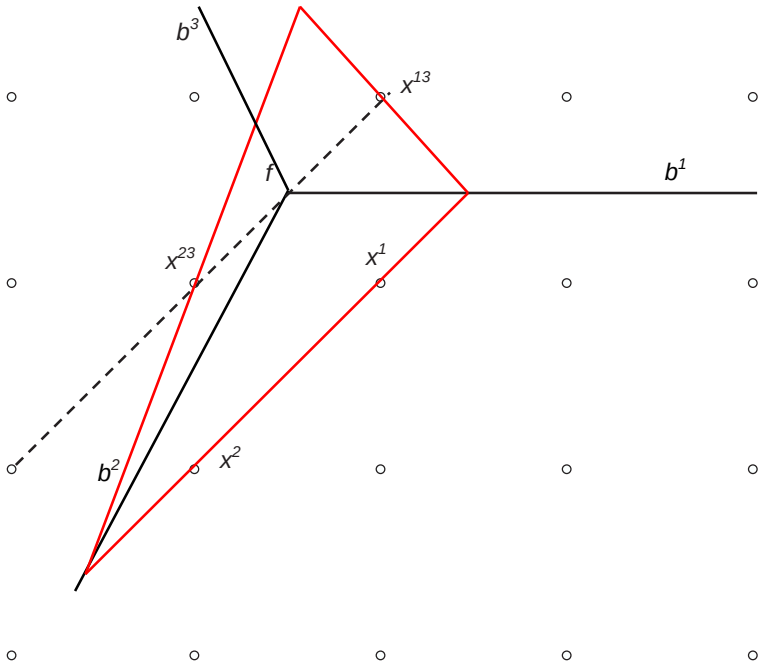

Figure 4.3: Step 2: completing the triangle.

problems with two constraints only. All the instances have been generated by slightly modifying the mixed integer knapsack problem generator kindly provided by Alper Atamturk, and are named as `mik2-x-y.n`, where `x` and `y` denote, respectively, the number of binary and continuous variables.

Tables 4.1–4.3 report the results obtained by adding one round of cuts to the LP relaxation. In particular, the tables compare the effect of "triangle cuts" with respect to the classical GMI cuts. Three different possibilities have been considered: i) adding the classical GMI cuts only, ii) adding the triangle cuts only, and iii) adding both families of cuts. For each of these configurations, the tables report the number of generated cuts (# cuts), the percentage gap closed (% gap) and the CPU seconds spent for the cut generation (sep.secs). The last table also reports the average values over all the 90 considered instances. As usual, when dealing with the GMIs, we generated one cut for each tableau row associated with a fractional integer-constrained basic variable. Triangle cuts are instead generated as discussed in the previous section.

The tables show that triangles are useful in practice to improve on the dual bound yielded by the classical GMI cuts. Behind an average improvement in the % gap (i.e., GMIs close an average gap of 13.57% while triangles 15.21%), it is worth noting that the gap provided by triangles is better than the one yielded by GMIs in 60 out of the 90 instances, while GMIs provide a better dual bound in the remaining 30 instances. Interestingly, as shown by the third tested configuration (i.e., GMI + triangles), the two families of cuts often collaborate. In particular, the average gap closed adding both cuts raises to 16.58, and we can observe that triangles dominate GMIs (in terms of

dual bound) in 42 cases out of 90, GMIs dominate triangles in 17 cases, while the two families collaborate in the remaining 31 problems.

Tables 4.4–4.6 reports the same results as Tables 4.1–4.3 but considering two round of cuts. Even in this case the last table reports the average values on all the 90 instances.

By adding two rounds of cuts the trend is confirmed and triangles seems even more effective. GMIs close an average gap of 19.55% while triangles 23.66%. Out of the 90 problems, triangles are better in 71 cases, and in 33 cases dominate GMIs, while GMIs are better in 19 cases and in 5 cases dominate triangles. Finally, on 52 out of the 90 tested problems the two families collaborate, and the average gap closed by adding both cuts together raises to 25.02%. When applying two round of cuts, it is worth noting that the number of generated cuts consistently increases when considering triangles, because the number of tableau rows of the second tableau becomes larger and all the pair of rows are considered in our cut generation procedure (i.e., no selection rule is applied).

4.4 Open questions

This chapter presented a first practical idea to generate two-row cuts arising from one type of maximal lattice-free triangles. Preliminary computational results on a large set of two-row problems have shown that triangles are effective in strengthening the relaxation obtained at the root node by adding the classical GMI cuts.

Many questions have still to be answered. First, other type of triangles need to be investigated, as well as quadrilaterals. Second, even in the simpler context of two-row problems, several different cuts can be generated, and hence some selection rules must be investigated. Third, the effect of two-row cuts on multiple-row problems (i.e., on general MIPs) has not been tested in this chapter. On the practical side, moving to general MIPs requires also to devise an effective way to select only few row pairs from the simplex tableau, because adding cuts arising from all the row pairs seems not to be a practical approach.

Table 4.1: GMI cuts vs. triangle cuts: 1 round of cuts, part I.

Instance	GMI			triangles			GMI + triangles		
	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs
mik2-50-10.01	2	29.34	0.0	7	34.57	0.2	9	35.45	0.2
mik2-50-10.02	2	8.43	0.0	4	10.06	0.1	6	10.06	0.1
mik2-50-10.03	2	45.31	0.0	5	54.83	0.2	7	54.83	0.2
mik2-50-10.04	2	10.45	0.0	5	17.48	0.1	7	17.48	0.1
mik2-50-10.05	2	1.92	0.0	8	3.66	0.3	10	3.66	0.3
mik2-50-10.06	2	20.59	0.0	5	20.04	0.1	7	22.52	0.1
mik2-50-10.07	2	5.31	0.0	5	4.63	0.1	7	6.45	0.1
mik2-50-10.08	2	5.42	0.0	4	2.92	0.2	6	5.42	0.2
mik2-50-10.09	2	7.91	0.0	3	13.34	0.1	5	13.34	0.1
mik2-50-10.10	2	10.50	0.0	5	20.26	0.2	7	20.26	0.2
mik2-50-20.01	2	18.07	0.0	6	19.00	0.2	8	19.04	0.2
mik2-50-20.02	2	12.86	0.0	7	16.64	0.3	9	17.64	0.3
mik2-50-20.03	2	10.54	0.0	5	15.92	0.4	7	16.03	0.4
mik2-50-20.04	2	9.14	0.0	4	9.74	0.2	6	9.74	0.2
mik2-50-20.05	2	19.20	0.0	5	21.02	0.2	7	21.02	0.2
mik2-50-20.06	2	28.46	0.0	5	20.82	0.3	7	28.46	0.3
mik2-50-20.07	2	26.20	0.0	6	19.45	0.3	8	26.20	0.3
mik2-50-20.08	2	26.71	0.0	6	25.44	0.3	8	28.30	0.3
mik2-50-20.09	2	16.61	0.0	5	13.65	0.3	7	18.03	0.2
mik2-50-20.10	2	23.59	0.0	3	17.61	0.1	5	24.05	0.1
mik2-50-30.01	2	26.75	0.0	5	26.92	0.6	7	30.84	0.6
mik2-50-30.02	2	11.48	0.0	5	10.73	0.4	7	11.48	0.4
mik2-50-30.03	2	11.94	0.0	6	12.89	0.4	8	12.89	0.4
mik2-50-30.04	2	27.36	0.0	6	27.10	0.5	8	30.83	0.5
mik2-50-30.05	2	14.25	0.0	6	17.67	0.3	8	17.67	0.3
mik2-50-30.06	2	13.70	0.0	5	19.23	0.3	7	20.97	0.3
mik2-50-30.07	2	14.80	0.0	6	15.28	0.2	8	15.29	0.2
mik2-50-30.08	2	21.43	0.0	6	27.64	0.3	8	29.21	0.3
mik2-50-30.09	2	10.65	0.0	3	9.88	0.2	5	10.65	0.2
mik2-50-30.10	2	11.22	0.0	4	11.61	0.3	6	11.61	0.3

Table 4.2: GMI cuts vs. triangle cuts: 1 round of cuts, part II.

Instance	GMI			triangles			GMI + triangles		
	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs
mik2-100-10.01	2	8.14	0.0	3	12.83	0.1	5	12.83	0.1
mik2-100-10.02	2	20.50	0.0	6	21.40	0.1	8	23.25	0.1
mik2-100-10.03	2	9.81	0.0	8	16.12	0.1	10	16.12	0.1
mik2-100-10.04	2	6.00	0.0	5	4.85	0.2	7	6.00	0.2
mik2-100-10.05	2	9.39	0.0	4	7.98	0.0	6	9.39	0.0
mik2-100-10.06	2	2.67	0.0	5	4.55	0.1	7	4.55	0.1
mik2-100-10.07	2	1.04	0.0	5	10.30	0.1	7	10.30	0.1
mik2-100-10.08	2	6.87	0.0	3	8.91	0.1	5	9.55	0.1
mik2-100-10.09	2	33.71	0.0	3	18.84	0.1	5	34.21	0.1
mik2-100-10.10	2	18.28	0.0	4	18.63	0.1	6	18.77	0.1
mik2-100-20.01	2	16.49	0.0	7	12.60	0.2	9	16.49	0.2
mik2-100-20.02	2	16.81	0.0	6	15.48	0.3	8	16.93	0.2
mik2-100-20.03	2	20.87	0.0	4	21.48	0.2	6	22.84	0.2
mik2-100-20.04	2	3.89	0.0	7	11.85	0.2	9	11.85	0.2
mik2-100-20.05	2	14.20	0.0	7	11.20	0.4	9	14.20	0.4
mik2-100-20.06	2	18.89	0.0	5	22.91	0.5	7	22.91	0.5
mik2-100-20.07	2	10.44	0.0	6	5.82	0.2	8	10.44	0.2
mik2-100-20.08	2	1.58	0.0	6	8.04	0.4	8	8.04	0.4
mik2-100-20.09	2	9.78	0.0	5	11.47	0.2	7	12.07	0.2
mik2-100-20.10	2	5.50	0.0	2	6.10	0.3	4	6.32	0.2
mik2-100-30.01	2	7.98	0.0	5	9.52	0.3	7	9.52	0.3
mik2-100-30.02	2	18.23	0.0	5	14.62	0.5	7	18.23	0.4
mik2-100-30.03	2	17.09	0.0	4	17.53	0.7	6	17.53	0.7
mik2-100-30.04	2	15.54	0.0	5	34.13	0.4	7	34.13	0.4
mik2-100-30.05	2	4.81	0.0	5	8.10	0.4	7	8.10	0.4
mik2-100-30.06	2	18.53	0.0	6	20.17	0.7	8	20.17	0.7
mik2-100-30.07	2	19.68	0.0	5	26.78	2.0	7	26.78	2.0
mik2-100-30.08	2	3.41	0.0	5	3.18	0.3	7	3.41	0.3
mik2-100-30.09	2	22.38	0.0	6	19.97	0.3	8	22.55	0.3
mik2-100-30.10	2	8.74	0.0	6	7.41	0.3	8	8.74	0.3

Table 4.3: GMI cuts vs. triangle cuts: 1 round of cuts, part III.

Instance	GMI			triangles			GMI + triangles		
	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs
mik2-150-10.01	2	8.27	0.0	6	17.37	0.3	8	17.37	0.3
mik2-150-10.02	2	16.84	0.0	6	16.56	0.2	8	17.68	0.2
mik2-150-10.03	2	5.85	0.0	6	11.31	0.1	8	11.31	0.1
mik2-150-10.04	2	2.30	0.0	3	2.65	0.1	5	3.25	0.1
mik2-150-10.05	2	6.73	0.0	5	6.91	0.1	7	7.61	0.1
mik2-150-10.06	2	6.56	0.0	5	7.07	0.1	7	7.07	0.1
mik2-150-10.07	2	4.75	0.0	5	7.81	0.1	7	7.81	0.1
mik2-150-10.08	2	12.17	0.0	6	11.90	0.1	8	12.55	0.1
mik2-150-10.09	2	14.30	0.0	4	4.81	0.1	6	14.30	0.1
mik2-150-10.10	2	43.32	0.0	6	38.72	0.2	8	43.32	0.2
mik2-150-20.01	2	10.25	0.0	4	18.99	0.1	6	18.99	0.1
mik2-150-20.02	2	18.06	0.0	1	15.34	0.2	3	18.06	0.2
mik2-150-20.03	2	6.98	0.0	4	12.54	0.2	6	12.54	0.2
mik2-150-20.04	2	8.98	0.0	4	24.23	0.3	6	24.23	0.3
mik2-150-20.05	2	13.11	0.0	6	15.85	0.3	8	16.00	0.3
mik2-150-20.06	2	22.54	0.0	7	26.86	0.5	9	26.86	0.5
mik2-150-20.07	2	5.18	0.0	5	7.67	0.2	7	7.67	0.2
mik2-150-20.08	2	15.19	0.0	7	17.28	0.6	9	17.28	0.6
mik2-150-20.09	2	11.08	0.0	6	18.13	0.3	8	18.13	0.2
mik2-150-20.10	2	18.85	0.0	4	13.72	0.3	6	19.40	0.3
mik2-150-30.01	2	16.55	0.0	6	20.21	0.4	8	23.69	0.4
mik2-150-30.02	2	3.71	0.0	2	2.88	0.2	4	3.71	0.2
mik2-150-30.03	2	8.06	0.0	5	11.04	0.2	7	11.04	0.2
mik2-150-30.04	2	9.85	0.0	3	9.53	0.6	5	10.40	0.6
mik2-150-30.05	2	25.45	0.0	5	26.62	0.6	7	26.62	0.7
mik2-150-30.06	2	4.69	0.0	4	6.09	0.4	6	6.09	0.4
mik2-150-30.07	2	3.68	0.0	2	5.02	0.2	4	5.02	0.2
mik2-150-30.08	2	16.36	0.0	4	24.81	0.7	6	24.81	0.6
mik2-150-30.09	2	4.84	0.0	5	6.28	0.5	7	6.28	0.5
mik2-150-30.10	2	5.43	0.0	6	7.56	0.4	8	7.56	0.4
avg.	2.0	13.57	0.0	5.0	15.21	0.3	7.0	16.58	0.3

Table 4.4: GMI cuts vs. triangle cuts: 2 round of cuts, part I.

Instance	GMI			triangles			GMI + triangles		
	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs
mik2-50-10.01	5	34.00	0.0	85	41.48	2.2	186	40.25	4.7
mik2-50-10.02	6	12.41	0.0	30	18.50	1.1	35	18.50	1.1
mik2-50-10.03	5	47.24	0.0	43	63.93	1.0	48	63.93	1.0
mik2-50-10.04	5	23.01	0.0	25	29.26	0.7	58	27.65	1.9
mik2-50-10.05	6	2.39	0.0	75	4.54	2.7	81	4.54	2.7
mik2-50-10.06	6	22.80	0.0	35	25.77	1.3	438	29.11	7.4
mik2-50-10.07	6	6.05	0.0	51	7.76	1.4	55	8.71	1.7
mik2-50-10.08	5	11.98	0.0	41	14.93	0.9	22	17.84	1.1
mik2-50-10.09	5	17.39	0.0	24	31.29	0.6	29	38.88	0.6
mik2-50-10.10	5	17.44	0.0	24	26.84	0.8	29	27.17	0.8
mik2-50-20.01	5	22.25	0.0	33	22.44	2.0	77	23.41	4.3
mik2-50-20.02	6	14.88	0.0	92	22.12	4.8	183	20.62	11.2
mik2-50-20.03	5	20.78	0.0	191	20.87	17.1	170	24.38	6.9
mik2-50-20.04	6	10.24	0.0	184	12.92	6.3	191	12.92	6.3
mik2-50-20.05	5	31.39	0.0	49	29.32	1.7	54	29.53	1.7
mik2-50-20.06	6	37.73	0.0	77	34.50	3.1	148	33.72	6.3
mik2-50-20.07	6	40.65	0.0	27	38.53	11.0	53	53.30	4.8
mik2-50-20.08	5	34.26	0.0	88	31.52	3.7	112	33.59	5.8
mik2-50-20.09	5	25.22	0.0	79	24.77	2.6	56	30.96	3.5
mik2-50-20.10	6	32.40	0.0	62	28.68	3.6	69	38.38	6.9
mik2-50-30.01	4	31.44	0.0	71	34.89	1.3	86	37.89	1.4
mik2-50-30.02	6	15.58	0.0	64	21.59	6.7	63	21.85	5.1
mik2-50-30.03	5	15.17	0.0	27	18.57	3.7	32	18.57	3.7
mik2-50-30.04	5	36.95	0.0	45	44.93	6.4	102	45.67	4.6
mik2-50-30.05	6	16.24	0.0	23	26.84	2.0	28	26.84	2.0
mik2-50-30.06	6	24.45	0.0	77	26.95	3.6	41	29.24	2.3
mik2-50-30.07	6	30.75	0.0	69	32.13	3.6	71	31.98	5.8
mik2-50-30.08	6	34.05	0.0	145	40.37	8.4	129	44.83	11.4
mik2-50-30.09	4	21.14	0.0	68	20.84	4.1	17	22.89	0.7
mik2-50-30.10	5	19.85	0.0	23	29.29	2.3	28	29.29	2.3

Table 4.5: GMI cuts vs. triangle cuts: 2 round of cuts, part II.

Instance	GMI			triangles			GMI + triangles		
	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs
mik2-100-10.01	6	10.41	0.0	26	18.69	0.7	31	18.92	0.7
mik2-100-10.02	6	22.54	0.0	35	31.13	0.8	88	32.51	2.1
mik2-100-10.03	5	14.25	0.0	205	24.92	4.1	212	25.15	4.2
mik2-100-10.04	4	7.76	0.0	34	18.71	1.0	9	7.76	0.2
mik2-100-10.05	5	18.25	0.0	26	20.49	0.9	18	20.99	2.5
mik2-100-10.06	5	9.17	0.0	31	13.17	1.0	36	13.17	1.0
mik2-100-10.07	5	4.88	0.0	31	15.73	1.0	36	15.73	1.0
mik2-100-10.08	5	21.04	0.0	35	18.55	0.9	61	18.42	1.6
mik2-100-10.09	6	37.45	0.0	26	29.09	1.3	101	40.52	2.0
mik2-100-10.10	5	29.86	0.0	69	32.90	1.9	72	33.17	2.0
mik2-100-20.01	5	23.71	0.0	25	18.99	16.8	57	26.15	2.2
mik2-100-20.02	5	25.30	0.0	40	29.69	1.7	40	36.39	1.6
mik2-100-20.03	5	22.80	0.0	217	25.10	9.1	124	24.55	4.5
mik2-100-20.04	5	9.63	0.0	68	19.20	5.1	74	19.38	5.2
mik2-100-20.05	6	15.65	0.0	56	14.82	5.6	51	16.21	2.7
mik2-100-20.06	5	27.33	0.0	93	40.24	3.5	99	40.24	3.5
mik2-100-20.07	6	13.54	0.0	73	13.71	4.9	94	18.25	3.6
mik2-100-20.08	6	1.88	0.0	23	16.04	1.9	28	16.73	1.9
mik2-100-20.09	6	19.11	0.0	39	22.39	50.1	47	20.45	1.5
mik2-100-20.10	5	9.08	0.0	14	9.32	1.5	98	10.57	3.2
mik2-100-30.01	6	10.60	0.0	61	25.06	4.0	67	25.06	4.0
mik2-100-30.02	5	21.38	0.0	42	19.30	2.1	36	22.45	3.7
mik2-100-30.03	6	19.29	0.0	31	24.57	1.9	36	24.62	1.9
mik2-100-30.04	5	39.73	0.0	50	56.70	8.2	55	56.70	8.2
mik2-100-30.05	5	7.84	0.0	25	9.27	2.0	30	9.47	2.0
mik2-100-30.06	5	19.96	0.0	74	29.39	7.4	80	29.39	7.3
mik2-100-30.07	4	23.28	0.0	24	33.56	3.4	29	33.73	3.3
mik2-100-30.08	5	5.75	0.0	28	7.71	2.9	32	6.55	6.3
mik2-100-30.09	5	25.33	0.0	30	24.32	2.5	101	25.51	8.5
mik2-100-30.10	5	13.10	0.0	34	13.60	1.9	39	14.48	2.2

Table 4.6: GMI cuts vs. triangle cuts: 2 round of cuts, part III.

Instance	GMI			triangles			GMI + triangles		
	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs	# cuts	% gap	sep.secs
mik2-150-10.01	6	20.79	0.0	66	25.16	4.0	72	25.16	3.9
mik2-150-10.02	6	19.12	0.0	170	25.23	3.1	128	25.34	2.8
mik2-150-10.03	5	14.71	0.0	91	16.61	2.0	97	16.65	2.0
mik2-150-10.04	5	2.61	0.0	49	4.01	1.3	170	3.74	5.1
mik2-150-10.05	5	9.34	0.0	73	11.18	3.4	92	13.39	2.0
mik2-150-10.06	5	9.74	0.0	77	10.93	8.7	83	10.93	8.7
mik2-150-10.07	4	8.02	0.0	84	10.47	1.7	90	10.48	1.6
mik2-150-10.08	6	17.14	0.0	40	22.26	0.9	105	24.77	3.8
mik2-150-10.09	5	15.20	0.0	17	10.19	0.4	31	15.94	0.7
mik2-150-10.10	6	52.35	0.0	80	50.61	1.7	93	54.27	2.6
mik2-150-20.01	6	14.23	0.0	66	21.94	4.8	72	21.94	4.8
mik2-150-20.02	6	24.31	0.0	19	26.70	1.4	97	27.88	3.7
mik2-150-20.03	6	8.77	0.0	82	26.59	4.0	88	26.59	3.9
mik2-150-20.04	5	15.26	0.0	80	26.80	3.4	86	26.80	3.4
mik2-150-20.05	5	20.91	0.0	93	32.26	4.5	101	31.89	6.0
mik2-150-20.06	5	27.36	0.0	214	30.06	9.1	221	30.06	9.2
mik2-150-20.07	6	6.05	0.0	20	11.95	2.8	25	11.95	2.8
mik2-150-20.08	6	23.78	0.0	35	25.74	3.6	40	25.74	3.6
mik2-150-20.09	5	19.69	0.0	36	23.44	1.2	41	23.44	1.2
mik2-150-20.10	6	27.87	0.0	38	20.59	2.5	80	31.27	4.5
mik2-150-30.01	6	18.13	0.0	76	25.94	10.7	160	27.72	12.3
mik2-150-30.02	5	6.38	0.0	31	4.20	3.1	16	9.54	1.9
mik2-150-30.03	5	16.73	0.0	30	19.34	1.6	35	19.61	1.6
mik2-150-30.04	5	17.88	0.0	23	17.42	2.7	48	20.29	3.7
mik2-150-30.05	6	31.92	0.0	137	29.34	9.3	144	29.41	9.3
mik2-150-30.06	5	11.62	0.0	37	16.53	3.8	42	16.76	3.8
mik2-150-30.07	5	12.14	0.0	25	16.65	3.1	30	16.69	3.1
mik2-150-30.08	6	22.07	0.0	40	33.80	3.3	46	33.80	3.3
mik2-150-30.09	5	10.85	0.0	34	12.99	3.6	39	13.07	3.5
mik2-150-30.10	6	14.08	0.0	75	17.34	13.8	81	17.36	13.8
avg.	5.4	19.55	0.0	60.3	23.66	4.2	78.1	25.02	3.9

Chapter 5

Integer Linear Programming Local Search for the Vehicle Routing Problem

5.1 Introduction

The Vehicle Routine Problem (VRP) calls for the determination of the optimal set of routes to be performed by a fleet of vehicles to serve a given set of customers, and it is one of the most important and studied combinatorial optimization problems.

Fifty years have elapsed since Dantzig and Ramser [64] introduced the problem in 1959. They described a real-world application concerning the delivery of gasoline to service stations and proposed the first mathematical programming formulation and algorithmic approach. A few years later, in 1964, Clarke and Wright [51] proposed an effective greedy heuristic that improved on the Dantzig-Ramser approach. Following these two seminal papers, hundreds of models and algorithms were proposed for the optimal and approximate solution of different version of VRP. Dozens of packages for the solution of various real-world VRPs are now available on the market, but VRP is still widely studied in the literature, and several exact and heuristic approaches are proposed every year.

In this chapter[1] we address the "classical" (Capacitated and Distance Constrained) version of VRP, which can be formally stated as follows. We are given a central *depot* and a set of n *customers*, which are associated with the nodes of a complete undirected graph $G = (V, E)$ (where $V = \{0, 1, \ldots, n\}$, node 0 represents the depot and $V \setminus \{0\}$ is the set of customers). Each edge $e \in E$ has an associated finite *cost* $c_e \geq 0$ and each customer $v \in V \setminus \{0\}$ has a *demand* $q_v \geq 0$ (with $q_0 = 0$ for the depot node). A fleet of k identical *vehicles* is located at the depot, each one with a *capacity* Q and a *total*

[1] The results of this chapter appear in: P. Toth and A. Tramontani, "An integer linear programming local search for capacitated vehicle routing problems", in B. Golden, S. Raghavan, E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 275–295, Springer, New York, 2008 [169].

distance-traveled (duration) limit D. The customers must be served by at most k cycles (*routes*), each cycle associated with one vehicle, starting and ending at the depot. Each route must have a duration (computed as the sum of the edge costs in the route) not exceeding the given limit D of the vehicles, and can visit a subset S of customers whose total demand $\sum_{v \in S} q_v$ does not exceed the given capacity Q. The problem consists of finding a feasible solution covering (i.e., visiting) exactly once all the customers and having a minimum overall cost, computed as the sum of the traveled edge costs; see, e.g., [166].

VRP is $\mathcal{NP}$-hard in the strong sense, as it generalizes the Bin Packing Problem and the Traveling Salesman Problem. The interest in VRP is motivated by both its practical relevance and its considerable difficulty: the largest benchmark instance that can be solved to proven optimality has 120 customers only, whereas instances with more than 200 customers can be effectively attached by heuristic and metaheuristic approaches only. Exact methods usually deal with the capacitated problem with no distance constraints and no empty routes allowed (i.e., customers must be served by exactly k routes and $D = \infty$). Heuristic and metaheuristic algorithms usually take into account both capacity and distance constraints, and often consider the number of routes as a decision variable. For a comprehensive survey on solution techniques for the Vehicle Routing Problem we refer the reader to [55, 56, 96, 168]. Recent exact algorithms have been proposed by Augerat et al. [12], Hadjiconstantinou et al. [103], Ralphs et al. [146], Baldacci et al. [28], Lysgaard et al. [119], Fukasawa et al. [92]. Effective metaheuristic algorithms have been recently proposed by Osman [130], Taillard [161], Gendreau et al. [94], Rochat and Taillard [151], Rego and Roucairol [147], Xu and Kelly [175], Berger and Barkaoui [32], Toth and Vigo [167], Prins [145], Reimann et al. [149], Li et al. [114], Tarantilis [162], Wassan [172], Kytöjoki et al. [109], Mester and Bräysy [125], Pisinger and Ropke [141].

In this chapter we present an Integer Linear Programming (ILP) Local Search algorithm for VRP, based on an exponential neighborhood which is explored by solving an ILP formulation. The method follows a destroy-and-repair paradigm, where the current solution is randomly destroyed (i.e., customers are removed in a random way) and repaired by following ILP techniques. Hence, the overall procedure can be considered as a general framework which could be extended to cover other variants of Vehicle Routing Problems.

Our starting point is the refinement heuristic procedure proposed by De Franceschi et al. [68]. Given an initial solution to be possibly improved, the procedure performs the following steps: (a) randomly select several customers from the current solution, and build the restricted solution obtained from the current one by extracting (i.e., short-cutting) the selected customers; (b) reallocate the extracted customers to the restricted solution by solving an ILP problem (denoted as Reallocation Model), in the attempt of finding a new improved solution. To explore different neighborhoods of the same solution, customers are selected by means of different randomized selection criteria. The method proposed in [68] often provides an improvement of the initial solution, but is rather expensive in terms of computing time. To get a more effective and deeper exploration of the solution space, we present a generalization of the neighborhood pro-

posed in [68] and investigate the corresponding ILP formulation. Since the Reallocation Model has a number of variables exponential in the number of the extracted customers, the solution of its LP relaxation has to be handled by using pricing and column generation techniques. In [68], such a relaxation was heuristically solved through an intensive pricing loop, in which a huge number of variables were iteratively generated and added to the model only if their reduced costs were under a fixed threshold. Variable generation was driven by heuristic criteria (see [68] for details). In this chapter we investigate the column generation problem associated with the LP relaxation of the Reallocation Model, which is shown to be $\mathcal{NP}$-hard, and propose a two-phase approach for the neighborhood exploration, which first reduces the neighborhood size through a simple heuristic criterion, and then explores the reduced neighborhood by solving the corresponding Reallocation Model formulation through the (heuristic) solution of the column generation problem associated with its LP relaxation.

The chapter is organized as follows. In Section 5.2 the exponential neighborhood we propose for VRP is described, and the ILP formulation corresponding to the neighborhood exploration is presented. The implementation of the Local Search algorithm is given in Section 5.3, while Sections 5.4, 5.5 and 5.6 describe in detail the basic steps of the method. In particular, Section 5.4 describes the heuristic criteria for selecting the customers to be extracted (i.e., the neighborhood to be explored), Section 5.5 presents the heuristic procedure we propose for reducing the neighborhood size, while Section 5.6 presents the column generation problem associated with the LP relaxation of the Reallocation Model. Computational experiments on benchmark capacitated VRP instances from the literature (with/without distance constraints) are reported in Section 5.7, comparing the proposed method with the approach presented in [68], and with the most effective metaheuristic techniques proposed for VRP. Some conclusions are finally drawn in Section 5.8.

5.2 Exponential neighborhood

Let $\mathcal{Z}$ be the set of all the feasible solutions of the VRP defined on G. For any given solution $z_0 \in \mathcal{Z}$ and node subset $\mathcal{F} \subseteq V \setminus \{0\}$, we define $z_0(\mathcal{F})$ as the *restricted solution* obtained from z_0 by *extracting* (i.e., by short-cutting) all the nodes $v \in \mathcal{F}$. Let $\mathcal{I} = \mathcal{I}(z_0, \mathcal{F})$ denote the set of all the edges in $z_0(\mathcal{F})$, and $\mathcal{S} = \mathcal{S}(\mathcal{F})$ the set of all the *sequences* which can be obtained through the recombination of nodes in $\mathcal{F}$ (i.e., the set of all the paths in $\mathcal{F}$). Each edge $i \in \mathcal{I}$ is viewed as a potential *insertion point* which can allocate one or more nodes in $\mathcal{F}$ through at most one sequence $s \in \mathcal{S}$.

With the above notation, for each $z_0 \in \mathcal{Z}$ and $\mathcal{F} \subseteq V \setminus \{0\}$, we define the neighborhood $N(z_0, \mathcal{F})$ as the set of all the feasible solutions $z \in \mathcal{Z}$ which can be obtained through the reallocation of all the extracted nodes $v \in \mathcal{F}$ to the restricted solution $z_0(\mathcal{F})$. This is obtained by allocating some sequences $s \in \mathcal{S}$ to some insertion points $i \in \mathcal{I}$, so that each node $v \in \mathcal{F}$ is covered exactly once by the allocated sequences and each insertion point $i \in \mathcal{I}$ allocates at most one sequence $s \in \mathcal{S}$. We say that the insertion point $i = (a, b) \in \mathcal{I}$ allocates the nodes $\{v_j \in \mathcal{F} : j = 1, \ldots, h\}$ through the

sequence $s = (v_1, v_2, \ldots, v_h) \in \mathcal{S}$, if the edge (a, b) in the restricted solution is replaced by the edges $(a, v_1), (v_1, v_2), \ldots, (v_h, b)$ in the new feasible solution.

$N(z_0, \mathcal{F})$ is an exponential neighborhood of the given solution z_0 which can be viewed as an extension of the neighborhood proposed by Sarvanov and Doroshko [155], and, independently, by Gutin [102], for the pure Traveling Salesman Problem (see [68] for details). Of course, $N(z_0, \mathcal{F})$ depends on the choice of $\mathcal{F}$, and in particular $N(z_0, \emptyset) = \{z_0\}$, while $N(z_0, V \setminus \{0\}) = \mathcal{Z}$ for any $z_0 \in \mathcal{Z}$, since any empty route in $z_0(\mathcal{F})$ is viewed as an insertion point $(0, 0)$. In the general case, $N(z_0, \mathcal{F})$ can be explored by solving a set-partitioning model (denoted as the Reallocation Model) which is theoretically $\mathcal{NP}$-hard, but effectively solvable in practice. The reallocation model corresponding to z_0 and $\mathcal{F}$ can be described as follows.

Let $\mathcal{R}$ denote the set of routes in the restricted solution. For any sequence $s \in \mathcal{S}$, let $V(s)$ be the node set of s, $c(s)$ the sum of the costs of the edges in the sequence, and $q(s)$ the sum of the demands q_v associated with the nodes $v \in V(s)$. For any node $v \in \mathcal{F}$, let $\mathcal{S}(v) := \{s \in \mathcal{S} : v \in V(s)\}$ denote the set of sequences $s \in \mathcal{S}$ containing node v. For each insertion point $i = (a, b) \in \mathcal{I}$ and for each sequence $s = (v_1, v_2, \ldots, v_h) \in \mathcal{S}$ we define γ_{si} as the extra-distance (i.e., the extra-cost) for assigning sequence s to insertion point i in its best possible orientation (i.e., $\gamma_{si} := c(s) - c_{ab} + \min\{c_{av_1} + c_{v_h b}, c_{av_h} + c_{v_1 b}\}$). For each route $r \in \mathcal{R}$, let $\mathcal{I}(r)$ denote the set of insertion points (i.e., edges) associated with r, while let $\tilde{q}(r)$ and $\tilde{c}(r)$ denote, respectively, the total demand and distance computed for route r, still in the restricted solution. With the above notation, the Reallocation Model is an Integer Linear Programming (ILP) problem based on the decision variables

$$x_{si} = \begin{cases} 1 & \text{if sequence } s \in \mathcal{S} \text{ is allocated to insertion point } i \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

and reads as follows:

$$\sum_{r \in \mathcal{R}} \tilde{c}(r) + \min \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}} \gamma_{si} x_{si} \tag{5.2}$$

subject to:

$$\sum_{s \in \mathcal{S}(v)} \sum_{i \in \mathcal{I}} x_{si} = 1 \qquad v \in \mathcal{F}, \tag{5.3}$$

$$\sum_{s \in \mathcal{S}} x_{si} \leq 1 \qquad i \in \mathcal{I}, \tag{5.4}$$

$$\sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}(r)} q(s) x_{si} \leq Q - \tilde{q}(r) \qquad r \in \mathcal{R}, \tag{5.5}$$

$$\sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}(r)} \gamma_{si} x_{si} \leq D - \tilde{c}(r) \qquad r \in \mathcal{R}, \tag{5.6}$$

$$x_{si} \in \{0, 1\} \qquad s \in \mathcal{S},\ i \in \mathcal{I}. \tag{5.7}$$

The objective function (5.2), to be minimized, gives the cost of the final VRP solution. Constraints (5.3) impose that each extracted node belongs to exactly one of the selected sequences, i.e., that it is covered exactly once in the final solution. Note that, if the costs satisfy the triangle inequality, one could replace $=$ by $\geq$ in (5.3), thus obtaining an ILP having the structure of a set-covering (instead of a set-partitioning) problem with side constraints. Constraints (5.4) avoid to allocate two or more sequences to an insertion point. Finally, constraints (5.5) and (5.6) impose that each route in the final solution fulfills the capacity and distance restrictions, respectively.

Solving the reallocation model to optimality corresponds to the complete exploration of the neighborhood $N(z_0, \mathcal{F})$. However, for some choices of $\mathcal{F}$, the neighborhood can be too large and cannot be completely explored with an acceptable computational effort (note that the number of variables x_{si} is exponential in the number of nodes in $\mathcal{F}$). Moreover, the quality of the feasible solutions in $N(z_0, \mathcal{F})$ depends on the choice of $\mathcal{F}$. Indeed, in order to explore the solution space close to z_0 in an effective way, three different aspects have to be considered:

(a) different neighborhoods $N(z_0, \mathcal{F})$ have to be explored, using different sets $\mathcal{F}$, selected by using different selection criteria;

(b) it is of crucial importance to reduce the neighborhood to explore without loosing possible improvements of the current solution z_0;

(c) even the reduced neighborhood can be too large and has to be explored only in a partial way.

5.3 Local Search algorithm

The choice of the extracted node set $\mathcal{F}$ is a key factor of the proposed approach. In particular, wrong choices of $\mathcal{F}$ lead to bad neighborhoods which contain no improved solutions with respect to z_0, even if z_0 is not a "good" quality solution. To get promising neighborhoods, we apply an iterative local search algorithm: at each iteration, a different neighborhood $N(z_0, \mathcal{F})$ is explored, using a different set $\mathcal{F}$ of extracted nodes, determined according to different selection criteria (see Section 5.4).

Moreover, as previously mentioned, the neighborhood $N(z_0, \mathcal{F})$ could be too large and its complete exploration could not be performed in a reasonable computing time. Therefore, at each iteration of the local search algorithm, we first determine a reduced neighborhood $N(z_0, \mathcal{F}, \lambda) \subseteq N(z_0, \mathcal{F})$ and its corresponding Reduced Reallocation Model (see Section 5.5), and then we perform only a partial exploration of the current reduced neighborhood, which corresponds to the selection of a small subset of all the potential variables x_{si}, with $(s, i) \in \mathcal{S} \times \mathcal{I}$, for the Reduced Reallocation Model (see Section 5.6).

Given an initial feasible solution z_0 for VRP (taken from the literature or found by any heuristic method), the proposed Local Search Algorithm (LSA) works as follows:

1. (Initialization). Initialize a list Θ of all the available selection criteria.

2. (Selection). Apply the next selection criterion in Θ to determine the set $\mathcal{F}$ of nodes to be extracted.

3. (Extraction). Extract the nodes selected in the previous step and construct the corresponding restricted VRP solution obtained by short-cutting the extracted nodes.

4. (Reduction). Determine a reduced neighborhood $N(z_0, \mathcal{F}, \lambda) \subseteq N(z_0, \mathcal{F})$ and build the Linear Programming (LP) relaxation of the corresponding Reduced Reallocation Model with an initial empty set of variables.

5. (Construction). Populate the Reduced Reallocation Model with a promising subset of variables, determined by using pricing and column generation techniques.

6. (Reallocation). "Freeze" the current set of variables, add the integrality requirements to the variables and solve the corresponding Reduced Reallocation Model by using a general-purpose MIP solver. Once an (almost) optimal ILP solution has been found, construct the corresponding new VRP solution and possibly update the incumbent solution. If the incumbent solution has been updated, then process each route in the new solution through a 3-OPT exchange heuristic (in the attempt of further improving it) and repeat from Step 1.

7. (Termination). If the list Θ is empty, then STOP; otherwise repeat from Step 2.

5.4 Node selection criteria

Selection criteria determine the set $\mathcal{F}$ of extracted nodes, and therefore the neighborhood $N(z_0, \mathcal{F})$ to explore. Several deterministic criteria have been considered and experimentally evaluated, but none of them seems to work better than randomized criteria. Therefore we use the same randomized selection criteria proposed by De Franceschi et al. in [68]. They can be briefly described as follows.

- Random-Alternate scheme: this criterion is akin to the Sarvanov and Doroshko [155] scheme for the pure Traveling Salesman Problem: for each route all the nodes in even position or all the nodes in odd position are selected, the position parity being randomly-determined (with equal probability) for any route.

- Scattered scheme: each node has a probability p of being extracted, where p is a fixed parameter; this scheme allows for the removal of consecutive nodes, i.e., of route subsequences.

- Neighborhood scheme: given a seed node v^*, then v^* is selected and each other node v is selected with a probability inversely proportional to the distance c_{vv^*} of v from v^* (such that a given percentage p of the nodes are selected on average).

At the beginning of the computation a list N containing all the customer nodes is created, associating with each node v^* a score equal to the average distance from v^* of the 10 nodes nearest to v^* and sorting the list N by increasing scores. At each application of the scheme, the next node v^* in the list (in a circular way) plays the role of the seed node.

RANDOM-ALTERNATE and SCATTERED schemes appear particularly suited to improve the first solutions, and they seem to be useful even to improve new solutions obtained by the application of the NEIGHBORHOOD scheme. On the contrary, the NEIGHBORHOOD scheme seems more appropriate to improve good quality solutions, whereas the other schemes fail. Therefore, in Step 1 we initialize the list Θ as $\Theta := \{R(3), S(3), N(n)\}$ with the following meaning: we first apply 3 times the RANDOM-ALTERNATE scheme, then we apply 3 times the SCATTERED scheme, and afterwards the NEIGHBORHOOD scheme is applied for each customer node. In this way, in the first iterations we look for "global" improvements by using completely randomized selection criteria, and afterwards we concentrate on the neighborhoods of different customers looking for "local" improvements.

5.5 Neighborhood reduction

Concerning the Reallocation Model (5.2)–(5.7), for each sequence $s \in \mathcal{S}$ and for each insertion point $i \in \mathcal{I}$, we say that s is feasible for i if s can be allocated to i without violating capacity and distance constraints for the route r_i containing i. With the same notation, we say that each node $v \in \mathcal{F}$ is feasible for $i \in \mathcal{I}$ if the sequence (v) is feasible for i. Then we define $\mathcal{F}_i^* := \{v \in \mathcal{F} : v \text{ is feasible for } i\}$ and $\mathcal{I}_v^* := \{i \in \mathcal{I} : v \in \mathcal{F}_i^*\}$.

During the neighborhood construction, whenever we have to find a new variable x_{si} for a given insertion point $i \in \mathcal{I}$, we are required to generate a new sequence s by considering all the sequences in $\mathcal{S}$. However, many sequences in $\mathcal{S}$ may be infeasible for i, and many feasible sequences can have a too high insertion cost γ_{si} with respect to other feasible sequences. Removing a priori all the infeasible sequences and all the "bad" sequences (i.e., the sequences with a too high insertion cost) would lead to a strong reduction of the computational effort needed for the generation of new variables. To this end, for each $i \in \mathcal{I}$ we determine a reduced node subset $\mathcal{F}_i \subseteq \mathcal{F}$. All the sequences generated for i are sequences $s \in \mathcal{S}_i$, where $\mathcal{S}_i := \{s \in \mathcal{S} : V(s) \subseteq \mathcal{F}_i\}$. Associating each insertion point $i \in \mathcal{I}$ only with a reduced node subset $\mathcal{F}_i \subseteq \mathcal{F}$ corresponds to reduce the neighborhood $N(z_0, \mathcal{F})$ of all the feasible solutions that can be reached from z_0. With the above reduction, we get a reduced neighborhood which can be explored by solving a Reduced Reallocation Model (RRM), which is similar to the original Reallocation Model (5.2)–(5.7) and reads as follows:

$$\sum_{r \in \mathcal{R}} \tilde{c}(r) + \min \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}_s} \gamma_{si} x_{si} \tag{5.8}$$

subject to:

$$\sum_{s\in\mathcal{S}(v)}\sum_{i\in\mathcal{I}_s} x_{si} = 1 \qquad v\in\mathcal{F}, \tag{5.9}$$

$$\sum_{s\in\mathcal{S}_i} x_{si} \le 1 \qquad i\in\mathcal{I}, \tag{5.10}$$

$$\sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{I}_s(r)} q(s)x_{si} \le Q-\tilde{q}(r) \qquad r\in\mathcal{R}, \tag{5.11}$$

$$\sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{I}_s(r)} \gamma_{si}x_{si} \le D-\tilde{c}(r) \qquad r\in\mathcal{R}, \tag{5.12}$$

$$x_{si}\in\{0,1\} \qquad s\in\mathcal{S},\ i\in\mathcal{I}_s, \tag{5.13}$$

where, for each $s\in\mathcal{S}$ and each $r\in\mathcal{R}$, $\mathcal{I}_s := \{i\in\mathcal{I} : s\in\mathcal{S}_i\}$ and $\mathcal{I}_s(r) := \mathcal{I}_s\cap\mathcal{I}(r)$. Of course, the approach is effective only if we are able to find a smart reduction without loosing any potential improvement of z_0 in $N(z_0,\mathcal{F})$.

Assuming that the triangle inequality holds, a first exact reduction can be performed by setting $\mathcal{F}_i := \mathcal{F}_i^*$ for any $i\in\mathcal{I}$. Since any feasible sequence s for i has to be a path in $\mathcal{F}_i^*$, the above setting is not a true reduction of the neighborhood $N(z_0,\mathcal{F})$, but it corresponds to a strong reduction of the solution space of the LP relaxation of the Reallocation Model. In addition, we perform a true neighborhood reduction based on the following heuristic assertion: if an extracted node $v\in\mathcal{F}$ has to be allocated to a certain route $r\in\mathcal{R}$, then it will probably be allocated to one of the insertion points $i\in\mathcal{I}(r)$ nearest to v. Therefore, an effective problem reduction has to satisfy the following requirements:

(a) any node v has to be associated with its *pivot* i_v (i.e., the insertion point v was extracted from). In this way we can always get z_0 as the new solution;

(b) any node v has to be associated with at least one insertion point $i\in\mathcal{I}(r)$ for each route $r\in\mathcal{R}$;

(c) if the insertion cost of the sequence (v) in the insertion point i is "too high", then v can be removed from $\mathcal{F}_i$ (i.e., any sequence s containing v is removed from $\mathcal{S}_i$).

For each $v\in\mathcal{F}$ and for each $i\in\mathcal{I}_v^*$, let γ_{vi} denote the insertion cost of the sequence (v) in i. For each $r\in\mathcal{R}$ and for each $v\in\mathcal{F}$, we define δ_{vr} as the average insertion cost of the sequence (v) in the route r, computed as

$$\delta_{vr} := \frac{\sum_{i\in\mathcal{I}(r)\cap\mathcal{I}_v^*}\gamma_{vi}}{|\mathcal{I}(r)\cap\mathcal{I}_v^*|}. \tag{5.14}$$

With the above notation, assuming that the triangle inequality holds, for each $i\in\mathcal{I}$ we set

$$\mathcal{F}_i := \{v\in\mathcal{F}_i^* : \gamma_{vi}\le\lambda\ \delta_{vr_i}\}\cup\{v\in\mathcal{F} : i=i_v\} \tag{5.15}$$

where λ is a fixed parameter. We denote the reduced neighborhood corresponding to the Reduced Reallocation Model (5.8)–(5.13) with $N(z_0, \mathcal{F}, \lambda)$. Note that $\forall\ \lambda \geq 0, N(z_0, \mathcal{F}, \lambda) \subseteq N(z_0, \mathcal{F})$, and $\lim_{\lambda \to \infty} N(z_0, \mathcal{F}, \lambda) = N(z_0, \mathcal{F})$.

5.6 Neighborhood construction

Once the reduced neighborhood $N(z_0, \mathcal{F}, \lambda)$ has been built, it could be entirely explored by solving the Reduced Reallocation Model (RRM) (5.8)–(5.13) to optimality. As mentioned before, for computing time reasons we decide to explore this neighborhood only in a partial way. Therefore we initialize RRM with a small subset of variables which ensure the model to be feasible, and then we solve its LP relaxation adding other variables by using column generation techniques. When no other variable with small (say) reduced cost can be added to the model, we "freeze" the current set of variables and we add the integrality requirements. The Partial Reduced Reallocation Model we get in this way corresponds to a partial reduced neighborhood which can be explored by solving the model through a general purpose MIP solver.

Let LP-RRM denote the Linear Programming relaxation of RRM. Then let $\mathcal{S}_B$ denote the set of all the *basic* sequences in $\mathcal{S}$ extracted from the incumbent solution z_0 (we say that $s \in \mathcal{S}$ is a *basic* sequence if it belongs to z_0 and no other sequence in $\mathcal{S}$ belonging to z_0 contains s). For each $s \in \mathcal{S}_B$ let $i_s \in \mathcal{I}$ denote the *pivot* insertion point the basic sequence s has been extracted from. Finally, let $\mathcal{V}_P$ and $\mathcal{S}_P$ denote the *variable pool* and the *sequence pool*, which contain, respectively, all the variables and all the sequences generated so far. With the above notation, the Neighborhood construction can be described in detail through the following steps:

1. (Initialization). Set $\mathcal{S}_P := \mathcal{S}_B$ and $\mathcal{V}_P := \{x_{si} : s \in \mathcal{S}_B, i = i_s\}$. For each insertion point $i \in \mathcal{I}$, construct a small number of new sequences $s \in \mathcal{S}_i$ that "fit well" with i (i.e., that have a small insertion cost with respect to i), add these sequences to $\mathcal{S}_P$ and the corresponding variables to $\mathcal{V}_P$. After this step, a feasible solution for RRM exists (e.g., the current solution z_0) and we can use reduced costs for evaluating the "goodness" of the other variables.

2. (Pricing). Apply a fast Pricing step with the sequences generated so far to reduce the computational effort required for the *Column generation* step:

   (a) add to $\mathcal{S}_P$ all the sequences $s \in \mathcal{S}$ with cardinality 1 and 2;

   (b) solve LP-RRM;

   (c) $\forall i \in \mathcal{I}$ and $\forall s \in \mathcal{S}_P \cap \mathcal{S}_i$, if the reduced cost rc_{si} of the variable x_{si} does not exceed a given threshold RC_{max} (i.e., $rc_{si} \leq RC_{max}$), then add x_{si} to $\mathcal{V}_P$;

   (d) if at least one variable has been added, repeat from Step 2b.

3. (Column generation). Look for "good" variables by (heuristically) solving the column generation problem associated with each insertion point in LP-RRM:

(a) for each $i \in \mathcal{I}$, try to solve the column generation problem associated with i finding as many variables with small reduced cost as possible;

(b) if at least one variable has been found, add all the variables to $\mathcal{V}_P$, solve LP-RRM and repeat from Step 3a.

The *Column generation* step represents the crucial step of the Neighborhood construction phase, and is described in Section 5.6.1. This is the most time consuming step, but it often allows to improve even good quality initial solutions. However, preliminary computational experiments showed that the *Initialization* and the *Pricing* steps are important as well, since they provide an initial nice structure for the Reduced Reallocation Model. If we initialize the variable pool as $\mathcal{V}_P := \{x_{si} : s \in \mathcal{S}_B, i = i_s\}$ and then apply only the *Column generation* step, we generally find the same final solution but in a much higher computing time, since in the first iterations of the *Column generation* step a huge number of useless variables are generated and added to the variable pool. Moreover, during all the steps of the Neighborhood construction phase, hashing techniques are used to handle the variable and sequence pools to avoid the generation of duplicated variables.

5.6.1 The column generation problem

Consider the Linear Programming relaxation LP-RRM of the Reduced Reallocation Model, and let π^1_v, π^2_i, π^3_r and π^4_r be the dual variables associated, respectively, with constraints (5.9), (5.10), (5.11) and (5.12) in LP-RRM, where $v \in \mathcal{F}$, $i \in \mathcal{I}$ and $r \in \mathcal{R}$. Then, for any $(s, i) \in \{(s, i) \in \mathcal{S} \times \mathcal{I} : i \in \mathcal{I}_s\}$, with $s = (v_1, \ldots, v_h)$ and $i = (a_i, b_i)$, the reduced cost rc_{si} of variable x_{si} is defined by

$$rc_{si} := \gamma_{si} - \sum_{v \in V(s)} \tilde{\pi}^1_v - \tilde{\pi}^2_i - q(s)\tilde{\pi}^3_{r_i} - \gamma_{si}\tilde{\pi}^4_{r_i} \tag{5.16}$$

where $\tilde{\pi} = (\tilde{\pi}^1_v, \tilde{\pi}^2_i, \tilde{\pi}^3_r, \tilde{\pi}^4_r)$ denotes the optimal dual solution of LP-RRM. Let $P(s, i) = (V_P, E_P)$ denote the path $(a_i, v_1, \ldots, v_h, b_i)$ in G corresponding to variable x_{si}, where $V_P := \{v_1, \ldots, v_h\} \subseteq V$ and $E_P := \{(a_i, v_1), \ldots, (v_h, b_i)\} \subseteq E$. We can rewrite the reduced cost of variable x_{si} as

$$rc_{si} := -\tilde{\pi}^2_i - c_i(1 - \tilde{\pi}^4_{r_i}) + \sum_{e \in E_P} c_e(1 - \tilde{\pi}^4_{r_i}) + \sum_{v \in V_P} -(\tilde{\pi}^1_v + q_v\tilde{\pi}^3_{r_i}) \tag{5.17}$$

where c_i denotes the cost of edge $i = (a_i, b_i)$. Now consider the graph $G(i, \tilde{\pi}) = (V_i, E_i)$, with $V_i := \{a_i, b_i\} \cup \mathcal{F}_i$ and $E_i := E(V_i) \setminus (a_i, b_i)$, where $E(V_i)$ denotes the set of edges in E having both endpoints in V_i. Associate with each edge $e \in E_i$ a cost $c'_e = c_e(1 - \tilde{\pi}^4_{r_i})$ and a weight c_e, and associate with each node $v \in \mathcal{F}_i$ a cost $q'_v = -(\tilde{\pi}^1_v + q_v\tilde{\pi}^3_{r_i})$ and a weight q_v. We say that a path $P = (V_P, E_P)$ from a_i to b_i in $G(i, \tilde{\pi})$ is a feasible path if

$$\sum_{v \in V_P} q_v \leq Q - \tilde{q}(r_i), \tag{5.18}$$

$$\sum_{e \in E_P} c_e \leq D - \tilde{c}(r_i) + c_i, \tag{5.19}$$

and its cost is

$$c'(P) = \sum_{e \in E_P} c'_e + \sum_{v \in V_P} q'_v. \tag{5.20}$$

With the above definitions the following proposition holds:

Proposition 5.1 *For any $i \in \mathcal{I}$, the column generation problem associated with i in LP-RRM is the problem of finding a feasible path P from a_i to b_i in $G(i, \tilde{\pi})$, with cost $c'(P) < \tilde{\pi}_i^2 + c_i(1 - \tilde{\pi}_{r_i}^4)$.*

Proof. For any $i \in \mathcal{I}$, the column generation problem associated with i in LP-RRM is the problem of finding a variable x_{si} with negative reduced cost, such that $q(s) \leq Q - \tilde{q}(r_i)$, $\gamma_{si} \leq D - \tilde{c}(r_i)$, and $s \in \mathcal{S}_i$. For any $s = (v_1, \ldots, v_h) \in \mathcal{S}_i$, let $P = (a_i, v_1, \ldots, v_h, b_i)$ be the corresponding path in $G(i, \tilde{\pi})$. x_{si} fulfills both capacity and distance constraints if and only if P is feasible. Moreover, the reduced cost of x_{si} is given by $r_{si} = c'(P) - \tilde{\pi}_i^2 - c_i(1 - \tilde{\pi}_{r_i}^4)$. □

As described above, the column generation problem for LP-RRM associated with any insertion point $i \in \mathcal{I}$ is in practice a Resource Constrained Elementary Shortest Path Problem (RCESPP) defined on a graph $G(i, \tilde{\pi})$, which also arises as column generation problem in the *classical* Set Partitioning formulation of VRP (see, e.g., Feillet et al. [76] and Righini and Salani [150]). It is worth noting that the size of $G(i, \tilde{\pi})$ and of the corresponding RCESPP strictly depend on $|\mathcal{F}_i|$. Indeed, the Neighborhood Reduction described in Section 5.5 corresponds to reducing the size of $G(i, \tilde{\pi})$ and hence of RCESPP for any insertion point $i \in \mathcal{I}$.

In the general case, $G(i, \tilde{\pi})$ contains negative cycles (i.e., cycles in which the sum of the costs c'_e associated with the edges and the costs q'_v associated with the nodes is negative): indeed, while dual variables $\pi_i^2, \pi_r^3, \pi_r^4$ are non positive, dual variables π_v^1 are free and usually assume positive values (note that they are always non negative if the triangle inequality holds). Positive values of variables π_v^1 usually lead to negative node costs q'_v and to negative cycles in graph $G(i, \tilde{\pi})$. Therefore, the column generation problem in LP-RRM is strongly $\mathcal{NP}$-hard.

Since the aim of the Neighborhood Construction phase is to find promising variables for the Reduced Reallocation Model in a short computing time, we solve the column generation problem through a simple heuristic, whose aim is to find as many variables with small reduced cost as possible. For any given graph $G(i, \tilde{\pi})$, the algorithm works as follows:

1. Find an initial feasible path $P = (a_i, v, b_i)$, with $v \in \mathcal{F}_i$ (such a path always exists if $\mathcal{F}_i \neq \emptyset$).

2. Evaluate all the 1-1 *feasible* exchanges between each node $w \in \mathcal{F}_i \setminus V_P$ and each node $v \in V_P$, and select the best one (with respect to the cost of the corresponding path); if this exchange leads to an improvement, perform it and repeat from Step 2.

3. Evaluate all the *feasible* insertions of each node $w \in \mathcal{F}_i \setminus V_P$ in each edge $(v_1, v_2) \in E_P$ and select the best one; if no feasible insertion exists, terminate; otherwise, force such an insertion even if it leads to a worse path and repeat from Step 2.

Whenever a new path is generated, the corresponding variable is added to the variable pool $\mathcal{V}_P$ if its reduced cost is smaller than a given threshold RC_{max} (say).

5.7 Computational results

The performance of the Local Search Algorithm (LSA) proposed in the previous sections was evaluated by considering two classes of experiments, corresponding to two different possibilities for finding the initial solutions to be possibly improved. In Class 1, the initial solution is obtained by means of the C code corresponding to the Granular Tabu Search algorithm proposed by Toth and Vigo [167]. In this way we get a self-contained algorithm which requires no initial solution to be given. In several cases, the Granular Tabu Search algorithm provides very good initial solutions, while in other cases it provides solutions which are quite far from the best-known ones reported in the literature. In Class 2, we start from an extremely-good feasible solution (in several cases, the best-known solution reported in the literature), with the aim of improving it (this is of course possible only if the initial solution is not optimal, as it is the case for some of them).

Our computational analysis considers two well known sets of Euclidean instances from the literature, that are generally used as standard benchmarks for the considered problem. The first set consists of the 14 instances (with $|V|$ varying from 51 to 200) proposed by Christofides, Mingozzi and Toth [47] (CMT instances). Both *real costs* and *rounded-integer costs* have been considered for these instances. The second set consists of the 20 large-scale instances (with $|V|$ varying from 201 to 484) proposed by Golden, Wasil, Kelly and Chao [97] (GWKC instances). Only *real costs* have been considered for the GWKC instances in the literature. For the second class of experiments we consider, in addition, instance E101-14u (also called E-n101-k14), with *rounded-integer costs*, from Vigo's web page [171], and the large-scale instance tai385, with *real costs*, from Taillard's web page [160]. All the instances but tai385 (with 386 nodes and 47 routes) are denoted as Tn-kx, according to the notation adopted by Toth and Vigo [166], where "T" is equal to "E" for VRP instances with capacity constraints only and "D" for VRP instances with both capacity and distance constraints, "n" and "k" indicate, respectively, the number of nodes and the maximum number of routes, and "x" refers to the paper where the instance was proposed.

Cordeau et al. [56] provide a computational comparison of recent VRP heuristics on the CMT and the GWKC instances with *real costs*. On the CMT instances, the

best solutions are obtained by Taillard [161] in 12 cases out of 14, Rochat and Taillard [151] for all the instances but E200-17c, Mester and Bräysy [125] for all the instances but D200-18c. On the GWKC instances, the best solutions are obtained by Mester and Bräysy [125] for all the 20 instances but D281-08k, D361-09k and E400-18k, Prins [145] for instance D281-08k, Pisinger and Ropke [141] for instance D361-09k and De Franceschi et al. [68] for instance E400-18k. If the two instance sets are considered together, the best performers in terms of accuracy and computing time are Mester and Bräysy [125] and Prins [145]. It should be noted that these two methods combine population search and local search approaches, thus allowing for a broad and deep exploration of the solution space. Results on VRP instances with *rounded-integer costs* are less reported in the papers considering heuristic methods. The best solutions on the CMT instances are obtained by Gendreau et al. [93], Xu and Kelly [175] and Wassan [172]. We refer the reader to [56, 125, 168] for a deeper analysis of the most effective heuristic and metaheuristic techniques proposed in the literature for VRP.

LSA has been tested on a Pentium M 1.86 GHz notebook with 1 GByte RAM, running under Microsoft Windows XP Operative System, and has been coded in $C++$ with Microsoft Visual $C++$ 6.0 compiler. The ILP solver used in the experiments is ILOG Cplex 10.0 [105] with a limit of 30,000 branching nodes. In the Reallocation step we provide to the ILP solver the feasible solution corresponding to the current incumbent VRP solution, where each basic sequence is just reallocated to its corresponding pivot insertion point. In this way the ILP solver can immediately initialize its own incumbent solution, so every subsequent update (if any) corresponds to an improved VRP solution (the run being interrupted as soon as the internal ILP lower bound gives no hope to find an improvement).

LSA setting depends on the parameters RC_{MAX}, p and λ, which are related to the neighborhood size. Although RC_{MAX} could be tuned considering the edge costs of the tested instances, we prefer to run all the experiments with a fixed value of RC_{MAX}, and we fix $RC_{MAX} = 1$. For the CMT instances we fix $p = 0.5$ and consider two different settings for λ, namely $\lambda = 1$ (*strong reduction*) and $\lambda = \infty$ (*feasibility reduction*). For the GWKC instances we fix $p = 0.3$ and $\lambda = 1$, in order to reduce the number of variables generated for the Reallocation Model (in some GWKC instances, if $\lambda = \infty$ we cannot handle the Reallocation Model, because of the excessive memory requirement).

Tables 5.1 and 5.2 report the results obtained by algorithm LSA for the CMT instances, with *rounded-integer cost* and *real costs*, respectively. The initial solutions are found by applying the Granular Tabu Search algorithm [167], and two different settings of λ are compared (i.e., $\lambda = \infty$ and $\lambda = 1$). The columns in the tables have the following meaning:

- *Prev. best* is the previously best known solution value from the literature; provable optimal values are marked by *;
- *Start* is the value of the initial solution;
- *Final* is the value of the solution found by LSA (the number of used routes is

given if it is smaller than the maximum one);

- *Time* is the total computing time (in seconds) required by LSA to terminate the search process.

For each instance, the best solution value reported in the table is represented in bold face. All the provably optimal solution values are obtained by Fukasawa et al. [92] for instances with *rounded-integer costs*, while instance E051-05e with *real costs* has been solved to optimality by Hadjiconstantinou et al. [103].

Table 5.1: Computational results for the 14 CMT instances with *rounded-integer costs*. Initial solutions obtained by means of the *C* code of Toth and Vigo [167].

			LSA ($\lambda = \infty$)		LSA ($\lambda = 1$)	
Instance	Prev. best	Start	Final	Time	Final	Time
D051-06c	**548**	551	**548**	15	**548**	10
D076-11c	**905**	915	**905**	146	**905**	133
D101-09c	**856**	858	**856**	66	**856**	52
D101-11c	**865**	866	**865**	506	**865**	67
D121-11c	1526	1536	**1524**	795	**1524**	670
D151-14c	1147	1162	**1146**	441	**1146**	281
D200-18c	**1392**	1422	**1392**	1826	**1392**	1001
E051-05e	* **521**	**521**	—	7	—	5
E076-10e	* **830**	836	832	66	832	57
E101-08e	* **815**	817	**815**	85	**815**	66
E101-10c	* **820**	**820**	—	56	—	20
E121-07c	* **1034**	1036	**1034**	95	**1034**	62
E151-12c	**1015**	1026	1024	1298	1024	367
E200-17c	1289	1304	(16)**1285**	3431	(16)**1285**	1959

The two tested configurations always provide the same final solution value, but *strong reduction* ($\lambda = 1$) outperforms *feasibility reduction* ($\lambda = \infty$) in terms of computing time. The results show that LSA is able to improve substantially the initial solution, even for the instances for which the Granular Tabu Search algorithm provides very good-quality solutions. Table 5.1 shows that, with *rounded-integer costs*, LSA improves the initial solution in 12 cases out of 14 (failing, of course, for instances E-051-05e and E101-10c, where the initial solution is proved to be optimal). In particular, in 7 cases out of 12, LSA is able to reach the best-known solution (which in 2 cases is optimal), and in 3 cases out of 12 it improves the previous best-known solution. Table 5.2 shows that, with *real costs*, LSA improves the initial solution in 10 cases out of 14 (i.e., whenever the initial solution is not the best-known one) and in 3 cases it is able to reach the best-known solution. Note that, for instance E200-17c, with both *rounded-integer costs* and *real costs*, LSA is able to find a solution with an empty route (i.e., it is able to

Table 5.2: Computational results for the 14 CMT instances with *real costs*. Initial solutions obtained by means of the *C* code of Toth and Vigo [167].

			LSA ($\lambda = \infty$)		LSA ($\lambda = 1$)	
Instance	Prev. best	Start	Final	Time	Final	Time
D051-06c	**555.43**	**555.43**	—	8	—	6
D076-11c	**909.68**	920.72	912.49	123	912.49	113
D101-09c	**865.94**	869.48	**865.94**	128	**865.94**	91
D101-11c	**866.37**	**866.37**	—	975	—	83
D121-11c	**1541.14**	1545.51	1543.73	646	1543.73	601
D151-14c	**1162.55**	1173.12	1164.12	453	1164.12	441
D200-18c	**1395.85**	1435.74	1403.71	6458	1403.71	3370
E051-05e	* **524.61**	**524.61**	—	8	—	6
E076-10e	**835.26**	838.60	835.32	53	835.32	37
E101-08e	**826.14**	828.56	**826.14**	81	**826.14**	77
E101-10c	**819.56**	**819.56**	—	66	—	41
E121-07c	**1042.11**	1042.87	**1042.11**	136	**1042.11**	92
E151-12c	**1028.42**	1033.21	1031.71	1157	1031.71	458
E200-17c	**1291.29**	1318.25	(16)1301.52	6202	(16)1301.52	2718

reduce the number of used routes from 17 to 16). In order to find feasible solutions for this instance using the maximum number of available routes, we ran LSA by associating an infinite cost with the empty routes (i.e., with the insertion points (0,0)). With *rounded-integer costs*, we found a solution of value 1288 in 3992 seconds, while, with *real costs*, we found a solution of value 1301.79 in 5420 seconds. Finally, we ran LSA with an unlimited number of iterations (i.e., when the list Θ of the available selection criteria is empty we restart from the INITIALIZATION step) and a time limit of 10 hours, looking for possible better solutions. With *rounded-integer costs*, LSA found a solution of value 1145 after 1141 seconds for instance D151-14c, and a solution of value 1378 after 11006 seconds for instance D200-18c. (Note that these solutions correspond to a further improvement on the corresponding previous best-known solutions.) With *real costs*, LSA found a solution of value 1162.99 after 1791 seconds for instance D151-14c, and a solution of value 1399.92 after 15676 seconds for instance D200-18c.

Table 5.3 reports the results obtained by algorithm LSA for the 20 large-scale GWKC instances. The columns have the same meaning as in the previous tables.

For these instances, the initial solutions found by the *C* code of Toth and Vigo [167] are quite far from the best-known solutions from the literature. The table shows that LSA always improves substantially the initial solutions, but it never reaches the best-known ones (although in some cases it reduces the number of used routes). This is probably due to the regular structure of these instances, which allows LSA to find only small improvements of the incumbent solution, and to the value of parameter p which

Table 5.3: Computational results for the 20 large-scale GWKC instances. Initial solutions obtained by means of the C code of Toth and Vigo [167].

			LSA ($\lambda = 1$)	
Instance	Prev. best	Start	Final	Time
D201-05k	**6460.98**	6697.53	6654.00	421
D241-10k	**5627.54**	5736.15	5728.91	373
D281-08k	**8412.80**	8963.32	(7)8535.35	1080
D321-10k	**8447.92**	8553.03	8459.73	1444
D361-09k	**10181.75**	10547.44	10276.81	4043
D401-10k	**11036.22**	11402.75	11115.95	2819
D441-11k	**11663.55**	12036.24	11847.68	3357
D481-12k	**13624.52**	14910.62	(10)13886.64	4921
E241-22k	**707.79**	711.07	709.23	5636
E253-27k	**859.11**	868.80	(26)862.84	25439
E256-14k	**583.39**	593.35	586.85	27257
E301-28k	**997.52**	1016.83	(27)1000.39	11495
E321-30k	**1081.31**	1096.18	1087.48	18330
E324-16k	**741.56**	751.66	743.35	50386
E361-33k	**1366.86**	1400.96	1372.38	68445
E397-34k	**1345.23**	1369.44	(33)1347.03	26492
E400-18k	**918.42**	936.04	927.96	86323
E421-41k	**1820.09**	1915.83	(38)1836.54	46219
E481-38k	**1622.69**	1652.32	(37)1623.52	80819
E484-19k	**1107.19**	1147.14	1132.75	58321

is fixed to 0.3 (i.e., at each iteration only 30% of the customers are extracted on average from the incumbent solution) to avoid an excessive memory requirement.

For the second class of experiments, in which we start from a good feasible solution taken from the literature, we selected the same initial solutions as in De Franceschi et al. [68]. In this way, we can compare LSA with the method proposed in [68]. Computational results for this class of experiments, concerning the 14 CMT instances with both *rounded-integer costs* and *real costs*, the 13 large-scale instances with only capacity constraints, and instance E101-14u, are reported in Table 5.4. The columns have the same meaning as in the previous tables. We also report the computational results taken from [68] (algorithm SERR), which were obtained on an AMD Athlon XP 2400+ PC with 1 GByte RAM, using ILOG Cplex 8.1 [105] as ILP solver. The computing times refer to machines with similar performance, and can be used for comparing the two approaches. In addition, we report the font of the initial solution we start from (*Source*), and, in the last column, all the new best-known solution values found by LSA (*New best*). Note that for instance E151-12c, with *rounded-integer costs*, the initial solution (taken from [160]) corresponds to instance M-n151-k12, which is the same as E151-12c with a different order of the nodes.

The table shows that LSA clearly outperforms the method proposed in [68] in terms of the quality of the solution found.[2] Moreover, in many cases, the computational effort for finding the final solution is strongly reduced. Finally, concerning the large-scale VRP instances, when starting from very good solutions, LSA is able to improve the best-known solution from the literature in 7 cases out of 13.

5.8 Conclusions

In this chapter we presented an Integer Linear Programming (ILP) Local Search algorithm for the classical Vehicle Routing Problem (VRP), based on an exponential neighborhood which is explored by solving an ILP formulation. We investigated the neighborhood structure and the column generation problem associated with the LP relaxation of the ILP formulation used for the neighborhood exploration. We showed that the column generation problem is $\mathcal{NP}$-hard, and we proposed a two-phase approach for an effective neighborhood exploration, which first reduces the neighborhood size through a simple heuristic procedure, and then explores the reduced neighborhood by solving the corresponding ILP problem through the (heuristic) solution of the column generation problem associated with its LP relaxation.

Computational results on 50 capacitated VRP instances from the literature (with/without distance constraints) showed that the proposed method can be used as a profitable tool for improving existing VRP solutions, and that even extremely-good quality solutions found by the most effective metaheuristic techniques proposed for VRP can be further improved. For 11 instances, the proposed method was able to improve the best-known

[2] In Table 5.4, the new best solution of value 1285, found by LSA for instance E-200-17c with *rounded-integer costs*, has an empty route. For the same instance LSA finds a solution of value 1288 with no empty routes.

Table 5.4: Comparison on benchmark instances with "good" CVRP/DCVRP initial solutions from the literature.

				SERR [68]		LSA ($\lambda = 1$)		
Instance	Prev. best	Start	Source	Final	Time	Final	Time	New best
D051-06c	**548**	**548**	[93]	—	—	—	5	—
D076-11c	**905**	907	[93]	**905**	178	**905**	30	—
D101-09c	**856**	**856**	[93]	—	—	—	48	—
D101-11c	**865**	866	[93]	**865**	1274	**865**	69	—
D121-11c	1526	1529	[93]	1526	26622	**1524**	337	**1524**
D151-14c	1147	1180	[93]	1161	44578	1146	734	**1145**
D200-18c	1392	1404	[93]	1398	4075	1385	1384	**1378**
E051-05e	* **521**	**521**	[93]	—	—	—	5	—
E076-10e	* **830**	832	[93]	831	279	831	25	—
E101-08e	* **815**	**815**	[93]	—	—	—	51	—
E101-10c	* **820**	824	[93]	**820**	18	**820**	38	—
E121-07c	* **1034**	1035	[93]	**1034**	89	**1034**	63	—
E151-12c	**1015**	1016	[160]	**1015**	377	**1015**	283	—
E200-17c	1289	1316	[93]	1307	48488	1292	5940	**1285**
D051-06c	**555.43**	**555.43**	[93]	—	—	—	6	—
D076-11c	**909.68**	913.23	[93]	—	—	911.76	40	—
D101-09c	**865.94**	**865.94**	[93]	—	—	—	60	—
D101-11c	**866.37**	**866.37**	[93]	—	—	—	83	—
D121-11c	**1541.14**	1551.63	[93]	1546.10	232466	1545.56	871	—
D151-14c	**1162.55**	1189.79	[93]	1178.02	7431	1164.55	915	—
D200-18c	**1395.85**	1421.88	[93]	1416.47	42262	1406.47	2759	—
E051-05e	* **524.61**	**524.61**	[93]	—	—	—	6	—
E076-10e	**835.26**	836.37	[93]	**835.26**	381	**835.26**	34	—
E101-08e	**826.14**	**826.14**	[93]	—	—	—	60	—
E101-10c	**819.56**	822.85	[93]	**819.56**	20	**819.56**	45	—
E121-07c	**1042.11**	1043.94	[93]	1043.42	115	**1042.11**	94	—
E151-12c	**1028.42**	1034.90	[93]	1034.50	397	1031.07	1023	—
E200-17c	**1291.29**	1311.35	[94]	1305.35	18386	1301.79	1686	—
E241-22k	**707.79**	**707.79**	[124]	—	—	—	5225	—
E253-27k	**859.11**	(26)**859.11**	[124]	—	—	—	4259	—
E256-14k	583.39	583.39	[124]	—	—	**582.64**	19903	**582.64**
E301-28k	**997.52**	(27)998.73	[124]	—	—	(27)998.69	9440	—
E321-30k	**1081.31**	**1081.31**	[124]	—	—	—	7194	—
E324-16k	741.56	742.04	[124]	741.70	61662	**739.53**	45516	**739.53**
E361-33k	1366.86	1366.86	[124]	—	—	**1366.54**	12717	**1366.54**
E397-34k	1345.23	(33)1345.23	[124]	—	—	(33)**1343.47**	31629	**1343.47**
E400-18k	918.42	918.45	[124]	918.42	5585	**916.62**	63207	**916.62**
E421-41k	**1820.09**	(38)1821.15	[124]	—	—	(38)1820.94	15721	—
E481-38k	1622.69	(37)1622.69	[124]	—	—	(37)**1622.39**	36867	**1622.39**
E484-19k	**1107.19**	**1107.19**	[124]	—	—	—	23792	—
E101-14u	* **1067**	1076	[171]	**1067**	2866	**1067**	139	—
tai385	24422.50	24435.50	[160]	24422.50	152287	**24421.11**	8271	**24421.11**

solution reported in the literature.

The presented method follows a destroy-and-repair paradigm, where the current solution is randomly destroyed (i.e., customers are removed in a random way) and repaired by following ILP techniques. Hence, the overall procedure can be considered as a general framework which can be extended to cover other variants of Vehicle Routing Problems.

Chapter 6

Integer Linear Programming Local Search for the Open Vehicle Routing Problem

6.1 Introduction

In Chapter 5 we presented an Integer Linear Programming (ILP) Local Search algorithm for the "classical" (Capacitated and Distance Constrained) Vehicle Routing Problem (VRP). As already discussed, the overall procedure follows a destroy-and-repair paradigm (i.e., the current solution is first randomly destroyed and then repaired by following ILP techniques) which can be considered as a general framework and can be extended to cover other variants of Vehicle Routing Problems.

Such a framework is extended in this chapter[1] to the Open Vehicle Routing Problem (OVRP), thus showing the flexibility and the effectiveness of the proposed technique. OVRP is a variant of the classical VRP in which the vehicles are not required to return to the depot after completing their service. Clearly, most of the theoretical and practical considerations developed in the previous chapter for VRP still apply to OVRP. However, as discussed in the following, this apparently minor change in the problem definition leads in a sense to a more general problem, thus requiring to extend the framework in order to capture the different structure of the addressed problem. Further, the main ingredients of the Local Search algorithm (i.e., node selection criteria, neighborhood definition, termination conditions) are slightly modified in this chapter with the aim of providing an overall simpler and more general procedure, less dependent on small implementation details. Hence, to avoid confusion between this chapter and the previous one, we decided to represent the overall procedure, pointing out the main differences with respect to the method presented in Chapter 5 for VRP.

The chapter is organized as follows. In Section 6.2 OVRP is formally stated and

[1]The results of this chapter appear in: M. Salari, P. Toth and A. Tramontani, "An ILP improvement procedure for the open vehicle routing problem", Technical Report OR-08-06, DEIS, University of Bologna, 2008 [153].

the main works proposed in the literature for the problem are discussed. In Section 6.3 we describe the neighborhood proposed for OVRP and the ILP formulation used to implicitly define and explore it. The implementation of the Local Search algorithm is given in Section 6.4, while Section 6.5 reports the computational experiments on benchmark capacitated OVRP instances from the literature (with/without distance constraints), comparing the presented method with the most effective metaheuristic techniques proposed for OVRP. Some conclusions are finally drawn in Section 6.6.

6.2 Problem statement and literature review

The Open Vehicle Routing Problem (OVRP) is a variant of the classical (Capacitated and Distance Constrained) Vehicle Routing Problem (VRP) in which the vehicles are not required to return to the depot after completing their service. OVRP can be formally stated as follows. We are given a central *depot* and a set of n *customers*, which are associated with the nodes of a complete undirected graph $G = (V, E)$ (where $V = \{0, 1, \ldots, n\}$, node 0 represents the depot and $V \setminus \{0\}$ is the set of customers). Each edge $e \in E$ has an associated finite *cost* $c_e \geq 0$ and each customer $v \in V \setminus \{0\}$ has a *demand* $q_v \geq 0$ (with $q_0 = 0$ for the depot node). A fleet of m identical *vehicles* is located at the depot, each one with a *fixed cost* F, a *capacity* Q and a *total distance-traveled (duration)* limit D. The customers must be served by at most m paths (*open routes*), each path associated with one vehicle, starting at the depot and ending at one of the customers. Each route must have a duration (computed as the sum of the edge costs in the route) not exceeding the given limit D of the vehicles, and can visit a subset S of customers whose total demand $\sum_{v \in S} q_v$ does not exceed the given capacity Q. The problem consists of finding a feasible solution covering (i.e., visiting) exactly once all the customers and having a minimum overall cost, computed as the sum of the traveled edge costs plus the fixed costs associated with the vehicles used to serve the customers.

OVRP is known to be $\mathcal{NP}$-hard in the strong sense, as it generalizes the Bin Packing Problem and the Hamiltonian Path Problem. At first glance, having open routes instead of closed ones looks like a minor change with respect to the classical VRP, and in fact OVRP can be also formulated as a VRP on a directed graph, by fixing to 0 the cost of each arc entering the depot. However, if the undirected case is considered, the open version turns out to be more general than the closed one. Indeed, as shown by Letchford et al. [113], any closed VRP on n customers in a complete undirected graph can be transformed into an OVRP on n customers, but there is no transformation in the reverse direction. Further, there are many practical applications in which OVRP naturally arises. This happens, of course, when a company does not own a vehicle fleet, and hence customers are served by hired vehicles which are not required to come back to the depot (see, e.g., Tarantilis et al. [165]). But the *open model* also arises in pick-up and delivery applications, where each vehicle starts at the depot, delivers to a set of customers and then it is required to visit the same customers in reverse order, picking up items that have to be backhauled to the depot. An application of this type is described in Schrage [158]. Further areas of application, involving the planning of train services and of school bus routes, are reported by Fu et al. [90].

OVRP has recently received an increasing attention in the literature. Exact branch-and-cut and branch-cut-and-price approaches have been proposed, respectively, by Letchford et al. [113] and Pessoa et al. [140], addressing the capacitated problem with no distance constraints and no empty routes allowed (i.e., $D = \infty$ and exactly m vehicles must be used). Heuristic and metaheuristic algorithms usually take into account both capacity and distance constraints, and consider the number of routes as a decision variable. In particular, an unlimited number of vehicles is supposed to be available (i.e., $m = \infty$) and the objective function is generally to minimize the number of used vehicles first and the traveling cost second, assuming that the fixed cost of an additional vehicle always exceeds any traveling cost that could be saved by its use (i.e., considering $F = \infty$). However, several authors address as well the variant in which there are no fixed costs associated with the vehicles (i.e., $F = 0$) and hence the objective function is to minimize the total traveling cost with no attention to the number of used vehicles (see, e.g., Tarantilis et al. [165]). Considering capacity constraints only (i.e., taking $D = \infty$), Sariklis and Powell [154] propose a two-phase heuristic which first assigns customers to clusters and then builds a Hamiltonian path for each cluster, Tarantilis et al. [163] describe a population-based heuristic, while Tarantilis et al. [164, 165] present threshold accepting metaheuristics. Taking into account both capacity and distance constraints, Brandão [40], Fu et al. [90, 91] and Derigs and Reuter [69] propose tabu search heuristics, Li et al. [115] describe a record-to-record travel heuristic, Pisinger and Ropke [141] present an adaptive large neighborhood search heuristic which follows a destroy-and-repair paradigm, while Fleszar et al. [88] propose a variable neighborhood search heuristic.

6.3 Reallocation Model

Let z be a feasible solution of the OVRP defined on G. For any given node subset $\mathcal{F} \subset V \setminus \{0\}$, we define $z(\mathcal{F})$ as the *restricted solution* obtained from z by *extracting* (i.e., by short-cutting) all the nodes $v \in \mathcal{F}$. Let $\mathcal{R}$ be the set of routes in the restricted solution, $\mathcal{I} = \mathcal{I}(z, \mathcal{F})$ the set of all the edges in $z(\mathcal{F})$, and $\mathcal{S} = \mathcal{S}(\mathcal{F})$ the set of all the *sequences* which can be obtained through the recombination of nodes in $\mathcal{F}$ (i.e., the set of all the elementary paths in $\mathcal{F}$). Each edge $i \in \mathcal{I}$ is viewed as a potential *insertion point* which can allocate one or more nodes in $\mathcal{F}$ through at most one sequence $s \in \mathcal{S}$. We say that the insertion point $i = (a, b) \in \mathcal{I}$ allocates the nodes $\{v_j \in \mathcal{F} : j = 1, \ldots, h\}$ through the sequence $s = (v_1, v_2, \ldots, v_h) \in \mathcal{S}$, if the edge (a, b) in the restricted solution is replaced by the edges $(a, v_1), (v_1, v_2), \ldots, (v_h, b)$ in the new feasible solution. Since the restricted routes, as well as the final ones, are open paths starting at the depot, in addition to the edges of the restricted solution we also consider the insertion points (called *appending insertion points* in the following) $i = (p_r, 0)$, where p_r denotes the last customer visited by route $r \in \mathcal{R}$, which allow to append any sequence to the last customer of any restricted route. Further, empty routes in the restricted solution are associated with insertion points $(0, 0)$.

For each sequence $s \in \mathcal{S}$, $c(s)$ and $q(s)$ denote, respectively, the cost of the elemen-

tary path corresponding to s and the sum of the demands of the nodes in s. For each insertion point $i = (a, b) \in \mathcal{I}$ and for each sequence $s = (v_1, v_2, \ldots, v_h) \in \mathcal{S}$, γ_{si} denotes the extra-cost (i.e., the extra-distance) for assigning sequence s to insertion point i in its best possible orientation (i.e., $\gamma_{si} := c(s) - c_{ab} + \min\{c_{av_1} + c_{v_h b}, c_{av_h} + c_{v_1 b}\}$). Note that, for the appending insertion points $i = (p_r, 0)$, γ_{si} is computed as $c(s) + \min\{c_{p_r v_1}, c_{p_r v_h}\}$. The extra-cost for assigning the sequence s to the insertion point $i = (0, 0)$ associated with an empty route is simply $c(s) + \min\{c_{0v_1}, c_{0v_h}\}$. For each route $r \in \mathcal{R}$, $\mathcal{I}(r)$ denotes the set of insertion points associated with r, while $\tilde{q}(r)$ and $\tilde{c}(r)$ denote, respectively, the total demand and the total distance computed for route r, still in the restricted solution.

For each $i \in \mathcal{I}$, suppose to define a subset $\mathcal{S}_i \subseteq \mathcal{S}$ of all the available sequences, containing only the sequences which can be allocated to the specific insertion point i (the definition of $\mathcal{S}_i$ will be discussed later in this section). Then, a neighborhood of the given solution z can be formulated (and explored) by solving an ILP problem (denoted as the *Reallocation Model*) based on the decision variables

$$x_{si} = \begin{cases} 1 & \text{if sequence } s \in \mathcal{S}_i \text{ is allocated to insertion point } i \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases}, \tag{6.1}$$

which reads as follows:

$$\sum_{r \in \mathcal{R}} \tilde{c}(r) + \min \sum_{i \in \mathcal{I}} \sum_{s \in \mathcal{S}_i} \gamma_{si} x_{si} \tag{6.2}$$

subject to:

$$\sum_{i \in \mathcal{I}} \sum_{s \in \mathcal{S}_i(v)} x_{si} = 1 \qquad v \in \mathcal{F} \ , \tag{6.3}$$

$$\sum_{s \in \mathcal{S}_i} x_{si} \leq 1 \qquad i \in \mathcal{I} \ , \tag{6.4}$$

$$\sum_{i \in \mathcal{I}(r)} \sum_{s \in \mathcal{S}_i} q(s) x_{si} \leq Q - \tilde{q}(r) \qquad r \in \mathcal{R} \ , \tag{6.5}$$

$$\sum_{i \in \mathcal{I}(r)} \sum_{s \in \mathcal{S}_i} \gamma_{si} x_{si} \leq D - \tilde{c}(r) \qquad r \in \mathcal{R} \ , \tag{6.6}$$

$$x_{si} \in \{0, 1\} \qquad i \in \mathcal{I}, \ \ s \in \mathcal{S}_i \ , \tag{6.7}$$

where, for any $i \in \mathcal{I}$ and $v \in \mathcal{F}$, $\mathcal{S}_i(v) \subseteq \mathcal{S}_i$ denotes the set of sequences covering customer v which can be allocated to insertion point i. The objective function (6.2), to be minimized, gives the traveling cost of the final OVRP solution. Constraints (6.3) impose that each extracted node belongs to exactly one of the selected sequences, i.e., that it is covered exactly once in the final solution. Constraints (6.4) avoid to allocate two or more sequences to the same insertion point. Finally, constraints (6.5) and (6.6) impose that each route in the final solution fulfills the capacity and distance restrictions, respectively. Note that, if there is a non-null fixed cost F associated with the vehicles, it can be taken into account by simply adding F to the cost of the edges incident at the

depot node.

The Reallocation Model (6.2)–(6.7) defines a neighborhood of a given solution z which depends on the extracted nodes $\mathcal{F}$ and on the subsets $\mathcal{S}_i$ ($i \in \mathcal{I}$). As discussed in the previous chapter, for any given $\mathcal{F}$, the choice of $\mathcal{S}_i$ is a key factor in order to allow an effective exploration of the solution space in the neighborhood of the given solution. The subsets $\mathcal{S}_i$ are built by following a two-phase approach. First, for each $i \in \mathcal{I}$, we define a subset $\mathcal{F}_i \subseteq \mathcal{F}$ of the extracted nodes, containing only the nodes which can be allocated through some sequences to the specific insertion point i. Then, we build the subsets $\mathcal{S}_i$ by iteratively solving the column generation problem associated with the Linear Programming (LP) relaxation of the Reallocation Model (LP-RM), by generating, for each specific insertion point $i \in \mathcal{I}$, variables x_{si} with *small* reduced cost obtained by considering only the extracted nodes in $\mathcal{F}_i$ (i.e., variables x_{si} such that all the nodes in s belongs to the set $\mathcal{F}_i$). Hashing techniques are used to avoid the generation of duplicated variables.

6.3.1 Building the Reallocation Model

The Reallocation Model (6.2)–(6.7) defined in the previous section is built through the following steps.

1. (Reduction). For each $i \in \mathcal{I}$, build a subset $\mathcal{F}_i \subseteq \mathcal{F}$ containing only the extracted nodes which can be allocated to the specific insertion point i. Subsets $\mathcal{F}_i$ are built by following the parametric procedure defined in the previous chapter, where the parameter λ is fixed to 1 (see Chapter 5, Section 5.5). In particular, any node $v \in \mathcal{F}$ is associated with at least one insertion point $i \in \mathcal{I}(r)$ for each restricted route $r \in \mathcal{R}$, and with its *pivot* i_v (i.e., the insertion point v was extracted from).

2. (Initialization). For each insertion point $i = (a_i, b_i) \in \mathcal{I}$, initialize subset $\mathcal{S}_i$ with the *basic* sequence extracted from i (i.e., the, possibly empty, sequence of nodes connecting node a_i and b_i in the given solution z) plus the feasible singleton sequence with the minimum insertion cost (i.e., the sequence (v), with $v \in \mathcal{F}_i$, with the minimum extra-cost among all the singleton sequences which can be allocated to i without violating the capacity and distance restrictions for the restricted route containing i). Initialize LP-RM with the initial set of variables corresponding to the current subsets $\mathcal{S}_i$, and solve LP-RM. (Note that, with such an initialization, each subset $\mathcal{S}_i$ contains the *basic* sequence extracted from insertion point i, and hence the current solution can always be obtained as a new feasible solution of the Reallocation Model.)

3. (Pricing). Apply a fast Pricing step with the sequences generated so far to reduce the computational effort required for the column generation step:

   (a) $\forall i \in \mathcal{I}$ and $\forall v \in \mathcal{F}_i$, consider the singleton sequence $s = (v)$. If the reduced cost rc_{si} of the variable x_{si} does not exceed a given threshold RC_{max} (i.e., $rc_{si} \leq RC_{max}$), then add x_{si} to LP-RM;

(b) if at least one variable has been added, solve LP-RM and repeat from Step 3a.

4. (Column generation). Look for "good" variables by (heuristically) solving the column generation problem associated with each insertion point in LP-RM:

   (a) for each $i \in \mathcal{I}$, solve the column generation problem associated with i and $\mathcal{F}_i$ (i.e., by considering only the sequences s of nodes belonging to $\mathcal{F}_i$), finding as many variables with small reduced cost as possible;

   (b) add to LP-RM all the variables x_{si} with $rc_{si} \leq RC_{max}$ found in the previous step. If at least one variable has been added, solve LP-RM and repeat from Step 4a.

For any fixed insertion point $i \in \mathcal{I}$, the column generation problem associated with i and $\mathcal{F}_i$ in LP-RM is exactly the same Resource Constrained Elementary Shortest Path Problem (RCESPP) described in the previous chapter, and it is solved through the same greedy heuristic (see Chapter 5, Section 5.6.1), with the aim of finding as many variables with small reduced cost as possible. However, while in the classical VRP context the RCESPP associated with all insertion points $i \in \mathcal{I}$ can be defined on a undirected graph $G(i, \tilde{\pi})$, in the OVRP context the appending insertion points $i = (p_r, 0)$, $r \in \mathcal{R}$, (i.e., the insertion points which allow to append a sequence to the end of a restricted route), need to be addressed on a mixed graph. Indeed, if the considered $i = (a_i, b_i) \in \mathcal{I}$ is an appending insertion point (i.e., b_i is the depot node), the column generation problem must be addressed on a mixed graph, where the edges incident at the depot are replaced by directed arcs (of different cost and weight) entering and leaving the depot.

6.4 Local Search Algorithm

The Reallocation Model described in the previous section allows for exploring a neighborhood of a given feasible solution, depending on the choice of the extracted customers in $\mathcal{F}$. We propose a Local Search algorithm for OVRP, based on model (6.2)–(6.7), which iteratively explores different neighborhoods of the current solution. Given an initial feasible solution z_0 for OVRP (taken from the literature or found by any heuristic method), the procedure works as follows.

1. (Initialization). Set $kt := 0$ and $kp := 0$. Take z_0 as the incumbent solution and initialize the current solution z_c as $z_c := z_0$.

2. (Node selection). Build set $\mathcal{F}$ by selecting each customer with a probability p.

3. (Node extraction). Extract the nodes selected in the previous step from the current solution z_c and construct the corresponding restricted OVRP solution $z_c(\mathcal{F})$, obtained by short-cutting the extracted nodes.

4. (Reallocation). Define the subsets $\mathcal{S}_i$ ($i \in \mathcal{I}(z_c, \mathcal{F})$) as described in Section 6.3. Build the corresponding Reallocation Model (6.2)–(6.7) and solve the model

by using a general-purpose ILP solver. Once an optimal ILP solution has been found, construct the corresponding new OVRP solution and possibly update z_c and z_0.

5. (TERMINATION). Set $kt := kt + 1$. If $kt = KT_{max}$, terminate.

6. (PERTURBATION). If z_c has been improved in the last iteration, set $kp := 0$; otherwise set $kp := kp + 1$. If $kp = KP_{max}$, "perturb" the current solution z_c and set $kp := 0$. In any case, repeat Step 2.

The procedure performs KT_{max} iterations and at each iteration explores a randomly generated neighborhood of the current solution z_c. However, if z_c is not improved for KP_{max} consecutive iterations, we introduce a random perturbation (see Step 6) in order to move to a different area of the solution space, so as to enforce the diversification of the search. In particular, when performing a *Perturbation* step, we randomly extract np customers from z_c (with np uniformly randomly chosen in $[np_{min}, np_{max}]$ and with each customer having the same probability to be extracted), and reinsert each extracted customer, in turn, in its best feasible position. If a customer cannot be inserted in any currently non-empty route (due to the capacity and/or distance restrictions), a new route is created to allocate the customer. In general, when performing the *Perturbation* step, several customers cannot be inserted in the non-empty routes of the current solution, and hence the new perturbed solution can use more vehicles than the current one.

6.5 Computational Results

The performance of the Local Search Algorithm (LSA) described in the previous sections was evaluated on the 16 benchmark instances usually addressed in the literature, taken from Christofides et al. [47] (instances `C1`–`C14`)[2] and from Fisher [87] (instances `F11`–`F12`), and on the 8 large scale benchmark instances proposed by Li et al. [115], and also addressed by Derigs and Reuter [69] (instances `O1`–`O8`). The number of customers of `C1`–`C14` and `F11`–`F12` ranges from 50 to 199. `C1`–`C5`, `C11`–`C12` and `F11`–`F12` have only capacity constraints, while `C6`–`C10` and `C13`–`C14` are the same instances as `C1`–`C5` and `C11`–`C12`, respectively, but with both capacity and distance constraints. Instances `O1`–`O8` have no distance restrictions and a number of customers varying from 200 to 480. As usual, for the problems with distance constraints, the route duration limit D is taken as the original value for the classical VRP multiplied by 0.9.

LSA needs an initial solution to be given, which in principle could be computed through any available constructive heuristic algorithm. We decided to run LSA starting from an extremely-good feasible solution available from the literature (in several cases, the best-known solution reported in the literature), with the aim of attempting to

[2]Instances `C1`–`C14` are exactly the same instances addressed also in the classical VRP context and denoted in the previous chapter as *CMT instances* (see Chapter 5, Section 5.7). In this chapter these instances are named in a different way (i.e., instances `C1`–`C14`), according to the usual notation of the other works proposed in the literature for OVRP.

improve it (this is of course impossible if the initial solution is provably optimal, as it is the case for some of them). In particular, we considered as initial solutions the ones obtained by Derigs and Reuter [69], Fu et al. [90, 91], Fleszar et al. [88] and Pisinger and Ropke [141].

LSA has been tested on a Pentium IV 3.4 GHz with 1 GByte RAM, running under Microsoft Windows XP Operative System, and has been coded in `C++` with Microsoft Visual `C++` 6.0 compiler. The ILP solver used in the experiments is ILOG CPLEX 10.0 [105]. LSA setting depends on the parameters RC_{max}, p, np_{min}, np_{max}, and on the number of iterations KP_{max} and KT_{max}. Although these parameters could be tuned considering the edge costs and the particular characteristics of the tested instances, we preferred to run all the experiments with a fixed set of parameters: $RC_{max} = 1$, $p = 0.5$ (i.e., 50% of the customers are selected on average), $np_{min} = 15$, $np_{max} = 25$, $KP_{max} = 50$ and $KT_{max} = 5,000$ (i.e., we perform globally 5,000 iterations, and the current solution is perturbed if it cannot be improved for 50 consecutive iterations). Finally, since several authors address the problem considering as objective function the minimization of the number of vehicles first and of the traveling cost second (i.e., assuming $F = \infty$), while other authors considered as objective function the minimization of the traveling cost (i.e., $F = 0$), we decided to run LSA without allowing to change the number of vehicles used in the initial solution. However, as stated in Section 6.4, the Perturbation step often requires additional routes to be created (to preserve the feasibility of the solution). In such cases, we add a small penalty θ to the cost of the edges incident at the depot, in order to force LSA to "recover" the solution in the following iterations. After some preliminary tests, we decided to fix $\theta = 12$ for the considered instances.

The computational results are reported in Tables 6.1–6.3. All the CPU times are expressed in seconds, and all the solution costs have been computed in double precision. Table 6.1 reports the computational results on the 16 instances `C1`–`C14` and `F11`–`F12` obtained by starting from the solutions provided by Fu, Eglese and Li and obtained through the algorithm proposed in [90]. In some cases, several solutions are provided for the same instance, obtained by using slightly different versions of their algorithm, with the same number of routes and different traveling cost. Among the different solutions for the same instance, we considered as initial solution for LSA the best one provided. The first column of the table gives the instance name. Columns 2–3 report the number of used vehicles (m) and the traveling cost of the initial solution (*cost*), while column 4 reports the CPU time of the corresponding algorithm (*time*), run on a Pentium IV 3 GHz. (CPU times have been taken from [91]. However, the cost of the initial solution for instance `C8` is better than the ones reported in [91], and hence for this initial solution we did not report the corresponding computing time.) Columns 5–7 report the computational results provided by LSA. For each instance, we report the final solution cost (*cost*), the CPU time required to reach the final solution (*b.time*) and the overall computing time required to perform all the 5,000 iterations (*t.time*). The last column reports the cost of the best known solution using the same number of vehicles as the initial solution. When LSA was not able to improve on the initial solution, we mark with a "—" the final solution cost and the corresponding *b.time*. Final solution

costs equal to the previously best known ones are underlined, new best solutions are in bold face, while provably optimal solutions, taken from Letchford et al. [113], are marked with an * (LSA was not run when the initial solution is provably optimal).

Table 6.2 reports the computational results on the same instances by starting from the best available solutions among the ones obtained by Derigs and Reuter [69], Fu et al. [90, 91], Fleszar et al. [88] and Pisinger and Ropke [141]. The table has the same structure as Table 6.1, but column 4 reports the source of the initial solution used in the experiments. For instances `C5`, `C7`, `C9`, `C13` and `C14`, the best available solutions for the case $F = \infty$ and the case $F = 0$ are different. In such cases, we considered both the solutions as initial solutions for LSA.

Finally, Table 6.3 reports the computational results on the 8 large scale instances `O1`–`O8` by starting from the solutions provided by Derigs and Reuter [69]. The table has the same structure as Table 6.1, but the CPU time related to the initial solution (column 4) was obtained on a Pentium IV 2.8 GHz.

Table 6.1: Computational results on the "classical" 16 benchmark instances starting from the solutions by Fu et al. [90, 91].

Instance	Initial solution			LSA			Prev. best sol.
	m	cost	time	cost	b.time	t.time	cost
C1	5	*416.06	0.8				*416.06
C2	10	567.14	7.8	—	—	84.2	567.14
C3	8	641.88	23.2	*639.74	106.0	119.9	*639.74
C4	12	738.94	6.8	733.13	21.2	156.6	733.13
C5	17	878.95	61.9	**868.81**	10.3	220.3	869.25
C6	6	412.96	0.6	—	—	45.1	412.96
C7	11	568.49	6.0	—	—	83.1	568.49
C8	9	646.31		644.63	0.1	136.2	644.63
C9	14	761.28	46.6	756.14	102.7	255.5	756.14
C10	17	903.10	51.9	878.54	323.9	460.2	875.07
C11	7	717.15	23.1	683.64	165.8	198.8	682.12
C12	10	534.71	4.2	*534.24	1.6	94.0	*534.24
C13	12	917.90	82.1	**894.19**	475.0	1165.3	896.50
C14	11	600.66	2.5	591.87	293.8	354.7	591.87
F11	4	*177.00	0.4				*177.00
F12	7	777.07	28.4	**769.55**	77.8	148.2	769.66
	Pentium IV 3 GHz			Pentium IV 3.4 GHz			

The tables show that LSA is able to improve even extremely-good quality solutions, obtained by some of the most effective metaheuristic techniques proposed for OVRP. It is worth noting that the solutions and the CPU times provided by Fu et al. [90, 91] (i.e., columns 2–4 of Table 6.1) are the best ones from among 20 runs of the corresponding randomized algorithm with different seeds. Hence, taking into account the different performance of the processors used for testing the different algorithms, the overall com-

Table 6.2: Computational results on the "classical" 16 benchmark instances starting from the best available solutions.

Instance	Initial solution			LSA			Prev. best sol.
	m	cost	source	cost	b.time	t.time	cost
C1	5	*416.06	[88, 91, 141]				*416.06
C2	10	567.14	[88, 91, 141]	—	—	84.2	567.14
C3	8	*639.74	[88]				*639.74
C4	12	733.13	[88, 141]	—	—	151.2	733.13
C5	16	896.08	[141]	892.37	276.5	450.2	879.37
C5	17	869.24	[69]	**868.93**	130.2	275.2	869.24
C6	6	412.96	[88, 91, 141]	—	—	45.1	412.96
C7	10	583.19	[141]	—	—	80.6	583.19
C7	11	568.49	[91]	—	—	83.1	568.49
C8	9	644.63	[88]	—	—	136.8	644.63
C9	13	757.84	[141]	**757.73**	33.9	412.5	757.84
C9	14	756.14	[69]	—	—	243.4	756.14
C10	17	875.07	[69]	**874.71**	2.6	454.7	875.07
C11	7	682.12	[88, 141]	—	—	178.3	682.12
C12	10	*534.24	[88, 141]				*534.24
C13	11	904.04	[88]	**899.16**	6.3	959.3	904.04
C13	12	917.90	[91]	**894.19**	475.0	1165.3	896.50
C14	11	591.87	[88, 141]	—	—	276.3	591.87
C14	12	581.81	[69]	—	—	364.2	581.81
F11	4	*177.00	[91, 141]				*177.00
F12	7	769.66	[88]	**769.55**	56.6	142.2	769.66

Table 6.3: Computational results on the 8 large scale benchmark instances starting from the solutions by Derigs and Reuter [69].

Instance	Initial solution			LSA			Prev. best sol.
	m	cost	time	cost	b.time	t.time	cost
O1	5	6018.52	467.0	—	—	182.2	6018.52
O2	9	4584.69	467.0	**4573.53**	34.6	284.0	4584.55
O3	7	7731.46	4047.0	—	—	304.6	7731.46
O4	10	7260.59	927.0	**7259.81**	34.4	438.9	7260.59
O5	9	9167.19	1186.0	**9165.40**	41.4	499.6	9167.19
O6	9	9805.45	1231.0	—	—	581.3	9803.80
O7	10	10348.57	3190.0	**10344.37**	8.6	653.0	10348.57
O8	10	12420.16	1969.0	—	—	623.6	12420.16
	Pentium IV 2.8 GHz			Pentium IV 3.4 GHz			

puting time required by LSA is comparable with the others reported in the tables, and in several cases the final improved solution is found very quickly. Our test-bed concerns in practice 35 different, non provably optimal, initial solutions which could be possibly improved, corresponding to 22 different instances. Out of these 35 solutions, LSA improves on the initial solution in 21 cases. For these cases, LSA reaches 6 times the previously best known solution (provably optimal in 2 cases), while finds 12 times a new best solution. Considering the 14 initial solutions which LSA does not improve, it is worth noting that 13 of them are the best solutions known in the literature (for the case $F = \infty$ or $F = 0$).

In order to look for possible better solutions, we performed some additional experiments. In particular, after the first 5,000 iterations, we ran LSA for 2,000 more iterations with a slightly different parameter setting. Starting from the solutions provided by Fu et al. [91], for instance `C5` with 17 vehicles LSA found a solution of cost 868.44 after 5220 iterations and 237.4 seconds, for instance `C10` a solution of cost 878.52 after 5151 iterations and 483.1 seconds, and for instance `C11` a solution of cost 683.15 after 6371 iterations and 380.1 seconds. Starting from the solutions provided by Derigs and Reuter [69], for instance `O4` LSA found a solution of cost 7251.74 after 6616 iterations and 869.4 seconds, and for instance `O5` a solution of cost 9162.93 after 6293 iterations and 716.2 seconds. (Note that the solutions obtained for instances `C5`, `O4` and `O5` correspond to a further improvement on the previous best-known solutions.)

Finally, still starting from the solutions by Fu et al. [91], we ran LSA with a different tuning of parameter p, to investigate how the neighborhood size affects the overall performance of the method, both in terms of quality of the solutions found and of CPU time. Let $z_{avg}(\overline{p})$ be the average final solution cost obtained on the 14 instances `C2`–`C14` and `F12` with $p = \overline{p}$, and let $ttime_{avg}(\overline{p})$ be the corresponding average CPU time in seconds. With $p = 0.3$, $p = 0.5$ and $p = 0.7$ we obtained the following results: $z_{avg}(0.3) = 684.55$ and $ttime_{avg}(0.3) = 71.9$, $z_{avg}(0.5) = 681.65$ and $ttime_{avg}(0.5) = 251.6$, $z_{avg}(0.7) = 683.32$ and $ttime_{avg}(0.7) = 460.0$. As expected, the average CPU time consistently increases with the number of extracted customers, while the best solution costs are obtained with the default setting of p (i.e., $p = 0.5$), thus indicating that extracting too many customers leads in general to worse solutions (i.e., $z_{avg}(0.7) > z_{avg}(0.5)$). This is not completely surprising, and it is essentially due to the column generation heuristic, which falls in troubles in finding good variables for the Reallocation Model when the current solution has been almost completely "destroyed" by the removal of too many customers.

The current best known solution costs for the tested instances are given in summary in Table 6.4, where we also report the number of customers n and the route duration limit D associated with the vehicles. Solution costs are given both for the case $F = \infty$ (i.e., when the objective is to minimize the number of used vehicles first and the traveling cost second) and the case $F = 0$ (i.e., when the objective is to minimize the traveling cost). As usual, the best known solution cost for the case $F = 0$ is reported only if the traveling cost is smaller than the corresponding one for the case $F = \infty$. For each instance whose best known solution was not improved by LSA we report the algorithms providing the corresponding best known costs. Previously best known solution costs

reached also by LSA (starting from a worse solution) are underlined, while new best solution costs found by LSA are in bold face. For the capacitated instances, in the case $F = \infty$, we also report the best known lower bound LB taken from [113] and [140].

Table 6.4: Current best known solution costs for the tested OVRP benchmark instances.

Inst.	n	D	Best known solution						
			$F = \infty$				$F = 0$		
			m	LB	cost	best heuristics	m	cost	best heursitiscs
C1	50		5	416.1	*416.06	[40], [69], [88], [90, 91], [115], [141]	6	412.96	[163], [164], [165]
C2	75		10	559.62	567.14	[69], [88], [90, 91], [115], [141]	11	564.06	[163], [164], [165]
C3	100		8	639.7	*639.74	[69], [88], [115]	9	639.57	[165]
C4	150		12	730.2	733.13	[69], [88], [115], [141]			
C5	199		16	848.5	879.37	[164]	17	**868.44**	
C6	50	180	6		412.96	[40], [69], [88], [90, 91], [115], [141]			
C7	75	144	10		583.19	[141]	11	568.49	[69], [90, 91], [115]
C8	100	207	9		644.63	[40], [69], [88], [115]			
C9	150	180	13		**757.73**		14	756.14	[69]
C10	199	180	17		**874.71**				
C11	120		7	657.1	682.12	[69], [88], [141]	10	678.54	[165]
C12	100		10	534.2	*534.24	[69], [88], [115], [141], [163], [164], [165]			
C13	120	648	11		**899.16**		12	**894.19**	
C14	100	936	11		591.87	[69], [88], [115], [141]	12	581.81	[69]
F11	71		4	177.0	*177.00	[69], [90, 91], [115], [141]			
F12	134		7	762.9	**769.55**				
O1	200		5		6018.52	[69], [115]			
O2	240		9		**4573.53**				
O3	280		7		7731.46	[69]			
O4	320		10		**7251.74**				
O5	360		8		9197.61	[115]	9	**9162.93**	
O6	400		9		9803.80	[115]			
O7	440		10		**10344.37**				
O8	480		10		12420.16	[69]			

6.6 Conclusions

The Integer Linear Programming (ILP) Local Search algorithm presented in the previous chapter for the classical Vehicle Routing Problem (VRP) has been extended in this chapter to the Open Vehicle Routing Problem (OVRP), a variant of VRP in which the vehicles are not required to return to the depot after completing their service. OVRP has recently received an increasing attention in the literature, and several heuristic and metaheuristic algorithms have been proposed for this problem, as well as exact approaches.

The algorithm has been extended in order to capture the different structure of the addressed problem. Further, the main ingredients of the framework have been slightly modified, with the aim of providing an overall simpler and more general procedure, less dependent on small implementation details.

Computational results on 22 benchmark instances from the literature showed the effectiveness and the flexibility of the method, which can be used as a profitable tool for improving existing OVRP solutions, and even extremely-good quality solutions found by the most effective metaheuristic techniques proposed for OVRP. Out of 30 best known solutions which are not provably optimal, in 10 cases the proposed method was able to improve on the best known solution reported in the literature.

Chapter 7

An extended formulation for the Traveling Salesman Problem with Time Windows

7.1 Introduction

This chapter[1] presents an extended formulation for the Traveling Salesman Problem with Time Windows (TSPTW), a well known generalization of the classical TSP where each node must be visited within a given time window. The polyhedral approaches proposed for this problem in the literature typically follow the one which has been proven to be extremely effective in the classical TSP context. Here we present an overall (quite) general idea which is based on a relaxed discretization of time windows. Such an idea leads to a strong formulation and to strong valid inequalities which can be effectively separated within a classical branch-and-cut framework.

The overall branch-and-cut algorithm has been tested on hard benchmark instances from the literature, arising from a practical scheduling application. The results show that the relaxed discretization of time windows is effective in practice for tackling TSPTW. Interestingly, several unsolved benchmark instances are here solved for the first time.

The chapter is organized as follows. Section 7.2 recalls some of the main results from the literature for TSPTW. In Section 7.3 we formally state the problem and present a new extended formulation for TSPTW, based on a relaxed discretization of time windows. Section 7.4 presents some valid inequalities for the proposed formulation and discuss their connections with other inequalities which are typically used to tackle TSPTW. Section 7.5 discuss the main steps of the separation procedure for the proposed inequalities. Section 7.6 describes a possible way for exploiting the relaxed discretization in practice, and describe how to deal with some crucial aspects of this method. Section 7.7 report computational results on 50 benchmark instances from the

[1]The results of this chapter are part of a joint work-in-progress with S. Dash, O. Günlük and A. Lodi.

literature, comparing the proposed approach with a more classical but effective one. Finally, some conclusions are drawn in Section 7.8.

7.2 Literature review

The *Traveling Salesman Problem with Time Windows* (TSPTW) is the problem of finding a minimum-cost path visiting a set of cities exactly once, where each city must be visited within a specific time window. This problem can be found in a variety of real-life applications such as routing, scheduling, manufacturing-and-delivery problems, and, for this reason, has been extensively studied (see Desrosiers et al. [70] for a survey). The problem is of course NP-hard because it generalizes the classical *Traveling Salesman Problem* (TSP) and Savelsberg [156] showed that even finding a feasible solution of TSPTW is NP-complete.

The first approaches dealing with TSPTW, back in the eighties, are due to Christofides, Mingozzi and Toth [48] and Baker [13] and considered a variant of the problem where the total schedule time has to be minimized. Both papers presented branch-and-bound schemes where the former used a so-called state-space-relaxation approach, whereas the latter exploited a time-constrained critical-path formulation.

In the nineties, several contributions were proposed. Langevin et al. [111] addressed the problem by using a two-commodity flow formulation within a branch-and-bound scheme. Dumas et al. [74] proposed a dynamic-programming approach with sophisticated elimination tests to reduce the state space, while. Mingozzi, Bianco and Ricciardelli [126] presented a dynamic-programming algorithm with a generalization of the state-space-relaxation technique which can also be applied to TSPTW problems with precedence constraints. Also in the nineties, the problem started to receive quite a lot of attention by the *Constraint Programming* (CP) community. Indeed, the double nature of TSPTW, which includes at the same time routing and scheduling characteristics, makes it suitable for CP techniques. First, small TSPTW instances were solved as subproblems of a large task assignment by Caseau and Koppstein [43]. Lately, the problem was instead the main focus of a paper by Pesant et al. [138] where the authors solved it by enriching a simple CP model with redundant constraints. A variant of the TSPTW, called TSP with *multiple* Time Windows, has been solved by the same authors (Pesant et al. [139]) with basically the same algorithmic approach (and slightly adapting the model), thus showing the flexibility of the CP paradigm.

The most recent approaches are due to Balas and Simonetti [27], Ascheuer, Fischetti and Grötschel [11] and Focacci, Lodi and Milano [89].

Specifically, Balas and Simonetti (2001) proposed a special dynamic-programming approach: under the assumption that, given an initial ordering of the cities, city i precedes city j if $j \geq i+k$ ($k > 0$), the authors discussed and experimented an algorithm that is linear in n and exponential in k. For those problems satisfying these conditions, the dynamic-programming procedure finds an optimal solution, while in the other cases it can be used as a linear time heuristic to explore an exponential-size neighborhood exactly.

Ascheuer, Fischetti and Grötschel [11] considered several formulations for the *Asym-*

metric version (ATSPTW) of the problem (among which a new one introduced in the companion paper, Ascheuer, Fischetti and Grötschel [10]), and computationally compared them within a branch-and-cut scheme. The framework incorporates up-to-date techniques tailored for the Asymmetric TSPTW such as data preprocessing, primal heuristics, local search, and variable fixing (obviously, besides the specific separation algorithms).

Finally, Focacci, Lodi and Milano [89] proposed a hybrid algorithm merging classical Operations Research techniques (as reduced-cost fixing, cutting planes and Lagrangean relaxations) for coping with the optimization perspective (the routing part), and CP propagation algorithms for the feasibility viewpoint (the scheduling part).

7.3 Problem definition and formulations

We formally define the problem by considering the more general directed case. Let $G = (V, A)$, be a directed graph where each node $i \in V$ represents a city with an associated time window $W_i = [R_i, D_i]$. We call R_i the *release time* and D_i the *deadline* of node i. For each $(i, j) \in A$ there is an associated travel cost $c_{ij} \geq 0$, and a travel time $\theta_{ij} > 0$. We assume that all data is integral. Moreover, there are two special nodes $p, q \in V$. A tour is a permutation of the nodes in V that starts with node p and ends at node q. Given a tour, we associate an *arrival* time and a *start* time with each node as follows: arrival and start time at node p is its release time r_p. For all other nodes the arrival time is the start time at the previous node in the tour plus the travel time from that node to the current one. The start time at node i is defined as the maximum of the arrival time at i and R_i. A tour is *feasible* if the start time at each node is contained in its time window. An optimal solution to the problem is a feasible tour with minimum cost. In this setting early arrivals are allowed, in the sense that a the tour can arrive at a node before its release time. However, in this case the tour "waits" until the release time of the node.

Note that our definition of the problem requires two special "start" and "end" nodes p and q. In some variations of the problem studied in the literature, any permutation of the nodes is potentially feasible. However, these instances can trivially be translated into our setting by creating two new artificial nodes to stand for p and q and adding zero-cost arcs from p to all other nodes and from all nodes to q having fixed travel time (say 1). In addition, the time windows are set as follows: $W_p = [m, m]$ and $W_q = [M, M]$ where m is less than all other release times, and M is larger than all other deadlines.

Throughout the paper we use $V^+(i)$ to denote the set of nodes that can follow i in a tour, that is $V^+(i) := \{j \in V : (i, j) \in A\}$. Similarly, $V^-(i) := \{j \in V : (j, i) \in A\}$. We next describe three mixed integer programming formulations for the problem. The first one is the so-called Big M Formulation. The following two formulations are new and we call them the Time indexed Formulation, and the Bucket Formulation.

7.3.1 Big M Formulation (BMF)

This formulation uses a binary variable x_{ij} for each arc $(i,j) \in A$ to denote whether or not node j follows node i in the tour. In addition, there is a variable s_i associated with each node $i \in V$ to represent its start time in the tour. The formulation is:

$$\min \quad \sum_{(i,j)\in A} c_{ij}x_{ij}, \tag{7.1}$$

$$\sum_{j\in V^+(i)} x_{ij} = 1 \qquad \forall\ i \in V \setminus \{q\}, \tag{7.2}$$

$$\sum_{k\in V^-(i)} x_{ki} = 1 \qquad \forall\ i \in V \setminus \{p\}, \tag{7.3}$$

$$s_i + \theta_{ij} - (1 - x_{ij})M_{ij} \leq s_j \qquad \forall\ (i,j) \in A, \tag{7.4}$$

$$R_i \leq s_i \leq D_i \qquad \forall\ i \in V, \tag{7.5}$$

$$x_{ij} \in \{0,1\} \qquad \forall\ (i,j) \in A, \tag{7.6}$$

where $M_{ij} = D_i - R_j + \theta_{ij}$. Note that the constraints (7.2) and (7.3) ensure that the x_{ij} variables with value 1 in a feasible integral solution correspond to the union of a path from p to q and a collection of directed cycles. In addition, the constraints (7.4) ensure that start times at the nodes are increasing along any path and therefore directed cycles cannot exist in the solution. Finally, the constraints (7.4) and (7.5) together ensure that the solution respects time windows and defines a feasible tour.

7.3.2 Time Indexed Formulation (TIF)

In this formulation, instead of the start time variables s_i, there are binary variables z_i^t associated with each $t \in W_i$ such that $z_i^t = 1$ if and only if the start time at node i is t. In addition, there are binary variables y_{ij}^t associated with each arc $(i,j) \in A$ and each $t \in W_i$ such that $y_{ij}^t = 1$ if and only if the start time at node i is t and arc (i,j) is present in the tour. We assume that y_{ij}^t is present in the formulation only if $t + \theta_{ij} \leq D_j$, i.e., one can start at time t at node i, and travel along arc (i,j) and arrive at j by its deadline.

We use $I_k(i,t)$ to denote the collection of possible start times at node k assuming that the start time at node i is t and arc (k,i) is selected. More precisely, we define

$$I_k(i,t) \ = \ \{\tau \in W_k : \max\{\tau + \theta_{ki}, R_i\} = t\}.$$

In other words, if the start time t at node i is R_i, then the start time at node k in a feasible tour is some $\tau \in W_k$ satisfying $\tau \leq R_i - \theta_{ki}$, otherwise the start time at node k is exactly $t - \theta_{ki}$ if it belongs to W_k.

The following is the Time Indexed Formulation (TIF) for ATSPTW:

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} & & \\
& \sum_{t\in W_i} z_i^t = 1 & \forall\ i \in V, & (7.7)\\
& \sum_{j\in V^+(i)} y_{ij}^t = z_i^t & \forall\ i \in V\setminus\{q\}, \forall\ t \in W_i, & (7.8)\\
\sum_{k\in V^-(i)} & \sum_{\tau\in I_k(i,t)} y_{ki}^\tau = z_i^t & \forall\ i \in V\setminus\{p\}, \forall\ t \in W_i, & (7.9)\\
& \sum_{t\in W_i} y_{ij}^t = x_{ij} & \forall\ (i,j) \in A, & (7.10)\\
& z_i^t \in \{0,1\} & \forall\ i \in V, \forall\ t \in W_i, & (7.11)\\
& y_{ij}^t \in \{0,1\} & \forall\ (i,j) \in A, \forall\ t \in W_i, & (7.12)\\
& x_{ij} \in \{0,1\} & \forall\ (i,j) \in A. &
\end{aligned}$$

We will show that TIF is a valid formulation of the ATSPTW problem – feasible tours correspond to integral solutions of TIF – and is stronger than BMF. The constraints (7.7) assert that a tour must start at each node i within the window W_i. Constraint (7.8) asserts that a tour with start time t at node $i \neq q$ must leave the node at time t along some arc (the "processing" time at i is zero), whereas constraint (7.9) asserts that tour has a start time at the previous node (say k; also $i \neq p$) contained in $I_k(i,t)$. Clearly any feasible tour can be mapped to an integral solution of TIF by setting $x_{ij} = 1$ if arc (i,j) is used in the tour, and $z_i^t = 1$ if the tour has start time t at node i. Further, if $\theta_{ij} > 0$ for all arcs in A, then every integral solution of TIF defines a feasible tour as there cannot be any directed cycle in the support of the solution.

Proposition 7.1 *The LP relaxation of TIF dominates the LP relaxation of BMF.*

Proof. We define a linear transformation which maps a solution of TIF to a solution of BMF by letting the x_{ij} values remain unchanged, and letting

$$s_i = \sum_{t\in W_i} t z_i^t \text{ for all } i \in V.$$

We will show that the values of s_i and x_{ij} satisfy the constraints defining BMF. Firstly, (7.2) holds as

$$\sum_{j\in V^+(i)} x_{ij} = \sum_{j\in V^+(i)} \sum_{t\in W_i} y_{ij}^t = \sum_{t\in W_i} \sum_{j\in V^+(i)} y_{ij}^t = \sum_{t\in W_i} z_i^t = 1.$$

The first equality above is implied by (7.10), the third equality by (7.8) and the last one by (7.7). To see that (7.3) holds, note that

$$\sum_{k\in V^-(i)} x_{ki} = \sum_{k\in V^-(i)} \sum_{t\in W_k} y_{ki}^t = \sum_{k\in V^-(i)} \sum_{t'\in W_i} \sum_{t\in I_k(i,t')} y_{ki}^t.$$

The last expression in the previous equation equals

$$\sum_{t' \in W_i} \sum_{k \in V^-(i)} \sum_{t \in I_k(i,t')} y_{ki}^t = \sum_{t' \in W_i} z_i^{t'} = 1$$

by equations (7.8) and (7.9). Now define $\overline{z}_i^t = z_i^t - y_{ij}^t$. Then

$$\sum_{t \in W_i} \overline{z}_i^t = \sum_{t \in W_i} z_i - \sum_{t \in W_i} y_{ij}^t = 1 - x_{ij}$$

by equations (7.7) and (7.10). Therefore

$$\begin{aligned} s_i = \sum_{t \in W_i} t z_i^t = & \quad \sum_{t \in W_i} t \overline{z}_i^t + \sum_{t \in W_i} t y_{ij}^t \\ \leq & \quad D_i \sum_{t \in W_i} \overline{z}_i^t + \sum_{t \in W_i} t y_{ij}^t = D_i(1 - x_{ij}) + \sum_{t \in W_i} t y_{ij}^t. \end{aligned} \tag{7.13}$$

Now, using the fact that $M_{ij} = D_i + \theta_{ij} - R_j$, we have

$$s_i + \theta_{ij} - (1 - x_{ij}) M_{ij} = s_i + \theta_{ij} x_{ij} - (1 - x_{ij})(D_i - R_j).$$

The inequality in (7.13) and (7.10) imply that the last term above is less than or equal to

$$R_j(1 - x_{ij}) + \theta_{ij} x_{ij} + \sum_{t \in W_i} t y_{ij}^t = R_j(1 - x_{ij}) + \sum_{t \in W_i} (t + \theta_{ij}) y_{ij}^t.$$

Similarly, writing $\overline{z}^\tau = z^\tau - \sum_{t \in I_i(j,\tau)} y_{ij}^t$, we have $\sum_{\tau \in W_j} \overline{z}^\tau = 1 - x_{ij}$. Further

$$\begin{aligned} s_j = \sum_{\tau \in W_j} \tau z_j^\tau = \sum_{\tau \in W_j} \tau \overline{z}^\tau + \sum_{\tau \in W_j} \tau \sum_{t \in I_i(j,\tau)} y_{ij}^t \quad & \geq R_j(1 - x_{ij}) + \sum_{\tau \in W_j} \tau \sum_{t \in I_i(j,\tau)} y_{ij}^t \\ & \geq R_j(1 - x_{ij}) + \sum_{t \in W_i} (t + \theta_{ij}) y_{ij}^t. \end{aligned}$$

□

7.3.3 Time Bucket Relaxation (TBR)

We next present a new formulation which is obtained by aggregating some of variables in the time-indexed formulation presented above. This formulation, as we describe below, is a relaxation for ATSPTW and some feasible solutions to this formulation may not correspond to feasible tours. As we discuss in the next section, we add the so-called *infeasible path* inequalities to this formulation to make it an exact formulation for ATSPTW.

We call this formulation the *time bucket formulation* (TBF) where, unlike the time-indexed formulation, we do not define variables associated with each time index. We instead, partition the time window of a node into a collection of non-overlapping intervals and define variables associated with these intervals. More precisely, the time

window W_i for a node $i \in V$ is divided in a set of buckets (intervals) $B_i = \{b_1^i, \ldots, b_L^i\}$, with $b = [r_b, d_b]$, $r_{b_1^i} = R_i$, $d_{b_L^i} = D_i$, and the start time of b_{j+1}^i is greater than the end time of b_j^i. We allow $W_i \neq \bigcup_{b \in B_i} [r_b, d_b]$ as long as each missing time instant $t \in W_i$ cannot be a valid starting time of the node i associated with a feasible tour. Using earlier notation, a sufficient condition for such a time instant is $\bigcup_{k \in V^-(i)} I_k(i,t) = \emptyset$.

As in the previous formulations, binary variables x_{ij} indicate whether or not arc $(i,j) \in A$ is a part of the tour. We define binary variables z_i^b associated with each bucket $b \in B_i$ of a node $i \in V$ such that $z_i^b = 1$ if and only if bucket b is selected for node i (b is then called the *starting bucket* at i). We also define binary variables y_{ij}^b associated with each arc $(i,j) \in A$ and each bucket $b \in B_i$ such that $y_{ij}^b = 1$ if and only if $x_{ij} = 1$ and $z_i^b = 1$.

We also use $I_k(i,b)$ to denote the collection of possible starting buckets at node k assuming that arc (k,i) is selected and the starting bucket at node i is b. More precisely, for a bucket $b_\ell \in B_i$

$$I_k(i, b_\ell) = \{b \in B_k \ : \ d_{b_{\ell-1}} < r_b + \theta_{ki} \leq d_{b_\ell}\}$$

where we assume $d_{b_0} = -\infty$. Conversely, if the starting bucket at node k is b and arc (k,i) is selected then the starting bucket at node i is denoted by $N_i(k,b)$ and is defined as the bucket $\beta \in B_i$ such that $b \in I_k(i,\beta)$. If $r_b + \theta_{ij} > Dj$, we define $N_i(k,b)$ to be null. The following is the time bucket relaxation (TBR) of ATSPTW:

$$\min \quad \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\sum_{b \in B_i} z_i^b = 1 \qquad \forall\ i \in V \tag{7.14}$$

$$\sum_{j \in V^+(i)} y_{ij}^b = z_i^b \qquad \forall\ i \in V \setminus \{q\}, \forall\ b \in B_i \tag{7.15}$$

$$\sum_{k \in V^-(i)} \sum_{\beta \in I_k(i,b)} y_{ki}^\beta = z_i^b \qquad \forall\ i \in V \setminus \{p\}, \forall\ b \in B_i \tag{7.16}$$

$$\sum_{b \in B_i} y_{ij}^b = x_{ij} \qquad \forall\ (i,j) \in A \tag{7.17}$$

$$z_i^b \in \{0,1\} \qquad \forall\ i \in V, \forall\ b \in B_i$$

$$y_{ij}^b \in \{0,1\} \qquad \forall\ (i,j) \in A, \forall\ b \in B_i$$

$$x_{ij} \in \{0,1\} \qquad \forall\ (i,j) \in A$$

$$y_{ij}^b = 0 \qquad \textit{for all } (i,j) \in A,\ b \in B_i \text{ such that } r_b + \theta_{ij} > Dj.$$

Note that equation (7.14) implies that for each node $i \in V$, there is precisely one $b \in B_i$ for which $z_i^b = 1$. Therefore, if $i \neq q$, equation (7.15) implies that precisely one $y_{ij}^b = 1$ for some $j \in V^+(i)$. Similarly, if $i \neq p$, equation (7.16), implies that precisely one $y_{ki}^\beta = 1$ for some $k \in V^-(i)$. Therefore, if $i \neq p$, there is precisely one node $j \in V^+(i)$ for which $x_{ij} = 1$ and if $i \neq p$, there is precisely one node $k \in V^+(i)$ for which

$x_{ki} = 1$. Consequently, a feasible solution to the TBR does not necessarily give a tour (a permutation of the nodes) but instead gives a directed path from p to q that does not necessarily visit all other nodes on the way. If there are nodes that do not appear on this path, they are partitioned into cycles. Therefore, as presented above, the TBR has feasible solutions that do not correspond to feasible tours for ATSPTW.

We next show that all feasible tours for the ATSPTW are feasible for the this formulation and therefore TBR is a relaxation for ASTPTW.

Proposition 7.2 *Any feasible tour for ATSPTW is also feasible for the TBR and therefore TBR is a relaxation for ATSPTW.*

Proof. Let $V = \{1 \dots n\}$ and without loss of generality consider a feasible tour $(1 \to 2 \to \dots \to n)$ for which there are feasible starting times $s_i \in W_i$ for each $i \in V$ such that $s_1 = R_1$ and $s_i = \max\{R_i, s_{i-1} + \theta_{i-1,i}\}$ for $i \geq 2$. For $i \in V$, remember that $B_i = \{b_1, \dots, b_L\}$, with $b = [r_b, d_b]$, $r_{b_1} = R_i$, $d_{b_L} = D_i$, and $r_{b_{j+1}} > d_{b_j}$.

For all $i \in V$, we next identify a time instant τ_i and the associated bucket $\beta(i)$ and then construct a feasible y vector using these buckets. Let $\tau_1 = R_1$ and $\beta(1)$ be the first bucket of node 1 and notice that $\tau_1 = r_{\beta(1)} = s_1$. For $i > 1$, we iteratively define $\tau_i = \max\{R_i, r_{\beta(i-1)} + \theta_{i-1,i}\}$ and we define $\beta(i)$ to be the bucket $b_\ell \in B_i$ such that $d_{b_{\ell-1}} < \tau_i \leq d_{b_\ell}$ where we let $d_{b_0} = -\infty$. Notice that as $R_i \leq \tau_i$, if $\tau_i \leq s_i$, then $\tau_i \in W_i$ and it is possible to identify the time window $\beta(i) \in B_i$.

We next inductively argue that $\tau_i \leq s_i$ for all $i \in V$. First note that $\tau_1 = s_1$, and assume that the claim is true for $i \in V$. As $\tau_i \leq s_i$, $\tau_i \in W_i$ and there exists a time window $\beta(i) \in B_i$. If $\tau_i \geq r_{\beta(i)}$, then using the definition of s_i and τ_i, we clearly have $\tau_{i+1} \leq s_{i+1}$. If, on the other hand, $\tau_i < r_{\beta(i)}$ then $\tau_i \in W_i$ is a time instant that cannot be a valid starting time of the node i associated with a feasible tour. Let $\hat{\tau} > \tau_i$ be the earliest possible starting time for a feasible tour. Clearly $\hat{\tau}$ is contained in some bucket for node i and $\hat{\tau} \geq r_{\beta(i)}$. Furthermore, as $s_i \geq \tau_i$ and as s_i is a valid starting time at node i, we have $s_i \geq \hat{\tau}$ implying $s_i \geq r_{\beta(i)}$ and therefore $\tau_{i+1} \leq s_{i+1}$ as desired. Therefore, time windows $\beta(i) \in B_i$ can be constructed for all $i \in V$ as claimed.

The last step is to set $x_{i,i+1} = z_i^{\beta(i)} = y_{i,i+1}^{\beta(i)} = 1$ for $i \in \{1, \dots, n-1\}$ and $z_n^{\beta(n)} = 1$ to obtain a feasible solution to TBR. As this solution has the same cost as the starting solution to ATSPTW, the proof is complete. □

Therefore, all feasible solutions and in particular the optimal solution to the ATSPTW corresponds to a feasible solution for the TBR. However, the converse is not true as feasible solutions to the TBR do not have to give feasible solutions to ATSPTW.

7.3.4 Time Bucket Formulation (TBF)

We next present the so-called *subtour elimination* and *infeasible path* constraints that help turn the TBR to a valid formulation for ATSPTW. Let S be a proper subset of V and $\delta(S)$ be the set of arcs in with their tail in S and head outside S, that is $\delta(S) = \{(i,j) \in A \ : \ i \in S, j \in V \setminus S\}$. Clearly, any feasible tour has at least one arc

that connects nodes in S to nodes in $V \setminus S$ assuming $q \notin S$. Therefore, the well known *subtour elimination* constraints:

$$\sum_{(i,j)\in\delta(S)} x_{ij} \geq 1 \tag{7.18}$$

are valid for ATSPTW for all $S \subset V$ with $q \notin S$. Note that if a feasible solution to the TBR also satisfies subtour elimination constraints for all $S \subset V$, then it gives a directed path from p to q that visits all remaining nodes.

Let $P = (v_1, v_2, \ldots, v_h)$ be a permutation of the nodes in $S \subseteq V$. We call P an infeasible path if it is not possible to construct feasible start times $s_{v_i} \in W_{v_i}$ that satisfy $s_{v_i} \geq s_{v_{i-1}} + \theta_{v_i,v_{i-1}}$ for all $i \geq 2$. Clearly, if P is an infeasible path, it cannot be a part of a feasible tour and therefore any feasible tour satisfies the *infeasible path* constraint

$$\sum_{i=1}^{h-1} x_{v_i,v_{i+1}} \leq h - 2 \tag{7.19}$$

for all infeasible path P. Notice that, if a solution to the TBR has no cycles and in addition contains no infeasible paths (in particular, if the solution itself is not an infeasible path from p to q) then it gives a feasible solution to ATSPTW.

Therefore, it is possible to formulate the ATSPTW using only the x variables and an exponential number of constraints. Ascheuer, Fischetti and Grötschel [11] exploited constraints (7.19) to derive an ILP formulation which uses only a set of binary variables associated with the arcs in the graph. We instead use this observation to obtain a new formulation for the ATSPTW which we call the *time-bucket formulation* (TBF). The new formulation is obtained by simply adding all possible subtour elimination constraints (7.18) and all possible infeasible path constraints (7.19) to the TBR. In practice, these constraints should clearly be used as cutting-planes in a branch-and-cut framework.

7.4 Valid inequalities for TBF

For $S \subseteq V$, let $\overline{S}$ stand for $V \setminus S$. If $S \subseteq V$ and $S' \subseteq \overline{S}$, we define $\delta(S, S') = \{(i, j) \in A : i \in S,\ j \in S'\}$ to be the set of arcs going from S to S'. Furthermore, we next define a "bucket" graph $G' = (\mathcal{B}, A_{\mathcal{B}})$ associated with the formulation where

$$\mathcal{B} = \bigcup_{i\in V} B_i \text{ and } A_{\mathcal{B}} = \bigcup_{(ij)\in A} \Big\{(b, b') \ :\ b \in B_i, b' \in B_j, b' = N_j(i, b)\Big\}.$$

Notice that there is a one-to-one correspondence between the variables y^b_{ij} of TBF and the arcs $A_{\mathcal{B}}$ of G' and there is a one-to-one correspondence between the z^b_i variables and the nodes $\mathcal{B}$. Therefore, every feasible solution of TBF corresponds to a path in G' that starts from a bucket in B_p and ends at a bucket of B_q that visits exactly one

bucket of each of the remaining nodes. Given a collection of buckets $B \subseteq \mathcal{B}$, we define $\overline{B} = \mathcal{B} \setminus B$. For disjoint $B \subseteq \mathcal{B}$ and $B' \subseteq \overline{B}$, we define

$$\Delta(B, B') = \{(i, j, b) \ : \ i, j \in V, b \in B_i \cap B, \ N_j(i, b) \in B'\}$$

where $N_j(i, b)$ is the the starting bucket at node j if arc (i, j) is selected and the starting bucket at node i is b. In other words, $\Delta(B, B')$ corresponds to collection of y variables associated with the cut $\delta(B, B')$ in G' (the arcs in G' that go from buckets in B to buckets in B').

For any $S \subseteq V$, $B(S) = \cup_{i \in S} B_i$ denotes the set of buckets associated with the nodes in S. For any $B \subseteq \mathcal{B}$, $V(B) = \{i \in V : B_i \subseteq B\}$ denotes the set of nodes which have all their buckets contained in B.

7.4.1 Subtour elimination constraints

Recall the subtour elimination constraints (SEC)

$$\sum_{(i,j) \in \delta(S)} x_{ij} \geq 1, \quad \text{(SECs)} \tag{7.20}$$

which are valid for ATSPTW for all $S \subset V$ with $q \notin S$. The following inequalities involving buckets generalize the SECs:

$$\sum_{(i,j,b) \in \Delta(B, \overline{B})} y_{ij}^b \geq 1, \quad \text{(BSECs)} \tag{7.21}$$

provided that $B \subseteq \mathcal{B} \setminus B_q$ and $B_t \subseteq B$ for some $t \in V$.

Proposition 7.3 *Inequalities (7.21) are valid for TBF and they subsume inequalities (7.20).*

Proof. Remember that every feasible solution of TBF corresponds to a path in G' that visits exactly one bucket of each node and and ends at a bucket of B_q. Therefore, the path has to cross the cut $\Delta(B, \overline{B})$ at least once as $B_q \subset \mathcal{B} \setminus B$ and $B_t \subseteq B$ for some $t \in V$. This observation shows that the inequality is valid. To see that these inequalities subsume (7.20), it suffices to observe that choosing $B = \cup_{i \in S} B_i$ in inequality (7.21) gives inequality (7.20) as $\sum_{(i,j) \in \delta(S)} x_{ij} = \sum_{(i,j,b) \in \Delta(B, \overline{B})} y_{ij}^b$. □

7.4.2 SOP Inequalities

We next present the so-called SOP (sequential ordering polytope) inequalities described in the literature. These inequalities have been presented by Balas, Fischetti and Pulleyblank [22] in the context of the Precedence-constrained TSP, and are known to be effective even for ATSPTW (see, e.g., [11]). SOP inequalities are based on a concept of "precedence" between nodes[2]. We say that a node $i \in V$ precedes another node $j \in V$

[2]Obviously, several precedences between nodes can be inferred by exploiting the node time windows.

if j has to be visited after i in any feasible tour. We denote this precedence as $i \prec j$. We extend this definition to subsets of nodes as follows. Let $S, S' \in V$, we say $S \prec S'$ if each node in S precedes every node in S'.

Let $S \subset V$, and define $\pi(S) := \{j \in V : j \prec i \text{ for some } i \in S\}$, and $\sigma(S) := \{j \in V : i \prec j \text{ for some } i \in S\}$. Further, let

$$\delta_\pi(S) = \delta(S \setminus \pi(S), \overline{S} \setminus \pi(S)) \text{ and } \delta_\sigma(S) = \delta(S \setminus \sigma(S), \overline{S} \setminus \sigma(S)).$$

Here $\delta_\pi(S)$ stands for the arcs (i, j) from S to $\overline{S}$ where $i, j \notin \pi(S)$. The following inequalities are valid for the BMF and therefore for the TBF:

$$\sum_{(i,j)\in\delta_\pi(S)} x_{ij} \geq 1, \quad (\pi\text{-cuts}) \tag{7.22}$$

$$\sum_{(i,j)\in\delta_\sigma(S)} x_{ij} \geq 1, \quad (\sigma\text{-cuts}) \tag{7.23}$$

where $S \subset V \setminus \{q\}$. It is known that constraints (7.22) and (7.23) dominate subtour elimination constraints (7.20) (see, e.g., [22]).

For any permutation $P = (v_1, \ldots, v_h)$ of nodes of V, define $\theta(P) = \sum_{i=1}^{h-1} \theta_{v_i, v_{i+1}}$. Then, given disjoint sets X, Y with $X \subseteq V \setminus \{q\}$ and $X \prec Y$, let

$$W = \{k \in V \setminus (X \cup Y) : \exists\, i \in X, j \in Y \text{ with } R_i + \theta(i, k, j) > D_j\}, \tag{7.24}$$

and note that a path from i to k to j arriving at node j after D_j even if starting at R_i at i cannot be part of a feasible tour. Further, let

$$Q := \{(u, v) \in A : \exists\, i \in X, j \in Y \text{ with } R_i + \theta(i, u, v, j) > D_j\}. \tag{7.25}$$

Finally, let $\tilde{W} = \pi(X) \cup \sigma(Y) \cup W$. Then, for any $S \subset V$ such that $X \subseteq S$, $Y \subseteq \overline{S}$, the following inequalities are also valid for BMF and therefore for TBF:

$$\sum_{(i,j)\in\delta(S\setminus\tilde{W}, \overline{S}\setminus\tilde{W})\setminus Q} x_{ij} \geq 1. \quad ((\pi, \sigma)\text{-cuts}) \tag{7.26}$$

In this chapter we refer inequalities (7.26) as (π, σ)-cuts. However, they are indeed a strengthened version of the classical (π, σ)-cuts for the Precedence-constrained TSP (see, e.g., [10, 11]).

7.4.3 Bucket SOPs

For buckets, we extend the concept of precedence as follows: for a node $i \in V$, we say that bucket $b \in B_i$ precedes node $j \in V$ if in any feasible solution of TBF that visits node i at bucket b (i.e., any feasible solution with $z_i^b = 1$), node j is visited after node i. We denote this precedence as $b \prec j$. In a similar fashion, we say that a node $j \in V$ precedes a bucket $b \in B_i$ if in any feasible TBF solution that visits node i at bucket b, node j is visited before node i. We denote this precedence as $j \prec b$. Note that the original node precedence relationship implies bucket-node precedences as follows:

$i \prec j \Rightarrow b \prec j$ for all $b \in B_i$ and $i \prec b$ for all $b \in B_j$. Furthermore, the bucket-node precedence relationship can be combined with node precedence relationships as follows: for a bucket b and nodes $j, k \in V$, $b \prec j$ and $j \prec k$ implies $b \prec k$, while $j \prec b$ and $k \prec j$ implies $k \prec b$.

Let $B \subseteq \mathcal{B}$ be a collection of buckets. We next abuse notation and use $\pi(B)$ to denote the set of buckets which precede nodes with all buckets contained in B, in other words

$$\pi(B) = \{b \in \mathcal{B} : b \prec i, \text{ where } i \in V \text{ and } B_i \subseteq B\}.$$

Similarly, we define $\sigma(B)$ as

$$\sigma(B) = \{b \in \mathcal{B} : i \prec b, \text{ where } i \in V \text{ and } B_i \subseteq B\}.$$

Furthermore, we define

$$\Delta_\pi(B) = \Delta(B \setminus \pi(B), \overline{B} \setminus \pi(B)) \text{ and } \Delta_\sigma(B) = \Delta(B \setminus \sigma(\overline{B}), \overline{B} \setminus \sigma(\overline{B})).$$

We next generalize π-cuts and σ-cuts in the context of TBF as follows: Let $B \subseteq \mathcal{B}$ such that $B_t \subseteq B$ for some node $t \in V$ and $B_q \subseteq \overline{B}$. We call the following inequalities

$$\sum_{(i,j,b) \in \Delta_\pi(B)} y_{ij}^b \geq 1 \quad (\pi_B\text{-cuts}) \tag{7.27}$$

and

$$\sum_{(i,j,b) \in \Delta_\sigma(B)} y_{ij}^b \geq 1 \quad (\sigma_B\text{-cuts}) \tag{7.28}$$

bucket SOP inequalities. It is easy to see that bucket SOP inequalities (7.27) and (7.28) dominate bucket subtour elimination constraints (7.21).

Proposition 7.4 *The Bucket SOP inequalities (7.27) and (7.28) are valid for TBF.*

Proof. As discussed earlier, a feasible solution to TBF corresponds to a directed path in G' that starts with a bucket in B_p, ends at a bucket in B_q and visits exactly one bucket associated with the remaining nodes. As $B_t \subseteq B$ for some $t \in V$ and $B_q \subseteq \overline{B}$ by assumption, the path must visit one of the buckets in B and must end in one of the buckets in B'. Therefore, the path must cross the cut $\delta(B, B')$ at least once. Let (b, b') be the last arc on the path that crosses from B to B' and let $b \in B_i$ and $b' \in B_j$. Notice that b and b' cannot be included in $\pi(B)$. If $b \in \pi(B)$, then there is a node k with $B_k \subseteq B$ such that $b \prec k$, which means that some bucket of B_k is visited after b, a contradiction. The same argument holds for b'. Therefore $(i, j, b) \in \Delta_\pi(B)$ and the inequality (7.27) is valid. One can similarly argue that if (b, b') is the first arc crossing from B to B' in a feasible solution to TBF, then $b, b' \notin \sigma(\overline{B})$. Therefore inequality (7.28) is valid. □

Furthermore, constraints (7.27) and (7.28) dominate constraints (7.22) and (7.23), respectively. Instead of proving it directly, we will show that constraints (7.22) and

(7.23) are actually dominated by a simpler subfamily of the inequalities described in the next section.

The same generalization used to derive π_B-cuts and σ_B-cuts can be applied to derive the (π_B, σ_B)-cuts, which dominate inequalities (7.26). Let again X, Y be disjoint subset of nodes, with $X \subseteq V \setminus \{q\}$ and $X \prec Y$, and define W and Q as in (7.24) and (7.25), respectively. If now $\tilde{W}$ and $\tilde{Q}$ are defined in the "bucket" space as

$$\tilde{W} := \pi(B(X)) \cup \sigma(B(Y)) \cup B(W) \tag{7.29}$$

and

$$\tilde{Q} = \{(i, j, b) : (i, j) \in Q, b \in B_i\}, \tag{7.30}$$

then, for any $B \in \mathcal{B}$ such that $X \subseteq V(B)$ and $Y \subseteq V(\overline{B})$, the following inequalities are valid for TBF:

$$\sum_{(i,j,b) \in \Delta(B \setminus \tilde{W}, \overline{B} \setminus \tilde{W}) \setminus \tilde{Q}} y_{ij}^b \geq 1. \quad ((\pi_B, \sigma_B)\text{-cuts}) \tag{7.31}$$

7.4.4 Simple Bucket SOPs

Let $S \subseteq V$ and take $B = \cup_{i \in S} B_i$. Then we call the corresponding inequality (7.27) a *simple π_B-cut* (s-π_B-cut). We similarly define *simple σ_B-cuts* (s-σ_B-cuts) as a special case of (7.28).

By abusing notation, s-π_B-cuts and s-σ_B-cuts can be restated more conveniently as follows. For any $S \subseteq V$, let define $\pi_B(S) = \{b \in \mathcal{B} : b \prec j \text{ for some } j \in S\}$ and $\sigma_B(S) = \{b \in \mathcal{B} : j \prec b \text{ for some } j \in S\}$, i.e., $\pi_B(S) = \pi(B(S))$ and $\sigma_B(S) = \sigma(B(S))$. In a similar way, we define

$$\Delta_{\pi_B}(S) = \Delta(B(S) \setminus \pi_B(S), B(\overline{S}) \setminus \pi_B(S)), \quad \text{i.e., } \Delta_{\pi_B}(S) = \Delta_\pi(B(S)),$$

and

$$\Delta_{\sigma_B}(S) = \Delta(B(S) \setminus \sigma_B(\overline{S}), B(\overline{S}) \setminus \sigma_B(\overline{S})), \quad \text{i.e., } \Delta_{\sigma_B}(S) = \Delta_\sigma(B(S)).$$

Then, the following constraint are clearly valid for any $S \subset V \setminus \{q\}$ (see Proposition 7.4):

$$\sum_{(i,j,b) \in \Delta_{\pi_B}(S)} y_{ij}^b \geq 1 \quad (s\text{-}\pi_B\text{-cuts}) \tag{7.32}$$

and

$$\sum_{(i,j,b) \in \Delta_{\sigma_B}(S)} y_{ij}^b \geq 1 \quad (s\text{-}\sigma_B\text{-cuts}) \tag{7.33}$$

By simply recalling the TBF constraints (7.17), and noting that $B(\pi(S)) \subseteq \pi_B(S)$ and $B(\sigma(S)) \subseteq \sigma_B(S)$ for any $S \subset V$, the following proposition is straightforward:

Proposition 7.5 *The simple Bucket SOP inequalities (7.32) and (7.33) dominate, respectively, the SOP inequalities (7.22) and (7.23).*

Since (7.32) and (7.33) are a particular case of (7.27) and (7.28), one gets immediately the following:

Corollary 7.1 *The Bucket SOP inequalities (7.27) and (7.28) dominate, respectively, the SOP inequalities (7.22) and (7.23).*

The same arguments above can be used to define *simple* (π_B, σ_B)*-cuts* (s-(π_B, σ_B)-cuts) as a particular case of constraints (7.31). To this end, let suppose to be given, again, disjoint subsets of nodes X, Y, with $X \subseteq V \setminus \{q\}$ and $X \prec Y$. Define W and Q as in (7.24) and (7.25), respectively, and define $\tilde{W}$ and $\tilde{Q}$ as in (7.29) and (7.30), respectively. Then, for any $S \in V$ such that $X \subseteq S$ and $Y \subseteq \overline{S}$, we get the following valid inequality for TBF:

$$\sum_{(i,j,b)\in\Delta(B(S)\setminus\tilde{W},B(\overline{S})\setminus\tilde{W})\setminus\tilde{Q}} y_{ij}^b \geq 1. \quad (s\text{-}(\pi_B, \sigma_B)\text{-cuts}) \tag{7.34}$$

Even in this case, the following proposition is straightforward:

Proposition 7.6 *s-(π_B, σ_B)-cuts* (7.34) *dominate* *(π, σ)-cuts* (7.26).

As a consequence, even (π_B, σ_B)-cuts (7.31) dominate (π, σ)-cuts (7.26).

7.4.5 Tournament constraints

Let $P = (v_1, v_2, \ldots, v_h)$, be a directed elementary path in G and let $|P| = h - 1$ denote the number of arcs on P. The path P is called an *infeasible path* if it cannot be contained in any feasible tour. It is known that deciding whether a given path is infeasible or not is $\mathcal{NP}$-complete. However, there are some simple conditions that imply infeasibility. In particular the path P is infeasible if

$$r_{v_1} + \theta(P) > d_{v_h}, \tag{7.35}$$

where $\theta(P) = \sum_{i=1}^{h-1} \theta_{v_i,v_{i+1}}$ denotes the length of the path. In addition, the path P is infeasible if for some node v_t, not on the path, we have both $P' = (v_t, v_1, v_2, \ldots, v_h)$ and $P'' = (v_1, v_2, \ldots, v_h, v_t)$ infeasible. The infeasibility of the paths P' and P'' can again be checked using (7.35). As discussed in Section 7.3.4, given an infeasible path P, the following *infeasible path elimination constraint* (IPEC)

$$\sum_{i=1}^{h-1} x_{v_i,v_{i+1}} \leq |P| - 1, \quad (\text{IPECs}) \tag{7.36}$$

is valid for both BMF and TBF. Furthermore, these inequalities can be strengthened to obtain the so-called *tournament constraints* (TOUR)

$$\sum_{(i,j)\in T(P)} x_{i,j} \leq |P| - 1, \quad (\text{TOURs}) \tag{7.37}$$

where $T(P) = \{(i,j) \in A \,:\, (i,j) = (v_i, v_j) \text{ for some } 1 \leq i < j \leq h\}$ are also valid for both BMF (see [11]) and TBF. Also note that if P_1 and P_2 are both infeasible paths such that P_1 is contained in P_2, then inequalities (7.36) and (7.37) associated with P_2 are dominated by the corresponding inequalities associated with P_1.

Let $S = \{v_1, \ldots, v_h\}$ be the collection of nodes associated with path $P = (v_1, \ldots, v_h)$ and let $\Psi(S)$ denote the collection of all possible paths that visit all nodes in S. As described in [11], if all paths in $\Psi(S)$ are infeasible then the following stronger inequality

$$\sum_{(i,j)\in A(P)} x_{i,j} \leq |P| - 1, \tag{7.38}$$

where $A(P) = \{(i,j) \in A \,:\, (i,j) = (v_i, v_j) \text{ for some } 1 \leq i, j \leq h\}$ is also valid for BMF and TBF. Whenever a violated tournament constraint (7.37) is identified it is customary to try to strengthen it this way.

7.4.6 Bucket tournament constraints

We extend inequalities (7.37) by considering time buckets as follows. Let $P = (v_1, \ldots, v_h)$ be a path with $B_{v_1} = \{b_1, \ldots, \ldots, b_k\}$. Let $b_t \in B_{v_1}$ be the first bucket of v_1 such that $r_{b_t} + \theta(P) > d_{v_h}$. In other words, if the starting bucket at node v_1 is one of $\{b_t, b_{t+1}, \ldots, b_k\}$, then a feasible tour cannot visit the nodes in $\{v_1, \ldots, v_h\}$ in the order specified by P. Note that P itself is not necessarily an infeasible path if the starting bucket at node v_1 is one of $\{b_1, \ldots, b_{t-1}\}$. It is therefore easy to see that the following inequality

$$\sum_{b\in\{b_t, b_{t+1}, \ldots, b_k\}} z^b_{v_1} + \sum_{(i,j)\in T(P)} x_{i,j} \leq |P| \tag{7.39}$$

is valid for the TBF. If one of the buckets in $\{b_t, b_{t+1}, \ldots, b_k\}$ is chosen for node v_1, then inequality (7.39) becomes the tournament constraint (7.38) as P becomes an infeasible path. If, on the other hand, one of the buckets in $\{b_1, \ldots, b_{t-1}\}$ is chosen, then inequality (7.39) is implied by the subtour elimination constraint.

We strengthen these inequalities further. For $v \in V \setminus S$, $S = \{v_1, \ldots, v_h\}$, let $L_v(b_t)$ denote the collection of time buckets at node v that will lead to a starting bucket in $\{b_t, b_{t+1}, \ldots, b_k\}$ if arc (v, v_1) is chosen. More precisely, $L_v(b_t) = \{b \in B_v \,:\, (b, b') \in \mathcal{B} \text{ for some } b' \in \{b_t, b_{t+1}, \ldots, b_k\}\}$. It is now easy to see that $\sum_{v\in V\setminus S} \sum_{b\in L_v(b_t)} y^b_{v,v_1} \geq \sum_{b\in\{b_t, b_{t+1}, \ldots, b_k\}} z^b_{v_1}$. As a consequence, the following *bucket tournament inequality* dominates (7.39):

$$\sum_{v\in V\setminus S} \sum_{b\in L_v(b_t)} y^b_{v,v_1} + \sum_{i\in S:(i,v_1)\in A} x_{i,v_1} + \sum_{(i,j)\in T(P)} x_{i,j} \leq |P| \quad \text{(BTOURs)} \tag{7.40}$$

Proposition 7.7 *Given any elementary path $P = (v_1, \ldots, v_h)$, BTOUR inequality (7.40) is valid for TBF.*

Proof. Notice that if the first term in inequality (7.40) can either be 0 or 1. If it is 1, then the second term has to be zero and therefore the inequality reduces to the valid

inequality (7.39). If, on the other hand, the first term in inequality (7.40) is zero, then the inequality is implied by the subtour elimination constraint. □

7.5 Separation routines

We next outline the main steps of the separation routines we developed for embedding all the inequalities presented in the previous section in a branch-and-cut algorithm based on BTF.

7.5.1 Separating bucket SOPs

In their most general version, SOP inequalities (7.22), (7.23) and (7.26) cannot be separated in polynomial time. However, Ascheuer, Fischetti and Grötschel [11] developed a heuristic polynomial separation procedure which turn outs to be effective in practice, by following some important theoretical results presented by Balas, Fischetti and Pulleyblank [22] in the context of the Precedence-constrained ATSP. Following their main ideas, we developed a heuristic polynomial procedure for bucket SOP inequalities. We next outline the main steps of the separation procedure for s-π_B-cuts (7.32) and s-(π_B, σ_B)-cuts (7.34). All the other families of bucket SOP inequalities are separated by following an identical approach.

Simple π_B-cuts separation

Given a fractional solution y^*, we consider, in turn, each node $v \in V \setminus \{p, q\}$, we initialize $S := \{v\}$ and we apply the following steps:

1. Define the reduced y-arc set $A^0_{\mathcal{B}} = \{(b, b') \in A_{\mathcal{B}} : b \notin \pi_B(S), b' \notin \pi_B(S)\}$. For each arc $(i, j) \in A$, denote as $B_i(j)$ the subset of buckets of B_i which are connected to node j in $A^0_{\mathcal{B}}$: i.e.,

$$B_i(j) = \{b \in B_i : (b, N_j(i, b)) \in A^0_{\mathcal{B}}\}. \tag{7.41}$$

   Then exploit the "linking" constraints (7.17) and compute, for each $(i, j) \in A$,

$$q_{ij} = \sum_{b \in B_i(j)} y^{*b}_{ij}. \tag{7.42}$$

   Build the sparse graph $G^0 = (V^0, A^0)$ in the x-space, with $A^0 = \{(i, j) \in A : q_{ij} > 0\}$ and V^0 induced by A^0.

   Compute the max-flow z^0 from cluster S to q in G^0 (where each arc has a capacity q_{ij}) and denote as $(S', \overline{S}')$ the corresponding min-cut. If $z^0 < 1$, then add the violated s-π_B-cut (7.32) corresponding to S to the cut pool.

2. Note that $S \subseteq S'$ and hence $\pi_B(S) \subseteq \pi_B(S')$. If $\pi_B(S) = \pi_B(S')$, then terminate. Otherwise set $S := S'$ and repeat from Step 1.

The procedure described above also acts as an exact separation for the classical subtour elimination constraints. Whenever the procedure fails in finding violated s-π_B-cuts, no violated SEC can occur in the fractional solution x^*. Note, however, that "extended" subtours in the y-space can occur. A similar approach has been developed for separating s-σ_B-cuts (7.33).

Simple (π_B, σ_B)-cuts separation

Following Ascheuer, Fischetti and Grötschel [11], we separate over constraints (7.34) by considering sets X and Y of the form $X = \{u\}$, $Y = \{w\}$ with $u \prec w$ ($u, w \in V \setminus \{p, q\}$). For each $u, v, w \in V$ such that $u \prec v \prec w$, it can be easily shown that all the s-(π_B, σ_B)-cuts arising from $X = \{u\}$, $Y = \{w\}$ are implied by those arising from $X = \{u\}$, $Y = \{v\}$ and those arising from $X = \{v\}$, $Y = \{w\}$. Hence, at any time we look for violated s-(π_B, σ_B)-cuts, we consider only the pairs of nodes u, w, $u \prec w$, such that the precedence between u and w cannot be inferred by any transitive relationship with some other node v. Given a fractional solution y^*, we consider, in turn, each pair of node u, w with the above mentioned properties (i.e., $u \prec w$ and no node v exists such that $u \prec v \prec w$), and we apply the following steps:

1. Define $\tilde{W}$ and $\tilde{Q}$ as in (7.29) and (7.30), respectively. Build the reduced y-arc set $A^0_{\mathcal{B}} = \{(b, b') \in A_{\mathcal{B}} : b \notin \tilde{W}, b' \notin \tilde{W}, (v(b), v(b'), b) \notin \tilde{Q}\}$, where for any bucket $\beta \in \mathcal{B}$, $v(\beta)$ denotes the node in V containing β. For each arc $(i, j) \in A$, compute $B_i(j)$ as in (7.41) and then q_{ij} as in (7.42).

2. Build the sparse graph $G^0 = (V^0, A^0)$ in the x-space, with $A^0 = \{(i, j) \in A : q_{ij} > 0\}$ and V^0 induced by A^0.

3. Compute the max-flow z^0 from u to w in G^0 (where each arc has a capacity q_{ij}) and denote as $(S', \overline{S}')$ the corresponding min-cut. If $z^0 < 1$, then add the violated s-π_B-cut (7.34) corresponding to S, u, w to the cut pool. In any case terminate.

As highlighted in the above detailed description of the separation procedures, the *simple* bucket SOPs allow to exploit the potential of the extended y-space for strengthening the classical SOP inequalities, while keeping the computation of max-flow in the x space. Similar procedure have been implemented for separating the more general version of bucket SOPs (i.e., inequalities (7.27), (7.28) and (7.31)). However, for those kind of constraints we are forced to explicitly work in the extended space.

7.5.2 Separating tournament and bucket tournament constraints

Given any fractional solution x^*, it can be proven that at most a polynomial number of (feasible or infeasible) elementary paths corresponding to violated tournament constraints (7.37) can occur. (This result is due to Savelsbergh and reported by Ascheuer, Fischetti and Grötschel [10].) Hence, all the violated tournament constraints can be enumerated in polynomial time. However, given a path P corresponding to a violated tournament, deciding if P is infeasible is $\mathcal{NP}$-complete. On the other hand, it can also be proven the following statement:

Proposition 7.8 *If x^* is integral and does not contain any cycle, then it is a feasible solution for ATSPTW iff, for any path $P = (v_1, \ldots, v_h)$ corresponding to a violated tournament constraints, $R_{v_1} + \theta(P) \leq D_{v_h}$.*

In practice, the results of Savelsbergh and the above proposition imply that a polynomial separation of tournaments constraints suffices to provide a correct formulation for ATSPTW. Exploiting this result, we developed the following separation procedure, similar to the one proposed in [11].

Given a fractional solution x^*, for each node $v \in V \setminus \{p, q\}$ we enumerate all the elementary paths on x^* ending at v, by following the flow in the fractional solution x^* through a depth-first strategy. For each $v \in V$, we implicitly explore a decision tree in which any node corresponds to an elementary path P ending at v. At each node of the tree, three different situations can occur:

(i) the TOUR (7.37) corresponding to P is not violated $\Rightarrow$ we stop extending the path P;

(ii) the TOUR corresponding to P is violated and P is infeasible according to the trivial check (7.35) $\Rightarrow$ we add the constraint and we stop extending the path P (by further extending the path we can only find dominated constraints);

(iii) the TOUR corresponding to P is violated but P is not proven to be infeasible according to the check (7.35) $\Rightarrow$ we extend P following the flow in x^*.

As discussed, the above procedure is polynomial and acts as an exact method whenever x^* is integer and with no cycles (note however that the procedure is heuristic if x^* is fractional). As it is customary, whenever we succeed in finding a violated TOUR, we try to strengthen it as a constraint (7.38).

Bucket tournament constraints (7.40) can effectively be separated by slightly modifying the procedure described above: at any node of the decision tree, if the current TOUR (7.37) is violated but P is not proven to be infeasible, we check if the bucket tournament (7.40) corresponding to P is also violated, and in the case we add it to the cut pool. Interestingly, note that, in our separation procedure, we stop extending the path as soon as there is no hope to find a violated tournament constraint. Whenever this happens, there is also no hope to find a violated bucket tournament (7.40).

7.6 Building the formulation

In order to exploit the potential of Time Bucket Formulation, we have to devise an effective strategy for building the buckets. In addition, we have to derive a profitable set of bucket-node precedences for separating Bucket SOP inequalities. Furthermore, it is also worth mentioning that Ascheuer, Fischetti and Grötschel [11] have shown that a deep preprocessing phase is of crucial importance for strengthening any ATSPTW formulation. To this end, one can first try to exploit the time windows to infer some precedence relationships among nodes. Then, the detected node precedences can be

used for fixing some x variables (i.e., for removing some arcs), while the removal of some arcs might allow to tighten the time windows themselves.

Assume w.l.o.g. that the triangle inequality holds for the traveling time matrix (i.e., $\theta_{ij} + \theta_{jk} \geq \theta_{ik}$ for any $i, j, k \in V \setminus \{p, q\}$, $i \neq j \neq k$). Given an ATSPTW instance, we perform the following preprocessing in order to build a possibly strong formulation.

1. (Node preprocessing.) We apply the same node preprocessing proposed in [11] which can be briefly summarized as follows[3]. First, we set $p \prec j \ \forall \ j \in V \setminus \{p\}$ and $j \prec q \ \forall \ j \in V \setminus \{q\}$. Then we iterate the following steps until we are able to enforce the formulation:

   (a) For each $i, j \in V \setminus \{p, q\}$, if $R_j + \theta_{ji} > D_i$, set $i \prec j$.

   (b) For each i, j, k such that $i \prec j$ and $j \prec k$, set $i \prec k$.

   (c) For each i, j such that $i \prec j$, remove arc (j, i). For each i, j, k such that $i \prec j \prec k$, remove arc (i, k).

   (d) Tightening the time windows through the following rules (see [11]):

$$
\begin{array}{ll}
R_k := \max\{R_k, \min_{i \in V^-(k)}\{R_i + \theta_{ik}\}\} & \forall\ k \in V \text{ s.t. } V^-(k) \neq \emptyset \\
R_k := \max\{R_k, \min\{D_k, \min_{j \in V^+(k)}\{R_j - \theta_{kj}\}\}\} & \forall\ k \in V \text{ s.t. } V^+(k) \neq \emptyset \\
D_k := \min\{D_k, \max\{R_k, \max_{i \in V^-(k)}\{D_i + \theta_{ik}\}\}\} & \forall\ k \in V \text{ s.t. } V^-(k) \neq \emptyset \\
D_k := \min\{D_k, \max_{j \in V^+(k)}\{D_j - \theta_{kj}\}\} & \forall\ k \in V \text{ s.t. } V^+(k) \neq \emptyset
\end{array}
$$

   (e) Arc removal based on 3-infeasible paths. For any $(i, j) \in A$, if there exists $k \in V$ such that (i, j, k) and (k, i, j) are both infeasible, then arc (i, j) can be removed.

   After this step, we end up with a sparse graph, a set of tightened time windows and a set of node precedences.

2. (Building the buckets.) We build buckets by exploiting the sparse graph obtained at Step 1 in the following way.

   For each node $i \in V$ and for each $t \in W_i$, we compute the sets $V_t^+(i) := \{j \in V^+(i) : t + \theta_{ij} \leq D_j\}$ and $V_t^-(i) := \{j \in V^-(i) : I_j(i, t) \neq \emptyset\}$. $V_t^+(i)$ is the set of nodes which can follow node i if the start time at node i is t. $V_t^-(i)$ is the set of nodes which can precede node i if the arrival time at node i is t.

   Then we build the buckets as follows. The first bucket of node i is defined as $b_1 := [R_i, R_i]$. The other buckets are computed by aggregating in intervals consecutive time instants t, t' such that $V_t^+(i) = V_{t'}^+(i)$ and $V_t^-(i) = V_{t'}^-(i)$.

   It is worth noting that holes are detected whenever $V_t^+(i) = \emptyset$ or $V_t^-(i) = \emptyset$, where a hole for node i is defined as a time instant $t \in W_i$ such that the start time at node i cannot be t in any feasible ATSPTW solution. In several cases, the buckets we obtain through such a procedure are nonconsecutive time intervals.

[3] The reader is referred to [11] for a detailed description of this steps.

3. (Imposing "bucket triangle inequality".) Consider three different nodes $i, j, k \in V$ and a bucket $b \in B_i$. Then, in the *complete* graph (i.e., the one provided on input before node preprocessing), let $\overline{b} = N_k(i, b)$ be the bucket of node k which is visited if selecting arc (i, k) and visiting node i at bucket b. In the same way, define $\tilde{b} = N_j(i, b)$ and $\hat{b} = N_k(j, \tilde{b})$. $\hat{b}$ is the bucket of node k which is visited if the path (i, j, k) is selected and node i is visited at bucket b. If, for any choice of nodes $i, j, k \in V$ $(i \neq j \neq k)$ and for any choice of $b \in B_i$, $\hat{b}$ does not precede $\overline{b}$, then the graph $G' = (\mathcal{B}, A_{\mathcal{B}})$ is said to be "bucket triangular".

   Bucket triangle inequality seems to be relevant for getting a strong Time Bucket Formulation. Indeed, if this property is not satisfied, negative waiting times introduced by buckets can easily "cheat" the LP relaxation. Further, bucket triangle inequality is also required to infer in a very simple way a profitable set of bucket-node precedences for separating Bucket SOP inequalities.

   Since the heuristic outlined at Step 2 typically provides a graph $G' = (\mathcal{B}, A_{\mathcal{B}})$ not bucket triangular, then we impose this property by splitting the buckets in a careful way, i.e., in the attempt of imposing such a condition without increasing too much the number of buckets and hence the size of the formulation.

4. (Bucket preprocessing.) Given the sparse graph and the node precedences obtained at Step 1, and given the buckets computed at Steps **??**–3, we apply a further preprocessing step, with the aim of removing bucket-arcs (i.e., y variables) and computing a suitable set of bucket-node precedences. This is accomplished through the following steps.

   (a) For any $i, j \in V$ such that $i \prec j$, set $b \prec j$ for all buckets $b \in B_i$ and $i \prec b$ for all buckets $b \in B_j$.

   (b) For any $i, j \in V$ and $b \in B_i$ such that $r_b + \theta_{ij} > D_j$, set $j \prec b$. For any $i, j \in V$ and $b \in B_j$ such that $R_i + \theta_{ij} > d_b$, set $b \prec i$.

   (c) For any $b \in B_i$ and any $j, k \in V$ such that $b \prec j$ and $j \prec k$, set $b \prec k$. For any $b \in B_i$ and any $j, k \in V$ such that $j \prec k$ and $k \prec b$, set $j \prec b$.

   (d) For any $b \in B_i$ and any $j \in V$ such that $b \prec j$, remove all bucket-arcs (b', b) with $b' \in B_j$. For any $b \in B_i$ and any $j \in V$ such that $j \prec b$, remove bucket-arc (b, b') with $b' \in B_j$.

   (e) For any $b \in B_i$ and any $j, k \in V$ such that $b \prec j \prec k$, remove bucket-arc (b, b') with $b' \in B_k$. For any $b \in B_i$ and any $j, k \in V$ such that $j \prec k \prec b$, remove all bucket-arcs (b', b) with $b' \in B_j$.

   (f) Bucket-arc removal based on 3-infeasible paths. For any $(b, b') \in A_{\mathcal{B}}$ connecting nodes i and j (i.e., $b \in B_i$ and $b' \in B_j$), if there exists $k \in V$ such that path (k, i, j) is infeasible and $r_b + \theta_{ij} + \theta_{jk} > D_k$, then bucket-arc (b, b') can be removed.

7.7 Computational results

Time bucket formulation has been tested within the CPLEX 10.0 [105] branch-and-cut framework. The branch-and-cut has been tested on a Pentium M 1.86 GHz notebook with 1 GByte RAM running under Microsoft Windows XP Operative System, and has been coded in `C++` with Microsoft Visual `C++` 6.0 compiler.

In this first set of experiments, no "ad hoc heuristic" or branching strategy has been implemented. In particular, we used default branching and we disabled all the CPLEX heuristics and all the CPLEX cutting planes, since the LP relaxation seems to be hard to manage and CPLEX heuristics often fall in troubles, spending a lot of time without finding any feasible solution.

All the cutting planes discussed in Section 7.4 has been embedded in the branch-and-cut by means of CPLEX callbacks. After some preliminary tests, we decided to separate cutting planes at each node of the search tree in the following order. We first separate s-π_B-cuts (7.32), then s-σ_B-cuts (7.33), then s-(π_B, σ_B)-cuts (7.34). Finally, we separate TOURs (7.37) and BTOURs (7.40) at the same time. Classical ATSPTW cutting planes (i.e., SOPs and SECs) are not separated, as dominated by simple bucket SOPs. Concerning the more general version of bucket SOPs (i.e., (7.27), (7.28), (7.31)), we decided to not separate them after several preliminary experiments, because it seems rather difficult to effectively exploit them. On the one side, a huge number of general violated bucket SOPs can usually be found even when the heuristic procedure for simple bucket SOPs fails in finding violated inequalities. On the other hand, those cuts are time consuming to separate and the benefit they provide in the lower bound is often negligible. Among the three families (classical, simple bucket and general bucket SOPs), the second one really seems to be the best, because it allows to exploit the potential of buckets by working in the x-space.

As testbed, we selected the same 50 ATSPTW instances considered by Ascheuer, Fischetti and Grötschel [11]. These instances derive from a practical scheduling application and have a number of nodes varying from 12 to 233. Looking at the results reported in [11], they can be divided in two completely different classes. The branch-and-cut proposed in [11] can easily solve 32 out of the 50 problems in a at most a few minutes, while cannot solve the remaining 18 problems in 5 hours of CPU time. As far as we known,, among these 18 hard problems, `rbg042a` as been solved to proven optimality by Focacci, Lodi and Milano [89], while the remaining 17 instances are still unsolved.

Table 7.1 report the results on the 32 so-called easy instances, comparing the method proposed in [11] (AFG) with TBF. For each problem, in the first columns the table reports the size of the instance (i.e., number of nodes and number of arcs) and the optimal solution value. Then, for each of the compared methods, we report the lower bound at the root node (after the addition of cutting planes and before branching), the number of branch-and-cut nodes enumerated and the overall CPU time in seconds (see the table key).

The table shows that both AFG and TBF are able to easily deal with these instances, as they both provide a strong lower bound at the root node. Comparing the CPU time is

Table 7.1: Time Bucket Formulation vs. Ascheuer et al. [11]: comparison on easy instances.

				Ascheuer et al. [11] (AFG)			Time Bucket Formulation (TBF)		
Prob.	$\|V\|$	$\|A\|$	opt	rLB	#nodes	CPU	rLB	#nodes	CPU
rbg010a	12	54	149	99.3	2	0.1	100.0	1	0.0
rbg016a	18	79	179	98.9	2	0.2	100.0	0	0.0
rbg016b	18	167	142	93.7	76	8.8	97.2	2	0.2
rbg017.2	17	200	107	100.0	0	0.0	100.0	0	0.0
rbg017a	19	176	146	100.0	0	0.1	100.0	0	0.0
rbg017	17	122	148	100.0	4	0.8	99.3	0	0.0
rbg019a	21	71	217	100.0	0	0.0	100.0	0	0.0
rbg019b	21	211	182	98.9	820	54.6	99.5	1	0.3
rbg019c	21	229	190	95.8	58	8.7	96.8	42	0.9
rbg019d	21	156	344	99.7	2	0.8	100.0	6	0.2
rbg020a	22	95	210	100.0	0	0.2	100.0	0	0.0
rbg021.2	21	237	182	100.0	0	0.2	100.0	0	0.1
rbg021.3	21	256	182	97.8	340	27.2	98.4	62	2.7
rbg021.4	21	264	179	98.9	72	5.8	100.0	1	0.2
rbg021.5	21	268	169	98.8	76	6.6	100.0	1	0.3
rbg021.6	21	358	134	99.3	2	1.4	100.0	1	0.3
rbg021.7	21	375	133	96.2	24	4.3	100.0	0	0.6
rbg021.8	21	380	132	97.7	254	17.4	98.5	10	1.4
rbg021.9	21	380	132	97.0	320	26.1	98.5	23	2.8
rbg021	21	229	190	95.8	58	8.8	96.8	42	0.9
rbg027a	29	479	268	99.3	6	2.3	99.3	2	1.4
rbg031a	33	388	328	100.0	0	1.7	100.0	0	0.2
rbg033a	35	421	433	100.0	0	1.9	99.8	6	0.9
rbg034a	36	535	403	99.5	2	1.0	100.0	6	1.9
rbg035a.2	37	940	166	95.2	96	64.8	100.0	3	5.3
rbg035a	37	477	254	100.0	0	1.8	100.0	2	0.2
rbg038a	40	486	466	100.0	13204	4232.2	100.0	9	1.5
rbg040a	42	539	386	92.0	1756	751.8	96.6	25	3.6
rbg050a	52	1629	414	100.0	6	18.6	100.0	2	24.5
rbg055a	57	765	814	99.9	2	6.4	100.0	19	3.5
rbg067a	69	843	1048	99.9	2	6.0	100.0	23	3.6
rbg125a	127	1824	1409	99.5	56	229.8	100.0	8	9.6
avg.				98.5	538.8	171.6	99.4	9.3	2.1

Key to Table 7.1

$\|V\|$	:	number of nodes
$\|A\|$	:	number of arcs after the preprocessing
opt	:	optimal solution value found by both AFG and BTF
rLB	:	lower bound at the root node before branching
#nodes	:	number of B&C nodes
CPU	:	overall CPU seconds obtained: on a SUN SPARC Station 10 for AFG on a Pentium M 1.86 GHz for BTF

rather difficult, due to the different machines[4], but the number of B&C nodes explored by TBF is usually much smaller.

Table 7.2 report the results on the 18 so-called hard instances, comparing again the two methods. Since we are mostly interested in understanding the capability of TBF to provide strong lower bounds, in this set of experiments we decided to provide

[4]The results reported for [**?**] have been obtained on a SUN SPARC Station 10 by using the branch-and-cut framework ABACUS [1].

to the method the best known as initial cutoff(i.e., we set as initial cutoff bestUB+1). For each problem, the table reports the size of the instance and the best UB known in the literature (the proven optimal solution in case for `rbg042a`). Then, for each of the compared methods, we report the best found UB, the lower bound at the root node (again, after the addition of cutting planes and before branching), the best available lower bound provided at the end of the computation, the number of branch-and-cut nodes enumerated and the overall CPU time in seconds. For TBF, we also report the CPU time spent in separation routines and the %gap closed with respect to AFG by considering both the root LB and the global LB (see the key of the table).

Table 7.2: Time Bucket Formulation vs. Ascheuer et al. [11]: comparison on hard instances.

				Ascheuer et al. [11] (AFG)				Time Bucket Formulation (TBF)							
Prob.	$\|V\|$	$\|A\|$	bestUB	UB	rLB	gLB	#nodes	UB	rLB	gLB	#nodes	totsecs	sepsecs	r %gap.	g %gap
rbg041a	43	628	403	417	361	382	23396	**402**	386	402	335	28.4	3.5	60.98	100.00
rbg042a	44	762	411	435	394	409	22300	**411**	403	411	1724	864.2	18.5	52.94	100.00
rbg048a	50	1288	492	527	454	455	25222	—	457	458	2080	> 10800	89.9	7.89	8.11
rbg049a	51	1083	488	501	408	418	17486	—	418	426	3715	> 10800	131.3	12.50	11.43
rbg050b	52	1175	527	542	447	453	8600	—	456	460	4081	> 10800	143.3	11.25	9.46
rbg050c	52	1396	536	536	507	509	25184	—	508	508	1227	> 10800	69.6	3.45	-3.70
rbg086a	88	926	1052	1052	1042	1049	12208	**1051**	1048	1051	30	7.3	2.2	66.67	100.00
rbg092a	94	1367	1109	1111	1084	1102	8828	**1093**	1089	1093	154	269.9	14.2	55.56	100.00
rbg132.2	132	3126	1125	1125	1053	1069	4336	**1083**	1078	1083	69	1474.1	28.1	83.33	100.00
rbg132	132	1575	1400	1400	1323	1348	7628	**1360**	1345	1360	72	13.2	5.3	59.46	100.00
rbg152.3	152	6191	1594	1594	1521	1525	2558	1547	1537	1538	75	> 10800	103.6	61.54	59.09
rbg152	152	2125	1792	1792	1759	1770	5038	**1783**	1776	1783	138	82.8	24.1	70.83	100.00
rbg172a	174	2837	1897	1897	1777	1787	3434	**1799**	1794	1799	120	176.2	41.4	77.27	100.00
rbg193.2	193	6031	2093	2093	1969	1981	1726	—	2009	2009	100	> 10800	176.2	32.26	25.00
rbg193	193	3050	2452	2452	2386	2388	2790	**2414**	2408	2414	4465	9542.8	1073.1	78.57	100.00
rbg201a	203	3287	2296	2296	2158	2159	3282	**2189**	2184	2189	272	551.8	120.1	83.87	100.00
rbg233.2	233	7588	2304	2304	2146	2152	1200	—	2183	2185	40	> 10800	116.0	23.42	21.71
rbg233	233	3766	2786	2786	2635	2647	1106	**2689**	2683	2689	4051	5844.4	1104.7	88.89	100.00
avg.							9795.7				1263.8	5247.5	181.4	51.70	68.39

Key to Table 7.2

$\|V\|$	:	number of nodes
$\|A\|$	:	number of arcs after the preprocessing
bestUB	:	best available upper bound from the literature. We used bestUB+1 as initial cutoff for TBF
UB	:	best upper bound found by AFG vs. best upper bound found by TBF
rLB	:	lower bound at the root node before branching
gLB	:	best available lower bound at the end of the computation 5 hours of time limit on a SUN SPARC Station 10 for AFG 3 hours of time limit on a Pentium M 1.86 GHz for TBF
#nodes	:	number of B&C nodes
totsecs	:	overall CPU seconds for TBF
sepsecs	:	CPU seconds spent in separation routines by TBF
r %gap	:	TBF root lower bound improvement: (100*(TBF rLB - AFG rLB))/(best UB - AFG rLB)
g %gap	:	TBF global lower bound improvement: (100*(TBF gLB - AFG gLB))/(best UB - AFG gLB)

The table shows that in this case TBF clearly outperforms AFG. First, TBF can solve 11 out of the 18 problems, and often the CPU time is just a few minutes. Second, the root LB is generally much stronger, thus dramatically reducing the number of B&C node required to solve the problem. As already stated, we deal with the absence of any primal heuristics by setting the initial cutoff as the best known solution value + 1. However, it is worth noting that in most cases such best upper bound is really far from the optimal solution value, and that TBF branch-and-cut typically find the optimal lower bound after few nodes, spending then the rest of the time to find a feasible solution matching the lower bound. This also confirms the difficulty of ATSPTW in practice.

Despite these convincing results, there are 7 instances still unsolved. On the first

4 instances, the improvement in the global lower bound with respect to AFG is much smaller than in the other cases (and, in particular, for instance `rbg050c` the global lower bound by AFG is better than the one by BTF). Probably, for such unsolved problems, the heuristic devised for building the formulation is not effective, and some other work is required in this direction. On the other 3 problems, instead, the formulation seems to be strong, but the number of B&C nodes enumerated in three CPU hours is really too small. Also this issue asks for devising a more clever way to define the formulation (i.e., the buckets).

The last table, Table 7.3, is aimed to understand the effectiveness of the bucket inequalities. Considering the 18 hard instances and solving the root node only, we compare the lower bound provided by the classical cuts which do not exploit the buckets in any way (i.e., classical SECs first, then SOPs (7.22), (7.23), (7.26) and finally TOURs (7.37)) with the lower bound obtained through the default setting.

Table 7.3: Node cuts vs. bucket cuts at the root node: comparison on hard instances.

				TBF + Node cuts			TBF + Bucket cuts			
Prob.	$\|V\|$	$\|A\|$	bestUB	rLB	#cuts	CPU	rLB	#cuts	CPU	r %gap
rbg041a	43	628	402	382.0000	3	0.3	385.9375	61	0.8	19.69
rbg042a	44	762	411	397.6071	14	1.7	402.0943	86	3.1	33.50
rbg048a	50	1288	492	457.0000	7	10.1	457.0000	51	18.7	0.00
rbg049a	51	1083	488	414.1707	4	2.5	417.5649	43	4.6	4.60
rbg050b	52	1175	527	451.3000	22	10.5	455.0559	80	16.9	4.96
rbg050c	52	1396	536	508.0000	8	22.7	508.0000	44	19.3	0.00
rbg086a	88	926	1051	1046.0781	40	1.1	1047.8000	102	2.0	34.98
rbg092a	94	1367	1093	1086.6458	59	4.9	1088.9544	163	8.0	36.33
rbg132.2	132	3126	1083	1073.6000	63	62.6	1077.7500	167	101.2	44.15
rbg132	132	1575	1360	1344.1667	44	2.2	1344.5000	59	2.8	2.11
rbg152.3	152	6191	1547	1534.9167	96	2306.8	1537.0000	211	1133.0	17.24
rbg152	152	2125	1783	1775.0556	44	3.8	1775.5000	70	4.9	5.59
rbg172a	174	2837	1799	1791.0000	91	12.8	1793.7000	155	18.6	33.75
rbg193.2	193	6031	2093	2006.8500	61	262.7	2008.5159	278	751.9	1.93
rbg193	193	3050	2414	2405.0000	40	10.5	2407.4286	116	13.5	26.98
rbg201a	203	3287	2189	2181.0000	108	16.0	2183.7000	153	22.7	33.75
rbg233.2	233	7588	2304	2177.6833	53	496.3	2182.6429	139	425.7	3.93
rbg233	233	3766	2689	2679.0000	56	22.7	2682.4286	238	46.8	34.29
avg.					45.2	180.6		123.1	144.1	18.77

Key to Table 7.3

$\|V\|$	:	number of nodes
$\|A\|$	:	number of arcs after the preprocessing
bestUB	:	best available upper bound (including also results from Table 7.2)
nLB	:	optimal solution of the LP relaxation + node cuts
bLB	:	optimal solution of the LP relaxation + bucket cuts
#cuts	:	number of generated cuts
CPU	:	overall CPU seconds for solving the root node
r %gap	:	percentage improvement obtained by the bucket cuts: (100*(bLB - nLB))/(bestUB - nLB)

The table shows that the buckets are indeed very useful for strengthening the formulation, and the theoretical dominance of bucket cuts w.r.t. node cuts is confirmed by the results. Behind an average gap closed of 18.77%, there are several cases in which

this value raises to more than 30%. On the other hand, there are several cases in which the improvement provided by bucket cuts is less than 5%. Interestingly, the cases in which the improvement seems to be not so relevant are mostly related to the instances that TBF cannot solve, thus confirming the fact that a more clever exploitation of the "bucket idea" is required.

7.8 Conclusions

In this chapter we presented an extended formulation for the Traveling Salesman Problem with Time Windows (TSPTW), a well known generalization of the classical TSP where each node must be visited within a given time window. In particular, we proposed a quite general idea which is based on a relaxed discretization of time windows. Such an idea leads to a strong formulation and to strong valid inequalities which can been effectively exploited within a classical branch-and-cut framework.

The overall branch-and-cut algorithm has been tested on hard benchmark instances from the literature, arising from a practical scheduling application, and the results have shown that the proposed formulation is effective in practice for tackling TSPTW. Interestingly, several unsolved benchmark instances have been solved for the first time.

As discussed, the formulation is based on the idea of dividing time windows in buckets (intervals). In this chapter, we proposed a first heuristic idea to devise buckets. Future works should address more clever ideas for building the buckets, as it seems to be of crucial importance for the effectiveness of the overall method. Probably, the formulation itself can be exploited in this sense, and primal and dual solutions of its LP relaxation should be investigated to derive a dynamic approach for the problem of finding an "almost optimal" set of buckets.

On the other hand, other directions of work might concern the possibility of extending the presented approach to other contexts in which the time window component seems to be relevant.

Chapter 8

Improving on Branch-and-Cut Algorithms for Generalized Minimum Spanning Trees

8.1 Introduction

Several variants of Generalized Minimum Spanning Tree Problems (GMSTPs) have been introduced in the literature in different papers by a number of authors. Roughly speaking, all these variants are generalizations of the classical *Minimum Spanning Tree Problem* (MSTP, see, e.g., [120]) on an undirected graph $G = (V, E)$ in which the node set V is partitioned into a given set of clusters[1], and the minimum tree has to "span" those clusters instead of simple nodes.

In particular, in this chapter[2] we are concerned with two specific variants, the most classical one in which *Exactly* one node in each cluster has to be visited (E-GMSTP), and the less studied problem in which *at Least* one node in each cluster has to be reached (L-GMSTP).

More precisely, V is partitioned into $|K|$ clusters V_k, $k \in K$. Each edge $e = \{i, j\} \in E$ has a cost $c_e \in \mathbb{R}^+$. The E-GMSTP is the problem of finding a minimum cost tree including exactly one node from each node set of the partition (see Figure 8.1 for a feasible solution of E-GMSTP). This problem was introduced by Myung, Lee and Tcha [127] who have shown it is strongly $\mathcal{NP}$-hard by a reduction from the *node-cover* problem. Mathematical formulations and exact methods have been discussed by Myung, Lee and Tcha [127] and Pop, Kern, and Still [142, 144], while a polynomial approximation algorithm has been proposed by Pop, Kern and Still [143].

In the L-GMSTP, instead, at least one node from each cluster of the partition must be included in the minimum cost tree (see Figure 8.2 for a feasible solution of L-GMSTP). This problem was introduced by Ihler, Reich and Widmayer [104] as a

[1] A variant in which the clusters may overlap is considered in [73].

[2] The results of this chapter appear in: C. Feremans, A. Lodi, P. Toth and A. Tramontani, "Improving on branch-and-cut algorithms for generalized minimum spanning trees", *Pacific Journal of Optimization*, 1:491–508, 2005 [80].

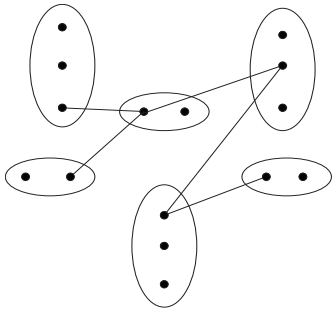

Figure 8.1: Feasible solutions for E-GMSTP.

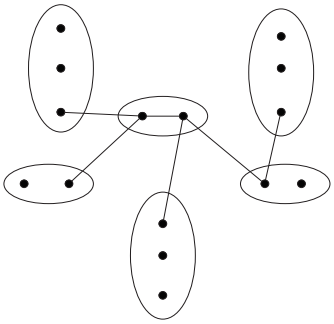

Figure 8.2: Feasible solutions for L-GMSTP.

particular case of the *Generalized Steiner Tree Problem* under the name *"Class Tree Problem"*. Ihler, Reich, Widmayer [104] have shown that the decision version of the L-GMSTP is $\mathcal{NP}$-complete even if G is a tree, and that there is no constant worst-case ratio polynomial-time algorithm unless $\mathcal{P} = \mathcal{NP}$, even if G is a tree on V with edge lengths 1 and 0. Heuristic algorithms have been proposed by Ihler, Reich, Widmayer [104] and by Dror, Haouari and Chaouachi [72].

The L-GMSTP problem reduces at a first glance to the E-GMSTP when the triangle inequalities hold, but this is not true as shown by the example in Figure 8.3. Indeed,

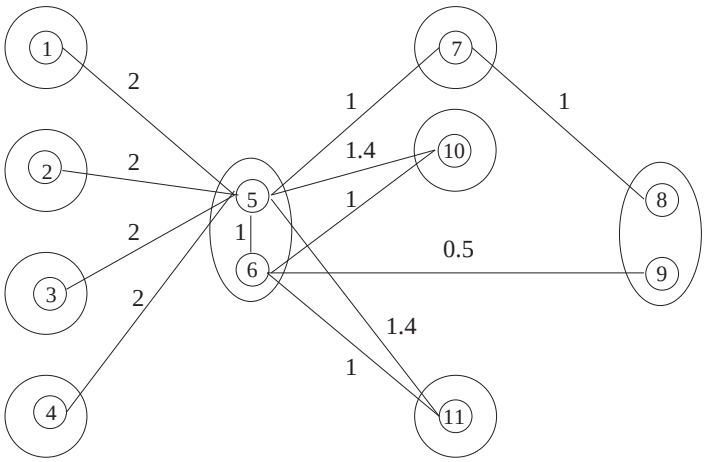

Figure 8.3: A graph for which E-GMSTP and L-GMSTP differ.

if the graph depicted in Figure 8.3 is completed through shortest paths, it satisfies the triangle inequalities and the value of the optimal solution of L-GMSTP is 12.5 (using both nodes 5 and 6), while the optimal value of the E-GMSTP solution is 12.8 (only using node 5).

Applications modeled by E-GMSTP are encountered in telecommunications, where metropolitan and regional networks must be interconnected by a tree containing a gateway from each network. For this internetworking, a node has to be chosen in each

local network as a hub and the hub nodes must be connected via transmission links such as optical fiber (see Myung, Lee and Tcha [127] for details).

The L-GMSTP has been used to solve an important real life network design problem arising in desert environments and consisting in designing a minimal length irrigation network which connects at least one node from each parcel to a water source (see Dror, Haouari and Chaouachi [72] for details).

This chapter presents several effective techniques to improve on the branch-and-cut approaches for E-GMSTP and L-GMSTP proposed by Feremans, Labbé and Laporte [78] and by Feremans [77] respectively. In particular, we improved on the performances through: *i*) new effective heuristic algorithms, *ii*) updated branching strategies, and *iii*) the use of general-purpose Chvátal-Gomory cuts (with and without the strengthening procedures proposed by Letchford and Lodi [112]).

Finally, a generalization of both problems requiring some clusters to be visited exactly once and the remaining clusters at least once is presented. Such a generalization is denoted as E/L-GMSTP and naturally appears when the considered network is somehow "mixed", i.e., involving clusters which may require a different behavior (fixed-charge costs).

The chapter is organized as follows. In Section 8.2 the Integer Linear Programming (ILP) formulation for E-GMSTP discussed in [78] and tested in [79] is recalled. In Section 8.3 an ILP formulation for L-GMSTP is proposed and its relationship with the one for E-GMSTP is discussed, while Section 8.4 discusses the proposed generalized problem. In Section 8.5 computational experiments are reported showing the effectiveness of the proposed techniques. Some conclusions are drawn in Section 8.6.

8.2 ILP Formulation for E-GMSTP

Myung, Lee and Tcha [127] have provided two basic formulations for the E-GMSTP using two sets of binary variables, namely $x_e, \forall e \in E$ and $y_i, \forall i \in V$. In the first formulation, called *ucut*, connectivity is ensured by *cutset constraints* of the form $x(\delta(S)) \geq y_i + y_j - 1$ $(i \in S \subset V, j \notin S)$, whereas in the second, called *usub*, cycles are prevented through *subpacking constraints* of the form $x(E(S)) \leq y(S) - 1$ $(S \subset V, \mu(S) \neq 0)$, where for any $S \subseteq V$ we define $\mu(S) = |\{k : V_k \subseteq S\}|$, i.e., the number of clusters included in S. As it is customary, for any $S \subseteq V$, $\delta(S)$ (resp. $E(S)$) denotes the set of edges having exactly one endpoint (resp. both endpoints) in S, and $x(\delta(S))$ (resp. $x(E(S))$) denotes the sum of the x-values on the subset $\delta(S)$ (resp. $E(S)$).

The undirected cutset formulation uses the fact that a feasible Generalized Spanning Tree (E-GST) is a connected subgraph of G containing one node per cluster and $|K|-1$ edges:

Undirected cutset formulation (ucut)

$$\min \sum_{e \in E} c_e x_e \tag{8.1}$$

subject to

$$y(V_k) = 1 \qquad k \in K, \tag{8.2}$$

$$x(E) = |K| - 1, \tag{8.3}$$

$$x(\delta(S)) \geq y_i + y_j - 1 \qquad i \in S \subset V, j \notin S, \tag{8.4}$$

$$x_e \in \{0,1\} \qquad e \in E, \tag{8.5}$$

$$y_i \in \{0,1\} \qquad i \in V. \tag{8.6}$$

Constraints (8.2) guarantee that each cluster is visited exactly once, while constraint (8.3) forces the tree structure. As already mentioned, constraints (8.4) assure connectivity. Finally, constraints (8.5) and (8.6) are the integrality requirements.

An E-GST can also be defined as an acyclic subgraph of G containing one node per cluster and $|K| - 1$ edges:

Undirected subpacking formulation (usub)

$$\min \sum_{e \in E} c_e x_e$$

subject to

$$\begin{aligned} y(V_k) &= 1 && k \in K, \\ x(E) &= |K| - 1, \\ x(E(S)) &\leq y(S) - 1 && S \subset V, \mu(S) \neq 0, \\ x_e &\in \{0,1\} && e \in E, \\ y_i &\in \{0,1\} && i \in V. \end{aligned} \tag{8.7}$$

The model is equivalent to the previous one with the only difference of the connectivity constraints (8.4) replaced by constraints (8.7).

Myung, Lee and Tcha [127] have also proved that $\mathcal{P}_{usub} \subseteq \mathcal{P}_{ucut}$, i.e., that the subpacking formulation dominates the cutset one in terms of continuous relaxation. The example depicted in Figure 8.4, provided by Magnanti and Wolsey [120] in the context of the Minimum Spanning Tree Problem, can also be used to show (with $|K| = 5, V_k = \{k\}\ \forall k \in K$) that this inclusion is strict. Indeed (8.2), (8.3) and (8.4) are satisfied while (8.7) is violated for $S = \{3,4,5\}$.

The subpacking formulation is the one used by Feremans, Labbé and Laporte [79] as a base for a branch-and-cut approach and such an approach is elaborated in Section 8.5.1.

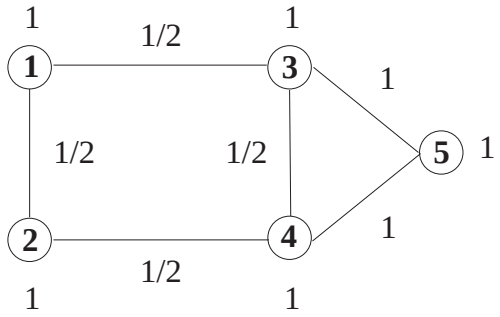

Figure 8.4: Example showing that $\mathcal{P}_{usub} \subset \mathcal{P}_{ucut}$.

8.3 ILP Formulation for L-GMSTP

The L-GMSTP can be formulated as an integer linear program as follows.

$$\min \sum_{e \in E} c_e x_e$$

s.t.

$$\begin{aligned}
y(V_k) &\geq 1 && k \in K, && (8.8)\\
x(E) &= y(V) - 1, && && (8.9)\\
x(\delta(S)) &\geq y_i + y_j - 1 && i \in S \subset V, j \notin S,\\
x_e &\in \{0,1\} && e \in E,\\
y_i &\in \{0,1\} && i \in V,
\end{aligned}$$

where constraints (8.8) and (8.9) replace constraints (8.2) and (8.3), respectively.

Notice that constraints

$$x(E(S)) \leq y(S) - 1 \qquad S \subset V, \mu(S) \neq 0,$$

i.e., constraints (8.7), valid for the E-GMSTP remain valid for the L-GMSTP. However, they do not dominate all the constraints (8.4) unlike for the E-GMSTP polytope. Indeed, consider the following example (see Figure 8.5) where all the constraints (8.7) are satisfied but at least one of the constraints (8.4) is violated. Let $V = \{1, 2, \ldots, 6\}, V_1 = \{1, 2, 3\}, V_2 = \{4, 5\}, V_3 = \{6\}$ and the graph is complete. If $y_i = 1$, $\forall i \in V, x_{12} = x_{14} = x_{24} = x_{35} = x_{56} = 1, x_e = 0$ otherwise, then the constraint (8.4) for $S = \{1, 2, 4\}, i = 1, j = 3$ is violated.

The above linear formulation for the L-GMSTP is not strong with respect to its linear relaxation. We can justify this claim in exploiting the same argument as the one used in Magnanti and Wolsey [120] for the Minimum Spanning Tree problem. Indeed, if each cluster is reduced to a single node, the L-GMSTP boils down to the classical MSTP. However, this formulation is particularly effective in a branch-and-cut context since the so-called cut-constraints (8.4) are easy to separate using max-flow algorithms. Such a formulation is elaborated and tested in Section 8.5.2.

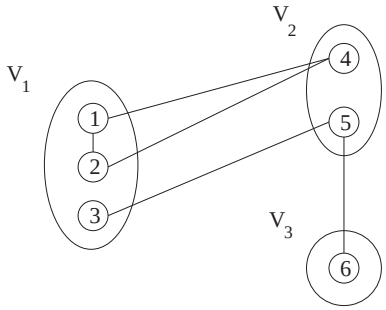

Figure 8.5: Constraints (8.4) are not all implied by constraints (8.7) for L-GMSTP.

8.3.1 From L-GMSTP to E-GMSTP

One way to solve the L-GMSTP consists of seeing it as a variant of the E-GMSTP. For this purpose, the following transformed graph has to be defined.

Let $G = (V, E)$, with V partitioned into clusters $V_1, V_2, \ldots, V_{|K|}$, be the graph of the L-GMSTP instance. The transformed graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ is defined as follows:

- $\widetilde{V}$ is equal to V, and the partition into clusters remains the same,
- for each pair of nodes $i, j \in \widetilde{V}$ belonging to different clusters and such that there exists a path between i and j in G, there is an edge $\{i, j\} \in \widetilde{E}$ with cost equal to the value of the shortest path from i to j in the original graph G.

The transformed graph $\widetilde{G}$ is $|K|$-partite complete if G is connected on V.

Proposition 8.1 *The optimal solution of the E-GMSTP solved on $\widetilde{G}$ can be transformed into a feasible solution of L-GMSTP on G. It gives then an upper bound on the value of the optimal solution of L-GMSTP.*

Proof. An edge in an E-GMSTP solution in $\widetilde{G}$ corresponds to a path in G. Removing the repeated edges and the cycles (in deleting the edge with highest cost in each cycle, one cycle at a time), we obtain a feasible solution to L-GMSTP with value less or equal to the corresponding solution in $\widetilde{G}$. □

It is not difficult to see that repeated edges and cycles can occur from the transformation.

It does not always exist an optimal solution of E-GMSTP on $\widetilde{G}$ such that in removing repeated edges and cycles, we get an optimal solution for L-GMSTP on G. It means that solving the E-GMSTP on $\widetilde{G}$ can only provide an upper bound to the L-GMSTP on G. To see this, it suffices to consider again the graph in Figure 8.3. The optimal solution of L-GMSTP on G is 12.5, while the optimal value of the E-GMSTP solution on $\widetilde{G}$ is 12.8 and corresponds to a feasible solution of L-GMSTP in G of value 12.8.

In [77], such a transformation of G into $\widetilde{G}$ with edges in $\widetilde{G}$ corresponding to the shortest paths with the maximum number of edges has been tested as a heuristic. This is done, in order to get a solution in $\widetilde{G}$ that is transformed into a solution in G with as many as possible repeated edges and cycles. To obtain shortest paths with maximum

number of edges, the following scaling is performed on G. The cost c_e is replaced by $100c_e - 1$. Three heuristics based on the solution of E-GMSTP have been tested in [77]: *1*) the transformation using the shortest paths, *2*) the transformation using the shortest paths with maximum number of edges, and *3*) L-GMSTP solved directly as E-GMSTP. (Such a third case clearly provides an upper bound for L-GMSTP since a feasible solution of E-GMSTP is also feasible for L-GMSTP.)

8.4 A generalization: the E/L-GMSTP

The E/L-GMSTP can be formulated as an integer linear program as follows.

$$\min \sum_{e \in E} c_e x_e$$

s.t.

$$\begin{aligned}
y(V_k) &= 1 && k \in K_E, && (8.10)\\
y(V_k) &\geq 1 && k \in K_L, && (8.11)\\
x(E) &= y(V) - 1, &&&&\\
x(\delta(S)) &\geq y_i + y_j - 1 && i \in S \subset V, j \notin S,&&\\
x_e &\in \{0,1\} && e \in E,&&\\
y_i &\in \{0,1\} && i \in V,&&
\end{aligned}$$

where $K = K_E \cup K_L$ is partitioned in two sets K_E and K_L such that the clusters in K_E (resp. K_L) must be visited exactly (resp. at least) once.

This problem is clearly a generalization of both E-GMSTP and L-GMSTP since both sets K_E and K_L can reduce to the empty set. Obviously, in case K_L is the empty set the model can be tightened to become the subpacking formulation described in Section 8.2.

8.5 Computational results

In this section we concentrate on the computation of provably optimal solutions for E-GMSTP, L-GMSTP and E/L-GMSTP. This is done by using the same branch-and-cut framework, which is specified so as to take into account the main differences among the three problems. This framework, developed for Generalized Spanning Tree problems by Feremans, Labbé and Laporte [79] and Feremans [77], has been improved in the current chapter to obtain better results through new branching rules, primal-heuristic algorithms and the use of general-purpose Chvátal-Gomory cuts.

First of all we briefly recall the branch-and-cut method of [79, 77]:

1. **Initialization**: Insert the linear program:

$$\{\min \sum_{e \in E} c_e x_e : (8.2), (8.3), x_e, y_i \geq 0, \quad \forall i \in V,\ \forall e \in E\}$$

for E-GMSTP or

$$\{\min \sum_{e \in E} c_e x_e : (8.8), (8.9), x_e, y_i \in [0,1], \quad \forall i \in V, \ \forall e \in E\}$$

for L-GMSTP, respectively, in a problem list $\mathcal{L}$. Initialize the incumbent solution $\overline{z}$ to infinity.

2. **Termination**: If $\mathcal{L}$ is empty, STOP. Otherwise, extract one subproblem from $\mathcal{L}$ according to a best-first rule.

3. **LP solution**: Solve the subproblem using an LP-solver and let x^* be its optimal solution and z^* its value. If $z^* \geq \overline{z}$, go to STEP 2. Otherwise[3], if the solution is integer and feasible update the incumbent solution $\overline{z}$ and go to STEP 2.

4. **Separation I**: separate the special cases of (8.7):

$$x(\delta(i)) \geq y_i \quad i \in V \tag{8.12}$$

and

$$x(E(\{i\} : V_k)) \leq y_i \quad i \in V \setminus (W \cup V_k), \ \forall k \in K \tag{8.13}$$

for E-GMSTP (where $W = \{i \in V : i \in V_k, |V_k| = 1\}$ is the set of nodes belonging to a cluster which is a singleton) or

$$x_e \leq y_i \ \text{ and } \ x_e \leq y_j \quad \forall\, e = \{i,j\} \in E \tag{8.14}$$

for L-GMSTP, respectively[4].
If violated inequalities are found, add them to the current subproblem and go to STEP 3.

5. **Separation II**: separate constraints (8.7).
If violated inequalities are found, add them to the current subproblem and go to STEP 3.

6. **Separation III**: separate
odd-cycle inequalities and *odd-hole* inequalities[5]
for E-GMSTP or

cut constraints (8.4)

for L-GMSTP, respectively[6].
If violated inequalities are found, add them to the current subproblem and go to STEP 3.

[3]Both local improvement and rounding procedures are applied in the E-GMSTP context (see [79] for details).

[4]For the proof of validity and separation details of (8.12)-(8.13) and (8.14) see [79] and [77], respectively.

[5]See [79] for proof of validity and separation details.

[6]Note that the constraints for E-GMSTP and L-GMSTP in STEP 6 play a rather different role. Indeed, constraints (8.4) are necessary for the correctness of the branch-and-cut in the L-GMSTP context, while odd-cycle and odd-hole are redundant constraints for E-GMSTP.

7. **Branching**: create two new subproblems by branching on a constraint (8.2) for E-GMSTP, or on the edge whose value is closest to 0.5 and with maximum cost for L-GMSTP, respectively.
   Add the subproblems to the list $\mathcal{L}$ and go to STEP 2.

8.5.1 Improving on E-GMSTP

The branch-and-cut method outlined in the previous section has been improved on as follows.

First, we changed the branching strategy by branching on nodes, i.e., on the y_i variable closest to 0.5. As it is customary, two subproblems are created having $y_i = 0$ and $y_i = 1$, respectively.

Second, we improved on the separation procedure: preliminary tests have shown that odd-cycle inequalities and odd-hole inequalities separation routines are rather time-consuming, and these constraints are seldom generated. Instead of these constraints we used the general-purpose Chvátal-Gomory cuts, with and without the strengthening procedures proposed by Letchford and Lodi [112]. The use of Chvátal-Gomory cuts has been recently re-discovered by several authors, and a number of ways of using them have been proposed. In our computational experiments we found very useful to separate a single round of Chvátal-Gomory cuts at the root node of our branch-and-cut tree when no other cuts have been identified and before resorting to branching. After the addition of such a first round, we restart from STEP 3 of the algorithm above.

Computational results comparing the algorithm in [79] with the improved one on both random-generated Euclidean instances from [79] and the set of instances proposed by Dror, Haouari, and Chaouachi [72] (and denoted in the following as "DHC" instances) are presented in Tables 8.1-8.2 and Table 8.3, respectively. All the codes have been implemented by using the branch-and-cut framework ABACUS [1] version 2.4 (alpha release) and Cplex 9.0 [105] as LP solver. We compare four algorithms, namely, the original algorithm presented in [79], and three versions (v. 0,1,2) of the improved algorithm depending on the type of Chvátal-Gomory cuts used, i.e., classical ones (v.0), strengthened of type 1 (v.1), and strengthened of type 2 (v.2) (see, Letchford and Lodi [112] for details). Table 8.1 is organized as follows. Each line refers to a set of five random Euclidean instances generated as in [79]. First, $|V|$, $|K|$ and $|E|$ indicate the number of nodes, clusters and edges, respectively, of the corresponding five instances. Then, we report the number of instances solved to optimality within the time limit of 3,600 CPU seconds on a Pentium M 1.6 Ghz notebook with 512 MByte of main memory (Succ), the average number of nodes of the branch-and-cut tree (Nodes), the average gap of the lower bound computed at the root node with respect to either the optimal solution value or the best know one (LB%), the average separation time in seconds (sepT) and, finally, the average overall time in seconds[7]. The last row of Table 8.1 reports for each algorithm the total number of solved instances and the average value over all the instances of the other entries.

[7] An instance which is not solved to optimality in the time limit has a computing time of 3,600 CPU seconds for computation of the average time.

Table 8.1: E-GMSTP: Euclidean Instances Comparison.

			Branch-and-cut [79]					Branch-and-cut, v.0					Branch-and-cut, v.1					Branch-and-cut, v.2				
$\|V\|$	$\|K\|$	$\|E\|$	Succ	Nodes	LB%	sepT	TT	Succ	Nodes	LB%	sepT	TT	Succ	Nodes	LB%	sepT	TT	Succ	Nodes	LB%	sepT	TT
120	20	7140	5	12.20	97.79	6.90	131.60	5	17.40	98.09	6.23	135.49	5	16.20	98.09	5.85	128.47	5	14.20	98.09	5.97	131.45
120	30	7140	5	13.40	98.69	12.98	104.02	5	5.80	98.69	5.09	70.75	5	7.40	98.79	5.18	73.76	5	8.20	98.90	6.18	85.81
120	40	7140	5	30.60	97.70	36.64	237.17	5	33.40	97.62	13.34	197.45	5	55.40	97.70	19.63	323.30	5	49.80	97.78	18.23	284.86
150	15	11175	5	1.40	99.75	1.82	60.87	5	1.80	99.75	2.04	84.93	5	1.80	99.75	2.24	90.71	5	2.20	99.75	2.25	82.69
150	25	11175	5	21.00	98.29	24.04	456.68	5	23.40	98.29	13.98	443.77	5	31.40	98.29	16.30	530.15	5	38.20	98.29	19.03	607.99
150	30	11175	5	25.80	97.75	33.59	429.71	5	25.00	97.85	16.57	394.15	5	23.40	97.85	16.20	393.55	5	23.00	97.96	16.59	405.27
160	20	12720	5	15.00	97.53	20.15	498.30	5	9.40	97.90	14.64	471.61	5	11.40	98.07	13.81	491.34	5	11.80	98.07	15.30	523.99
160	40	12720	3	81.80	95.59	228.94	2124.00	4	109.40	95.63	86.75	2226.96	3	115.40	95.45	95.67	2256.79	2	125.40	95.68	101.33	2402.99
180	15	16110	5	1.80	100.00	4.49	224.44	5	1.80	99.70	3.88	252.71	5	1.80	99.70	4.02	261.15	5	1.80	99.70	4.12	270.94
180	30	16110	5	13.80	98.55	45.57	846.88	5	15.80	98.55	28.23	774.31	5	14.20	98.55	26.16	715.95	5	12.60	98.55	26.49	707.36
200	20	19900	5	11.00	98.45	36.00	1075.30	5	9.40	98.65	20.71	939.69	5	7.40	98.65	21.27	940.80	5	7.40	98.65	22.50	956.61
200	25	19900	4	22.20	96.48	80.56	2483.33	4	17.00	96.54	58.54	2197.82	4	17.00	96.58	56.69	2187.61	3	16.60	96.62	56.61	2329.57
200	40	19900	1	41.00	91.70	212.13	3509.48	2	43.00	91.75	105.84	3208.37	1	40.20	91.78	100.81	3143.80	2	44.20	91.79	98.83	3053.37
			58	22.38	97.56	57.22	937.06	60	24.05	97.62	28.91	876.77	58	26.38	97.63	29.53	887.49	57	27.34	97.68	30.26	910.99

Table 8.2: E-GMSTP: Euclidean Hard Instances Comparison.

			Branch-and-cut [79]			Branch-and-cut, v.0			Branch-and-cut, v.1			Branch-and-cut, v.2		
$\|V\|$	$\|K\|$	$\|E\|$	Nodes	UB	TT	Nodes	UB	TT	Nodes	UB	TT	Nodes	UB	TT
160	40	12720	123	249	> 3600.00	135	249	3436.41	111	249	2620.05	165	260	> 3600.00
			21	226	502.65	17	226	387.13	7	226	291.22	19	226	375.77
			145	254	> 3600.00	185	252	> 3600.00	171	248	> 3600.00	169	251	> 3600.00
			93	239	1977.30	183	239	2888.31	247	244	> 3600.00	251	241	> 3600.00
			27	247	940.04	27	247	822.94	41	247	1172.67	23	247	839.17
200	25	19900	9	128	2536.06	7	128	2010.21	15	128	2644.38	7	128	1989.08
			15	145	> 3600.00	21	145	> 3600.00	21	154	> 3600.00	21	151	> 3600.00
			7	121	427.56	7	121	572.13	5	121	509.41	3	121	499.64
			57	136	3563.77	39	136	3167.65	33	136	2665.40	39	136	> 3600.00
			23	133	2289.26	11	133	1639.12	11	133	1518.86	13	133	1959.15
200	40	19900	25	264	> 3600.00	31	267	> 3600.00	27	286	> 3600.00	33	249	> 3600.00
			27	244	> 3600.00	47	239	> 3600.00	39	223	> 3600.00	57	224	> 3600.00
			43	208	3147.41	19	208	1874.64	11	208	1319.01	9	208	1245.63
			61	219	> 3600.00	67	216	3367.23	67	222	> 3600.00	47	216	3221.22
			49	251	> 3600.00	71	249	> 3600.00	57	243	> 3600.00	75	230	> 3600.00

Table 8.3: E-GMSTP: "DHC" Instances Comparison.

					Branch-and-cut [79]				Branch-and-cut, v.0				Branch-and-cut, v.1				Branch-and-cut, v.2			
ID	$\|V\|$	$\|K\|$	$\|E\|$	Opt	Nodes	LB%	sepT	TT	Nodes	LB%	sepT	TT	Nodes	LB%	sepT	TT	Nodes	LB%	sepT	TT
1	25	4	50	23	1	100.00	0.00	0.20	1	100.00	0.00	0.14	1	100.00	0.00	0.15	1	100.00	0.00	0.11
2	25	8	100	45	1	100.00	0.00	0.07	1	100.00	0.00	0.07	1	100.00	0.00	0.07	1	100.00	0.00	0.06
3	25	10	150	37	3	100.00	0.00	0.07	1	100.00	0.00	0.07	1	100.00	0.00	0.08	1	100.00	0.00	0.08
4	50	5	150	18	1	100.00	0.00	0.10	1	100.00	0.00	0.07	1	100.00	0.00	0.07	1	100.00	0.00	0.08
5	50	10	300	27	1	100.00	0.00	0.09	1	100.00	0.00	0.09	1	100.00	0.00	0.08	1	100.00	0.00	0.09
6	75	8	200	55	1	100.00	0.00	0.11	1	100.00	0.00	0.11	1	100.00	0.00	0.11	1	100.00	0.00	0.11
7	75	10	300	67	19	100.00	0.01	0.17	1	100.00	0.00	0.17	1	100.00	0.00	0.18	1	100.00	0.00	0.17
8	75	15	400	58	39	100.00	0.01	0.13	1	100.00	0.00	0.15	1	100.00	0.00	0.12	1	100.00	0.00	0.13
9	100	7	300	37	1	100.00	0.00	0.10	1	100.00	0.00	0.11	1	100.00	0.00	0.10	1	100.00	0.00	0.10
10	100	10	500	49	11	100.00	0.00	0.20	1	100.00	0.02	0.20	1	100.00	0.00	0.19	1	100.00	0.00	0.20
11	150	8	300	65	5	100.00	0.02	0.33	1	100.00	0.02	0.34	1	100.00	0.03	0.33	1	100.00	0.00	0.34
12	150	12	500	79	5	100.00	0.34	3.18	1	100.00	0.57	3.64	1	100.00	0.34	3.48	1	100.00	0.51	3.74
13	200	10	500	65	23	95.38	0.46	7.67	9	96.92	0.50	7.17	5	96.92	0.40	5.23	5	96.92	0.34	4.52
14	200	20	1000	53	1	100.00	0.04	0.47	1	100.00	0.01	0.46	1	100.00	0.04	0.45	1	100.00	0.04	0.46
15	250	10	500	60	3	100.00	0.01	0.43	1	100.00	0.01	0.46	1	100.00	0.02	0.45	1	100.00	0.01	0.42
16	250	25	1000	123	3	100.00	0.39	2.91	1	100.00	0.27	2.87	1	100.00	0.36	2.92	1	100.00	0.31	2.94
17	300	20	1000	95	27	100.00	0.10	1.42	1	100.00	0.13	1.43	1	100.00	0.10	1.43	1	100.00	0.05	1.43
18	300	30	2000	85	7	100.00	0.25	3.07	1	100.00	0.24	3.10	1	100.00	0.29	3.08	1	100.00	0.26	3.12
19	300	40	3000	88	67	100.00	3.54	12.00	7	98.86	1.22	8.74	7	98.86	1.29	8.72	3	100.00	1.08	7.55
20	500	50	5000	109	71	95.41	343.88	1813.08	51	95.41	137.38	1182.65	35	95.41	143.96	1160.76	51	95.41	234.33	2265.11
					14.50	99.54	17.45	92.29	4.20	99.56	7.02	60.60	3.20	99.56	7.34	59.40	3.80	99.62	11.85	114.54

Table 8.2 reports the disaggregated results for the three sets of randomly-generated instances for which not all the instances were solved to optimality (hard instances).

Tables 8.1 and 8.2 show that the improved version v.0 obtains the best results for what concerns the number of solved instances and the computing time. As for the lower bound at the root node, the use of the Chvátal-Gomory cuts generally leads to better values, mainly the strengthening procedures proposed in [112] are applied.

Table 8.3 has the same structure as Table 8.1, but the first column indicates the identifier of the single instance (ID) and no column indicating the number of successes is present because the all set of 20 instances is solved to optimality by the four algorithms. In addition, the fifth column indicates the optimal value for the instance (Opt).

Finally, we tested our improved branch-and-cut algorithm v.0 with respect to that proposed in [79] on generalized Traveling Salesman instances generated according to Fischetti, Salazar and Toth [86]. As also reported in [79], the branch-and-cut algorithms solve all the problems with up to 226 nodes at the root node by using only constraints (8.7) (i.e., without additional cuts) but instance `ts225`. Thus, the two algorithms perform in the same way. On `ts225`, we obtained a speedup of 1.12 (computed as the ratio of the computing times of the original algorithm and of the improved one), a slightly reduced number of branching nodes (15 instead of 23) and a better lower bound at the root node due to the use of Chvátal-Gomory cuts (2 units improvement on a absolute gap of 22 units).

8.5.2 Improving on L-GMSTP

In addition to the two improvements already described in the previous section for E-GMSTP, and also used for L-GMSTP (i.e., an effective branching strategy and the use of general-purpose Chvátal-Gomory cuts) in the context of L-GMSTP we applied two new improvements.

In the original version of the branch-and-cut method presented in [77], no heuristic algorithm was used in STEP 3. We propose a greedy heuristic based on the classical algorithm of Prim for MSTP. This heuristic is divided in two phases: the first is a rounding procedure which starts from a fractional solution while the second one is a simple local improvement. More precisely:

Phase 1): a maximum-priority spanning tree is computed using Prim's algorithm (a priority proportional to x_e^* is associated with each edge e and, among edges of the same priority, the inter-cluster edges are considered first). This phase stops when a feasible solution $T = (V', E')$ for L-GMSTP is reached.

Phase 2): the solution is improved by removing all the redundant nodes of degree 1 (a node i is redundant if $|(V' \setminus \{i\}) \cap V_k| \geq 1,\ k : i \in K$). The corresponding adjacent edges are removed from T one at a time according to nonincreasing edge cost while the remaining edges still form a feasible solution.

Moreover, a more prudent separation policy has been implemented for constraints (8.4). Indeed, we decided to apply the separation routine based on max-flow computation in case the solution is integer but infeasible, while in case of a fractional solution such that neither constraints (8.7) nor their special cases (8.12) and (8.14) are violated we resort to branching, i.e., we execute STEP 7.

Computational results comparing the algorithm in [77] with the improved one on the "DHC" instances are presented in Table 8.4. Table 8.4 has the same structure as Table 8.3 but the first algorithm is now the branch-and-cut approach presented in [77]. Moreover, we report in column six the number of selected edges in the optimal solution ($|E'|$).

Table 8.4 shows that the first eighteen instances require very short computing time. As for instances `19` and `20` the improved version v.0 obtains the best computing times, while the best lower bound values are obtained by the strengthening procedures proposed in [112].

Before ending this section it has to be noted that Duin, Volgenant and Voß [73] report computational results on the exact solution of L-GMSTP computed through a transformation to the *Steiner Tree Problem* (STP). In particular, a code for the STP is used to compute optimal solutions of the L-GMSTP and then assert the quality of heuristic algorithms for L-GMSTP proposed by the same authors. Incidentally, the computing times on the "DHC" instances are rather short suggesting that this transformation is a competitive way of solving L-GMSTP.

8.5.3 E/L-GMSTP Results

Preliminary results on the introduced generalization of E-GMSTP and L-GMSTP are presented in this section by naturally adapting the branch-and-cut framework described in the previous section and using a subset of the "DHC" instances for which the optimal solutions computed in the previous sections for E-GMSTP and L-GMSTP were different. These results are reported in Table 8.5. The table presents the results for the version of the algorithm, among the three versions considered in the previous sections, which has on average the best results, i.e., v.0. The structure of the table is once again as the one of the previous ones, but for the number of clusters for which we specify $|K_E|$ (resp. $|K_L|$), i.e., the number of clusters for which exactly (resp. at least) one node has to be reached.

The table shows these instances as well can be effectively solved through the proposed branch-and-cut algorithm.

8.6 Conclusion

In this chapter we have considered three $\mathcal{NP}$-hard generalizations of the classical MSTP which often arise in practical applications, e.g., in the telecommunication and agricultural settings.

The relationships among these problems, and in particular with respect to their ILP formulations, have been discussed and branch-and-cut approaches have been extensively tested. More precisely, we improved on existing branch-and-cut algorithms from the literature [79, 77] by using new effective primal heuristics, more powerful branching strategies, and general-purpose Chvátal-Gomory cuts.

These modifications have been proved to be effective through computational results and the branch-and-cut approaches seem to be a flexible and powerful tool to handle such problems.

Table 8.4: L-GMSTP: "DHC" Instances Comparison.

						Branch-and-cut [77]				Branch-and-cut, v.0				Branch-and-cut, v.1				Branch-and-cut, v.2			
ID	$\|V\|$	$\|K\|$	$\|E\|$	Opt	$\|E'\|$	Nodes	LB%	sepT	TT	Nodes	LB%	sepT	TT	Nodes	LB%	sepT	TT	Nodes	LB%	sepT	TT
1	25	4	50	23	3	1	100.00	0.00	0.07	1	100.00	0.00	0.07	1	100.00	0.00	0.07	1	100.00	0.00	0.07
2	25	8	100	41	9	1	100.00	0.00	0.08	1	100.00	0.00	0.19	1	100.00	0.00	0.13	1	100.00	0.00	0.08
3	25	10	150	36	10	3	100.00	0.00	0.09	3	100.00	0.00	0.10	3	100.00	0.00	0.10	3	100.00	0.00	0.11
4	50	5	150	18	4	1	100.00	0.01	0.12	1	100.00	0.00	0.09	1	100.00	0.00	0.10	1	100.00	0.00	0.09
5	50	10	300	27	9	1	100.00	0.00	0.10	1	100.00	0.00	0.10	1	100.00	0.01	0.09	1	100.00	0.00	0.10
6	75	8	200	55	7	1	100.00	0.00	0.15	1	100.00	0.00	0.14	1	100.00	0.01	0.14	1	100.00	0.00	0.14
7	75	10	300	67	9	19	95.52	0.13	0.76	19	95.52	0.09	0.68	13	95.52	0.05	0.63	19	95.52	0.08	0.76
8	75	15	400	53	16	39	94.34	0.23	0.90	15	94.34	0.02	0.41	15	94.34	0.03	0.40	11	94.34	0.02	0.37
9	100	7	300	37	6	1	100.00	0.00	0.15	1	100.00	0.00	0.16	1	100.00	0.01	0.16	1	100.00	0.01	0.16
10	100	10	500	48	10	11	97.92	0.07	0.72	3	100.00	0.07	0.59	3	100.00	0.03	0.47	3	100.00	0.09	0.63
11	150	8	300	50	8	5	92.00	0.01	0.33	3	98.00	0.03	0.39	1	100.00	0.03	0.30	1	100.00	0.04	0.29
12	150	12	500	68	14	5	98.53	0.13	0.97	3	98.53	0.12	1.05	1	100.00	0.17	0.89	1	100.00	0.21	0.99
13	200	10	500	44	12	23	86.36	0.14	1.53	7	90.91	0.07	0.86	7	93.18	0.07	0.94	5	93.18	0.06	0.97
14	200	20	1000	50	20	1	100.00	0.00	0.38	1	100.00	0.00	0.41	1	100.00	0.03	0.41	1	100.00	0.01	0.45
15	250	10	500	60	9	3	96.67	0.00	0.72	3	98.33	0.01	0.71	3	98.33	0.04	0.70	1	100.00	0.01	0.60
16	250	25	1000	117	26	3	100.00	0.68	4.00	1	100.00	0.25	2.21	1	100.00	0.20	2.20	1	100.00	0.28	2.90
17	300	20	1000	88	23	27	95.45	0.97	4.95	27	95.45	0.44	5.18	23	95.45	0.45	4.21	19	95.45	0.65	4.51
18	300	30	2000	85	29	7	97.65	0.98	4.83	3	97.65	0.57	4.20	11	97.65	0.55	5.81	17	97.65	0.72	6.85
19	300	40	3000	88	39	67	98.86	11.48	30.08	23	98.86	1.20	9.74	11	98.86	1.39	8.78	19	98.86	1.35	8.44
20	500	50	5000	105	54	71	97.14	300.97	462.19	27	97.14	14.69	94.17	55	98.10	13.95	125.37	51	98.10	16.31	135.58
						14.50	97.52	15.79	25.66	7.20	98.24	0.88	6.07	7.70	98.57	0.85	7.60	7.90	98.66	0.99	8.20

Table 8.5: E/L-GMSTP: Modified "DHC" Instances.

							Branch-and-cut v.0			
ID	$\|V\|$	$\|K_E\|$	$\|K_L\|$	$\|E\|$	Opt	$\|E'\|$	Nodes	LB%	sepT	TT
2	25	1	7	100	45	7	1	100.00	0.01	0.08
3	25	1	9	150	37	9	1	100.00	0.00	0.04
8	75	1	14	400	58	14	7	98.28	0.03	0.27
10	100	1	9	500	49	9	1	100.00	0.01	0.29
11	150	1	7	300	61	8	15	78.69	0.05	0.96
12	150	1	11	500	75	12	13	94.67	0.34	2.62
13	200	1	9	500	46	12	19	86.96	0.09	1.39
14	200	1	19	1000	53	19	1	100.00	0.00	0.54
16	250	1	24	1000	119	26	7	98.32	0.71	4.60
17	300	1	19	1000	91	20	31	92.31	0.72	7.13
20	500	3	47	5000	108	53	475	94.44	52.04	1085.49

Bibliography

[1] ABACUS, A Branch-And-CUt System, `http://www.informatik.uni-koeln.de/abacus/`.

[2] T. Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4:4–20, 2007.

[3] T. Achterberg. *Constraint Integer Programming*. PhD thesis, ZIB, Berlin, 2007.

[4] T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4:77–86, 2007.

[5] T. Achterberg, T. Koch and A. Martin. Branching roles revisited. *Operations Research Letters*, 33:42–54, 2005.

[6] K. Andersen, G. Cornuéjols and Y. Li. Reduce-and-split cuts: Improving the performance of mixed integer Gomory cuts. *Management Science*, 51:1720–1732, 2005.

[7] K. Andersen, Q. Louveaux, R. Weismantel and L.A. Wolsey. Inequalities from two rows of a simplex tableau. In M. Fischetti and D.P. Williamson, editors, *Integer Programming and Combinatorial Optimization - IPCO 2007*, volume 4513 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, Berlin Heidelberg, 2007.

[8] G. Andreello, A. Caprara and M. Fischetti. Embedding cuts in a branch and cut framework: a computational study with $\{0, \frac{1}{2}\}$-cuts. *INFORMS Journal on Computing*, 19:229–238, 2007.

[9] D. Applegate, R.E. Bixby, V. Chvátal and W.J. Cook. Finding cuts in the tsp. Technical Report 95-05, DIMACS, 1995.

[10] N. Ascheuer, M. Fischetti and M. Grötschel. A polyhedral study of the asymmetric travelling salesman problem with time Windows. *Networks*, 36:69–79, 2000.

[11] N. Ascheuer, M. Fischetti and M. Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut, *Mathematical Programming*, 90:475–506, 2001.

[12] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef and G. Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. Techinal Report RR 949-M, Université Joseph Fourier, Grenoble, 1995.

[13] E.K. Baker. An exact algorithm for the time-constrained travelling salesman problem. *Operations Research*, 31:938–945, 1983.

[14] E. Balas. Intersection cuts – a new type of cutting planes for integer programming. *Operations Research*, 19:19–39, 1971.

[15] E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146–164, 1975.

[16] E. Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[17] E. Balas. A modified lift-and-project procedure, *Mathematical Programming*, 79:19–31, 1997.

[18] E. Balas. Disjunctive programming: properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89:3–44, 1998.

[19] E. Balas and P. Bonami. Generating lift-and-project cuts from the LP simplex tableau: open source implementation and testing of new variants. *Technical Report*, Tepper School of Business, CMU, 2008.

[20] E. Balas, S. Ceria and G. Cornuéjols. Mixed 0–1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42:1229–1246, 1996.

[21] E. Balas, S. Ceria, G. Cornuéjols and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

[22] E. Balas, M. Fischetti and W.R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68:241–265, 1995.

[23] E. Balas, R. Jeroslow. Strengthening Cuts for Mixed Integer Programs. *European Journal of Operations Research*, 4:224–234, 1980.

[24] E. Balas and M. Perregaard. Lift-and-project for mixed 0–1 programming: recent progress. *Discrete Applied Mathematics*, 123:129–154, 2002.

[25] E. Balas and M. Perregaard. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer gomory cuts for 0–1 programming. *Mathematical Programming*, 94:221–245, 2003.

[26] E. Balas and A. Saxena. Optimizing over the split closure. *Mathematical Programming*, 113:219–240, 2008.

[27] E. Balas, N. Simonetti. Linear time dynamic programming algorithms for new classes of restricted TSPs: a computational study. *INFORMS Journal on Computing*, 13:56–75, 2001.

[28] R. Baldacci, E.A. Hadjiconstantinou and A. Mingozzi. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52:723–738, 2004.

[29] A. Basu, P. Bonami, G. Cornuejols and F. Margot. On the relative strength of split, triangle and quadrilateral cuts. *Optimization Online*, `http://www.optimization-online.org/DB_HTML/2008/08/2060.html`.

[30] E.M.L. Beale and J.A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In J. Lawrence, editor, *OR 69. Proceedings of the Fifth International Conference on Operational Research*, pages 447–454. Tavistock Publications, 1970.

[31] M. Benichou, J.M. Gauthier, P. Girodet and G. Hentges. Experiments in mixed-integer programming. *Mathematical Programming*, 1:76–94, 1971.

[32] J. Berger and M. Barkaoui. A new hybrid genetic algorithm for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, 54:1254–1262, 2003.

[33] L. Bertacco. *Exact and Heuristic Methods for Mixed Integer Linear Programs*. PhD thesis, Università degli Studi di Padova, 2006.

[34] L. Bertacco, M. Fischetti and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4:63–76, 2007.

[35] J.E. Beasley. OR–Library: a collection of test data sets for a variety of Operations Research (OR) problems, `http://people.brunel.ac.uk/\~{}mastjjb/jeb/info.html`.

[36] R.E. Bixby, S. Ceria, C.M. McZeal, M.W.P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0, `http://www.caam.rice.edu/\~{}bixby/miplib/miplib.html`.

[37] P. Bonami *Étude et mise en oeuvre d'approches polyédriques pour la résolution de programmes en nombres entiers ou mixtes généraux*. PhD thesis, Université de Paris 6, 2003.

[38] P. Bonami, G. Cornuéjols, A. Lodi and F. Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, DOI 10.1007/s10107-008-0212-2, 2008.

[39] V. Borozan and G. Cornuéjols. Minimal valid inequalities for integer constraints. Technical Report, July 2007, `http://integer.tepper.cmu.edu/`.

[40] J. Brandão. A tabu search algorithm for the open vehicle routing problem. *European Journal of Operational Research*, 157:552-564, 2004.

[41] A. Caprara and M. Fischetti. $\{0, \frac{1}{2}\}$ Chvátal-Gomory cuts. *Mathematical Programming*, 74:221–235, 1996.

[42] A. Caprara and A. Letchford. On the separation of split cuts and related inequalities. *Mathematical Programming*, 94:279–294, 2003.

[43] Y. Caseau and P. Koppstein. A rule-based approach to a time-constrained traveling salesman problem. *Proceedings of the 2^{nd} International Symposium of Artificial Intelligence and Mathematics*, Fort Lauderdale, FL., 1992.

[44] Cbc. `https://projects.coin-or.org/Cbc`.

[45] S. Ceria and J. Soares. Disjunctive cuts for mixed 0–1 programming: duality and lifting. Technical Report, GSB, Columbia University, 1997.

[46] T. Christof, A. Löbel. PORTA, POlyhedron Representation Transformation Algorithm, `http://www.zib.de/Optimization/Software/Porta/`.

[47] N. Christofides, A. Mingozzi and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth and C. Sansi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979.

[48] N. Christofides, A. Mingozzi and P. Toth. State space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11:145–164, 1981.

[49] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[50] V. Chvátal. Resolution search. *Discrete Applied Mathematics*, 73:81–99, 1997.

[51] G. Clarke and J.V. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Reasearch*, 12:568–581, 1964.

[52] W. Cook. Fifty-plus years of combinatorial integer programming. To appear in M. Jünger, T. Liebling, D. Naddef, W. Pulleyblank, G. Reinelt, G. Rinaldi, L. Wolsey, editors, *50 Years of Integer Programming 1958–2008*, Springer-Verlag, 2008.

[53] W.J. Cook, S. Dash, R. Fukasawa and M. Goycoolea. Numerically accurate Gomory mixed-integer cuts. Technical report, School of Industrial and Systems Engineering, Georgia Tech, 2007. `http://mgoycool.uai.cl/papers/cdfg08_ijoc.pdf`.

[54] W.J. Cook, R. Kannan and A. Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47:155–174, 1990.

[55] J-F. Cordeau, G. Laporte, M. W. P. Savelsbergh and D. Vigo. Vehicle routing. In C. Barnhart and G. Laporte, editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 367–428. North–Holland, Amsterdam, 2007.

[56] J-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte and J-S. Sormany. New heuristics for the vehicle routing problem. In A. Langevin and D. Riopel, editors, *Logistics Systems: Design and Optimization*, pages 279–297. Springer–Verlag, New York, 2005.

[57] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112:3–44, 2008.

[58] G. Cornuéjols and Y. Li. A connection between cutting plane theory and the geometry of numbers. *Mathematical Programming*, 93:123-127, 2002.

[59] G. Cornuéjols, L. Liberti and G. Nannicini. Improved strategies for branching on general disjunctions. Technical Report working paper, LIX, École Polytechnique, 2008.

[60] G. Cornuj́ols and F. Margot. On the facets of mixed integer programs with two integer variables and two constraints. *Mathematical Programming*, DOI 10.1007/s10107-008-0221-1, 2008.

[61] H. Crowder, E. Johnson and M.W. Padberg. Solving large scale zero-one linear programming problem. *Operations Research*, 31:803–834, 1983.

[62] E. Danna, E. Rothberg and C. Le Pape. Exploiting relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90, 2005.

[63] G.B. Dantzig. Linear programming and extensions. *Princeton University Press*, Princeton, New Jersey, 1963.

[64] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6:80, 1959.

[65] Dash Optimization, Xpress-MP, `http://www.dashoptimization.com`.

[66] S. Dash, O. Günlük and A. Lodi. On the MIR closure of polyhedra. in M. Fischetti, D.P. Williamson, editors, *Integer Programming and Combinatorial Optimization - IPCO 2007*, volume 4513 of *Lecture Notes in Computer Science*, pages 337-351. Springer-Verlag, Berlin Heidelberg, 2007.

[67] S. Dash, O. Günlük and A. Lodi. MIR closures of polyhedral sets. *Mathematical Programming*, DOI 10.1007/s10107-008-0225-x, 2008.

[68] R. De Franceschi, M. Fischetti and P. Toth. A new ILP-based refinement heuristic for vehicle routing problems. *Mathematical Programming*, 105:471–499, 2006.

[69] U. Derigs and K. Reuter. A simple and efficient tabu search heuristic for solving the open vehicle routing problem. *Journal of the Operational Research Society*, DOI 10.1057/jors.2008.107, 2008.

[70] J. Desrosiers, Y. Dumas, M.M. Solomon and F. Soumis. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser, editors, *Network Routing*, pages 25–139. Elsevier, Amsterdam, 1995.

[71] S. Dey and L.A. Wolsey. Lifting integer variables in minimal inequalities corresponding to lattice-free triangles. In A. Lodi, A. Panconesi and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization - IPCO 2008*, volume 5035 of *Lecture Notes in Computer Science*, pages 463–475. Springer-Verlag, Berlin Heidelberg, 2008.

[72] M. Dror, M. Haouari and J. Chaouachi. Generalized Spanning Trees. *European Journal of Operational Research*, 120:583–592, 2000.

[73] C.W. Duin, A. Volgenant and S. Voß. Solving group Steiner problems as Steiner problems. *European Journal of Operational Research*, 154:323–329, 2004.

[74] Y. Dumas, J. Desrosiers, E. Gelinas and M.M. Solomon. An optimal algorithm for the travelling salesman problem with time windows. *Operations Research*, 43:367–371, 1995.

[75] D.G. Espinoza. Computing with multi-row gomory cuts. In A. Lodi, A. Panconesi and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization - IPCO 2008*, volume 5035 of *Lecture Notes in Computer Science*, pages 214–224. Springer-Verlag, Berlin Heidelberg, 2008.

[76] D. Feillet, P. Dejax, M. Gendreau and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44:216–229, 2004.

[77] C. Feremans. *Generalized Spanning Trees and Extensions*. PhD thesis, Université Libre de Bruxelles, 2001.

[78] C. Feremans, M. Labbé and G. Laporte. A comparative analysis of several formulations for the generalized minimum spanning tree problem. *Networks*, 39:29–34, 2002.

[79] C. Feremans, M. Labbé and G. Laporte. Polyhedral analysis of the generalized minimum spanning tree problem. *Networks*, 43:71–86, 2004.

[80] C. Feremans, A. Lodi, P. Toth and A. Tramontani. Improving on branch-and-cut algorithms for generalized minimum spanning trees. *Pacific Journal of Optimization*, 1:491–508, 2005.

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

[82] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2002.

[83] M. Fischetti and A. Lodi. Optimizing over the first Chvátal closure. *Mathematical Programming*, 110:3–20, 2007.

[84] M. Fischetti, A. Lodi and D. Salvagnin. Just MIP it! In V. Maniezzo, T. Stützle and S. Voss, editors, *MATHEURISTICS: Hybridizing metaheuristics and mathematical programming*, Operations Research/Computer Science Interfaces Series. Springer, 2008.

[85] M. Fischetti, A. Lodi and A. Tramontani. On the separation of disjunctive cuts. Technical Report OR-08-2, DEIS, University of Bologna, 2008.

[86] M. Fischetti, J.J. Salazar and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45:378–394 1997.

[87] M. Fisher. Optimal solutions of vehicle routing problems using minimum k-trees. *Operations Research*, 42:626–642, 1994.

[88] K. Fleszar, I.H. Osman and K.S. Hindi. A variable neighbourhood search for the open vehicle routing problem. *European Journal of Operational Research*, 195:803–809, 2009.

[89] F. Focacci, A. Lodi and M. Milano. A Hybrid Exact Algorithm for the TSPTW. *INFORMS Journal on Computing*, 14:403–417, 2002.

[90] Z. Fu, R. Eglese and L.Y.O Li. A new tabu search heuristic for the open vehicle routing problem. *Journal of the Operational Research Society*, 56:267-274, 2005.

[91] Z. Fu, R. Eglese and L.Y.O Li. Corrigendum: A new tabu search heuristic for the open vehicle routing problem. *Journal of the Operational Research Society*, 57:1018, 2006.

[92] R. Fukasawa, H. Longo, J. Lysgaard, M.P. de Aragão, M. Reis, E. Uchoa and R.F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106:491–511, 2006.

[93] M. Gendreau, A. Hertz and G. Laporte. A tabu search heuristic for the VRP. Technical Report CRT-777, 1991.

[94] M. Gendreau, A. Hertz and G. Laporte, A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[95] F.W. Glover and G.A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.

[96] B. Golden, S. Raghavan and E. Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, New York, 2008.

[97] B.L. Golden, E.A. Wasil, J.P. Kelly and I-M. Chao. Metaheuristics in vehicle routing. In T.G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 33–56. Kluwer Academic, Boston, 1998.

[98] R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[99] R.E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The Rand Corporation, 1960.

[100] R.E. Gomory and E.L. Johnson. Some continuous functions related to corner polyhedra, Part I. *Mathematical Programming*, 3:23–85, 1972.

[101] Z. Gu, G.L. Nemhauser and M.W.P. Savelsbergh. Mixed flow covers for mixed 0-1 integer programs. *Mathematical Programming*, 85:439–467, 1999.

[102] G.M. Gutin. On an approach to solving the traveling salesman problem (in Russian). *Proceedings of the USSR Conference on System Research* (Moscow, USSR), 184–185, 1984.

[103] E. Hadjiconstantinou, N. Christofides and A. Mingozzi. A new exact algorithm for the vehicle routing problem based on q-paths and k-shortest paths relaxations. *Annals of Operations Research*, 61:21–43, 1995.

[104] E. Ihler, G. Reich and P. Widmayer. Class steiner trees and VLSI-design. *Discrete Applied Mathematics*, 90:173–194, 1999.

[105] ILOG Cplex, `http://www.ilog.com/products/cplex`.

[106] E.L. Johnson. On the group problem for mixed integer programming. *Mathematical Programming Study*, 2:137–179, 1974.

[107] E.L. Johnson and M.W. Padberg. Degree-two inequalities, clique facets and biperfect graphs. *Annals of Discrete Mathematics*, 16:169–187, 1982.

[108] M. Karamanov and G. Cornuéjols. Branching on general disjunctions. Technical report, Tepper School of Business, CMU, 2005, revised 2008.

[109] J. Kytöjoki, T. Nuortio, O. Bräysy and M. Gendreau. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, 34:2743–2757, 2007.

[110] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[111] A. Langevin, M. Desrochers, J. Desrosiers and F. Soumis. A two-commodity flow formulation for the traveling salesman and makespan problem with time windows. *Networks*, 23:631–640, 1993.

[112] A.N. Letchford and A. Lodi. Strengthening chvátal-gomory cuts and fractional cuts. *Operations Research Letters*, 30:74–82, 2002.

[113] A.N. Letchford, J. Lysgaard and R.W. Eglese. A branch-and-cut algorithm for the capacitated open vehicle routing problem. *Journal of the Operational Research Society*, 58:1642–1651, 2007.

[114] F. Li, B.L. Golden and E.A. Wasil. Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, 32:1165-1179, 2005.

[115] F. Li, B. Golden and E. Wasil. The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers & Operations Research*, 34:2918-2930, 2008.

[116] J.T. Linderoth and M.W.P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11:173–187, 1999.

[117] A. Lodi. MIP computation and beyond. To appear in M. Jünger, T. Liebling, D. Naddef, W. Pulleyblank, G. Reinelt, G. Rinaldi, L. Wolsey, editors, *50 Years of Integer Programming 1958–2008*, Springer-Verlag, 2008.

[118] Q. Louveaux and L.A. Wolsey. Lifting, superadditivity, mixed integer rounding and single node flow sets revisited. *4OR*, 1:173–207, 2003.

[119] J. Lysgaard, A.N. Letchford and R.W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100:423–445, 2004.

[120] T.L. Magnanti and L.A. Wolsey. Optimal trees. In M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser, editors, volume 7 of *Handbooks in Operations Research & Management Science*, pages 503–615. Elsevier Science, Amsterdam, 1995.

[121] A. Mahajan and T.K. Ralphs. Experiments with branching using general disjunctions. Technical Report COR@L Lab, Lehigh University, 2008.

[122] H. Marchand. *A Polyhedral Study of the mixed knapsack set and its use to solve mixed integer programs.* PhD thesis, Université Catholique de Louvain, 1998.

[123] F. Margot. Testing cut generators for mixed-integer linear programming. Technical Report E-43, Tepper School of Business, CMU, 2007.

[124] D. Mester and O. Bräysy. Active guided evolution strategies for large scale vehicle routing problems. Working paper, University of Haifa, Israel, 2004.

[125] D. Mester and O. Bräysy. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & Operations Research*, 34:2964–2975, 2007.

[126] A. Mingozzi, L. Bianco and S. Ricciardelli. Dynamic programming strategies for the travelling salesman problem with time windows and precedence constraints. *Operations Research*, 45:365–377, 1997.

[127] Y.S. Myung, C.H. Lee and D.W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26:231–241, 1995.

[128] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 1988.

[129] G.L. Nemhauser and L.A. Wolsey. A recursive procedure to generate all cuts for 0-1 mixed integer programs. *Mathematical Programming*, 46:379–390, 1990.

[130] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Reasearch*, 41:421–451, 1993.

[131] J. Ostrowsky, J. Linderoth, F. Rossi and S. Smriglio. Constraint orbital branching. In A. Lodi, A. Panconesi and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization - IPCO 2008*, volume 5035 of *Lecture Notes in Computer Science*, pages 225–239. Springer-Verlag, Berlin Heidelberg, 2008.

[132] J. Owen and S. Mehrotra. Experimental results on using general disjunctions in branch-and-bound for general-integer linear program. *Computational Optimization and Applications*, 20:159–170, 2001.

[133] M.W. Padberg. A note on 0-1 programming. *Operations Research*, 23:833–837, 1975.

[134] M.W. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6:1–7, 1987.

[135] M.W. Padberg and G. Rinaldi. A branch and cut algorithm for the resolution of large-scale symmetric traveling salesmen problems. *SIAM Review*, 33:60–100, 1991.

[136] M.W. Padberg, T.J. Van Roy and L.A. Wolsey. Valid inequalities for fixed charge problems. *Operations Research*, 33:842–861, 1985.

[137] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[138] G. Pesant, M. Gendreau, J.-Y. Potvin and J.M. Rousseau. An exact constraint logic programming algorithm for the travelling salesman problem with time windows. *Transportation Science*, 32:12–29, 1998.

[139] G. Pesant, M. Gendreau, J.-Y. Potvin and J.M. Rousseau. On the flexibility of constraint programming models: from single to multiple Time Windows for the travelling salesman problem. *European Journal of Operational Research*, 117:253–263, 1999.

[140] A. Pessoa, M. Poggi de Aragão and E. Uchoa. Robust branch-cut-and-price algorithms for vehicle routing problems. In B. Golden, S. Raghavan, E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 297–325. Springer, New York, 2008.

[141] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34:2403–2435, 2007.

[142] P.C. Pop, W. Kern and G.J. Still. The generalized minimum spanning tree problem. Memorandum 1542, Department of Applied Mathematics, University of Twente, `http://dmmp.ewi.utwente.nl/research/publications.shtml`, 2000.

[143] P.C. Pop, W. Kern and G.J. Still. An approximation algorithm for the generalized minimum spanning tree problem with bounded cluster size. Memorandum 1577, Department of Applied Mathematics, University of Twente, `http://dmmp.ewi.utwente.nl/research/publications.shtml`, 2001.

[144] P.C. Pop, W. Kern and G.J. Still. A new relaxation method for the generalized minimum spanning tree problem. *European Journal of Operational Research*, 170:900–908, 2006.

[145] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31:1985–2002, 2004.

[146] T.K. Ralphs, L. Kopman, W.R. Pulleyblank and L.E. Trotter. On the capacitated vehicle routing problem. *Mathematical Programming*, 94:343–359, 2003.

[147] C. Rego and C. Roucairol. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory and Applications*, pages 661–675. Kluwer, Boston, MA, 1996.

[148] W. Rei, J.-F. Cordeau, M. Gendreau and P. Soriano. Accelerating Benders decomposition by local branching. Technical Report C7PQMR PO2006-02-X, C.R.T., Montréal, 2006.

[149] M. Reimann, K. Doerner and R.F. Hartl. D-ants: Savings based ants divide and conquer for the vehicle routing problem. *Computers & Operations Research*, 31:563–591, 2004.

[150] G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51:155–170, 2008.

[151] Y. Rochat and E.D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.

[152] E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19:534–541, 2007.

[153] M. Salari, P. Toth and A. Tramontani. An ILP improvement procedure for the open vehicle routing problem. Technical Report OR-08-06, DEIS, University of Bologna, 2008.

[154] D. Sariklis and S. Powell. A heuristic method for the open vehicle routing problem. *Journal of the Operational Research Society*, 51:564-573, 2000.

[155] V.I. Sarvanov and N.N. Doroshko. The approximate solution of the travelling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time (in Russian), in *Software: Algorithms and Programs*, 31:11–13. Mathematical Institute of the Byelorussian Academy of Sciences, Minsk, 1981.

[156] M.W.P. Savelsberg. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[157] M.P.W. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[158] L. Schrage. Formulation and structure of more complex/realistic routing and scheduling problems. *Networks*, 11:229-232, 1981.

[159] R.M. Stallman and G.J. Sussman. Forward reasoning and dependency directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135–196, 1977.

[160] E.D. Taillard. Eric Taillard's Page, Vehicle Routing Instances, `http://mistic.heig-vd.ch/taillard/problemes.dir/vrp.dir/vrp.html`.

[161] E.D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

[162] C.D. Tarantilis. Solving the vehicle routing problem with adaptative memory programming methodology. *Computers & Operations Research*, 32:2309–2327, 2005.

[163] C.D. Tarantilis, D. Diakoulaki and C.T. Kiranoudis. Combination of geographical information system and efficient routing algorithms for real life distribution operations. *European Journal of the Operational Research*, 152:437-453, 2004.

[164] C.D. Tarantilis, G. Ioannou, C.T. Kiranoudis and G.P. Prastacos. A threshold accepting approach to the open vehicle routing problem. *RAIRO Operations Research*, 38:345-360, 2004.

[165] C.D. Tarantilis, G. Ioannou, C.T. Kiranoudis and G.P. Prastacos. Solving the open vehicle routing problem via a single parameter metaheuristic algorithm. *Journal of the Operational Research Society*, 56:588-596, 2005.

[166] P. Toth and D. Vigo. An overview of vehicle routing problems. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, pages 1–26. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

[167] P. Toth and D. Vigo. The granular tabu search and its application to the vehicle routing problem. *INFORMS Journal on Computing*, 15:333–346, 2003.

[168] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

[169] P. Toth and A. Tramontani. An integer linear programming local search for capacitated vehicle routing problems. In B. Golden, S. Raghavan, E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 275–295. Springer, New York, 2008.

[170] T.J. Van Roy and L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45–57, 1987.

[171] D. Vigo. VRPLIB: A Vehicle Routing Problem LIBrary, `http://www.or.deis.unibo.it/research.html`.

[172] N.A. Wassan. A reactive tabu search for the vehicle routing problem. *Journal of the Operational Research Society*, 57:111-116, 2006.

[173] H.P. Williams. *Model solving in mathematical programming*. Wiley, Chichester, 1993.

[174] L.A. Wolsey. Facets for a linear inequality in 0-1 variables. *Mathematical Programming*, 8:165–178, 1975.

[175] J. Xu and J.P. Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30:379–393, 1996.

[176] G. Zambelli. On degenerate multi-row gomory cuts. *Operations Research Letters*, 37:21–22, 2009.

[177] A. Zanette, M. Fischetti and E. Balas. Can pure cutting plane algorithms work? In A. Lodi, A. Panconesi and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization - IPCO 2008*, volume 5035 of *Lecture Notes in Computer Science*, pages 416–434. Springer-Verlag, Berlin Heidelberg, 2008.