CUTTING PLANES FOR LARGE MIXED INTEGER PROGRAMMING MODELS

A Thesis
Presented to
The Academic Faculty

by

Marcos G. Goycoolea

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Industrial and Systems Engineering in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
December 2006

CUTTING PLANES FOR LARGE MIXED INTEGER PROGRAMMING MODELS

Approved by:

William J. Cook, Adviser
School of Industrial and Systems Engineering
Georgia Institute of Technology

Robin Thomas
Department of Mathematics
Georgia Institute of Technology

George L. Nemhauser
School of Industrial and Systems Engineering
Georgia Institute of Technology

Zonghao Gu
Ilog, Inc.

Ellis L. Johnson
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: November 9, 2006

ACKNOWLEDGEMENTS

The author was supported in part by ONR Grant N00014-03-1-0040, by NSF Grant DMI-0245609, and by the Presidential and Goizueta Fellowships of Georgia Tech.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introducion

1.1 Overview

In this thesis I focus on cutting planes for large Mixed Integer Programs (MIPs) in the context of Branch and Cut algorithms. More specifically, I describe two independent cutting-plane studies. The first of these deals with cutting planes for the Traveling Salesman Problem (TSP), the second with cutting planes for general Mixed Integer Programming.

In the first chapter of this thesis, where I describe my work concerning the TSP, I introduce a new family of constraints called the Generalized Domino-Parity (GDP) Inequalities. I present the following results for this class of inequalities:

- The class of GDP inequalities is valid for the Traveling Salesman Problem and Graphical Traveling Salesman Problem.
- The class of GDP inequalities generalizes most of the well-known TSP inequalities; including combs, domino-parity constraints, clique-trees, bipartitions, paths, and stars.
- A sub-class of the GDP inequalities which contains all clique-trees (having a fixed number of handles) can be separated exactly, in polynomial time, on planar graphs.

In the second chapter of this thesis I describe work concerning an efficient algorithm for the Mixed Integer Knapsack Problem (MIKP). Studying the MIKP is an important first

step in being able to derive valid inequalities for Mixed Integer Programming Problems consisting of a single row. More precisely,

- I present a specialized simplex-based algorithm for solving the linear programming relaxation of the mixed integer knapsack problem. This algorithm is combined with a pre-processing algorithm which can quickly detect unbounded or infeasible problem instances, and is able to reduce problem size by means of variable aggregation and variable elimination techniques.

- I discuss how to implement an effective branch and bound algorithm for solving instances of MIKP. The main feature of this algorithm is that it exploits cost and lexicographic dominance in order to significantly improve solution times. I present computational tests showing the effectiveness of the proposed branch-and-bound algorithm, and show that this algorithm outperforms the commercial mixed integer programming solver CPLEX. While it is not surprising that a problem-specific algorithm such as the one proposed should outperform a general-use optimization software, it is important to note that only alternative for solving this type of problem is mixed integer programming.

- I discuss how the domination-based methodology can be extended to general integer programming problems.

In the third chapter of this thesis I initiate a study of cutting planes for general MIPs. The long term goal of this study is to better understand cutting planes derived from single-row relaxations, and relaxations consisting of two or a few rows. The part of this study which I discuss in this thesis pertains to single row systems, and specifically focuses much on the well-known Mixed Integer Rounding (MIR) Inequalities. More specifically, I address the following issues:

- How best to implement a Mixed Integer Rounding cut separation algorithm. Here I study how best to set certain parameters and how to deal with issues arising from numerical stability.

- I address a question raised by Sanjeeb Dash and Oktay Günluk [40]. This question concerns the difficulty of finding cutting planes derived from single-row relaxations which outperform the MIR inequality in practice. In order to answer this question an algorithm is proposed which requires the use of the MIKP solver presented in the second chapter.

1.2 The traveling salesman problem (TSP)

Let $G = (V, E)$ be a complete graph with edge costs $(c_e : e \in E)$. The symmetric traveling salesman problem, or TSP, consists in finding a minimum-cost simple cycle (or tour) traversing all nodes of G. In the Dantzig, Fulkerson, and Johnson [37] cutting-plane method for the TSP, an integer programming model of the problem is defined using 0-1 variables x_e indicating if edge e is to be used in the optimal tour or not. By solving the Linear Programming (LP) relaxation of this formulation, and iteratively finding linear inequalities (or cutting planes) which were satisfied by all tour variables, Dantzig, Fulkerson and Johnson were able to solve a problems with 49 nodes. Considering the limited computational resources of the time, this was quite an achievement, and to date, this approach remains the most successful exact solution procedure for solving the TSP (see Jünger, Reinelt, and Rinaldi [70] and Naddef [84] for surveys of the broad literature on this approach).

The key step to the Dantzig, Fulkerson, and Johnson approach lies in being able to efficiently identify violated inequalities which are valid for the TSP. This step is often called *separation*. Much of the TSP literature is devoted to the study of classes of inequalities that are valid for the TSP, and which can be separated efficiently. Unfortunately, aside from the well-known subtour elimination constraints and the separation algorithm of blossom-inequalities (due to Padberg and Rao [91]), polynomial-time separation algorithms for tsp inequalities have proven to be elusive.

Besides the subtour elimination inequalities, perhaps the two best-known classes of constraints for the TSP are *combs* and *clique-trees* (see Chvátal [31], Grötschel and Padberg [65], and Grötschel and Pulleyblank [66]). The absence of efficient separation algorithms for these classes of inequalities has lead to the use of various heuristic methods

for handling them within cutting-plane algorithms. These heuristics are effective in many cases (see Padberg and Rinaldi [92], Applegate et al. [7], and Naddef and Thienel [86]), but additional exact methods could be critical in pushing TSP codes on to larger test instances.

An interesting new approach to TSP separation problems was adopted by Letchford [73], building on earlier work of Fleischer and Tardos [50]. Given an LP solution vector x^*, the support graph G^* is the sub-graph of G induced by the edge-set $E^* = \{e \in E : x^*_e > 0\}$. Letchford [73] introduced a new class of TSP inequalities which generalizes combs, called domino-parity constraints, and provided a separation algorithm in the case where G^* is a planar graph.

Letchford's results are very interesting for several reasons:

- It is a long-standing open question whether or not comb inequalities can be separated in polynomial time. While Letchford's result does not resolve this issue, it may be an important first step in that direction.

- An initial computational study of this algorithm by Boyd et al. [28], combining a computer implementation with by-hand computations, showed that the method can produce strong cutting planes for instances with up to 1,000 nodes. Cook, Espinoza, and Goycoolea [34] further studied Letchford's algorithm, presenting a range of procedures for improving its practical performance, and showed that it was very effective for solving instances with up to 30,000 nodes. In fact, by using the domino-parity cutting planes, they managed to solve two of the three remaining unsolved TSPLIB instances.

- Restricting the separation of comb inequalities to the case in which the support graph is planar is very practical for computation. While it is usually not the case (see Cook, Espinoza, and Goycoolea [34]) that the support graph G^* is planar, the graph G^* tends to be very sparse, and very "planar-like". Both reported implementations of Domino-Parity inequalities exploit this by using a procedure, which, given an LP solution vector x^*, "approximates" it by a vector x^{**} such that G^{**} is planar, and then proceeds to separate the point x^{**}. This approximation scheme, usually performed

by doing edge-contractions and edge-deletions, turns out to be remarkably effective. Furthermore, Cook, Espinoza, and Goycoolea [34] show that under certain conditions this approximation can be done "safely", i.e., in such a way as to guarantee that there is a violated inequality for x^{**} if and only if there is such an inequality for x^*. It would be interesting to see how this procedure of approximating the point to be separated could be applied to other integer programming applications.

- By generalizing comb inequalities to the class of Domino-Parity constraints, Letchford showed that behind comb inequalities, there is a much richer combinatorial structure. When Letchford's result first appeared, however, there was some concern that the class of Domino-Parity constraints might not be of much interest as a super-class of combs, as it may contain many weak valid inequalities. However, a recent result by Naddef [87] showed that under very generous conditions, Domino-Parity constraints are facet-defining.

In this work I present an extension to the work of Letchford, by introducing a new class of inequalities called the Generalized Domino Parity (GDP) constraints. Just as Domino-Parity constraints generalize comb-inequalities, I show that GDP constraints generalize the most well-known multiple-handled constraints, including clique-tree, bipartition, path, and star inequalities. Furthermore, I show that a sub-set of GDP constraints containing all of the clique-tree inequalities can be separated in polynomial time, provided that the support graph G^* is planar, and provided that we restrict the number of handles to a fixed, maximum size h.

1.3 The mixed integer knapsack problem (MIKP)

Consider a positive integer n. For each $k \in \{1, \ldots, n\}$ let $a_k, c_k \in \mathbb{Q}$, $l_k, \in \mathbb{Q} \cup \{-\infty\}$, and $u_k \in \mathbb{Q} \cup \{+\infty\}$. Let $b \in \mathbb{Q}$, and consider $I \subseteq \{1, \ldots, n\}$. The following problem will henceforth be referred to as the Mixed Integer Knapsack Problem (MIKP) :

$$
\begin{aligned}
\text{(MIKP)} \quad & \max \sum_{k=1}^{n} c_k x_k \\
& \text{s.t.,} \\
& \sum_{k=1}^{n} a_k x_k \leq b \\
& l_k \leq x_k \leq u_k, \ \forall k = 1, \ldots, n \\
& x_k \in \mathbb{Z}, \ \forall k \in I
\end{aligned}
$$

In Chapter 3 I present an algorithm for solving MIKP, that is, an algorithm which either (a) proves MIKP is infeasible, (b) proves MIKP is unbounded, or (c) finds an optimal solution to MIKP.

There are many variants of the MIKP which have been studied in the literature. In these it is traditionally assumed that all objective function coefficients, all constraint coefficients, and all variables must take integer and non-negative values. In addition:

- In the Knapsack Problem (KP), $l_i = 0$ and $u_i = 1$ for all $i \in 1, \ldots, n$.
- In the Bounded Knapsack Problem (BKP), $l_i = 0$ and $u_i < +\infty$ for all $i \in 1, \ldots, n$.
- In the Unbounded Knapsack Problem (UKP), $l_i = 0$ and $u_i = +\infty$ for all $i \in 1, \ldots, n$.

For a complete survey of these problems, as well as an up-to-date account of the most effective solution methodologies, the books by Kellerer, Pferschy, and Pisinger [72], and Martello and Toth [80] provide an excellent source of material.

Most modern algorithms for solving KP, BKP, and UKP are based either on branch-and-bound (following the work of Horowitz and Sahni [67]) or dynamic programming (following the work of Bellman [20]). However, the most efficient codes seldom make explicit use of Linear Programming. Further, I am not aware of any research having been conducted on mixed-integer variants of the knapsack problem, or variants of the knapsack problem in which some, but not all, of the variables are allowed to be unbounded as occurs in MIKP.

In Chapter 3 I present an LP-based branch-and-bound algorithm for MIKP. Solving MIKP is fundamentally different than solving KP, BKP, and UKP, given that (1) it is not clear how continuous variables should be introduced in the current solution methodologies

for KP, BKP, and UKP, and (2) MIKP works simultaneously with both bounded and unbounded variables.

The methodology that I propose is a linear-programming-based algorithm which exploits dominance conditions. One interesting aspect of this approach is that it differs from traditional linear-programming based algorithms by allowing feasible solutions to be pruned during the branching phase. The idea is that feasible solutions will only be pruned if either (a) they are not optimal (cost-domination-criteria), or (b) if they are optimal, but somewhere else in the tree it is known that there is another optimal solution (lexicographic-domination-criteria).

As I will show, the proposed algorithm performs quite well in practice, outperforming the general-use mixed integer programming solver CPLEX. While it is not surprising that a problem-specific algorithm such as the one proposed should outperform a general-use optimization software, it is important to note that only alternative for solving this type of problem is mixed integer programming. Notwithstanding, it seems clear from the computational study that the success of the algorithm is mostly due to the way domination is exploited. And this use of domination seems something that is actually not so problem-specific, but rather, something that can be applied to general mixed integer programming problems. Because of this success, a final section is included in this chapter where the main ideas behind the domination methodology are generalized. It would be interesting if the methodologies employed for this problem could effectively be implemented so as to improve the performance of general integer programming branch and bound algorithms.

1.4 The mixed integer rounding (MIR) cut

Consider $b \in \mathbb{R}$, $a, c \in \mathbb{R}^n$ and $l, u \in \mathbb{R}^n \cup \{+\infty, -\infty\}$. Let n_1, n_2 be integers such that $n = n_1 + n_2$, and consider:

$$Q = \{x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2} : ax \leq b,\ l \leq x \leq u\}.$$

Deriving strong valid inequalities for Q is of great practical importance to Mixed Integer Programming. This is because valid inequalities for single inequality systems such as Q can

be used as cutting-planes for multiple-row systems. In fact, let $A \in \mathbb{R}^{m \times n}$ and consider

$$P = \{x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2} : Ax \leq b,\ l \leq x \leq u\}.$$

If we assume that a is a conic combination of rows obtained from A we have $P \subseteq Q$. Hence, any inequality which is valid for Q will be valid for P.

Say that any valid inequality derived from a single row system is a *knapsack cut*. To date, the most successful cutting planes used for general mixed integer programming are all knapsack cuts. In fact, the Gomory Mixed Integer cut, and the Knapsack Cover inequalities are both classes of knapsack cuts. For a detailed study of the importance of these inequalities, see Bixby, Gu, Rothberg and Wunderling [24].

For the purposes of this thesis we will study the Gomory Mixed Integer cut as a member of a slightly broader class of inequalities known as the Mixed Integer Rounding (MIR) Inequalities. What is remarkable is that there exists a very simple "template" which can be used to derive an MIR from any single valid inequality of an MIP. When derived from the "correct" valid inequality, the MIR can be tremendously effective.

A fundamental question concerning MIR cuts is the following: What other inequalities derived from single valid inequalities are there? If the MIR is so effective, is there perhaps another class of inequalities which can easily be derived from single valid inequalities which is comparable?

Various researchers have tried to address these questions. Two important directions are as follows: The first direction starts from the observation that MIR inequalities are facets of the master cyclic group polyhedron (see Gomory and Johnson, [59], [60]). The work of Aaroz, Evans, Gomory and Johnson [9], Gomory, Johnson, and Evans [61], Dash and Günluk [42], [41], Dash, Goycoolea, and Günluk [38], and Fischetti and Saturni [49] attempt, both theoretically and computationally, to derive other facets of the cyclic group polyhedron. The second direction starts from the observation that MIR inequalities can be derived by lifting simpler systems with less variables. The work of Atamtürk [10], [11], Atamtürk and Rajan [12], Agra and Constantino [2], and Richard, de Farias, and Nemhauser [99] are examples which follow this approach. Though many new families of cuts have been derived through

these approaches, and much insight has been gained regarding the underlying structure of mixed integer cuts, little success has been achieved in being able to improve upon the performance of the MIR computationally.

Define as a *group cut* any valid inequality of the master cyclic group polyhedron. Recently, Dash and Günluk [40] made a startling computational observation: Observing that MIRs were the only group cuts to consistently yield positive results in separation algorithms, they devised an experiment which after adding MIR inequalities derived from tableau rows of a problem, tested if any other group cuts derived from those same tableau rows could possibly be violated. After performing this experiment on a very large set of MIP instances, they found that in a significant number of them, a notable 35% of the problems, no group cuts were violated in any of the tableau rows after the MIRs were added. This experiment, of course, would largely explain many of the unsuccessful attempts at adding additional classes of cuts.

For the second part of my thesis work I will begin a further exploration of this phenomenon by means of computational tests. For this, I focus on two main steps:

- **Implementation of an MIR inequality separation tool**

  The MIR inequality, as described in the literature, is not effective if used "as-is". Many techniques need to be used to obtain an effective implementation. Some of these techniques include: Scaling, bound complementation, using integrality of artificial variables, relaxing cut coefficients to increase numerical stability, etc. In Chapter 4 I present a computational section outlining the relative importance of different implementation features for separating MIR inequalities. An important part of working with MIR inequalities consists in effectively dealing with issues arising from numerical instability. These issues – if not properly dealt with – can very often lead to the generation of inequalities which are invalid. An extended discussion on this issue is also included, and some computational results illustrating the effectiveness of "safely" generated MIR inequalities are presented.

- **An Empirical follow up of Dash and Günluk's observation**

In Chapter 4 I also follow up on Dash and Günluk's observation by conducting an extended version of their experiment. The set-up is as follows: I solve each LP to optimality, save a copy of the tableau rows, add the GMI inequalities, and resolve. Then, instead of checking if there are any violated group-relaxation cuts, I check if there are any violated knapsack cuts. Furthermore, I do these tests both with finite and rational arithmetic. In order to do this separation I use the mixed integer knapsack solver developed in Chapter 3. The methodology follows the frameworks of Boyd [25] for Fenchel cut separation and of Applegate, Bixby, Chvátal and Cook [8] for Local Cut separation. Formally, let x^* correspond to the current fractional LP solution. Let $X = \{x^1, \ldots, x^k\}$ represent the extreme-point solutions of Q, and let $R = \{r^1, \ldots, r^q\}$ represent the extreme-rays of Q. Then, there exists no cutting plane separating x^* if and only if there exists $\lambda \in \mathbb{R}^k$ and $\alpha \in \mathbb{R}^q$ such that,

$$\begin{aligned} &\sum_{i=1}^{k} \lambda_i = 1 \\ &\sum_{i=1}^{k} \lambda_i x^i + \sum_{j=1}^{q} \alpha_j r^j = x^* \\ &\alpha_i \geq 0 \end{aligned}$$

Thus, the problem of testing separability reduces to determining if an LP is feasible or not. This problem is tackled with column generation, and using the mixed integer knapsack solver as a pricing oracle.

CHAPTER 2

A new class of valid inequalities for the traveling salesman problem

2.1 Introduction

Let $G = (V, E)$ be a complete graph with edge costs $(c_e : e \in E)$. If we identity each node in V with a "city", and each edge cost c_e with the "distance" or "cost" associated to traveling between a pair of cities, the symmetric traveling salesman problem, or TSP, consists in finding a minimum-cost tour by which to visit every city in G exactly once, and return back to the starting point. Formally, this problem can be restated as that of finding a minimum-cost Hamilton tour in G.

In the Dantzig, Fulkerson, and Johnson [37] cutting-plane method for the TSP, an integer programming model of the problem is defined using 0-1 variables x_e indicating if edge e is to be used in the optimal tour or not. By solving the Linear Programming (LP) relaxation of this formulation, and iteratively finding linear inequalities (or cutting planes) which were satisfied by all tour variables, Dantzig, Fulkerson and Johnson were able to solve a problem with 49 nodes which consisted in visiting all of the state capitals in the US. Considering the limited computational resources of the time, this was quite an achievement, and to date, this approach remains the most successful exact solution procedure for solving the TSP (see Jünger, Reinelt, and Rinaldi [70] and Naddef [84] for surveys of the broad literature on this

approach).

Rather than trying to identify just any linear inequality violated by the LP relaxation, the method of Dantzig, Fulkerson, and Johnson actually focuses on identifying specific classes of inequalities, and the separation is combined with a branch-and-bound scheme in order to effectively finish solving the problem. Several important classes of inequalities have been studied in the literature, the most important of which are the *subtour elimination constraints*.

For any $S \subseteq V$, let $\delta(S)$ denote the set of edges with exactly one end in S and let $E(S)$ denote the set of edges having both ends in S. For disjoint sets $S, T \subseteq V$, let $E(S:T)$ denote the set of edges having one end in S and one end in T. For any set $F \subseteq E$, define $x(F) := \sum(x_e : e \in F)$.

The subtour elimination constraints can formally be stated as follows:

$$x(\delta(S)) \geq 2 \qquad \forall\, \emptyset \neq S \subsetneq V. \tag{2.1}$$

An important property of these constraints is that the corresponding separation problem can be solved efficiently, that is, given a non-negative vector x^* a violated constraint can be found in polynomial time, provided one exists.

Many classes of inequalities that are valid for the TSP have been proposed which extend the subtour elimination constraints in different ways. However, for the most part polynomial-time separation algorithms have proven to be elusive.

The absence of other efficient separation algorithms has lead to the use of various heuristic methods for handling TSP inequalities within cutting-plane algorithms. The heuristics are effective in many cases (see Padberg and Rinaldi [92], Applegate, Bixby, Chvátal and Cook [7], and Naddef and Thienel [85], [86]), but additional exact methods could be critical in pushing TSP codes on to larger test instances.

An interesting new approach to TSP separation problems was adopted by Letchford [73], building on earlier work of Fleischer and Tardos [50]. Given an LP solution vector x^*, the support graph G^* is the subgraph of G induced by the edge set $E^* = \{e \in E : x^*_e > 0\}$. Letchford [73] introduced a new class of TSP inequalities, called domino-parity constraints,

and provided a separation algorithm in the case where G^* is a planar graph. An initial computational study of this algorithm by Boyd, Cockburn, and Vela [28], combining a computer implementation with by-hand computations, showed that the method can produce strong cutting planes for instances with up to 1,000 nodes. Cook, Espinoza, and Goycoolea [34] automated the methodology and extended it with heuristics and ad-hoc bounds. A computer implementation which used these techniques together with Concorde [7] had tremendous success, solving two previously unsolved instances of the TSP obtained from TSPLIB [98].

In this chapter we present a generalization of Letchford's results. We begin in Section 2.2 by describing in detail the classes of comb, clique-tree, bipartition, star, and domino-parity inequalities. We proceed, in Section 2.3, to define a new class of inequalities for the TSP which we call the generalized domino-parity (GDP) constraints, and show that these generalize all the afore-mentioned inequalities. In Section 2.4 we prove that violated GDP constraints may be characterized much in the same way as Letchford [73] characterizes violated domino-parity constraints. In Section 2.5 we use this characterization to give a polynomial time algorithm which, for any fixed h, separates a super-class of clique-tree inequalities.

2.2 Background: classes of TSP inequalities

2.2.1 The bipartition, clique-tree, and comb inequalities

Consider the families $\mathcal{H} = \{H^1, \ldots, H^h\}$ and $\mathcal{T} = \{T^1, \ldots, T^t\}$, where, $\emptyset \subsetneq H^i \subsetneq V$ for $i = 1, \ldots, h$ and $\emptyset \subsetneq T^j \subsetneq V$ for $j = 1, \ldots, t$. Assume that,

1. $H^i \cap H^j = \emptyset$ for $1 \leq i < j \leq h$,
2. $T^i \cap T^j = \emptyset$ for $1 \leq i < j \leq t$,
3. $T^j \setminus H^i \neq \emptyset$ for $1 \leq i \leq h, 1 \leq j \leq t$.

We will say that every set $H \in \mathcal{H}$ is a *handle*, and every set $T \in \mathcal{T}$ is a *tooth*.

For every $j = 1, \ldots, t$ define $t_j = |\{i \in 1, \ldots, h : T^j \cap H^i \neq \emptyset\}|$, and assume $t_j \geq 1$. If $T^j \setminus \bigcup\{H^i : i = 1, \ldots, h\}$ is non-empty, define $\beta_j = 1$, else define $\beta_j = t_j/(t_j - 1)$. For every $i = 1, \ldots, h$ define $h_i = |\{j \in 1, \ldots, t : H^i \cap T^j \neq \emptyset\}|$, and assume h_i is odd.

Boyd and Cunningham [27] proved that the following constraint, known as the *bipartition inequality*, is valid for the TSP:

$$\sum_{i=1}^{h} x(\delta(H^i)) + \sum_{j=1}^{t} \beta_j x(\delta(T^j)) \geq h + \sum_{i=1}^{h} h_i + 2\sum_{j=1}^{t} \beta_j. \tag{2.2}$$

Define a graph whose node-set is the union of $\mathcal{H}$ and $\mathcal{T}$. Define an edge between H^i and T^j in this graph if $H^i \cap T^j \neq \emptyset$. This graph is called the *intersection graph* defined by $\mathcal{H}$ and $\mathcal{T}$. Note that an intersection graph is always bipartite. If every tooth in $T^j \in \mathcal{T}$ is such that $\beta_j = 1$, and in addition, the intersection graph defined by $\mathcal{H}$ and $\mathcal{T}$ is a tree, it is easy to see that (2.2) is equivalent to,

$$\sum_{i=1}^{h} x(\delta(H^i)) + \sum_{j=1}^{t} x(\delta(T^j)) \geq 2h + 3t - 1. \tag{2.3}$$

This constraint, introduced by Grötschel and Pulleyblank [66], is known as a *clique-tree* inequality. A clique-tree inequality having a single handle is known as a *comb inequality* (see Chvátal [31], and Grötschel and Padberg [65]).

In Figure 2.1 we illustrate a bipartition constraint with three handles and ten non-degenerate teeth. This constraint has the form,

$$\sum_{i=1}^{3} x(\delta(H_i)) + \sum_{j=1}^{10} x(\delta(T_j)) \geq 44.$$

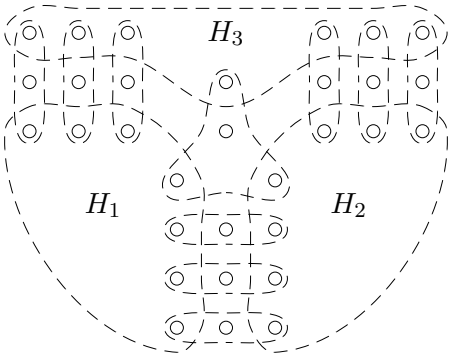


Figure 2.1: Example of a bipartition constraint on three handles.

2.2.2 The star inequalities

As before, consider the collections $\mathcal{H} = \{H^1, \ldots, H^h\}$ and $\mathcal{T} = \{T^1, \ldots, T^t\}$, where, $\emptyset \subsetneq H^i \subsetneq V$ for $i = 1, \ldots, h$ and $\emptyset \subsetneq T^j \subsetneq V$ for $j = 1, \ldots, t$. Now, assume:

1. $H^1 \subset H^2 \subset \ldots \subset H^h$,
2. $T^j \cap T^k = \emptyset \qquad \forall 1 \leq j < k \leq t$,
3. $H^1 \cap T^j \neq \emptyset$ for $1 \leq j \leq t$,
4. $T^j \setminus H^h \neq \emptyset$ for $1 \leq j \leq t$,
5. $(H^{i+1} \setminus H^i) \setminus \bigcup_{j=1}^{t} T^j = \emptyset$ for $1 \leq i \leq h-1$.

We say that $\hat{I} = \{l, l+1, \ldots, l+r\} \subseteq \{1, \ldots, h\}$ is an *interval* corresponding to tooth T_j if (i) $H^i \cap T^j = H^k \cap T^j$ for all $i, k \in \hat{I}$ and if (ii) $H^i \cap T^j \neq H^k \cap T^j$ for all $i \in \hat{I},\ k \notin \hat{I}$; that is, if $\hat{I}$ is a maximal index set of (successive) handles which have the same intersection with T^j. Consider $\alpha \in \mathbb{N}^h$, and $\gamma \in \mathbb{N}^t$. Assume that $\mathcal{H}$ and $\mathcal{T}$ satisfy the interval property with regards to α and γ; that is, assume that for each $1 \leq j \leq t$, and each interval $\hat{I}$ of T^j, we have $\gamma_j \geq \sum_{i \in \hat{I}} \alpha_i$. The following constraint, known as a *star inequality* (see Fleischmann [51]), is valid for the TSP:

$$\sum_{i=1}^{h} \alpha_i x(\delta(H^i)) + \sum_{j=1}^{t} \gamma_j x(\delta(T^j)) \geq (t+1) \sum_{i=1}^{h} \alpha_i + 2 \sum_{j=1}^{t} \gamma_j. \tag{2.4}$$

2.2.3 The domino-parity inequalities

A *domino* is a pair $\{T_1, T\}$ such that $\emptyset \subsetneq T_1 \subsetneq T \subsetneq V$. Let r be a positive integer and suppose that $E_1, \ldots, E_r$ are edge sets, i.e., $E_j \subseteq E$ for $j = 1, \ldots, r$. For each $e \in E$, define $\mu_e = |\{j \in \{1, \ldots, r\} : e \in E_j\}|$. That is, μ_e denotes the number of edge sets in which e appears. Let $H \subsetneq V$. Sets $E_1, \ldots, E_r$ are said to *support the cut* $\delta(H)$ if $\delta(H) = \{e \in E : \mu_e$ is odd$\}$. Observe that if $E_1, \ldots, E_r$ supports the cut $\delta(H)$ and x corresponds to a tour , then $\sum_{e \in E} \mu_e x_e$ is even valued. In fact, $\sum_{e \in E} \mu_e x_e = x(\delta(H)) + \sum_{e \in \delta(H)} x_e(\mu_e - 1) + \sum_{e \in E \setminus \delta(H)} \mu_e x_e$, and every term on the right is even-valued. Thus, the support of a cut is a kind of generalization of a cut.

Let p be a positive odd integer, and consider $\mathcal{T}$, a collection of p dominoes. Let $H \subseteq V$. Suppose that $F \subseteq E$, together with the sets $\{E(T_1 : T \setminus T_1)\}_{\{T_1, T\} \in \mathcal{T}}$, supports the cut $\delta(H)$ and define μ_e^H accordingly. The following constraint, known as the *domino-parity inequality* (see Letchford [73]) is valid for the TSP:

$$\sum_{e \in E} \mu_e^H x_e + \sum_{\{T_1, T\} \in \mathcal{T}} x(\delta(T)) \geq 3p + 1. \tag{2.5}$$

It is easy to see that domino-parity constraints generalize comb inequalities. In fact, let $\mathcal{T}$ define the teeth of a comb inequality, and let H be its handle. For every $T \in \mathcal{T}$ define a domino $\{T \cap H, T\}$. It is not difficult to verify that these dominoes, together with H, define a domino-parity inequality (see Letchford [73]).

2.3 The generalized domino-parity (GDP) inequalities

In this section, we construct a generalization of the domino-parity inequalities and show that this generalized class of constraints strictly contains bipartition and star inequalities, much in the same way as domino-parity inequalities contain comb inequalities.

2.3.1 Multi-dominoes

Consider a non-negative integer k and a family of node sets $\hat{T} = \{T_1, T_2, \ldots, T_k, T\}$ such that $\emptyset \neq T_i \subsetneq T \subsetneq V,\ \forall i = 1, \ldots, k$. We say that this family defines a *multi-domino* if for any set $\emptyset \neq K \subseteq \{1, \ldots, k\}$, the edges $\bigcup \{E(T_i : T \setminus T_i) : i \in K\}$ define a $|K| + 1$ (or greater) cut in the subgraph of the complete graph G induced by T.

Consider a positive integer k and a family of node sets $\hat{T} = \{T_1, T_2, \ldots, T_k, T\}$ satisfying $\emptyset \neq T_i \subsetneq T \subsetneq V,\ \forall i = 1, \ldots, k$. $\hat{T}$ is said to define a *degenerate multi-domino* if $\{T_1, \ldots, T_k\}$ defines a partition of T.

Note that unless otherwise specified, we will use the term multi-domino to refer both to degenerate and non-degenerate multi-dominoes.

In general, given a multi-domino $\hat{T} = \{T_1, \ldots, T_k, T\}$, we will say that T is its *ground-set*, and $T_1, \ldots, T_k$ are its *halves*. If a multi-domino $\hat{T}$ has k halves, we say that it is a *k-domino*, and write $\kappa(\hat{T}) = k$. Observe that k-dominoes (both degenerate and non-degenerate) satisfy the following recursive condition: If you remove any number $0 < r \leq k$ of halves from a k-domino (leaving the ground set intact), you obtain a $k - r$ domino which is non-degenerate. In addition, note that the definition of a 1-domino is equivalent to the domino definition of Letchford [73], and a 0-domino consists of a singleton containing a ground set and no halves. Whenever a multi-domino has more than one half, we will say that it is *large*. Finally, observe that for notation purposes, we will distinguish a multi-domino $\hat{T}$ from its ground set T by using a hat (“ ˆ ”) symbol.

If a k-domino $\hat{T}$ is degenerate, define $\beta(\hat{T}) = \frac{k}{k-1}$. Otherwise, define $\beta(\hat{T}) = 1$.

Lemma 2.1. Let $\hat{T} = \{T_1, T_2, \ldots, T_k, T\}$ be a $k-$domino. If x satisfies all subtour constraints, then

$$\frac{\beta(\hat{T})}{2}(x(\delta(T)) - 2) + \sum_{i=1}^{k} x(E(T_i : T \setminus T_i)) \geq k.$$

Proof. Assume x satisfies all subtour constraints. If $k = 0$ the result trivially follows from the subtour elimination constraints, so assume $k \geq 1$ and let $B_1, B_2, \ldots, B_r$ correspond to the partition of T obtained by removing the edge sets $E(T_1 : T \setminus T_1), E(T_2 : T \setminus T_2), \ldots, E(T_k : T \setminus T_k)$ from the subgraph of G induced by T. Note that

$$\sum_{i=1}^{r} x(\delta(B_i)) = x(\delta(T)) + \sum_{i=1}^{r} x(E(B_i : T \setminus B_i)).$$

It follows that

$$\frac{\beta(\hat{T})}{2}\left(x(\delta(T)) - 2\right) = \frac{\beta(\hat{T})}{2}\left(\sum_{i=1}^{r}\left(x(\delta(B_i)) - x(E(B_i : T \setminus B_i))\right) - 2\right). \tag{2.6}$$

However, note that if $\hat{T}$ is non-degenerate, then $\beta(\hat{T}) = 1$ and

$$\sum_{i=1}^{r} x(E(B_i : T \setminus B_i)) \leq 2\sum_{i=1}^{k} x(E(T_i : T \setminus T_i)).$$

On the other hand, if $\hat{T}$ is degenerate, then $\beta(\hat{T}) \leq 2$ and each T_i can be assumed equal to B_i. Thus, in either case, we have

$$\frac{\beta(\hat{T})}{2}\sum_{i=1}^{r} x(E(B_i : T \setminus B_i)) \leq \sum_{i=1}^{k} x(E(T_i : T \setminus T_i)). \tag{2.7}$$

Finally, note that if $\hat{T}$ is non-degenerate, then $r > k$ and $\beta(\hat{T}) = 1$. Likewise, if $\hat{T}$ is degenerate then $r = k$ and $\beta(\hat{T}) = k/(k-1)$. Thus, in both cases, $\beta(\hat{T})(2r-2)/2 \geq k$, and

$$\frac{\beta(\hat{T})}{2}\left(\sum_{i=1}^{r} x(\delta(B_i)) - 2\right) \geq \frac{\beta(\hat{T})}{2}(2r - 2) \geq k. \tag{2.8}$$

Putting together (2.6), (2.7), and (2.8) we get the desired result. ∎

2.3.2 Defining the GDP inequalities

Recall that comb inequalities require an odd number of teeth to intersect the handle of the constraint. However, for domino-parity inequalities, this requirement is relaxed, and though no conditions are imposed in terms of intersections, an odd number of teeth is still associated to the handle, but in a more abstract way through the notion of "supporting a cut". Again, in bipartition inequalities, handles are required to intersect an odd number of teeth. Since we are interested in generalizing domino-parity constraints to a class of inequalities containing bipartitions, we will need to generalize this association between handles and teeth to multiple handle configurations. In order to do this, we will map teeth, which in our new inequalities will be represented by multi-dominoes, to handles, by means of a function Φ, which associates each half of a multi-domino to a handle.

Consider a positive integer h and a family $\mathcal{T}$ of multi-dominoes. Let Φ define a map between halves of the multi-dominoes in $\mathcal{T}$ and numbers in $\{1,\dots,h\}$. That is, for every multi-domino $\hat{T} \in \mathcal{T}$, such that $\kappa(\hat{T}) \geq 1$, and every $j \in 1,\dots,\kappa(\hat{T})$, let $\Phi(\hat{T},j)$ take a value in $\{1,\dots,h\}$. We say that Φ is an h-tooth association defined over $\mathcal{T}$, and whenever $\Phi(\hat{T},j) = i$ for some $i \in \{1,\dots,h\}$ we will say that the i-th handle and $\hat{T}$ are associated to each other by means of Φ.

Theorem 2.1. Consider a family of node sets $\mathcal{H} = \{H_1,\dots,H_h\}$, and a family of multi-dominoes $\mathcal{T}$. Let Φ define an h-tooth association over $\mathcal{T}$, and assume that $|\Phi^{-1}(i)|$ is odd, for each $i = 1,\dots,h$. For each $H_i \in \mathcal{H}$ define $F_i \subseteq E$ such that $\{F_i\}$ and $\{E(T_j : T \setminus T_j) : \Phi(\hat{T},j) = i\}$ support the cut $\delta(H_i)$ in G and define μ^i accordingly. Then the inequality

$$\sum_{i=1}^{h} \mu^i x + \sum_{\hat{T}\in\mathcal{T}} \beta(\hat{T}) x(\delta(T)) \geq h + \sum_{i=1}^{h} h_i + 2\sum_{T\in\mathcal{T}} \beta(\hat{T}). \tag{2.9}$$

is satisfied by all tours, where $h_i = |\Phi^{-1}(i)|$ for each $i = 1,\dots,h$.

Proof. We use induction on h, the case $h = 0$ following from the validity of the subtour constraints. Let $\hat{x}$ be the incidence vector of a tour. If there exists $i_o \in \{1,\dots,k\}$ such that $\mu^{i_o}\hat{x} > h_{i_o} - 1$, then, since $\mu^{i_o}\hat{x}$ is even valued (see Section 2.2.3), we have $\mu^{i_o}\hat{x} \geq h_{i_o} + 1$. For every $\hat{T} \in \mathcal{T}$ define $\hat{T}^* = \{T_1, T_2, \dots, T_{\kappa(T)}, T\} \setminus \{T_j :$

$\Phi(\hat{T}, j) = i_o\}$. Note that for each $\hat{T} \in \mathcal{T}$ we have $\beta(\hat{T}^*) \leq \beta(\hat{T})$. Thus, by induction, the inequality obtained by removing handle H_{i_o}, replacing each $\hat{T} \in \mathcal{T}$ by $\hat{T}^*$, and using the same association Φ, renumbering appropriately,

$$\sum_{i=1, i\neq i_o}^{k} \mu^i x + \sum_{\hat{T}^* \in \mathcal{T}} \beta(\hat{T}^*) x(\delta(T)) \geq (h-1) + \sum_{i=1, i\neq i_o}^{k} h_i + 2 \sum_{\hat{T}^* \in \mathcal{T}} \beta(\hat{T}^*)$$

is valid. Then (2.9) follows since $(\beta(\hat{T}) - \beta(\hat{T}^*))\hat{x}(\delta(T)) \geq (\beta(\hat{T}) - \beta(\hat{T}^*))2$, and $\mu^{i_o}\hat{x} \geq h_{i_o} + 1$.

Now assume that $\mu^i \hat{x} \leq h_i - 1$ for each $i = 1, \ldots, h$. From Lemma 2.1 we have for each $T \in \mathcal{T}$

$$\beta(\hat{T})(\hat{x}(\delta(T)) - 2) \geq 2\kappa(T) - 2 \sum_{j=1}^{\kappa(T)} \hat{x}(E(T_j : T \setminus T_j)). \tag{2.10}$$

Noting that,

$$\sum_{\hat{T} \in \mathcal{T}} \kappa(\hat{T}) = \sum_{i=1}^{h} h_i,$$

and,

$$\sum_{i=1}^{h} \hat{x}(F_i) + \sum_{\hat{T} \in \mathcal{T}} \sum_{j=1}^{\kappa(\hat{T})} \hat{x}(E(T_j : T \setminus T_j)) = \sum_{i=1}^{h} \mu^i \hat{x}$$

and then summing over (2.10) we obtain,

$$\begin{aligned}
\sum_{T \in \mathcal{T}} \beta(\hat{T})(\hat{x}(\delta(T)) - 2) &\geq 2 \sum_{\hat{T} \in \mathcal{T}} \kappa(\hat{T}) - 2 \sum_{\hat{T} \in \mathcal{T}} \sum_{j=1}^{\kappa(\hat{T})} \hat{x}(E(T_j : T \setminus T_j)) \\
&\geq 2 \sum_{i=1}^{h} h_i - 2 \sum_{i=1}^{h} \mu^i \hat{x} \\
&= \sum_{i=1}^{h} h_i + \sum_{i=1}^{h} \left(h_i - \mu^i \hat{x}\right) - \sum_{i=1}^{h} \mu^i \hat{x} \\
&\geq \sum_{i=1}^{h} h_i + h - \sum_{i=1}^{h} \mu^i \hat{x}.
\end{aligned}$$

∎

We refer to the inequalities (2.9) as *generalized domino-parity (GDP) inequalities*. As in other well-known TSP inequalities, we will denote the sets $H_1, \ldots, H_h$ as *handles*, and the multi-dominoes $\hat{T} \in \mathcal{T}$ as *teeth*. We will say that teeth with more than one half are *large teeth*. If a multi-parity constraint has h handles, we say that it is an *h-parity constraint*.

When $h = 1$, and every tooth is non-degenerate and restricted to having at most one half, this class coincides with that of the *domino-parity inequalities* of Letchford [73]. In order to represent generalized domino-parity inequalities we will identify them in terms of the tuples $(\mathcal{H}, \Phi, \mathcal{T})$, or equivalently, the tuples $(\mathcal{F}, \Phi, \mathcal{T})$, corresponding to the handles (or sets F_i), the h-tooth association, and the teeth which define them.

2.3.3 Stars and bipartitions are GDP inequalities

Proposition 2.1. The class of $h-$parity inequalities contains the class of bipartition inequalities having h handles.

Proof. Consider a bipartition inequality with handles $\mathcal{H} = \{H^1, \ldots, H^h\}$ and teeth $\mathcal{T} = \{T^1, \ldots, T^t\}$. Define a set of multi-dominoes $\mathcal{T}'$ and an h-tooth association Φ by repeating the following procedure:

Step 1. Choose a tooth $T^j \in \mathcal{T}$ and define a zero-domino $\hat{T}^j \in \mathcal{T}'$ having ground set T^j.

Step 2. For each handle $H^i \in \mathcal{H}$ such that $T^j \cap H^i \neq \emptyset$: Let $r = \kappa(\hat{T}^j)$. Add half $T^j \cap H^i$ to $\hat{T}$, and define $\Phi(\hat{T}, r+1) = i$.

It is easy to see that this procedure leads to a valid h-parity inequality which coincides with the original bipartition inequality. ∎

Given that bipartition inequalities generalize clique-tree inequalities, Proposition 2.1 also tells us that h-parity inequalities generalize clique-tree inequalities on h handles.

Proposition 2.2. The class of GDP inequalities contains the class of star inequalities.

Proof. Consider a star inequality:

$$\sum_{i=1}^{h} \alpha_i x(\delta(H^i)) + \sum_{j=1}^{t} \gamma_j x(\delta(T^j)) \geq (t+1) \sum_{i=1}^{h} \alpha_i + 2 \sum_{j=1}^{t} \gamma_j. \tag{2.11}$$

Assume $\mathcal{T} = \{T^1, \ldots, T^t\}$, $\mathcal{H} = \{H^1, \ldots, H^h\}$, α, and γ are defined as in Section 2.2. We construct a family of teeth $\mathcal{T}'$ and handles $\mathcal{H}'$ as follows: For each set $H^i \in \mathcal{H}$ we define α_i copies of H_i in $\mathcal{H}'$. We will denote the k-th copy of handle $H^i \in \mathcal{H}$ as

$H'(i,k)$. Likewise, for each set $T^j \in \mathcal{T}$ we will define γ_j zero-dominoes in $\mathcal{T}'$, each having T^j as a ground set. We will denote the k-th tooth defined from $T^j \in \mathcal{T}$ as $\hat{T}'(j,k)$. Note that $h' = \sum_{i=1}^{h} \alpha_i$ will correspond to the total number of handles in $\mathcal{H}'$, and $t' = \sum_{i=1}^{t} \gamma_j$ will correspond to the total number of teeth in $\mathcal{T}'$. We now focus on adding the appropriate halves to each tooth in $\mathcal{T}$, and on defining an appropriate h'-tooth association Φ over $\mathcal{T}'$ such that $|\Phi^{-1}(i)|$ is odd for each $i = 1, \ldots, h'$. The idea will be that to each handle in $\mathcal{H}'$ we will associate *exactly* t teeth in $\mathcal{T}'$ (thus ensuring the parity condition) and to each tooth in $\mathcal{T}'$ we will assign *at most* h handles in $\mathcal{H}'$.

To define the halves of the teeth in $\mathcal{T}'$ and to define Φ we proceed algorithmically. For this, let $\mathcal{I}$ be the set of all intervals associated to the star inequality.

Step 1. Choose $T^j \in \mathcal{T}$ and an associated interval I. Define $\mathcal{H}'(I) = \{H'(i,k) \in \mathcal{H}' : i \in I,\ 1 \leq k \leq \alpha_i\}$ and $\mathcal{T}'(j) = \{\hat{T}'(j,k) \in \mathcal{T}' : 1 \leq k \leq \gamma_j\}$. Since $\sum_{i \in I} \alpha_i \leq \gamma_j$ it follows that $|\mathcal{H}'(I)| \leq |\mathcal{T}'(j)|$.

Step 2. Choose $H'(i,k) \in \mathcal{H}'(I)$ and $\hat{T}'(j,p) \in \mathcal{T}'(j)$. Denote $H'(i,k)$ by H'_q and $\hat{T}'(j,p)$ by $\hat{T}'$ (in order to use our indexing conventions). Let $r = \kappa(\hat{T}') + 1$, add a half $T_{r+1} = H^i \cap T^j$ to $\hat{T}'$, and define $\Phi(\hat{T}, r+1) = q$. Remove $H'(i,k)$ from $\mathcal{H}'(I)$ and remove $\hat{T}'(j,k)$ from $\mathcal{T}'(j)$. If $\mathcal{H}'(I)$ is non-empty, go back to Step 2. Otherwise, proceed to Step 3.

Step 3. Remove I from $\mathcal{I}$. If T^j has no more associated intervals, remove T^j from $\mathcal{T}$. If $\mathcal{T}$ is non-empty, go to Step 1. Otherwise, stop.

To see that this construction leads to a valid GDP inequality, we first show that $|\Phi^{-1}(q)| = t$ for all handles $H'_q \in \mathcal{H}$. To see this, observe that for every handle $H^i \in \mathcal{H}$ and every tooth $T^j \in \mathcal{T}$ there exists exactly one interval I associated to T^j such that $i \in I$. Consider the iteration of Step 1 when T^j and I are the current tooth-interval pair. In this iteration, all α_i copies of H_i in $\mathcal{H}'$ will be in set $\mathcal{H}'(I)$. Since $|\mathcal{H}'(I)| \leq |\mathcal{T}'(j)|$, each of these copies will be assigned to some tooth $\hat{T}' \in \mathcal{T}'$ by Φ. Given that for every other interval associated to T^j these copies of H^i will not

be assigned to any teeth, the conclusion follows.

Next, we show that the multi-dominoes in $\mathcal{T}'$ are well defined. Given that every half added to multi-dominoes in Step 2 is contained in the corresponding ground set, we just need to show that the recursive cut condition holds. To see this, consider a multi-domino $\hat{T}'(j,k) \in \mathcal{T}'$. Observe that all of the halves of $\hat{T}'(j,k)$ were added in different iterations of Step 2. Thus, every half of $\hat{T}'(j,k)$ is of the form $T^j \cap H^i$, where the sets H^i are such that their respective indices are in different intervals of tooth T^j. Given that any two sets H_{i_1} and H_{i_2} corresponding to different intervals of a same tooth $T^j \in \mathcal{T}$ are such that one of the corresponding halves strictly contains the other, it follows that all of the halves of $\hat{T}'(j,k)$ can be sorted by strict inclusion. It is easy to see that this implies that the recursive cut condition holds.

Thus, we have that $(\mathcal{H}', \Phi, \mathcal{T}')$ defines a GDP inequality.

To see that the right-hand-side of the GDP inequality coincides with that of the original star inequality, recall that $h' = \sum_{i=1}^{h} \alpha_i$ and $|\mathcal{T}'| = \sum_{i=1}^{t} \gamma_j$. Further, for every handle $H'_q \in \mathcal{H}'$ we have that $h'_q = t$, and for every tooth $\hat{T}' \in \mathcal{T}'$ we have that $\beta(\hat{T}') = 1$. Thus, the right-hand side of the GDP inequality is $h' + \sum_{q=1}^{h'} h'_q + |\mathcal{T}'| = \sum_{i=1}^{h} \alpha_i + t\sum_{i=1}^{h} \alpha_i + 2\sum_{i=1}^{t} \gamma_j$, which coincides with the right-hand-side of the corresponding star inequality.

To see that the left hand side coincides, we first show that every handle $H'(i,k) \in \mathcal{H}'$ has assigned t teeth whose ground sets are disjoint of each other. In fact, consider a handle $H'(i,k) \in \mathcal{H}'$ and a ground set T^j. All of the teeth in $\mathcal{T}'$ whose ground-set is T^j are in $\mathcal{T}'(j)$; all other teeth in $\mathcal{T}'$ have ground sets which are disjoint from T^j. Observe that there is a single interval I associated to T^j such that $H'(i,k) \in \mathcal{H}'(I)$. Thus there is only one iteration of Step 2 in which handle $H'(i,k)$ could be assigned a tooth with ground set T^j. Since in Step 2 each handle is assigned at most one tooth, it follows that all of the teeth assigned to $H'(i,k)$ have disjoint ground sets.

Given that for each handle $H'_q \in \mathcal{H}'$ all of the assigned teeth are disjoint, it follows that $\mu^q x = x(\delta(H'_q))$. Further, given that for each handle $H_i \in \mathcal{H}$ there are α_i

copies in $\mathcal{H}'$ it follows that $\sum_{q=1}^{h'} \mu^q x = \sum_{i=1}^{h} \alpha_i x(\delta(H_i))$. Analogously, we know that $\mathcal{T}'$ can be partitioned into the collections $\mathcal{T}'(j)$ with $j = 1, \ldots, t$, and that each collection $\mathcal{T}'(j)$ has γ_j multi-dominoes with ground-set T^j. Thus, $\sum_{\hat{T}' \in \mathcal{T}'} x(\delta(T')) = \sum_{j=1}^{t} \gamma_j x(\delta(T^j))$. From this we conclude that the left-hand side of the resulting GDP inequality and that of the original star inequality coincide. ∎

What the preceding discussion highlights is that star inequalities are nothing more than GDP inequalities such that: (a) for every pair of handles, one must contain the other, and (b) every pair of teeth either have completely disjoint ground sets, or the ground sets are exactly alike.

Note that in addition to containing bipartition and star inequalities, the class of GDP inequalities contains many other new and different structures. For instance, note that a tooth in a GDP inequality can be associated many times to the same handle. Also, GDP inequalities allow many configurations which may be obtained by combining star and bipartition inequalities, as well as abstractions of star and bipartition inequalities where teeth are allowed to intersect each other.

It is easy to see that not all GDP inequalities define facets of the TSP polytope, but the class does provide a common framework for possibly extending Letchford's algorithm to super-classes of other inequalities that have proven to be effective in TSP codes.

2.4 Properties of violated GDP inequalities

In this section we describe necessary and sufficient conditions for an h-parity constraint to be violated.

2.4.1 A characterization of violated GDP inequalities

Define the *weight* of k-domino $\hat{T} = \{T_1, T_2, \ldots, T_k, T\}$ to be

$$w(\hat{T}) := \beta(\hat{T})(x(\delta(T)) - 2) + \sum_{i=1}^{k} x(E(T_i : T \setminus T_i)) - k. \tag{2.12}$$

Lemma 2.2. Consider an h-parity inequality defined by $\mathcal{H}$, $\mathcal{T}$, and Φ. Let F_i be such that $\{E(T_j : T \setminus T_j) : \Phi(\hat{T}, j) = i\}$ and $\{F_i\}$ support the cut $\delta(H_i)$ for each $i = 1, \ldots, h$. The

slack of the h-parity inequality is

$$\sum_{\hat{T}\in\mathcal{T}} w(\hat{T}) + \sum_{i=1}^{h} x(F_i) - h.$$

Proof. The slack is,

$$\begin{aligned}
& \sum_{i=1}^{h} \mu^i x + \sum_{\hat{T}\in\mathcal{T}} \beta(\hat{T}) x(\delta(T)) - h - \sum_{i=1}^{h} h_i - 2\sum_{\hat{T}\in\mathcal{T}} \beta(\hat{T}) \\
= & \sum_{i=1}^{h} x(F_i) + \sum_{\hat{T}\in\mathcal{T}} \sum_{j=1}^{\kappa(\hat{T})} x(E(T_j : T \setminus T_j)) + \sum_{\hat{T}\in\mathcal{T}} \left(\beta(\hat{T}) \left(x(\delta(T)) - 2 \right) - \kappa(\hat{T}) \right) - h \\
= & \sum_{i=1}^{h} x(F_i) + \sum_{\hat{T}\in\mathcal{T}} \left(\beta(\hat{T}) \left(x(\delta(T)) - 2 \right) + \sum_{j=1}^{\kappa(\hat{T})} x(E(T_j : T \setminus T_j)) - \kappa(\hat{T}) \right) - h \\
= & \sum_{i=1}^{h} x(F_i) + \sum_{\hat{T}\in\mathcal{T}} w(\hat{T}) - h.
\end{aligned}$$

∎

Note that Lemma 2.1 and Lemma 2.2 together imply that if x satisfies all subtour elimination constraints, then a violated h-parity constraint must satisfy

$$0 \leq \frac{\beta(\hat{T})}{2}(x(\delta(T)) - 2) \leq w(\hat{T}) < h \qquad \forall \hat{T} \in \mathcal{T}. \tag{2.13}$$

Further, note that in many classes of well-known TSP inequalities, the handles are disjoint and halves of a multi-domino correspond to tooth-handle intersections (for example, clique-tree inequalities and bipartition inequalities - see Proposition 2.1). In these cases it is not difficult to see that every k-domino $\hat{T}$ participating in such inequalities will satisfy $w(\hat{T}) \geq \sum_{j=1}^{k} x(\delta(T_j)) - k - 2$. Thus, if all subtour elimination constraints are satisfied, $w(\hat{T}) \geq k - 2$. This means that it is possible to bound the number of teeth having three or more halves which participate in violated h-parity constraints.

Lemma 2.3. There exists an h-parity inequality with slack s^* iff there exist a family of multi-dominoes $\mathcal{T}$, an h-tooth association Φ, and sets $R_i \subseteq E^*$ for all $i \in \{1, \ldots, h\}$ such that:

1. $|\Phi^{-1}(i)|$ is odd, for all $i = 1, \ldots, h$.
2. $\{E^*(T_j : T \setminus T_j) : \Phi(\hat{T}, j) = i\}$ and $\{R_i\}$ support a cut in G^* for all $i = 1, \ldots, h$.

3. $s^* = \sum_{i=1}^{h} x^*(R_i) + \sum_{\hat{T}\in\mathcal{T}} w(\hat{T}) - h.$

Proof. From Theorem 2.1 and Lemma 2.2, there exists an h-parity inequality with slack s^* iff there exists a family of node sets $\mathcal{H} = \{H_1, \ldots, H_h\}$ in G, a family of edge sets $\{F_1, \ldots, F_h\}$ in G, a family of multi-dominoes $\mathcal{T}$ in G, and an h-tooth association Φ defined over $\mathcal{T}$, such that:

(a) $|\Phi^{-1}(i)|$ is odd, for $i = 1, \ldots, h$.

(b) $\{E(T_j : T \setminus T_j) : \Phi(\hat{T}, j) = i\}$ and $\{F_i\}$ support the cut $\delta(H_i)$ in G for all $i = 1, \ldots, h$.

(c) $s^* = \sum_{i=1}^{h} x^*(F_i) + \sum_{\hat{T}\in\mathcal{T}} w(\hat{T}) - h.$

Necessity is trivial, since we can just take $R_i = F_i \cap E^*$. So we focus on sufficiency. Assume that $\mathcal{T}$ and Φ define an h-tooth association, and sets $R_i \subseteq E^*$ for $i \in 1, \ldots, h$ are such that (ii) and (iii) hold. For each $i \in 1, \ldots, h$ let $H_i \subseteq V$ be one shore of the cut supported by $\{E^*(T_j : T \setminus T_j) : \Phi(\hat{T}, j) = i\}$ and R_i. A set F_i satisfying (b)-(c) can be obtained from R_i by adding edges $e \in \delta(H_i)$ such that $x^*_e = 0$. ∎

2.4.2 Planar duality and violated GDP inequalities

We henceforth assume that G^* is a planar graph and let $\bar{G}^*$ denote the planar dual of G^*. For any subset $F \subseteq E(G^*)$, denote by $\bar{F}$ the corresponding edges in $\bar{G}^*$. For each $\bar{e} \in \bar{G}^*$ let $x^*_{\bar{e}} = x^*_e$.

A graph is called *Eulerian* if every node has even degree. (As in Letchford [73], we do not require that Eulerian graphs be connected.)

Let r be a positive integer and suppose that $E_1, \ldots, E_r$ are edge sets satisfying $E_i \subseteq E^*$, $i = 1, \ldots, r$. As before, let $\mu_e = |\{i : e \in E_i\}|$. The collection $\{\bar{E}_i : i = 1, \ldots, r\}$ is said to *support an Eulerian subgraph* in $\bar{G}^*$ if the edges $\bar{e}$ for which μ_e is odd form an Eulerian subgraph in $\bar{G}^*$.

We say that a cut $C \subseteq E(G)$ is *minimal* if removing any subset of edges from C results in an edge set which does not define a cut. Observe that for any set $A \subsetneq V$ the cut $\delta(A)$ can always be decomposed into an edge disjoint union of minimal cuts. A well known result (see

Mohar and Thomassen [83]) is that if G is planar and $C \subseteq E(G)$ is a minimal cut, then $\overline{C}$ is a simple cycle in $\bar{G}^*$. Since every eulerian subgraph can be decomposed into edge disjoint simple cycles, this result implies that $\{\bar{E}_i : i = 1.\ldots, r\}$ supports an Eulerian subgraph in $\bar{G}^*$ iff $\{E_i : i = 1, \ldots, r\}$ supports a cut in G^*. This observation implies the following dual version of Lemma 2.3.

Lemma 2.4. There exists an h-parity inequality having slack s^* iff there exist a family of multi-dominoes $\mathcal{T}$, an h-tooth association Φ, and sets $\bar{R}_i \subseteq \bar{E}^*$ for all $i \in \{1, \ldots, h\}$ such that:

1. $|\Phi^{-1}(i)|$ is odd, for all $i = 1, \ldots, h$.
2. $\{\overline{E^*(T_j : T \setminus T_j)} : \Phi(\hat{T}, j) = i\}$ and $\{\bar{R}_i\}$ support an Eulerian subgraph in $\bar{G}^*$ for all $i = 1, \ldots, h$.
3. $s^* = \sum_{i=1}^{h} x^*(\bar{R}_i) + \sum_{\hat{T} \in \mathcal{T}} w(\hat{T}) - h$.

Proof. Follows from the definitions. ∎

In Figure 2.2 we illustrate the relevant edges of a two-parity constraint in the dual graph $\bar{G}^*$. This example has a total of six teeth. Also we assume that one of the teeth strictly contains two other teeth. In the illustration the dark circles represent nodes in $\bar{V}^*$, the solid lines represent edges in $\bar{R}_i$ for $i = 1, 2$, the dashed lines represent edges in $\overline{\delta(T)}$ for $T \in \mathcal{T}$, and the dotted lines represent edges in $\overline{E(T_j : T \setminus T_j)}$ for $j \in 1, \ldots, h$ such that $\Phi(T, j) \in \{1, 2\}$. Note that the solid and dotted lines together support two Eulerian subgraphs, one for each handle.

Lemma 2.5. Let edge sets $\{E_1, \ldots, E_k\}$ support an eulerian subgraph in G.

(a) Let $S \subseteq E(G)$ be eulerian, and assume $S = S_1 \cup S_2 \cup \ldots \cup S_k$, where the sets $S_1, \ldots, S_k$ are pairwise disjoint. Then, if $\{E_1, \ldots, E_k\}$ supports the eulerian subgraph D, the edge sets $\{E_1 \Delta S_1, E_2 \Delta S_2, E_3 \Delta S_3, \ldots, E_k \Delta S_k\}$ support the eulerian subgraph $D \Delta S$.

(b) Consider any edge set $S \subseteq E(G)$, then $\{E_1 \Delta S, E_2 \Delta S, E_3, \ldots, E_k\}$ supports the same eulerian subgraph as $\{E_1, E_2, \ldots, E_k\}$ in G.

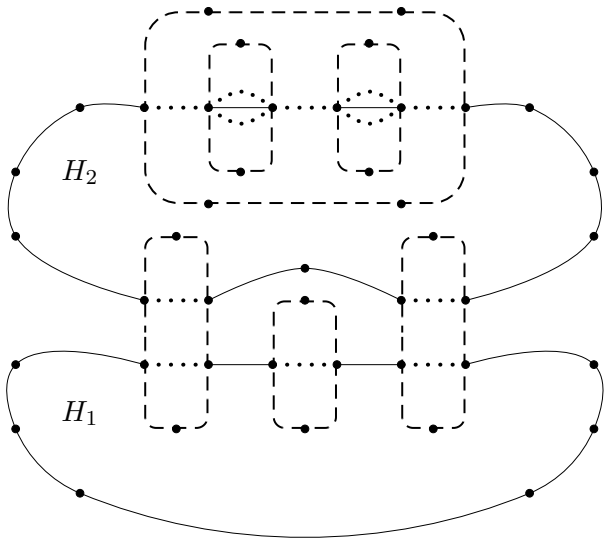


Figure 2.2: A two-parity constraint represented in $\bar{G}^*$

Proof. (a) For each edge $e \in E$, let μ_e represent the number of sets in $\{E_1, \ldots, E_k\}$ containing edge e. Let μ'_e represent the number of sets in $\{E_1 \Delta S_1, \ldots, E_k \Delta S_k\}$ containing edge e. Observe that $\mu_e = \mu'_e$ for each edge $e \in E \setminus S$. Now consider $e \in S_i$. Observe that $e \in E_i \cap S_i$ implies $\mu'_e = \mu_e - 1$, and $e \in S_i \setminus E_i$ implies $\mu'_e = \mu_e + 1$. Now consider the sets $D = \{e \in E : \mu_e \text{ is odd }\}$ and $D' = \{e \in E : \mu'_e \text{ is odd }\}$. From the previous arguments it follows that $e \in S \cap D$ implies $e \notin D'$. Likewise, $e \in S \setminus D$ implies $e \in D'$. Thus, $D' = (D \setminus (S \cap D)) \cup (S \setminus D) = (D \setminus S) \cup (S \setminus D) = D \Delta S$. Given that D is eulerian and S is eulerian, it follows that $D \Delta S$ is eulerian. Hence, $\{E_1 \Delta S_1, \ldots, E_k \Delta S_k\}$ supports the eulerian subgraph $D \Delta S$.

(b) Let $e \in S$. There are four cases to consider: (i) $e \notin E_1$ and $e \notin E_2$. In this case, e is added to both E_1 and E_2, thus μ_e is incremented by two, and so the parity remains the same. (ii) $e \in E_1$ and $e \notin E_2$. In this case e will be removed from E_1 but added to E_2, so μ_e will not change. (iii) $e \notin E_1$ and $e \in E_2$. In this case e will be removed from E_2 but added to E_1, so μ_e will not change. (iv) $e \in E_1$ and $e \in E_2$. In this case e will be removed from both E_1 and E_2, so μ_e will decrease by two, and its parity will not change. ∎

Lemma 2.6. Let edge sets $\{E_1, \ldots, E_k\}$ support a cut in G, and assume G is planar.

(a) Let $S \subseteq E(G)$ define a cut $\delta(A)$ for some $A \subsetneq V$, and assume $S = S_1 \cup S_2 \cup \ldots \cup S_k$, where the sets $S_1, \ldots, S_k$ are pairwise disjoint. Then, $\{E_1 \Delta S_1, \ldots, E_k \Delta S_k\}$ supports a cut in G, though not necessarily the same cut supported by $\{E_1, \ldots, E_k\}$.

(b) Consider any edge set $S \subseteq E(G)$, then $\{E_1 \Delta S, E_2 \Delta S, E_3, \ldots, E_k\}$ supports the same cut as $\{E_1, E_2, \ldots, E_k\}$ in G.

Proof.

Follows directly from Lemma 2.5 and planarity of G. ∎

Lemma 2.7. Consider a fractional solution x^* such that G^* is planar, and an h-parity constraint $(\mathcal{F}, \Phi, \mathcal{T})$ such that $\hat{T} = \{T_1, T\} \in \mathcal{T}$ is a degenerate one-domino. It is possible to replace $\hat{T}$ with a non-degenerate tooth and obtain another h-parity constraint with less than or equal slack than that of $(\mathcal{F}, \Phi, \mathcal{T})$.

Proof. Consider any edge $e = \{u, v\} \in E$ and define a non-degenerate one-domino $\hat{T}' = \{\{u\}, \{u, v\}\}$. Observe that $w(\hat{T}') = 1 - x^*_e$. Assume that $\Phi(\hat{T}, 1) = i$ and let $\{F_i, E(T_1 : T \setminus T_1), E_1, \ldots, E_k\}$ be the support set of the i-th handle, where sets $E_1, \ldots, E_k$ correspond to the edge sets derived from other multi-dominoes. From part (b) of Lemma 2.6, we know that $\{F_i, E(T_1 : T \setminus T_1), E_1, \ldots, E_k\}$ and $\{F_i \Delta \{e\}, E(T_1 : T \setminus T_1) \Delta \{e\}, E_1, \ldots, E_k\}$ support the same handle. Further, observe that $\hat{T}$ degenerate implies that $E(T_1 : T \setminus T_1) = \emptyset$ and so $E(T_1 : T \setminus T_1) \Delta \{e\} = \{e\} = E(T'_1, T')$. Let $F'_i = F_i \Delta \{e\}$. Observe that,

$$\begin{aligned} x^*(F_i) + w(\hat{T}) &= x^*(F_i) + 2x(\delta(T)) - 3 \\ &\geq x^*(F_i) + 1 = x^*(F_i) + x^*_e - x^*_e + 1 \\ &\geq x^*(F'_i) + 1 - x^*_e = x^*(F'_i) + w(\hat{T}'). \end{aligned}$$

Let $\mathcal{F}' = \mathcal{F} \setminus \{F_i\} \cup \{F'_i\}$ and $\mathcal{T}' = \mathcal{T} \setminus \{\hat{T}\} \cup \{\hat{T}'\}$. Further, define Φ' equal to Φ but for the fact that $\Phi'(\hat{T}', 1) = i$. From Lemma 2.3 we know that $(\mathcal{F}', \Phi', \mathcal{T}')$ defines a valid h-parity inequality, and from Lemma 2.2 it follows that the slack of this new inequality is less than or equal that of $(\mathcal{F}, \Phi, \mathcal{T})$. Thus we conclude our result. ∎

Lemma 2.8. Consider a fractional solution x^* such that G^* is planar, and an h-parity constraint $(\mathcal{F}, \Phi, \mathcal{T})$ with a non-degenerate tooth $\hat{T} = \{T_1, T\} \in \mathcal{T}$. Assume that both $\delta(T)$ and $\delta(T_1)$ are not minimal cuts. It is possible to replace $\hat{T}$ with a non-degenerate tooth $\hat{T}' = \{T_1', T'\}$ such that (a) $\delta(T')$ and $\delta(T_1')$ are minimal cuts, (b) $\delta(T') \subseteq \delta(T)$ and $\delta(T_1') \subseteq \delta(T_1)$, and (c) $\delta(T_1') \cup \delta(T') \subsetneq \delta(T_1) \cup \delta(T)$. Further, the new h-parity constraint has less than or equal slack than that of $(\mathcal{F}, \Phi, \mathcal{T})$.

Proof. Let $C = \overline{\delta(T)}$ and $D = \overline{\delta(T_1)}$. Observe that C and D can be decomposed into edge disjoint unions of simple cycles $C_1, C_2, \ldots, C_q$ and $D_1, D_2, \ldots, D_r$ respectively. Given that $\hat{T}$ non-degenerate there must exist $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, r\}$ such that $C_i \neq D_j$. Given that $T_1 \subseteq T$ we have that C and D do not cross with respect to the planar embedding. Thus there exist $\emptyset \subsetneq T_1' \subsetneq T' \subsetneq V$ such that $\delta(T') = C_i$ and $\delta(T_1') = D_i$. It follows that $\{T_1', T'\}$ defines a one-domino satisfying (a) and (b). If $\delta(T') = \delta(T)$ or $\delta(T_1') = \delta(T_1)$ it is easy to see that $\delta(T_1') \cup \delta(T') \subsetneq \delta(T_1) \cup \delta(T)$. Thus, assume $\delta(T') \subsetneq \delta(T)$ and $\delta(T_1') \subsetneq \delta(T_1)$. Observe that for $\delta(T_1') \cup \delta(T') = \delta(T_1) \cup \delta(T)$ it must follow that $\delta(T) \setminus \delta(T') \subseteq \delta(T_1')$ and $\delta(T_1) \setminus \delta(T_1') \subseteq \delta(T')$. Thus, since $\delta(T')$ and $\delta(T_1')$ are minimal cuts, this in turn must imply that $\delta(T) \setminus \delta(T') = \delta(T_1')$ and $\delta(T_1) \setminus \delta(T_1') = \delta(T')$, which means $\delta(T_1) = \delta(T_1') \cup (\delta(T_1) \setminus \delta(T_1')) = (\delta(T) \setminus \delta(T')) \cup (\delta(T_1) \setminus \delta(T_1')) = (\delta(T) \setminus \delta(T')) \cup \delta(T') = \delta(T)$. However, this means $\hat{T}$ is degenerate, which contradicts our initial hypothesis.

We now show that it is possible to construct an h-parity constraint with teeth $\mathcal{T}' = \mathcal{T} \setminus \{\hat{T}\} \cup \{\hat{T}'\}$ having less than or equal slack.

Assume that $\Phi(\hat{T}, 1) = i$ and let $\{F_i, E(T_1 : T \setminus T_1), E_1, \ldots, E_k\}$ be the support set of the i-th handle, where sets $E_1, \ldots, E_k$ correspond to the edge sets derived from other multi-dominoes. Let $D = \delta(T) \setminus \delta(T')$ and $D_1 = \delta(T_1) \setminus \delta(T_1')$. Observe that,

$$E(T_1' : T' \setminus T_1') = (E(T_1 : T \setminus T_1) \setminus D_1) \cup (\delta(T_1') \cap D).$$

In fact,

$$\begin{aligned} E(T_1' : T' \setminus T_1') = \delta(T_1') \setminus \delta(T') &= (\delta(T_1) \setminus D_1) \setminus (\delta(T) \setminus D) \\ &= (\delta(T_1) \setminus D_1 \setminus \delta(T)) \cup ((\delta(T_1) \setminus D_1) \cap D) \\ &= ((\delta(T_1) \setminus \delta(T)) \setminus D_1) \cup (\delta(T_1') \cap D). \end{aligned}$$

Next, observe that $E(T_1 : T \setminus T_1) \setminus D_1 = E(T_1 : T \setminus T_1) \setminus (D_1 \cap E(T_1 : T \setminus T_1)) = E(T_1 : T \setminus T_1)\Delta(D_1 \cap E(T_1 : T \setminus T_1))$. Thus, from part (b) of Lemma 2.6, $\{F_i, E(T_1 : T \setminus T_1), E_1, \ldots, E_k\}$ supports the same cut as $\{F_i\Delta(D_1 \cap E(T_1 : T \setminus T_1)), E(T_1 : T \setminus T_1) \setminus D_1, E_1, \ldots, E_k\}$. Also, since D is eulerian, and $D = (D \cap \delta(T_1')) \cup (D \setminus \delta(T_1'))$, it follows from part (a) of Lemma 2.6 that since $\{F_i\Delta(D_1 \cap E(T_1 : T \setminus T_1)), E(T_1 : T \setminus T_1) \setminus D_1, E_1, \ldots, E_k\}$ supports a cut, then so does $\{(F_i\Delta(D_1 \cap E(T_1 : T \setminus T_1)))\Delta(D \setminus \delta(T_1')), (E(T_1 : T \setminus T_1) \setminus D_1) \cup (\delta(T_1') \cap D), E_1, \ldots, E_k\}$. Thus, defining $F_i' = (F_i\Delta(D_1 \cap E(T_1 : T \setminus T_1)))\Delta(D \setminus \delta(T_1'))$ we have that $\{F_i', E(T_1' : T' \setminus T_1'), E_1, \ldots, E_k\}$ supports a cut. Finally, note that $x^*(F_i') \leq x^*(F_i) + x^*(E(T_1 : T \setminus T_1) \cap D_1) + x^*(D \setminus \delta(T_1'))$ and that $x^*(E(T_1' : T' \setminus T_1')) \leq x^*((E(T_1 : T \setminus T_1)) \setminus D_1) + x^*(D \cap \delta(T_1'))$. Thus, $x^*(F_i') + x^*(E(T_1' : T' \setminus T_1')) \leq x^*(F_i) + x^*(E(T_1 : T \setminus T_1)) + x^*(D)$. Hence,

$$\begin{aligned} x^*(F_i') + w(\hat{T}') &= x^*(F_i') + x^*(\delta(T')) + x^*(E(T_i' : T' \setminus T_i')) - 3 \\ &\leq x^*(F_i) + x^*(E(T_1 : T \setminus T_1)) + x^*(\delta(T')) + x^*(D) - 3 \\ &\leq x^*(F_i) + x^*(E(T_1 : T \setminus T_1)) + x^*(\delta(T)) - 3 \\ &\leq x^*(F_i) + w(\hat{T}). \end{aligned}$$

Let $\mathcal{F}' = \mathcal{F} \setminus \{F_i\} \cup \{F_i'\}$ and $\mathcal{T}' = \mathcal{T} \setminus \{\hat{T}\} \cup \{\hat{T}'\}$. Further, define Φ' equal to Φ but for the fact that $\Phi'(\hat{T}', 1) = i$. From Lemma 2.3 we know that $(\mathcal{F}', \Phi', \mathcal{T}')$ defines a valid h-parity inequality, and from Lemma 2.2 it follows that the slack of this new inequality is less than or equal that of $(\mathcal{F}, \Phi, \mathcal{T})$. ∎

Lemma 2.9. Consider a fractional solution x^* such that G^* is planar. There exists a maximally violated h-parity constraint $(\mathcal{F}, \Phi, \mathcal{T})$ such that every one-domino $\hat{T} = \{T_1, T\} \in \mathcal{T}$ satisfies:

(Q1) $\hat{T}$ is non-degenerate,

(Q2) $\delta(T)$ and $\delta(T_1)$ define minimal cuts, and

(Q3) $\delta(T \setminus T_1)$ defines a minimal cut.

Proof. From Lemmas 2.7 and 2.8 it is easy to see that there exists a maximally violated h-parity constraint $(\mathcal{F}, \Phi, \mathcal{T})$ such that (Q1) and (Q2) hold for every one-domino. Assume that $\hat{L} = \{L_1, L\} \in \mathcal{T}$ is a one-domino not satisfying (Q3). Let $\hat{T} = \{T_1, T\}$ be a one-domino where $T_1 = L \setminus L_1$ and $T = L$. It is easy to see that tooth $\hat{L}$ can be substituted by $\hat{T}$ to obtain an identical constraint. Further, observe that $\delta(L) \cup \delta(L_1) = \delta(T) \cup \delta(T_1)$. Since $\hat{T}$ does not satisfy (Q2) we can apply Lemma 2.8 to substitute $\hat{T}$ by yet another one-domino $\hat{T}' = \{T_1', T'\}$ satisfying (Q1) and (Q2), and such that $\delta(T') \cup \delta(T_1') \subsetneq \delta(T) \cup \delta(T_1)$. By repeatedly applying this procedure a finite number of times (because we can only remove so many edges from the original set $\delta(L) \cup \delta(L_1)$) we will eventually obtain a tooth satisfying (Q1)-(Q3). We can apply this procedure to all one-dominoes not satisfying (Q1)-(Q3) to eventually obtain another maximally violated h-parity constraint satisfying the required conditions. ∎

This allows us to use the following result:

Theorem 2.2 (Letchford [73])**.** Let x^* be a fractional solution satisfying all of the subtour elimination constraints. Let $\hat{T} = \{T_1, T\}$ be a one-domino satisfying (Q1), (Q2), and (Q3). If G^* is a planar graph, then there exist two vertices $s, t \in \bar{G}^*$ such that each of the following edge sets is an (s, t) path in $\bar{G}^*$:

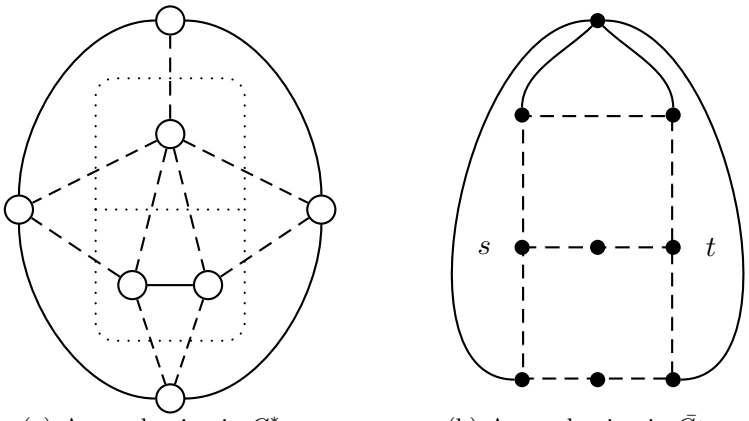
(i) $\overline{E^*(T_1 : T \setminus T_1)}$,

(ii) $\overline{E^*(T_1 : V \setminus T_1)}$,

(iii) $\overline{E^*(T \setminus T_1 : V \setminus T)}$.

Furthermore, these three (s, t) paths are edge disjoint and also have no vertices in common other than s and t.

An example of this result is depicted in Figure 2.3. In illustration (a) a one-domino is drawn in graph G^*. In this picture, the dotted lines represent the boundary of the domino and of its half, the dashed lines represent the edges in $\delta(T) \cup E(T_1 : T \setminus T_1)$, and the solid lines represent other edges in the graph. In illustration (b) the edges defining the same domino are depicted in graph $\bar{G}^*$. Again, the dashed lines correspond to the edges in $\delta(T) \cup E(T_1 : T \setminus T_1)$, and the solid lines correspond to the remaining edges. As can be seen in the latter illustration, the dashed lines define three edge disjoint paths which join vertices s and t, and which do not have any other vertices in common.



(a) A one-domino in G^* (b) A one-domino in $\bar{G}^*$

Figure 2.3: Representation of a domino in G^* and $\bar{G}^*$.

2.5 Separating GDP inequalities in planar graphs

Let $\mathcal{D}(h, l, r)$ represent the set of all generalized domino-parity inequalities $(\mathcal{H}, \Phi, \mathcal{T})$ such that (a) the number of handles is not greater than h, e.g. $|\mathcal{H}| \leq h$, (b) the number of large teeth is not greater than l, e.g. $|\hat{T} \in \mathcal{T} : \kappa(\hat{T}) \geq 2| \leq l$, and (c) no tooth has more than r halves, e.g. $\kappa(\hat{T}) \leq r$ for all $\hat{T} \in \mathcal{T}$.

In this section we present a polynomial-time algorithm which, for fixed integers h, l, r , finds a maximally violated constraint in $\mathcal{D}(h, l, r)$ in polynomial time, provided G^* is planar.

The algorithm proceeds in two steps. First, a set of candidate teeth is generated. Second, an enumeration scheme tests different associations Φ with the candidate teeth in order to identify a maximally violated inequality.

2.5.1 Step 1: Generating a set of candidate teeth

We begin with a simple result which establishes a bound on the weight of teeth participating in violated inequalities.

Lemma 2.10. Let x^* be a fractional solution, and let $\hat{T}$ correspond to a k-domino participating in a violated h-parity constraint. Then,

$$x^*(\delta(T)) + \sum_{j=1}^{k} x^*(E(T_j : T \setminus T_j)) < h + k + 2. \tag{2.14}$$

Proof. We know that

$$w(\hat{T}) = \beta(\hat{T})(x^*(\delta(T)) - 2) + \sum_{i=1}^{k} x^*(E(T_i : T \setminus T_i)) - k.$$

From equation (2.13) we know that $w(\hat{T}) < h$. In addition, we know $\beta(\hat{T}) \geq 1$. Thus,

$$(x^*(\delta(T)) - 2) + \sum_{i=1}^{k} x^*(E(T_i : T \setminus T_i)) - k \leq w(\hat{T}) < h.$$

The result immediately follows. ∎

Consider a fractional solution x^* and non-negative integers h, l, r. We say that a set of multi-dominoes $\mathcal{L}^*$ is *complete* for $\mathcal{D}(h, l, r)$ if every constraint $(\mathcal{H}, \Phi, \mathcal{T}) \in \mathcal{D}(h, l, r)$ which is violated by x^* satisfies $\mathcal{T} \subseteq \mathcal{L}^*$.

The importance of Lemma 2.10 is that it allows us to construct a complete set of teeth, as indicated by the following proposition.

Proposition 2.3. Consider x^* satisfying all of the subtour elimination constraints and non-negative integers h, l, r. It is possible to construct a complete set of multi-dominoes for $\mathcal{D}(h, l, r)$ in $O(n^{r^2+(h+3)(r+1)+1})$.

Proof. Consider a k-domino $\hat{T}$ (with $k \leq r$) participating in a violated h-parity constraint. Lemma 2.10 implies $x^*(\delta(T)) < h + k + 2$. In addition, because $\delta(T_j) \subseteq T \cup E(T_j : T \setminus T_j)$ for each $j = 1, \ldots, k$, it also implies that $x^*(\delta(T_j)) < h + k + 2$. Thus, it can be seen that the building blocks the multi-dominoes required are the sets $A \subsetneq V(G^*)$ such that $x^*(\delta(A)) < h + r + 2$.

Let $\mathcal{A} = \{A \subsetneq V : x^*(\delta(A)) < h + r + 2\}$. Since (due to the sub-tour elimination constraints) all sets $A \subsetneq V(G^*)$ satisfy $x^*(\delta(A)) \geq 2$, we know that they are all within a factor of $\alpha = (h + r + 2)/2$ of the min-cut. From Karger [71] it follows that $|\mathcal{A}| \leq (2n)^{h+r+2}$. Using the algorithm of Nagamochi [88] it is possible to completely enumerate $\mathcal{A}$ in $O(m^2 n + n^{h+r+2} m)$, where m is the number of edges in G. Since in planar graphs $m = O(n)$, we have that the entire enumeration of $\mathcal{A}$ can be performed in $O(n^{h+r+3})$.

Given the set $\mathcal{A}$ it is now possible to build $\mathcal{L}^*$. Start by enumerating k from 1 to r. Next, enumerate all possible ground sets $T \in \mathcal{A}$. Third, enumerate all possible subsets $\{T_1, \ldots, T_k\} \subseteq \mathcal{A}$. If $\hat{T} = \{T_1, \ldots, T_k, T\}$ defines a k-domino, and in addition, its weight satisfies the bound prescribed by Lemma 2.10 store it in $\mathcal{L}^*$. Otherwise, discard and keep iterating.

Let $f(n, k)$ be the time required to test if $\{T_1, \ldots, T_k, T\}$ defines a k-domino. For each $k = 1, \ldots, h$ this second part of the algorithm will require $|\mathcal{A}|\binom{|\mathcal{A}|}{k} f(n, k)$ iterations. Since $|\mathcal{A}| = O(n^{h+r+2})$, this is equal to $O(n^{(h+r+2)(k+1)} f(n, k))$. Thus, after enumerating over all k the running time will be $O(r n^{(h+r+2)(r+1)} f(n, r))$.

Finally, note that $f(n, k) \leq O(n 2^k)$, since to test if $\{T_1, \ldots, T_k, T\}$ defines a k-domino, it suffices to (a) check if $T_1, \ldots, T_k \subseteq T$, taking $O(n)$ time, and (b) check all possible subsets of $\{T_1, \ldots, T_k\}$ to see if they define the appropriate cut, taking $O(n 2^k)$.

Thus we conclude that the total running time required to generate $\mathcal{L}^*$ is bounded by $n^{h+r+3} + r 2^r n^{(h+r+2)(r+1)+1}$, which is $O(n^{r^2+(h+3)(r+1)+1})$.

Note that we can discard from $\mathcal{L}^*$ any one-dominoes which do not satisfy (Q1)-(Q3) as in Lemma 2.9, and that this does not affect the overall running time complexity of the procedure. ∎

2.5.2 Step 2: Putting it all together

Before actually getting to the main separation result, it is necessary to establish some basic graph theoretic results.

Proposition 2.4. Consider a planar graph G. Let S_1, S_2 be two eulerian subgraphs of G such that $S_1 \cup S_2 = E(G)$. Further, assume that G does not contain more than $h-1$ edge disjoint cycles. Then, the number of nodes having odd degree in the subgraph induced by $S_1 \setminus S_2$ is bounded by $8h-4$.

Proof. Let $F(G)$ represent the faces of graph G. Define two faces of G as being *adjacent* if their respective frontiers share a common edge. First, observe that $|F(G)| < 4h$. In fact, if $|F(G)| \geq 4h$, from the four-color theorem (See Appel and Haken [6], Robertson, Sanders, Seymour and Thomas [100]) it would follow that there exists a set of h non-adjacent faces. This in turn would imply that there exist h edge disjoint cycles in $E(G)$ (obtain them taking the frontier of the faces), which is contradictory with our problem hypotheses.

From Euler's Formula we know that if $K(G)$ is the set of connected components in G, then $|E(G)| = |V(G)| + |F(G)| - 1 - |K(G)|$. Thus, since $|K(G)| \geq 1$, it follows that $|E(G)| < |V(G)| + 4h - 2$. For each $v \in V(G)$ let $d(v)$ represent the degree of v in G. Observe that because $E(G)$ is the union of two eulerian subgraphs, $d(v) \geq 2$ for all $v \in V(G)$. Define $\delta_v = 1$ if $d(v) \geq 3$, and $\delta_v = 0$ otherwise. It is clear that $2 + \delta_v \leq d(v)$ for all $v \in V(G)$. Since $\sum_{v \in V(G)} d(v) = 2|E(G)|$ it follows that

$$2|V(G)| + \sum_{v \in V(G)} \delta_v \leq \sum_{v \in V(G)} d(v) = 2|E(G)| < 2|V(G)| + 8h - 4.$$

Thus, $\sum_{v \in V(G)} \delta_v < 8h-4$. However, every node $v \in V(G)$ having odd degree is such that $\delta_v = 1$. Thus, the number of nodes in G having odd degree is less than $8h-4$.

Finally, consider the subgraph induced by $S_1 \setminus S_2$. Because the number of nodes having an odd degree in this graph is equal to the number of nodes having an odd degree in G, the result follows. ∎

Consider a graph $G = (V, E)$, and a set $T \subseteq V$. We say that $J \subseteq E$ is a T-Join if T is equal to the set of vertices of odd degree in the graph (V, J). For a thorough background on T-Joins, see Schrijver [102].

Proposition 2.5. Consider a graph $G = (V, E)$. Assume that each edge $e \in E$ has non-negative weight x_e, and that $E = R \cup B$, where R and B are disjoint. Say that every edge in R is red, and every edge in B is blue. Consider a set $T \subseteq V$ such that $|T|$ is even. It is possible to find a minimum weight T-Join having an odd (or even) number of red edges in $O(2^{|T|} + |T|^2|V|^2 + |V|^3)$.

Proof. The more specific problem of finding a minimum-weight T-Join was shown to be strongly polynomial by Edmonds [44]. However, this same problem subject to the parity constraint has proven elusive, and it remains unknown whether it can be solved in polynomial time for general graphs. What we now proceed to solve is a more constrained version of this problem in which we allow the exponent of the polynomial to be a function of $|T|$.

Say that $F \subseteq E$ is *odd* if $|F \cap R|$ is odd; otherwise, say that F is *even*. Observe that if J is a T-Join in G, then $J = C \cup P_1 \cup P_2 \cup \ldots \cup P_k$, where each set P_i is a path with end-points in T, and set C is (a possibly empty) eulerian subgraph. Further, observe that there exists a minimum-weight odd (and even) T-Join such that: the paths P_i are pairwise edge disjoint, the set C is either empty, or an odd simple cycle, and that if P_i is odd (or even), then P_i is a minimum-weight odd (or even) path connecting its end-points.

Let C^1 be a minimum-weight odd cycle in in G'. Observe that such a cycle can be computed in $O(|V|^3)$ (see Gerards and Schrijver [52]). For each pair of distinct nodes $s, t \in T$ define P^0_{st} as a minimum-weight even path joining s and t. Likewise, define P^1_{st} as a minimum-weight odd path joining s and t. Observe that given $s, t \in T$ finding P^0_{st} and P^1_{st} can be achieved in $O(|V|^2)$ by solving a shortest path problem in an appropriate graph (see Gerards and Schrijver [52]). Computing all of the paths can thus be achieved in $O(|T|^2|V|^2)$. Define a graph G' with node set T. For each pair of distinct nodes $s, t \in T$ define an even edge in G' having end-points s, t and weight $w(P^0_{st})$, and define an odd edge in G' having end-points s, t and weight $w(P^1_{st})$. The minimum weight odd (or even) perfect-matching problem in G' can be solved by

enumeration in $O(2^{|T|})$. Let $M^o \subseteq E(G')$ be the optimal even solution of this sub-problem, and let M^1 be the optimal odd solution of this sub-problem. By unraveling the edges in M^o and M^1 to their corresponding paths, and iteratively removing pairs of repeated edges until each edge appears at most once in the resulting subgraphs, it is not difficult to see that we obtain T-Joins J^o and J^1 of even and odd parity. It is not difficult to see that a minimum weight even T-Join is given either by J^o or $J^1 \Delta C^1$, and that a minimum weight odd T-Join is given either by J^1 or $J^o \Delta C^1$. ∎

Proposition 2.6. Consider a graph $G = (V, E)$. Assume that the edge set E is partitioned into three subsets R, Y, B where each edge in R is labeled red, each edge in Y is labeled yellow, and each edge in B is labeled blue. In addition, assume that each edge $e \in E$ has a non-negative weight w_e. We say that a set of edges $D \subseteq E$ defines an RYB subgraph if D is eulerian and $Y \subseteq D$. Further, if $|D \cap R|$ is odd, we say that D is odd, otherwise, we say that D is even. It is possible to find a minimum-weight odd (or even) RYB subgraph in $O(2^{|T|} + |T|^2|V|^2 + |V|^3)$, where T is the set of nodes having an odd degree in the subgraph induced by Y.

Proof. Let $T = \{v \in V : v$ is the end-point of an odd number of edges $e \in Y\}$. Observe that for any eulerian subgraph $D \subseteq E$ such that $Y \subseteq D$ we will have that $D \setminus Y$ is a T-join. Thus, our problem reduces to finding a minimum weight T-Join $J \subseteq E \setminus Y$ such that $|J \cap R|$ is odd (even). Thus the result follows from Proposition 2.5. ∎

Consider non-negative integers h, l, r and a complete set of multi-dominoes $\mathcal{L}^*$ for $\mathcal{D}(h, l, r)$. Consider a collection of large teeth $\mathcal{T}^+ \subseteq \mathcal{L}^*$ and an h-tooth association Φ^+ defined over $\mathcal{T}^+$. We say that $(\mathcal{T}, \Phi, \mathcal{R})$ defines an *extension* of $\mathcal{T}^+$ and Φ^+ if the following conditions are met:

- $\mathcal{R} = \{R_1, \ldots, R_h\}$, where $R_i \subseteq E^*$, for $i = 1, \ldots, h$.
- $\mathcal{T} \subseteq \mathcal{L}^*$ and $\mathcal{T} \setminus \mathcal{T}^+$ is a collection of one-dominoes.
- Φ is an h-tooth association over $\mathcal{T}$, and Φ restricted to $\mathcal{T}^+$ coincides with Φ^+.

- $|\Phi^{-1}(i)|$ is odd, for $i = 1, \ldots, h$.
- $\{E(T_j : T \setminus T_j) : \Phi(\hat{T}, j) = i\}$ and $\{R_i\}$ supports a cut in G^*, for all $i = 1, \ldots, h$.

Proposition 2.7. Consider non-negative integers h, l, r and a fractional solution x^* satisfying all of the subtour elimination constraints. Assume G^* is planar and let $\mathcal{L}^*$ be a complete set of multi-dominoes for $\mathcal{D}(h, l, r)$. Let $\mathcal{T}^+ \subseteq \mathcal{L}^*$ be a collection of large multi-dominoes such that $|\mathcal{T}^+| \leq l$, and let Φ^+ be an h-tooth association defined over $\mathcal{T}^+$. It is possible to identify an extension $(\mathcal{T}, \Phi, \mathcal{R})$ of $\mathcal{T}^+, \Phi^+$ minimizing $s^* = \sum_{i=1}^{h} x^*(R_i) + \sum_{\hat{T} \in \mathcal{T}} w(\hat{T}) - h$ in $O(n^3)$, when h, l, r are treated as constants.

Proof. Since G^* is a planar graph, it suffices to construct $\Phi, \mathcal{T}$ and $\mathcal{R}$ satisfying the conditions of Lemma 2.4. That is we must satisfy,

(i) $\mathcal{T}^+ \subseteq \mathcal{T}$ and every $\hat{T} \in \mathcal{T} \setminus \mathcal{T}^+$ is a one-domino.

(ii) Φ restricted to $\mathcal{T}^+$ is equal to Φ^+, and $|\Phi^{-1}(i)|$ is odd, for all $i = 1, \ldots, h$.

(iii) $\{\overline{E^*(T_j : T \setminus T_j)} : \Phi(\hat{T}, j) = i\}$ and $\{\bar{R}_i\}$ support an Eulerian subgraph in $\bar{G}^*$ for all $i = 1, \ldots, h$.

(iv) $s^* = \sum_{i=1}^{h} x^*(\bar{R}_i) + \sum_{\hat{T} \in \mathcal{T}} w(\hat{T}) - h$ is of minimum value.

Observe that this can be broken down into h independent problems. For each $i \in 1, \ldots, h$ determine a collection of one-dominoes $\mathcal{T}^i \subseteq \mathcal{L}^*$ and a set $\bar{R}_i \subseteq \bar{E}$ such that,

(a) $|\mathcal{T}^i| + |\{(\hat{T}, j) : \hat{T} \in \mathcal{T}^+ \text{ and } \Phi^+(\hat{T}, j) = i\}|$ is odd.

(b) $\{\overline{E^*(T_j : T \setminus T_j)} : \hat{T} \in \mathcal{T}^+ \text{ and } \Phi^+(\hat{T}, j) = i\}$ and $\{E^*(T_1 : T \setminus T_1) : \hat{T} \in \mathcal{T}^i\}$ and $\{\bar{R}_i\}$ support an eulerian subgraph in $\bar{G}^*$.

(c) $s_i^* = x^*(\bar{R}_i) + \sum_{\hat{T} \in \mathcal{T}^i} w(\hat{T})$ is of minimum value.

In fact, if we solve each of these h problems, it is just a matter of defining $\mathcal{T} = \mathcal{T}^+ \cup \mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^h$, and $\Phi(\hat{T}, j) = \Phi^+(\hat{T}, j)$ if $\hat{T} \in \mathcal{T}^+$ and $\Phi(\hat{T}, 1) = i$ for each $\hat{T}$ in $\mathcal{T}^i$. If we do so, it is easy to see that $s^* = s_1^* + \ldots s_h^* - h + \sum_{\hat{T} \in \mathcal{T}^+} w(\hat{T})$ and that (i)-(iv) will be satisfied.

Consider any tooth $\hat{T} \in \mathcal{T}^+$ and $j \in 1, \ldots, \kappa(\hat{T})$. Let $S_1 = \overline{\delta(T_j)}$ and $S_2 = \overline{\delta(T)}$. Observe that S_1 and S_2 define eulerian subgraphs in $\bar{G}^*$. Further, since $S_1 \cup S_2 =$

$\overline{\delta(T)} \cup \overline{E(T_j : T \setminus T_j)}$, and since $x^*(\delta(T)) + x^*(E(T_j : T \setminus T_j)) < h + r + 2$ (see Lemma 2.10), it follows that $S_1 \cup S_2$ cannot contain more than $(h + r + 2)/2$ edge disjoint cycles (due to the subtour elimination constraints). Thus, from Proposition 2.4 we conclude that the number of nodes having odd degree in $\overline{E(T_j : T \setminus T_j)}$ is less than or equal to $4(h + r + 1)$.

Now consider the multigraph G' obtained from node set $V(\bar{G}^*)$ and edges obtained from the union (allowing parallel edges) of the sets $\overline{E(T_j : T \setminus T_j)}$ such that $\hat{T} \in \mathcal{T}^+$ and $\Phi(\hat{T}, j) = i$. Let $O(G')$ represent all of the nodes having odd degree in G'. Observe that $|O(G')| \leq 4lr(h + r + 1)$. In fact, we know there are at most l teeth in $\mathcal{T}^+$, and we know that each tooth $\hat{T} \in \mathcal{T}^+$ is such that $\kappa(\hat{T}) \leq r$. Thus, there can be at most lr pairs $(\hat{T}, j)$ such that $\Phi(\hat{T}, j) = i$ with $\hat{T} \in \mathcal{T}^+$ and $j \in 1, \ldots, \kappa(\hat{T})$. The bound follows from the fact that each set $\overline{E(T_j : T \setminus T_j)}$ can contribute at most $4(h + r + 1)$ odd-degree nodes to G'.

Iteratively remove from G' pairs of parallel edges until no more such pairs remain. Let Y_i be the edge set remaining (e.g., Y_i will be the set of edges appearing in an odd number of sets $\overline{E^*(T_j : T \setminus T_j)}$ with $\hat{T} \in \mathcal{T}^+$ and $\Phi^+(\hat{T}, j) = i$). Observe that each time we remove a pair of parallel edges the number of odd-degree nodes in G' does not change. Thus, the number of odd-degree nodes in the subgraph induced by Y is not more than $4lr(h + r + 1)$, or simply $O(lrh + lr^2)$.

Define another multi-graph, named M_i, having node set $V(\bar{G}^*)$. Add edge set Y_i to G'', label each of these edges as *yellow*, and assign to each a weight of zero. For each edge $e \in \bar{E}$ add an edge e' to M_i having the same end-points (call this set of edges B). Label each of these edges *blue*, and assign to them a weight equal to x^*_e. Finally, for each each one-domino in $\mathcal{L}^*$, identify the end-nodes s and t corresponding to its domino-paths, and add an s-t edge labeled *red* to M_i with weight equal that of the domino (call this edge set R).

Consider a set of one-dominoes $\mathcal{T}^i \subseteq \mathcal{L}^*$ and a set of edges $\bar{R}_i \subseteq \bar{E}$ satisfying (a)-(b). Let R' be the set of red edges in M_i corresponding to the one-dominoes in $\mathcal{T}^i$, and

let B' be the set of blue edges in M_i corresponding to edges in $\bar{R}_i$. Observe that $D = B' \cup Y \cup R'$ defines an RYB subgraph of M_i, as defined in Proposition 2.6, and that the weight of D equals exactly $x^*(\bar{R}_i) + \sum_{\hat{T} \in \mathcal{T}^i} w(\hat{T})$.

Likewise, consider an RYB subgraph $D \subseteq E(M_i)$. If $|\{\hat{T} \in \mathcal{T}^+ : \Phi(\hat{T}, j) = i, \text{ for some } j \in 1, \ldots, \kappa(\hat{T})\}|$ is even, assume D is odd. Otherwise, assume D is even. Let $\mathcal{T}^i \subseteq \mathcal{L}^*$ be the one-dominoes associated to red edges in D, and let $\bar{R}_i \subseteq \bar{E}$ correspond to the blue edges in D. Observe that $\mathcal{T}^i$ and $\bar{R}_i$ satisfy the conditions (a)-(b). Further, the weight of D is equal to exactly, $x^*(\bar{R}_i) + \sum_{\hat{T} \in \mathcal{T}^i} w(\hat{T})$.

Thus, we can see that the problem of identifying the sets $\mathcal{T}^i$ and $\bar{R}_i$ such that s_i^* is minimized, reduces to the problem of finding a minimum weight RYB graph (of the appropriate parity) in M_i.

The running time of this procedure will be determined by the amount of time required to find h minimum-weight RYB subgraphs (one for each graph M_i) of the appropriate parity. From Proposition 2.6, this will equal $O(h2^{lrh+lr^2} + h(lrh + lr^2)^2|\bar{V}|^2 + |V|^3)$.

∎

We are now ready for our main result, which we prove by means of an algorithm.

Theorem 2.3. Consider x^* such that all of the subtour elimination constraints are satisfied, and such that the support graph G^* is planar. It is possible to identify a maximally violated constraint in $\mathcal{D}(h, l, r)$ in $O(n^{lr^2+l(h+3)(r+1)+l+3})$.

Proof. The algorithm works by iterating over three loops: First, we enumerate over all collections $\mathcal{T}^+ \subseteq \mathcal{L}^*$ such that $|\mathcal{T}^+| = l$. Observe that there are $\binom{|\mathcal{L}^*|}{l}$ such sets, that is $O(n^{l(r^2+(h+3)(r+1)+1)})$. Second, for each of these sets $\mathcal{T}^+$ we enumerate over all possible h-tooth associations Φ^+ defined on $\mathcal{T}^+$. Given $\mathcal{T}^+$, observe that there are at most h^r such associations. Third, for each of these pairs $\mathcal{T}^+, \Phi^+$ we identify an h-parity extension $(\mathcal{T}, \Phi, \mathcal{R})$ of $\mathcal{T}^+, \Phi^+$ minimizing the quantity $s^* = \sum_{i=1}^{h} x^*(R_i) + \sum_{\hat{T} \in \mathcal{T}} w(\hat{T}) - h$. This last step requires $O(n^3)$. As we have seen in Lemma 2.3, each of these h-parity extensions can be used to obtain an h-parity constraint

with slack s^*. Thus, among all of the extensions generated, we keep the one with smallest value s^*. Note that this h-parity constraint will be optimal (e.g., most violated). In fact, every optimal h-parity constraint must be an extension of some pair $\mathcal{T}^+, \Phi^+$, thus every possible h-parity constraint will be indirectly considered by the algorithm. Putting everything together we get an algorithm that runs in $O(n^{l(r^2+(h+3)(r+1)+1)} \cdot h^r \cdot n^3) = O(n^{lr^2+l(h+3)(r+1)+l+3})$. ∎

Corollary 2.1. Consider x^* such that all of the subtour elimination constraints are satisfied, and such that the support graph G^* is planar. It is possible to separate a super-class of clique-tree inequalities having h handles in $o(n^{h^4})$.

Proof. Observe that every clique-tree on h handles has at most $h-1$ large teeth. Further, each large tooth can intersect at most h handles. Thus, considering $l = r = h$ in Theorem 2.3 we get the desired result. ∎

2.6 Final remarks

In this chapter we have constructed a new, very general class of valid inequalities for the TSP which contains many other well-known valid TSP inequalities. In addition, we have shown that assuming planarity of the support graph, and fixing the number of handles and large teeth, it is possible to separate a super-class of bipartition and clique-tree inequalities in polynomial time. However, the algorithm we have presented is very slow for practical separation purposes. In this section we begin by discussing some algorithmic speed-ups. Then, we remark on the relationship between this result and that proposed by Carr [30] — who proposed a similar algorithm for non-planar graphs.

By carefully observing the proof of Theorem 2.3, it is possible to see that the bottleneck of the separation algorithm lies in enumerating all of the candidate sets of large teeth. In order to improve upon this key step it is necessary to go back to Proposition 2.3 and reduce the size of the list of candidate teeth $\mathcal{L}^*$. It seems likely that every k-domino $\hat{T} = \{T_1, \ldots, T_k, T\}$ participating in a violated h-parity constraint satisfies $x(\delta(T_j)) < 4$. The reasoning for this is that halves of a tooth in a violated inequality should be

connected, and so should their complementation. Proving this likely requires using an inductive condition whereby it is assumed that before separating h-parity constraints all r-parity constraints with $r < h$ have been solved. The benefit of proving this would lie in that instead of enumerating all possible subsets of $\mathcal{A}$ having size k, one could instead enumerate subsets of $\mathcal{B} = \{B \subsetneq V : x(\delta(x)) < 4\}$ — which is considerably smaller. Another possible speed-up might be to improve upon the algorithm of Nagamochi et. al. [88] by taking into account planarity and the subtour elimination constraints. Perhaps one alternative to using Nagamochi et. al. algorithm would be to design an algorithm which works by solving shortest path problems, such as the one used by Letchford [73] to enumerate one-dominoes. An interesting way of doing this might consist in generalizing (if possible) Lemma 2.9 to show that there exist maximally violated h-parity constraints where every k-domino $\hat{T} = \{T_1, \ldots, T_k, T\}$ is such that $\delta(T), \delta(T_1), \ldots, \delta(T_k), \delta(T \setminus T_1), \ldots, \delta(T \setminus T_k)$ are all minimal cuts. This might allow for a different separation approach which would work by solving shortest path problems instead of enumerating partial tooth-handle associations and then completing them.

Next, consider the following Theorem, proven by Carr [30].

Theorem 2.4 (Carr 1997)**.** Consider positive integers h, t and a fractional solution x^* satisfying all of the subtour elimination constraints. It is possible to separate the class of bipartition inequalities having h handles and t teeth in polynomial time.

It is easy to see that this result is very similar to that in Theorem 2.3. The main difference is that in non-planar graphs it seems necessary to specify the total number of teeth to be separated, whereas in the case of planar graphs it suffices to specify the total number of large teeth. An interesting observation is that in both cases it is not strictly necessary to specify a bound on the number of teeth having three or more halves. This is because of the observation following Lemma 2.2 — that the number of teeth having more than three halves is naturally bounded by the number of handles. The proof of Carr's theorem follows a scheme similar to that of Theorem 2.3. Instead of enumerating h-tooth associations, Carr enumerates what he calls "backbones" which, essentially, are the same

thing. It seems likely that Carr's proof could be easily extended to separate GDP inequalities by taking into account the observations made in this chapter, with the restriction that the number of one-dominoes need be specified in case of non-planar graphs.

CHAPTER 3

Mixed integer knapsack problems

3.1 Introduction

Consider a positive integer n. For each $k = 1, \ldots, n$ let $a_k, c_k \in \mathbb{Q}$, $l_k, \in \mathbb{Q} \cup \{-\infty\}$, and $u_k \in \mathbb{Q} \cup \{+\infty\}$. Let $b \in \mathbb{Q}$, and consider $I \subseteq \{1, \ldots, n\}$. The following problem will henceforth be referred to as the Mixed Integer Knapsack Problem (MIKP) :

$$\begin{aligned} \text{(MIKP)} \quad & \max \sum_{k=1}^{n} c_k x_k \\ & \text{s.t.,} \\ & \sum_{k=1}^{n} a_k x_k \leq b \\ & l_k \leq x_k \leq u_k, \ \forall k = 1, \ldots, n \\ & x_k \in \mathbb{Z}, \ \forall k \in I \end{aligned}$$

Throughout this chapter we will assume that for each $k = 1, \ldots, n$ either $a_k \neq 0$ or $c_k \neq 0$, else, we could remove variable x_k from consideration as it does not affect the problem.

In this chapter we present an algorithm for solving MIKP. That is, an algorithm which either (a) proves MIKP is infeasible, (b) proves MIKP is unbounded, or (c) finds an optimal solution to MIKP.

3.1.1 Background

There are many variants of the MIKP which have been studied in the literature. In these it is traditionally assumed that all objective function coefficients, all constraint coefficients, and all variables must take integer and non-negative values. In addition:

- In the Knapsack Problem (KP), $l_i = 0$ and $u_i = 1$ for all $i \in 1, \ldots, n$.
- In the Bounded Knapsack Problem (BKP), $l_i = 0$ and $u_i < +\infty$ for all $i \in 1, \ldots, n$.
- In the Unbounded Knapsack Problem (UKP), $l_i = 0$ and $u_i = +\infty$ for all $i \in 1, \ldots, n$.

Knapsack problems and their variants have a rich history in the research literature. For a complete survey of these problems, as well as an up-to-date account of the most effective solution methodologies, the books by Kellerer, Pferschy, and Pisinger [72], and Martello and Toth [80] provide an excellent source of material. In what follows we give a brief overview of the most important advances.

Most of the exact algorithms which have been proposed for KP, BKP, and UKP employ either Branch and Bound or Dynamic Programming, and are typically classified into two variants: primal-methods, which work by keeping a feasible solution at each iteration, and primal-dual methods, in which infeasibility is permitted in intermediary stages.

Most branch-and-bound algorithms follow the primal framework originally proposed by Horowitz and Sahni [67] for KP. Martello and Toth have greatly extended this approach by use of ad-hoc bounding techniques, basic pre-processing and heuristics. Their extensions include algorithms for KP [77], [78], and [81], BKP [76] and [80], as well as UKP [76] and [79]). Pisinger proposed an alternative branch-and-bound algorithm for KP [93] which worked within a primal-dual framework. This approach seemed to have similar success as the approaches of Martello and Toth [72].

Dynamic Programming algorithms have their roots in the well-known primal algorithm of Bellman [20]. However, a different primal-dual approach was presented by Pisinger [96]. Pisinger has shown that dynamic programming can be very effective methodology for tackling instances of both KP [94] and BKP [97]. Greenberg and Feldman [64], and

Andonov, Poirriez, and Rajopadhye [5] have shown the same for UKP.

It is interesting to note that the most successful algorithm for KP to date is neither a branch-and-bound algorithm nor a dynamic-programming algorithm. The COMBO algorithm of Martello, Pisinger, and Toth [75] actually uses a mix of branch and bound and dynamic programming in order to tackle difficult problems.

We now focus on some of main ideas which have been utilized in the afore-mentioned algorithms.

The Core Concept A key development in the solution of KP, BKP, and UKP was the observation that only a relatively small subset of variables typically participate in optimal solutions. This observation, formalized in 1980 by Balas and Zemel [19], led to the concept of *core*, an apriori estimate of what that small subset of variables might be. The concept of a core has evolved much in recent years, and has been applied in many different ways to improve both branch-and-bound algorithms, as well as dynamic-programming algorithms. The basic idea is that the optimization is first restricted to variables in the core, and then, gradually, the core size is increased so as to encompass the entire problem. Interestingly, Pisinger observed [95] that branch and bound algorithms are very sensitive to the selection of a core, whereas dynamic algorithms tend to have a much more robust performance and benefit more from the concept.

Reducing BKP to KP Observe that an instance of BKP can be easily reduced to an instance of KP by means of a simple transformation. In fact, every non-negative integer a can be represented uniquely as a partial sum of powers of two. As an example, $29 = 2^0+2^2+2^3+2^4$. This makes it possible to substitute each variable x_i in BKP by $\lfloor \log(u_i) \rfloor + 1$ binary variables. Again, as an example, consider x_i such that $u_i = 29$. We can substitute variable x_i by binary variables $x_i^0,\ x_i^1,\ x_i^2,\ x_i^3,\ x_i^4$, where $a_i^0 = a_i,\ a_i^1 = 2a_i,\ a_i^2 = 4a_i,\ a_i^3 = 8a_i,\ a_i^4 = (29-8)a_i = 11a_i$, and $c_i^0 = c_i,\ c_i^1 = 2c_i,\ c_i^2 = 4c_i,\ c_i^3 = 8c_i,\ c_i^4 = (29-8)c_i = 11c_i$.

Kellerer, Pferschy, and Pisinger [72] report that this transformation had the following impact on branch-and-bound methodologies for BKP:

"During the seventies many papers appeared on specialized branch-and-bound

> methods for various integer programming problems related to the knapsack family. A few of them dealt explicitly with BKP. However, we are not aware of any relevant publication on this topic since 1979. It seems that further research was more or less stopped by the observation that these branch-and-bound algorithms customized for BKP were generally outperformed by the application of branch-and-bound methods on the instance of KP derived from the transformation ..."

In fact, most of the successful BKP-specific branch-and-bound algorithms actually work by applying this transformation in different ways.

Dominance and Periodicity A key development in the solution of UKP instances has been the use of *dominance* and *periodicity*.

The earliest studies of domination and periodicity date back to the work of Gilmore and Gomory [54] in 1963, though these concepts have been extended by many different authors. The most recent extension of their work is due to Andonov, Poirriez, and Rajopadhye [5]. This extension is the most general, and will have a direct relationship to what will be pursued in this chapter.

Periodicity can be described as follows in the context of UKP: Consider variables x_i and x_j such that c_i/a_i is greater than c_j/a_j. This indicates that variable x_i is "better" in some way, as it yields a greater return in the objective function per unit of capacity used up in the knapsack constraint. Now consider integers k_i and k_j such that $a_i k_i = a_j k_j$. From the previous observations it is clear that there exists an optimal solution of UKP such that $x_j \leq k_j - 1$. In fact, every solution such that $x_j \geq k_j$ can be "improved" by removing k_j units from x_j and adding k_i to x_i. This will clearly preserve feasibility, and in addition, because of the ratio condition, this cannot yield a solution which is worse.

Consider an instance of type UKP, and $i \neq j$ in $1, \ldots, n$. Say that x_i simply dominates x_j if $a_j \geq a_i$ and $c_j \leq c_i$. What Gilmore and Gomory fundamentally observed was that if x_i simply dominates x_j, then, there exists an optimal solution in which x_j is never used; thus, x_j can be eliminated from consideration in the problem.

Martello and Toth [79] later extended this notion in the following way: Say that x_i

multiply dominates x_j if $\lfloor a_i/a_j \rfloor \geq c_i/c_j$. Again, it is easy to see that if x_i multiply dominates x_j, the same principle applies.

Finally, Andonov et. al. [5] generalized this notion as follows: Say that a set of indices $I \subseteq \{1, \ldots, n\}$ is such that the corresponding variables threshold-dominate x_j if there exists an integer multiplier α and a vector π composed of n integers such that, $\sum_{i \in I} \pi_i a_i \leq \alpha a_j$ and $\sum_{i \in I} \pi_i c_i \geq \alpha c_j$. Again, given that all the variables are unbounded, this condition implies that there it is possible to impose $x_j \leq \alpha - 1$ in the presence of the variables x_i with $i \in I$.

These ideas have been used in a wide-variety of ways. Most notably, the dynamic-programming algorithm of Andonov et. al. [5], by using these ideas, has become the most effective algorithm for tackling UKP to date.

It is interesting to note that the notions of dominance and periodicity have not been used to tackle instances of KP and BKP. In fact, in the approach proposed in this chapter exactly this will be done, in the even more general class of problems MIKP.

The Mixed Integer Knapsack Problem Curiously enough, we are not aware of any work pursuing the more general class of problems MIKP. In fact, we are not aware of any work in which knapsack problems with continuous variables are studied, nor of any work on knapsack problems having both bounded and unbounded variables.

3.1.2 About this chapter: Motivation and contribution

In this study we present a new branch-and-bound algorithm for MIKP. Solving MIKP is fundamentally different than solving KP, BKP, and UKP, given that (1) it is not clear how continuous variables should be introduced in the current solution methodologies for KP, BKP, and UKP, and (2) MIKP works simultaneously with both bounded and unbounded variables.

The methodology that we propose is a linear-programming-based algorithm which exploits dominance conditions similar to those used in algorithms for UKP, but in the context of problems with bounds. In addition to cost-domination ideas such as those used for UKP, lexicographic-domination conditions are used to eliminate problems with symmetry. One

interesting aspect of this approach is that it differs from traditional linear-programming based algorithms by allowing feasible solutions to be pruned during the branching phase. The idea is that feasible solutions will only be pruned if either (a) they are not optimal (cost-domination-criteria), or (b) if they are optimal, but somewhere else in the tree it is known that there is another optimal solution (lexicographic-domination-criteria).

As will be seen in the computational section, solving instances of MIKP can be very difficult, even for very effective mixed integer programming solvers such as CPLEX [69]. The proposed algorithm is shown to be very effective in solving instances of MIKP, much more effective than CPLEX in fact, both in the amount of time taken to solve problems as by the size of the branch and bound tree explored to find the optimal solution.

As we will see in Chapter 4, being able to solve MIKP efficiently can have important applications to better being able to generate cutting planes for single-row relaxations of general mixed integer programming problems. In addition, being able to solve instances of MIKP by linear-programming based branch-and-bound algorithms may provide important insight into how general mixed-integer-programming problems should be tackled as well.

The organization of this chapter is as follows:

In Section 3.2 an easy way of identifying unbounded solutions is presented. In Section 3.3 a way of pre-processing instances of MIKP is presented. In Section 3.4 the issue of quickly solving the LP-relaxation of MIKP is addressed. In Section 3.5 a simple branch-and-bound algorithm for MIKP is presented. In Section 3.6 this simple branch-and-bound algorithm is enhanced by introducing domination-criteria. In Section 3.7 the effectiveness of the domination-enhanced branch-and-bound scheme is analyzed computationally, and compared to that of the general mixed-integer-programming solver CPLEX.

3.2 Infeasible, unbounded, and trivial instances of MIKP

In this section we give address two issues: infeasibility and unboundedness. As we will see, detecting infeasibility and unboundedness is very easy. After describing how to detect infeasible instances, a simple characterization of unboundedness for MIKP is stated. This characterization allows us to construct a single-pass linear time algorithm for detecting if

MIKP is unbounded. In those cases that MIKP is unbounded, the algorithm provides a proof by means of ray. In addition, some of the techniques used to characterize unboundedness also allows us to detect a certain class of instances which we call "trivial". As the name suggests, these "trivial" instances are very easy to solve.

In order to detect infeasibility it is possible to do a single-pass through the variables in any order. First, observe that if there is any variable x_i with $i = 1, \ldots, n$ such that $a_i > 0$ and $l_i = -\infty$, or such that $a_i < 0$ and $u_i = -\infty$, then the problem is feasible. In fact, if such a variable exists, regardless of what values the other variables take, x_i can be set in such a way as to make the knapsack constraint be satisfied. Assume this is not the case, and attempt to construct a feasible solution x^o as follows: For each $i \in 1, \ldots, n$ such that $a_i \geq 0$ set $x_i^o = \lceil l_i \rceil$ if $i \in I$, and $x_i^o = l_i$ if not. For each $i \in 1, \ldots, n$ such that $a_i < 0$ set $x_i^o = \lfloor u_i \rfloor$ if $i \in I$ and $x_i^o = u_i$ if not. Clearly x^o satisfies all of the bound and integrality constraints. In addition, each variable x_i^o is such the quantity $a_i x_i^o$ is as small as possible. Thus, if $\sum_{i=1}^{n} a_i x_i^o \leq b$ we conclude that the problem is feasible, and otherwise, that it is infeasible.

The concept of "efficiency" which we now proceed to define, will be used throughout this chapter. Intuitively, the efficiency of a variable tells us how valuable it is relative to the amount of capacity it uses up in the knapsack constraint.

Definition 3.1. Consider $k \in \{1, \ldots, n\}$. Define,

$$e_k = \begin{cases} c_k/a_k & \text{if } a_k \neq 0 \\ +\infty & \text{if } a_k = 0 \text{ and } c_k > 0 \\ -\infty & \text{if } a_k = 0 \text{ and } c_k < 0 \end{cases}$$

We say that e_k is the *efficiency* of variable x_k.

In order to characterize unboundedness we will classify some of the unbounded variables according to the definitions below.

Definition 3.2. Consider $k \in \{1, \ldots, n\}$.

We say that x_k is a *potentiator* if,

$$(a_k \leq 0,\ c_k > 0,\ u_k = +\infty) \text{ or } (a_k \geq 0,\ c_k < 0,\ l_k = -\infty).$$

We say that x_k is an *accumulator* if,

$$(a_k < 0,\ c_k = 0,\ u_k = +\infty) \text{ or } (a_k > 0,\ c_k = 0,\ l_k = -\infty).$$

We say that x_k is an *incrementor* if,

$$(a_k > 0,\ c_k > 0,\ u_k = +\infty) \text{ or } (a_k < 0,\ c_k < 0,\ l_k = -\infty).$$

We say that x_k is a *decrementor* if,

$$(a_k > 0,\ c_k \geq 0,\ l_k = -\infty) \text{ or } (a_k < 0,\ c_k \leq 0,\ u_k = +\infty).$$

Observe that if x_k is an accumulator, then it is also a decrementor. Furthermore, consider a free variable x_k. Observe that x_k is a decrementor if and only if it is an incrementor.

Intuitively, when MIKP has a potentiator, the problem should be unbounded. In fact, consider a feasible solution x to MIKP. If $a_k \leq 0,\ c_k > 0$ and $u_k = +\infty$, then, if we arbitrarily increase x_k, the problem will remain feasible, yet the objective function value will increase. Likewise, if $c_k < 0$ we can arbitrarily decrease x_k to achieve the same effect.

Similarly, if MIKP has an incrementor x_i and a decrementor x_j such that $e_i > e_j$, the problem should be unbounded. This is because the variable x_j can be used to "cancel" out the effect of x_i in the knapsack constraint, while yet being such that the net difference in objective value is positive.

If MIKP has an accumulator, the problem should be easy to solve. This is because accumulators allow us to arbitrarily reduce the left-hand side of the knapsack constraint without incurring any cost to the objective function.

Formally, these observations are summarized in the following lemmas.

Lemma 3.1. If MIKB is feasible and admits a potentiator, then MIKP is unbounded.

Proof. Let x_i be a potentiator. If $c_i > 0$ define $\pi \in \mathbb{Q}^n$ so that $\pi_i = 1$ and $\pi_i = 0$ for $k \in \{1, \ldots, n\} \setminus \{i\}$. Otherwise, define π so that $\pi_i = -1$ and $\pi_k = 0$ for $k \in \{1, \ldots, n\} \setminus \{i\}$. It is easy to see that in either case π defines a ray and $\pi c > 0$. Hence, MIKP is unbounded. ∎

Lemma 3.2. If MIKB is feasible, and admits an incrementor x_i and a decrementor x_j such that $e_i > e_j$, then MIKP is unbounded.

Proof. Assume the conditions of the lemma, and define $\pi \in \mathbb{Q}^n$ such that,

$$\pi_k = \begin{cases} 0 & \text{if } k \in \{1, \ldots, n\} \setminus \{i, j\}, \\ a_i & \text{if } k = i \text{ and } c_i \geq 0, \\ -a_j & \text{if } k = i \text{ and } c_i < 0, \\ a_i & \text{if } k = j \text{ and } c_j \geq 0, \\ -a_j & \text{if } k = j \text{ and } c_j < 0. \end{cases}$$

Observe that π is a ray. In addition, because $e_i > e_j$ and $\pi a = 0$ it follows that $\pi c > 0$. Hence, MIKP is unbounded. ∎

Proposition 3.1. MIKP is unbounded if and only one of the following conditions hold,

- MIKP is feasible and admits a potentiator x_j.
- MIKP is feasible and admits an incrementor x_i and a decrementor x_j such that $e_i > e_j$.

Proof. We have already seen that if either condition holds, then MIKP is unbounded. Hence, we focus on the converse.

Suppose that MIKP is unbounded and that it does not admit a potentiator.

Let π be a ray such that $\pi c > 0$ and such that $r = |\{k \in 1, \ldots, n : \pi_k \neq 0\}|$ is minimized.

The intuition behind this proof is as follows: Among those indices k such that $\pi_k \neq 0$, there must exist k such that x_k is an incrementor and there must exist k such that x_k is a decrementor. Choosing x_i to be the most efficient such incrementor, and x_j the least efficient such decrementor, it is possible to prove that a necessary condition for π to be a ray is that $e_i > e_j$. Thus, the proposition would hold.

Note that if $\pi_k \neq 0$, then, $c_k > 0$ implies $a_k > 0$. Likewise, $c_k < 0$ implies $a_k < 0$. If not, we would have either that x_k is potentiator, or that setting $\pi_k = 0$ we obtain another ray with smaller value r.

If $\pi_k \neq 0$ and $c_k = 0$ then $a_k \neq 0$. Otherwise, setting $\pi_k = 0$ we can contradict minimality of r.

Let $K^+ = \{k \in 1, \ldots, n : \pi_k c_k > 0\}$ and $K^- = \{k \in 1, \ldots, n : \pi_k c_k \leq 0\}$. Observe that $k \in K^+$ implies x_k is an incrementor and $\pi_k a_k > 0$. In fact, assume $k \in K^+$. We have $\pi_k > 0$ implies $u_k = \infty$ and $c_k > 0$. Thus, $a_k > 0$, and so x_k is an incrementor. On the other hand, $\pi_k < 0$ implies $l_k = -\infty$ and $c_k < 0$. Thus $a_k < 0$ and again, x_k is an incrementor.

Observe that $k \in K^-$ implies x_k is a decrementor and $\pi_k a_k \leq 0$. In fact, assume $k \in K^-$. We have $\pi_k > 0$ implies $u_k = \infty$ and $c_k \leq 0$. If $c_k = 0$ then $\pi_k > 0$ implies $a_k < 0$. Thus x_k is a decrementor. If $c_k < 0$ this implies $a_k < 0$, and again x_k is a decrementor. On the other hand, $\pi_k < 0$ implies $l_k = -\infty$ and $c_k \geq 0$. If $c_k = 0$ then $\pi_k < 0$ implies $a_k > 0$. If $c_k > 0$ we know $a_k > 0$. Thus, in either case x_k is a decrementor.

Note that because $\pi a \leq 0$ and $\pi c > 0$ must be satisfied, both K^+ and K^- must be non-empty sets. Without loss of generality, we will now assume that $k_1 > k_2$ implies $e_{k_1} \geq e_{k_2}$.

Let $k_1 = \min\{k : k \in K^+\}$ and let $k_2 = \max\{k : k \in K^-\}$. We now prove that $e_{k_1} > e_{k_2}$.

Suppose that $e_{k_1} \leq e_{k_2}$. Because of the way in which k_1 and k_2 are defined, it follows that $e_k \leq e_{k_1}$ for all $k \in K^+$. Likewise, it follows that $e_k \geq e_{k_2}$ for all $k \in K^-$.

Assume that $c\pi > 0$, and let $K = \{k \in 1, \ldots, n : \pi_k \neq 0\}$.

We have that,

$$\begin{aligned} 0 &< \pi c = \sum_{k\in K} \pi_k c_k = \sum_{k\in K} \pi_k e_k a_k \\ &= \sum_{k\in K^+} \pi_k e_k a_k + \sum_{k\in K^-} \pi_k e_k a_k \\ &\leq e_{k_1} \sum_{k\in K^+} \pi_k a_k + e_{k_2} \sum_{k\in K^-} \pi_k a_k \\ &\leq e_{k_2} \sum_{k\in K} \pi_k a_k \\ &= e_{k_2} \pi a. \end{aligned}$$

However, because $e_{k_2} \geq 0$, it follows from above that $\pi a > 0$ and $e_{k_2} > 0$. However, this is contradictory with π being a ray.

We thus have that $e_{k_1} > e_{k_2}$. However, k_1 and k_2 now satisfy the second condition of the proposition. ∎

Note that even if MIKP is bounded, it may still admit an accumulator.

Definition 3.3. Consider an instance of MIKP which is feasible and not unbounded. If MIKP has an accumulator, we say that MIKP is *trivial*.

Proposition 3.2. Assume that MIKP is feasible and not unbounded. In addition, let j correspond to an accumulator of MIKP. For each $k \in 1, \ldots, n$ such that $k \neq j$ define:

- $U_k = \begin{cases} \lfloor u_k \rfloor & \text{if } k \in I, \\ u_k & \text{otherwise.} \end{cases}$
- $L_k = \begin{cases} \lceil l_k \rceil & \text{if } k \in I, \\ l_k & \text{otherwise.} \end{cases}$
- $x_k = \begin{cases} U_k & \text{if } (c_k > 0) \text{ or } (c_k = 0 \text{ and } u_k < \infty), \\ L_k & \text{if } (c_k < 0) \text{ or } (c_k = 0 \text{ and } l_k > -\infty), \\ 0 & \text{if } c_k = 0 \text{ and } x_k \text{ is free.} \end{cases}$

In addition, if $a_j < 0$ let

$$x_j = \max \left\{ \left\lceil -\frac{\sum_{k\neq j} a_k x_k - b}{a_j} \right\rceil, \; l_k \right\}.$$

Otherwise, if $a_j > 0$ let

$$x_j = \min \left\{ \left\lfloor -\frac{\sum\limits_{k \neq j} a_k x_k - b}{a_j} \right\rfloor, \; u_k \right\}.$$

Then, x is well-defined and corresponds to an optimal solution of MIKP.

Proof. To see that x is well-defined, note that j is also a decrementor. Because j corresponds to a variable with null efficiency, and because the problem is not unbounded, there can be no incrementors or potentiators. Thus, for $k \in \{1, \ldots, n\} \setminus \{j\}$ we have that $c_k > 0$ implies $u_k < \infty$. Likewise, $c_k < 0$ implies $l_k > -\infty$.

It is easy to see that by the way in which x_j is defined, x is feasible for MIKP.

To see that x is optimal, simply note that whenever $c^k > 0$, the variable x_k takes as large a value as the bounds permit. Likewise, when $c_k < 0$ the variable x_k is as small as the bounds permit. ∎

Algorithm 1 is a one-pass algorithm for detecting if MIKP is unbounded or trivial. In case that the instance is found to be unbounded or trivial, the algorithm returns indices corresponding to a potentiator, an accumulator, or an incrementor/decrementor pair.

3.3 Preprocessing an instance of MIKP

In this section we are concerned with reducing an instance of MIKP to another, equivalent instance of MIKP which is easier to solve. A series of procedures for pre-processing an instance of MIKP are now presented. These are presented as distinct and independent procedures which should be performed one after another in steps. However, in an efficient implementation it is possible to perform most of these in one single linear-time pass. Most of these ideas are not new, and are the same as the pre-processing techniques used for general mixed integer programming problems. For a thorough introduction to pre-preprocessing see Savelsbergh [101].

Throughout this section we adopt the following conventions: $\infty + \infty = \infty$, if $x > 0$ then $x \cdot \infty = \infty$, if $x < 0$ then $x \cdot \infty = -\infty$, $\infty \cdot \infty = \infty$, $\lfloor -\infty \rfloor = -\infty$, $\lceil \infty \rceil = \infty$, and $-\infty \cdot \infty = -\infty$.

Algorithm 1 Detecting unbounded and trivial solutions

Input: c, a, b, e, l, u

$k_o \leftarrow -1$
$k_1 \leftarrow -1$; $k_2 \leftarrow -1$
$e^+ \leftarrow -\infty$; $e^- \leftarrow +\infty$
status $\leftarrow$ problem is not unbounded and is not trivial
for $i = 1$ to n **do**
  if x_i is a potentiator **then**
    $k_o \leftarrow i$
    status $\leftarrow$ potentiator
    return
  else if x_i is an incrementor and $e_i > e^+$ **then**
    $e^+ \leftarrow e_i$
    $k_1 \leftarrow i$
  else if x_i is an decrementor and $e_i < e^-$ **then**
    $e^- \leftarrow e_i$
    $k_2 \leftarrow i$
  end if
end for
if $e^+ > e^-$ **then**
  status $\leftarrow$ incrementor/decrementor pair
  return
else if $e^- = 0$ **then**
  status $\leftarrow$ accumulator
end if
return

Step 1. Test if MIKP is infeasible, trivial or unbounded. This can be done as proposed in the previous section. If the problem is infeasible, trivial or unbounded there is no need for further pre-processing, as the problem has been solved.

Note that if MIKP is feasible, not trivial and not unbounded, then it has no potentiators and no accumulators. In addition if variable x_i is an incrementor, and x_j a decrementor, then $e_i \leq e_j$.

Step 2. Strengthen bounds. Consider a variable x_k. Define,

$$U_k = \begin{cases} +\infty & \text{if } a_k \leq 0, \\ \left(b - \sum\limits_{i \neq k : a_i > 0} a_i l_i - \sum\limits_{i \neq k : a_i < 0} a_i u_i \right) / a_k & \text{otherwise.} \end{cases}$$

$$L_k = \begin{cases} -\infty & \text{if } a_k \geq 0, \\ \left(b - \sum\limits_{i \neq k : a_i > 0} a_i u_i - \sum\limits_{i \neq k : a_i < 0} a_i l_i \right) / a_k & \text{otherwise.} \end{cases}$$

Observe that these values are well-defined. That is, there are no divisions by zero and we never evaluate the expression $\infty - \infty$.

If $k \notin I$ then re-define $u_k := \min\{u_k, U_k\}$ and $l_k := \max\{l_k, L_k\}$. Otherwise, if $k \in I$ re-define $u_k := \min\{u_k, \lfloor U_k \rfloor\}$ and $l_k := \max\{l_k, \lceil L_k \rceil\}$. Observe that these bounds, if finite, are tight.

Step 3. Fix values of variables. Consider a variable x_k.

If $a_k \leq 0$ and $c_k \geq 0$ we may fix $x_k = u_k$. In fact, since x_k is not an accumulator or potentiator, $u_k < +\infty$. On the other hand, because the bounds have already been strengthened, we know that u_k is an admissible value for x_k.

Else, if $a_k \geq 0$ and $c_k \leq 0$ we may fix $x_k = l_k$. In fact, since k is not an accumulator or potentiator, $l_k > -\infty$. On the other hand, because the bounds have already been strengthened, we know that l_k is an admissible value for x_k.

Note that after fixing variables as described above, we can substitute out the values in MIKP and obtain a smaller problem with a new right-hand side. After doing these substitutions we obtain a smaller instance of MIKP such that each variable x_k satisfies ($a_k > 0$ and $c_k > 0$) or ($a_k < 0$ and $c_k < 0$).

Step 4. Complement variables. For simplicity, we will substitute variables so that the lower-bound is always non-negative. Consider a variable x_k. If $-\infty < l_k < 0$ substitute $x'_k = (x_k - l_k)$. If $l_k = -\infty$ substitute $x'_k = u_k - x_k$.

Step 5. Sort data. Sort the variables in order of decreasing efficiency. Break first ties if variables are of integer type or not. Break second ties by value of a_k.

Step 6. Aggregate variables. Consider two variables x_i and x_j for which $a_i = a_j$, $c_i = c_j$ and $(i \in I) = (j \in I)$. These two variables can be aggregated into a single variable x_k such that $a_k = a_i$, $c_k = c_i$, $l_k = l_i + l_j$, $u_k = u_i + u_j$, and $k \in I$ if and only if $i \in I$. This elimination of symmetry will be very helpful later in speeding up the branch and bound algorithm. Note that identifying variables to be aggregated is very easy and can be done in linear time given the way the variable indices are sorted.

After pre-processing we obtain the following problem:

$$\begin{aligned} \text{PP-MIKP} \quad & \max \sum_{k \in P \cup N} c_k x_k \\ & \text{s.t.,} \\ & \sum_{k \in P \cup N} a_k x_k \leq b \\ & l_k \leq x_k \leq u_k, \ \forall k \in P \cup N \\ & x_k \in \mathbb{Z}, \ \forall k \in I \end{aligned}$$

where, $P = \{k : c_k > 0 \text{ and } a_k > 0\}$, and $N = \{k : c_k < 0 \text{ and } a_k < 0\}$.

PP-MIKP satisfies the following conditions:

- PP-MIKP is feasible.
- PP-MIKP is not unbounded, and is not trivial.
- The variable indices are sorted, as described in Step 5.
- All variables x_k are such that ($a_k > 0$ and $c_k > 0$) or ($a_k < 0$ and $c_k < 0$). If the former condition holds, we say $k \in P$. If the latter condition holds, we say $k \in N$.
- For each $k \in P \cup N$, we have $l_k \geq 0$.
- For each $k \in P \cup N$, all finite bounds are tight; that is, there exists a feasible solution to MIKP which achieves the bound.
- There are no two identical variables.

3.4 Solving the LP relaxation of PP-MIKP

In this section we discuss how to solve the linear programming relaxation of PP-MIKP. That is, we focus in solving the problem,

$$\begin{aligned} \text{LP-PP-MIKP} \quad & \max \sum_{k \in P \cup N} c_k x_k \\ & \text{s.t.,} \\ & \sum_{k \in P \cup N} a_k x_k \leq b \\ & l_k \leq x_k \leq u_k, \ \forall k \in P \cup N. \end{aligned}$$

Note that LP-PP-MIKP is nothing more than a linear programming problem. As thus, any Simplex-based linear programming software package would do to solve it. However, this single row variant of a linear programming problem is so much simpler than its multiple-row variants that it merits a closer look.

In this section we present a bare-bones linear-programming approach for solving LP-PP-MIKP. In essence, what is proposed is not new; it is simply a special case of the Simplex algorithm, which extends in a simple way Dantzig's algorithm for solving the linear programming relaxation of bounded, positive coefficient knapsack problems. However, looking at the mechanics of solving this problem reveals that it is very easy. The algorithm proposed is trivial to implement, and, as we will see in the computational results section, runs considerably faster than available commercial packages designed for more general problems. In addition, with little effort it is possible to implement this to work with exact arithmetic.

3.4.1 Characterizing optimality

Definition 3.4. We say that $x^* \in \mathbb{R}^{|P|+|N|}$ is *tight* for PP-MIKP if,

$$\sum_{k \in P \cup N} a_k x_k^* = b.$$

Lemma 3.3. If there exists a tight feasible solution for LP-PP-MIKP, then there exists a tight optimal solution for LP-PP-MIKP.

Proof. Let $\hat{x}$ be an optimal solution of LP-PP-MIKP which is not tight. If $\hat{x}_i = u_i$ for all $i \in P$ and $\hat{x}_j = l_j$ for all $j \in N$, then there clearly exists no tight feasible solution of LP-PP-MIKP.

If there exists $i \in P$ such that $\hat{x}_i < u_i$, we can increase $\hat{x}_i$ by a small amount, maintaining feasibility, yet at the same time increasing the objective function value. Likewise, if there exists $j \in N$ such that $\hat{x}_j > l_j$, we can decrease $\hat{x}_j$ by a small amount, maintaining feasibility, yet at the same time increasing the objective function value. Either case contradicts optimality of $\hat{x}$. ∎

Definition 3.5. Say that $x^* \in \mathbb{R}^{|P|+|N|}$ is *k-efficient* for LP-PP-MIKP if $l_k \leq x_k^* \leq u_k$ and,

- $i \in P$ and $i > k$ implies $x_i^* = l_i$.

- $i \in P$ and $i < k$ implies $x_i^* = u_i$.
- $i \in N$ and $i > k$ implies $x_i^* = u_i$.
- $i \in N$ and $i < k$ implies $x_i^* = l_i$.

Theorem 3.1. If x^* is tight and k-efficient for PP-MIKP, then x^* is an optimal solution of LP-PP-MIKP.

Proof. Let x^* be tight and k-efficient for LP-PP-MIKP. From Lemma 3.3, we know that there exists at least one tight optimal solution of LP-PP-MIKP. Define $\Delta(x, x^*) = \{i \in P \cup N : x_i \neq x^*\}$, and let $\hat{x}$ represent a tight optimal solution of LP-PP-MIKP minimizing $|\Delta(x, x^*)|$. If $|\Delta(\hat{x}, x^*)| = 0$ we are done. Hence, assume $|\Delta(\hat{x}, x^*)| > 0$.

Consider $i \in \Delta(\hat{x}, x^*)$. Note that $a_i x_i^* \neq a_i \hat{x}_i$ and $\sum_{i \in P \cup N} a_i x_i^* = \sum_{i \in P \cup N} a_i \hat{x}_i$ implies that there must exist $j \in \Delta(\hat{x}, x^*)$ such that $i \neq j$, and such that $(a_j x_j^* - a_j \hat{x}_j)(a_i x_i^* - a_i \hat{x}_i) < 0$.

Without loss of generality, assume that $a_i(x_i^* - \hat{x}_i) > 0$ and $a_j(x_j^* - \hat{x}_j) < 0$.

Note that $i \leq k$ and $j \geq k$.

In fact, $i \in P$ implies $a_i > 0$, hence $x_i^* > \hat{x}_i$. However, this is only possible if $x_i^* > l_i$, thus $i \leq k$. If $i \in N$ we have $a_i < 0$, thus $x_i^* < \hat{x}_i$. Again, this is only possible if $x_i^* < u_i$, thus $i \leq k$.

Analogously, $j \in P$ implies $a_j > 0$, hence $x_j^* < \hat{x}_j$. However, this is only possible if $x_i^* < u_i$, thus $j \geq k$. If $j \in N$ we have $a_j < 0$, thus $x_j^* > \hat{x}_j$. Again, this is only possible if $x_j^* > l_i$, and so $j \geq k$.

This implies that $e_i \geq e_j$.

Now consider a very small value, $\varepsilon > 0$, and define x^ε as follows,

$$x_k^\varepsilon = \begin{cases} \hat{x}_j - \varepsilon \frac{a_i(x_i^* - \hat{x}_i)}{a_j} & \text{if } k = j, \\ \hat{x}_i + \varepsilon(x_i^* - \hat{x}_i) & \text{if } k = i, \\ \hat{x}_k & \text{otherwise.} \end{cases}$$

Note that:

$$cx^{\varepsilon} - c\hat{x} = -\varepsilon c_j \frac{a_i(x_i^* - \hat{x}_i)}{a_j} + \varepsilon c_i(x_i^* - \hat{x}_i) = \varepsilon a_i(x_i^* - \hat{x}_i)\left(\frac{c_i}{a_i} - \frac{c_j}{a_j}\right) \geq 0,$$

and,

$$ax^{\varepsilon} = a\hat{x} - \varepsilon a_j \frac{a_i(x_i^* - \hat{x}_i)}{a_j} + a_i\varepsilon(x_i^* - \hat{x}_i) = a\hat{x} = b.$$

Furthermore, $x_i^{\varepsilon} = \varepsilon x_i^* + (1-\varepsilon)\hat{x}_i$. Thus, $l_i \leq x_i^{\varepsilon} \leq u_i$.

Also, since $a_j(x_j^* - \hat{x}_j) < 0$, we know that $x_j^* > \hat{x}_j$, implies $a_j < 0$, and $x_j^* < \hat{x}_j$ implies $a_j > 0$. Thus, since $\varepsilon > 0$, and $a_i(x_i^* - \hat{x}_i) > 0$, we conclude $l_j \leq x_j^{\varepsilon} \leq u_j$.

Hence, either $cx^{\varepsilon} - c\hat{x} > 0$, in which case we contradict the optimality of $\hat{x}$, or, we conclude that x^{ε} is tight and optimal for LP-PP-MIKP.

Finally, if x^{ε} is optimal, observe that as ε increases, $|x_i^{\varepsilon} - x_i^*|$ and $|x_j^{\varepsilon} - x_j^*|$ decrease. Thus, by sufficiently increasing ε we will obtain a tight optimal solution of LP-PP-MIKP such that $|\Delta(x^{\varepsilon}, x^*)| < |\Delta(\hat{x}, x^*)|$, contradicting the fact that $\hat{x}$ minimizes $|\Delta(\hat{x}, x^*)|$. ∎

3.4.2 The phase I algorithm

The Phase I Algorithm takes as input an instance of LP-PP-MIKP, and does one of two things: (a) It proves that the instance is infeasible, or (b) it generates x, a k-efficient solution of the instance, having non-negative slack.

The algorithm begins by defining $k = \max\{j \in P \cup N : j \in N \text{ and } u_j = +\infty\}$, assuming that if the latter set is empty, then $k = -1$. If $k = -1$ it generates the solution x, where, for $j \in P \cup N$,

$$x_j = \begin{cases} l_j & \text{if } j \in P, \\ u_j & \text{if } j \in N. \end{cases}$$

If $k > -1$ it generates the solution x, where, for $j \in (P \cup N) \setminus \{k\}$,

$$x_j = \begin{cases} u_j & \text{if } j \in P \text{ and } j < k, \\ l_j & \text{if } j \in P \text{ and } j > k, \\ u_j & \text{if } j \in N \text{ and } j > k, \\ l_j & \text{if } j \in N \text{ and } j < k \end{cases}$$

and where,

$$x_k = \max\left\{ -\frac{1}{a_k}\left(\sum_{j \neq k} a_j x_j - b\right),\ l_k \right\}.$$

Observe that if $k = -1$ then the algorithm may generate an infeasible solution. In this case it is easy to see that the problem itself is infeasible. Also note that when the algorithm generates a feasible solution, this solution will be efficient. Thus, if the solution is tight it will be optimal. Algorithm 2 shows how a Phase I algorithm for LP-PP-MIKP may be implemented.

3.4.3 The primal phase II algorithm

The primal phase II algorithm takes as input an efficient solution of LP-PP-MIKP with non-negative slack, and finds from this an optimal solution to the problem. For this, it works by either increasing the values of positive coefficient variables, or decreasing the values of negative coefficient variables in a successive manner until the tightness condition is met, or until all of the variables are at their bounds and the iteration can't proceed.

Algorithm 2 The Phase I Algorithm

Input: c, a, l, u, P, N.

Output: $k, x_k, objective, activity, status$.

$k \leftarrow -1$
$activity \leftarrow 0$
$objective \leftarrow 0$
for $j = n - 1$ to 0 **do**
  if $j \in N$ and $u_j = +\infty$ **then**
    $k \leftarrow j$
    $x_k \leftarrow l_j$
    break
  end if
end for
for $j = 0$ to $n - 1$ **do**
  if $j \leq k$ **then**
    if $j \in P$ **then**
      $activity \leftarrow activity + u_j|a_j|$
      $objective \leftarrow objective + u_j|c_j|$
    else
      $activity \leftarrow activity - l_j|a_j|$
      $objective \leftarrow objective - l_j|c_j|$
    end if
  else
    if $j \in P$ **then**
      $activity \leftarrow activity + l_j|a_j|$
      $objective \leftarrow objective + l_j|c_j|$
    else
      $activity \leftarrow activity - u_j|a_j|$
      $objective \leftarrow objective - u_j|c_j|$
    end if
  end if
end for
if $activity < b$ **then**
  $status \leftarrow$ feasible solution
  return
else if $activity = b$ **then**
  $status \leftarrow$ optimal solution
  return
else if $activity > b$ and $k > -1$ **then**
  $x_k \leftarrow (activity - b)/|a_k|$
  $status \leftarrow$ optimal solution
  return
else
  $status \leftarrow$ problem infeasible
  return
end if

In order to preserve efficiency at each step it iterates by modifying the variables in order of decreasing efficiency. The algorithm always terminates with an optimal solution of the problem. Algorithm 3 shows how a Primal Phase II procedure for LP-PP-MIKP may be implemented.

3.4.4 The dual phase II algorithm

The dual phase II algorithm takes as input an efficient solution of LP-PP-MIKP with non-positive slack, and finds from this, an optimal solution to the problem. For this, it works by either decreasing the values of positive coefficient variables, or increasing the values of negative coefficient variables in a successive manner until the tightness condition is met, or until all of the variables are at their bounds and the iteration can't proceed. In order to preserve efficiency at each step it iterates by modifying the variables in order of increasing efficiency. The algorithm either terminates with an optimal solution of the problem, or a proof that the problem is infeasible. Algorithm 4 shows how a Dual Phase II procedure for LP-PP-MIKP may be implemented.

3.5 A branch and bound algorithm for MIKP

3.5.1 Variable branching

At each step of the branch-and-bound algorithm, we are presented with an instance of PP-MIKP, and an optimal solution to the corresponding linear relaxation, LP-PP-MIKP. This solution is both tight and efficient, and is fully characterized by the values $k, x_k, activity$ and $objective$. If variable k is of continuous type, we know that the current solution is optimal for PP-MIKP. Likewise, if variable k is of integer type, and in addition, x_k takes on an integer value, then the current solution is also optimal for PP-MIKP.

Now, assume that k is of integer type and x_k currently takes on a fractional value, say f_k. The simplest branching rule consists in defining two branches: In the first (which we call the *down direction*), we change the upper bound of variable x_k so that $x_k \leq \lfloor f_k \rfloor$. In the other, (which we call the *up direction*), we change the lower bound of variable x_k so that $x_k \geq \lceil f_k \rceil$.

Algorithm 3 Primal Phase II Algorithm

Input: $c, a, l, u, P, N, k, x_k, objective, activity.$

Output: $k, x_k, objective, activity, status.$

while $activity < b$ **do**
  if $k \in P$ **then**
    if $u_k < +\infty$ and $a_k * (u_k - x_k) < (b - activity)$ **then**
      $activity \leftarrow activity + a_k * (u_k - x_k)$
      $objective \leftarrow objective + (u_k - x_k) * c_k$
    else
      $objective \leftarrow objective + (b - activity) * c_k/a_k$
      $x_k \leftarrow x_k + (b - activity)/a_k$
      $activity \leftarrow b$
      $status \leftarrow$ tight optimal
      return
    end if
  else
    if $|a_k| * (x_k - l_k) < (b - activity)$ **then**
      $activity \leftarrow activity + |a_k| * (x_k - l_k)$
      $objective \leftarrow objective + (x_k - l_k) * |c_k|$
    else
      $objective \leftarrow objective + (b - activity) * |c_k|/|a_k|$
      $x_k \leftarrow x_k - (b - activity)/|a_k|$
      $activity \leftarrow b$
      $status \leftarrow$ tight optimal
      return
    end if
  end if
  $k \leftarrow k + 1$
  if $k \in P$ **then**
    $x_k \leftarrow l_k$
  end if
  if $k \in N$ **then**
    $x_k \leftarrow u_k$
  end if
end while
$status \leftarrow$ optimal
return

Algorithm 4 Dual Phase II Algorithm

Input: $c, a, l, u, P, N, k, x_k, objective, activity.$

Output: $k, x_k, objective, activity, status.$

while $activity > b$ **do**
  if $k \in P$ **then**
    if $a_k * (x_k - l_k) < (activity - b)$ **then**
      $activity \leftarrow activity - a_k * (x_k - l_k)$
      $objective \leftarrow objective - c_k * (x_k - l_k)$
    else
      $objective \leftarrow objective - (activity - b) * c_k/a_k$
      $x_k \leftarrow x_k - (activity - b)/a_k$
      $activity \leftarrow b$
      $status \leftarrow$ tight optimal
      return
    end if
  else
    if $u_k < +\infty$ and $|a_k| * (u_k - x_k) < (activity - b)$ **then**
      $activity \leftarrow activity - |a_k| * (u_k - x_k)$
      $objective \leftarrow objective - |c_k| * (u_k - x_k)$
    else
      $objective \leftarrow objective - (activity - b) * |c_k|/|a_k|$
      $x_k \leftarrow x_k + (activity - b)/|a_k|$
      $activity \leftarrow b$
      $status \leftarrow$ tight optimal
      return
    end if
  end if
  $k \leftarrow k - 1$
  if $k \in P$ **then**
    $x_k \leftarrow u_k$
  else
    $x_k \leftarrow l_k$
  end if
end while
$status \leftarrow$ problem infeasible
return

After branching we would like to avoid solving the new linear programming relaxation of PP-MIKP to optimality from scratch; rather, we would prefer to hot-start the solve from the current solution that we have. This is very easy to do. In the down-direction we round down x_k, and in the up-direction we round up x_k. Then, *activity* and *objective* are updated. If *activity* increased, we now have an efficient solution with negative slack, so we use the dual phase II algorithm to re-solve. Otherwise, we have an efficient solution with positive slack, so we use the primal phase II algorithm to re-solve.

Algorithm 5 illustrates this procedure.

3.5.2 Reduced cost bound improvements

In order to speed up the run-time of branch and bound algorithms, it is often desirable to improve variable bounds as early as possible in the branching tree. One way of doing so consists in using information derived from the optimal tableau of an LP-relaxation, and combining this information with valid lower bounds.

Consider the use of a slack variable s. We can re-formulate PP-MIKP as,

$$\begin{aligned}
\text{PP-MIKP} \quad & \max \sum_{k \in P \cup N} c_k x_k \\
& \text{s.t.,} \\
& \sum_{k \in P \cup N} a_k x_k + s = b \\
& s \geq 0 \\
& l_k \leq x_k \leq u_k, \ \forall k \in P \cup N \\
& x_k \in \mathbb{Z}, \ \forall k \in I.
\end{aligned}$$

Choose any $i \in I$ and substitute,

$$x_i = \frac{1}{a_i} \left(b - s - \sum_{k \in P \cup N \setminus \{i\}} a_k x_k \right)$$

in the objective function. We obtain the equivalent objective function representation:

$$(b - s) \frac{c_i}{a_i} + \sum_{k \in P \cup N \setminus \{i\}} \left(c_k - a_k \frac{c_i}{a_i} \right) x_k.$$

Algorithm 5 Branching

Input: $c, a, l, u, P, N, k, x_k, objective, activity, branch\_direction$.

Output: $k, x_k, objective, activity, status$.

if $branch\_direction = down$ **then**
  $u_k \leftarrow \lfloor x_k \rfloor$
  $activity \leftarrow activity + a_k(u_k - x_k)$
  $objective \leftarrow objective - c_k(u_k - x_k)$
  $x_k \leftarrow u_k$
  if $a_k > 0$ **then**
    PrimalPhaseII$(c, a, l, u, P, N, k, x_k, objective, activity)$
      $\hookrightarrow k, x_k, objective, activity, status$
    return
  else
    DualPhaseII$(c, a, l, u, P, N, k, x_k, objective, activity)$
      $\hookrightarrow k, x_k, , objective, activity, status$
    return
  end if
else
  $l_k \leftarrow \lceil x_k \rceil$
  $activity \leftarrow activity - a_k(x_k - l_k)$
  $objective \leftarrow objective + c_k(x_k - l_k)$
  $x_k \leftarrow l_k$
  if $a_k > 0$ **then**
    DualPhaseII$(c, a, l, u, P, N, k, x_k, objective, activity)$
      $\hookrightarrow k, x_k, , objective, activity, status$
    return
  else
    PrimalPhaseII$(c, a, l, u, P, N, k, x_k, objective, activity)$
      $\hookrightarrow k, x_k, objective, activity, status$
    return
  end if
end if

Define $z_o^i = be_i$ and $\bar{c}_k^i = (c_k - a_k e_i)$. Problem PP-MIKP can be re-written as,

$$\begin{aligned} \text{PP-MIKP}(i) \quad & \max z_o^i - e_i s + \sum_{k \in P \cup N} \bar{c}_k^i x_k \\ & \text{s.t.,} \\ & \sum_{k \in P \cup N} a_k x_k + s = b \\ & s \geq 0 \\ & l_k \leq x_k \leq u_k, \ \forall k \in P \cup N \\ & x_k \in \mathbb{Z}, \ \forall k \in I. \end{aligned}$$

Now consider the following relaxation of PP-MIKP(i), where the knapsack constraint has been eliminated from consideration:

$$\begin{aligned} \text{RPP-MIKP}(i) \quad & \max z_o^i - e_i s + \sum_{k \in P \cup N} \bar{c}_k^i x_k \\ & \text{s.t.,} \\ & s \geq 0 \\ & l_k \leq x_k \leq u_k, \ \forall k \in P \cup N \\ & x_k \in \mathbb{Z}, \ \forall k \in I. \end{aligned}$$

Let z^* denote the optimal solution of PP-MIKP, and let z_i^* denote the optimal solution of the relaxation RPP-MIKP(i). Clearly, $z_i^* \geq z^*$.

Suppose we have a lower bound z_{lb} on the optimal objective function value. That is, z_{lb} satisfies $z^* \geq z_{lb}$. An easy way to obtain such a lower bound is from feasible solutions to PP-MIKP. In fact, let x correspond to a feasible solution of PP-MIKP. Define $z(x) = \sum_{k \in P \cup N} c_k x_k$. Clearly, $z(x) \leq z^*$.

Now consider a variable x_j with $j \in I$.

Suppose you would like to impose $x_j \geq \alpha_j$, where $\alpha_j > l_i$.

If, after re-defining $u_j = \alpha_j - 1$, we have that $z_i^* \leq z_{lb}$, then the imposed bound is valid for the original problem. In fact, $z_i^* \leq z_{lb}$ implies $z^* \leq z_{lb}$. Hence, if the condition holds, every solution which does not satisfy the considered lower bound has a low objective value, and so, the bound is valid.

Analogously, suppose we want to impose $x_j \leq \beta_j$, where $\beta_j < u_j$. Then, if after re-defining $l_j = \beta_j + 1$, we have that $z_i^* \leq z_{lb}$, then the new upper bound is valid for PP-MIKP

by the same reasoning.

In general, solving system RPP-MIKP(i) is trivial, since there is no constraint linking the different variables. For this, simply fix every variable x_j such that $\bar{c}^i_j > 0$ to its upper bound, and every variable such that $\bar{c}^i_j \leq 0$ at its lower bound. Finally, set $s = 0$. In general, however, it is not even necessary to do this.

Assume x^* is the optimal solution of LP-PP-MIKP. We know that there exists i such that x^* is i-efficient, and we can assume x^* is tight (else x^* would be optimal for PP-MIKP as well).

Because x^* is i-efficient and optimal for PP-MIKP it follows that for $k \neq i$,

$$\bar{c}^i_k > 0 \Rightarrow x^*_k = u_k \text{ and } \bar{c}^i_k \leq 0 \Rightarrow x^*_k = l_k.$$

Thus, $z^* = z^*_i$, and x^* solves both LP-PP-MIKP(i) and LP-RPP-MIKP(i).

Furthermore, it is easy to update the value of z^*_i after bound changes have been made, and check if $z^*_i \leq z_{lb}$. Let $\Delta = z^* - z_{lb}$.

Proposition 3.3. Consider $j \in I$ such that $\bar{c}^i_j > 0$. Let δ_j be a non-negative integer such that $\delta_j \bar{c}^i_j \geq \Delta$. We may change the lower bound of x_j to,

$$L_j = \max\{l_j, u_j - \delta_j\rfloor\}.$$

Proof. Consider a solution (x', s') of PP-MIKP(i) such that $x'_j < u_j - \delta_j$. We have,

$$\begin{aligned} \bar{c}^i_j x_j &< \bar{c}^i_j u_j - \bar{c}^i_j \delta_j \\ &\leq \bar{c}^i_j u_j - \bar{c}^i_j \Delta / \bar{c}^i_j \\ &\leq \bar{c}^i_j u_j - \Delta. \end{aligned}$$

Thus,

$$\begin{aligned} z^i_o - e_i s' + \bar{c}^i_j x' &= z^i_o - e_i s' + \sum_{k \neq j} \bar{c}^i_k x'_k + \bar{c}^i_j x'_j \\ &< z^i_o - e_i s' + \sum_{k \neq j} \bar{c}^i_k x'_k + (\bar{c}^i_j u_j - \Delta) \\ &\leq z^i_o - e_i s^* + \sum_{k \neq j} \bar{c}^i_k x^*_k + \bar{c}^i_j u_j - \Delta \\ &= z^* - \Delta = z_{lb}. \end{aligned}$$

Thus, x' is such that $cx' \leq z_{lb}$. ∎

Proposition 3.4. Consider $j \in I$ such that $\bar{c}^i_j < 0$. Let δ_j be a non-negative integer such that $-\delta_j \bar{c}^i_j \geq \Delta$. We may change the upper bound of x_j to,

$$U_j = \min\{u_j, l_j + \delta_j\}.$$

Proof. Consider a solution (x', s') of PP-MIKP such that $x'_j > l_j + \delta_j$. We have,

$$\begin{aligned} \bar{c}^i_j x_j &< \bar{c}^i_j l_j + \bar{c}^i_j \delta_j \\ &\leq \bar{c}^i_j l_j - \Delta. \end{aligned}$$

Thus,

$$\begin{aligned} z^i_o - e_i s' + \bar{c}^i_j x' &= z^i_o - e_i s' + \sum_{k \neq j} \bar{c}^i_k x'_k + \bar{c}^i_j x'_j \\ &< z^i_o - e_i s' + \sum_{k \neq j} \bar{c}^i_k x'_k + (\bar{c}^i_j l_j - \Delta) \\ &\leq z^i_o - e_i s^* + \sum_{k \neq j} \bar{c}^i_k x^*_k + \bar{c}^i_j l_j - \Delta \\ &= z^* - \Delta = z_{lb}. \end{aligned}$$

Thus, x' is such that $cx' \leq z_{lb}$. ∎

3.6 Domination, branch and bound, and MIKP

As will be seen later in the computational results section, using a simple branch and bound algorithm as described in the previous sections is not enough to successfully tackle problems of practical size. In this section we concentrate on using a property called domination to improve the performance of branch and bound algorithms. The main idea of what will be done is that every time a variable bound is changed, this may have implications that lead us to change other bounds as well. By carefully identifying such implications, it is often possible to fix not just one, but several bounds at each node of the branch and bound tree.

3.6.1 Cost-Domination

Definition 3.6. Consider x^1 and x^2, two feasible solution of PP-MIKP. We say that x^1 *cost-dominates* x^2 if,

$$cx^1 > cx^2 \text{ and } ax^1 \leq ax^2. \tag{3.1}$$

It is easy to see that a necessary condition for x to be optimal is that it is not cost-dominated by any other solution. Let us begin by giving some simple sufficient conditions for cost-domination.

Definition 3.7. Consider indices $i, j \in I$, and non-zero integers k_i, k_j. If,

$$a_i k_i + a_j k_j \geq 0 \text{ and } c_i k_i + c_j k_j < 0 \tag{3.2}$$

and in addition,

$$l_i - u_i \leq k_i \leq u_i - l_i \text{ and } l_j - u_j \leq k_j \leq u_j - l_j \tag{3.3}$$

we say that (i, j, k_i, k_j) define an *integer cost-domination tuple*.

Proposition 3.5. Consider a solution x of PP-MIKP and let (i, j, k_i, k_j) be an integer cost-domination tuple.

- If $k_i \geq 0$ and $k_j \geq 0$, then

  $$x_i \geq l_i + k_i \text{ and } x_j \geq l_j + k_j$$

  implies x is cost-dominated.

- If $k_i \geq 0$ and $k_j \leq 0$, then

  $$x_i \geq l_i + k_i \text{ and } x_j \leq u_j + k_j$$

  implies x is cost-dominated.

- If $k_i \leq 0$ and $k_j \geq 0$, then

  $$x_i \leq u_i + k_i \text{ and } x_j \geq l_j + k_j$$

  implies x is cost-dominated.

- If $k_i \leq 0$ and $k_j \leq 0$, then

  $$x_i \leq u_i + k_i \text{ and } x_j \leq u_j + k_j$$

  implies x is cost-dominated.

Proof. Define x' so that,

$$x'_l = \begin{cases} x_i - k_i & \text{if } l = i, \\ x_j - k_j & \text{if } l = j, \\ x_l \text{ otherwise.} \end{cases}$$

$k_i \geq 0$ and $x_i \geq l_i + k_i$ implies $l_i \leq x_i - k_i \leq u_i$.

$k_i \leq 0$ and $x_i \leq u_i + k_i$ implies $l_i \leq x_i - k_i \leq u_i$.

Thus, in either case, we have $l_i \leq x'_i \leq u_i$.

Analogously, $l_i \leq x'_j \leq u_i$.

Also, observe that

$a_i x'_i + a_j x'_j = a_i x_i - a_i k_i + a_j x_j - a_j k_j \leq a_i x_i + a_j x_j$

and,

$c_i x'_i + c_j x'_j = c_i x_i - c_i k_i + c_j x_j - c_j k_j < c_i x_i + c_j x_j$

Thus, x' is feasible for PP-MIKP, and x' cost-dominates x. ∎

The following proposition follows directly.

Proposition 3.6. Consider an integer cost-domination tuple (i, j, k_i, k_j). Every optimal solution of PP-MIKP satisfies the following conditions:

If $k_i \geq 0$ and $k_j \geq 0$,

$$x_i \geq l_i + k_i \Rightarrow x_j \leq l_j + k_j - 1.$$

If $k_i \geq 0$ and $k_j \leq 0$,

$$x_i \geq l_i + k_i \Rightarrow x_j \geq u_j + k_j + 1.$$

If $k_i \leq 0$ and $k_j \leq 0$,

$$x_i \leq u_i + k_i \Rightarrow x_j \geq u_j + k_j + 1.$$

If $k_i \leq 0$ and $k_j \geq 0$,

$$x_i \leq u_i + k_i \Rightarrow x_j \leq l_j + k_j - 1.$$

Proof. Every optimal solution to PP-MIKP must be non-cost-dominated. Hence, the result follows from Proposition 3.5. ∎

Define for every non-zero rational number q the function

$$sign(q) = \begin{cases} 1 & \text{if } q > 0 \\ -1 & \text{otherwise} \end{cases}$$

Observation 3.1. Consider distinct pairs (i, j, k_i^1, k_j^1) and (i, j, k_i^2, k_j^2), each satisfying (3.2) and (3.3). Further, assume that $sign(k_i^1) = sign(k_i^2)$ and $sign(k_j^1) = sign(k_j^2)$. Then, if $1 \leq |k_i^1| \leq |k_i^2|$ and $1 \leq |k_j^1| \leq |k_j^2|$, the implications derived from (i, j, k_i^1, k_j^1) are stronger than those derived from (i, j, k_i^2, k_j^2). Thus, if $(i, j, k_i^1, k_j^1) \in D$, there is no need to store (i, j, k_i^2, k_j^2) in D as well.

3.6.2 Lexicographic-Domination

Definition 3.8. Consider x^1 and x^2, two feasible solution of PP-MIKP. We say that x^1 *lexicographically-dominates* x^2 if,

$$cx^1 = cx^2 \text{ and } ax^1 \leq ax^2 \tag{3.4}$$

and in addition, there exists $i \in \{1, \ldots, n\}$ such that $x_i^1 < x_i^2$ and:

$$\forall k \in \{1, \ldots, (i-1)\},\ x_k^1 = x_k^2.$$

Definition 3.9. Consider indices $i, j \in I$ such that $i < j$, and non-zero integers k_i, k_j. If,

$$a_i k_i + a_j k_j = 0 \text{ and } c_i k_i + c_j k_j = 0 \tag{3.5}$$

and in addition,

$$1 \leq k_i \leq u_i - l_i \text{ and } l_j - u_j \leq k_j \leq u_j - l_j \tag{3.6}$$

we say that (i, j, k_i, k_j) define an integer lexicographic-domination tuple.

Proposition 3.7. Consider a solution x of PP-MIKP and let (i, j, k_i, k_j) be an integer lexicographic-domination tuple.

- If $k_j \geq 0$, then

$$x_i \geq l_i + k_i \text{ and } x_j \geq l_j + k_j$$

  implies x is lexicographically-dominated.

- If $k_j \leq 0$, then

$$x_i \geq l_i + k_i \text{ and } x_j \leq u_j + k_j$$

  implies x is lexicographically-dominated.

Proof. Define x' so that,

$$x'_l = \begin{cases} x_i - k_i & \text{if } l = i, \\ x_j - k_j & \text{if } l = j, \\ x_l \text{ otherwise.} \end{cases}$$

$k_i \geq 0$ and $x_i \geq l_i + k_i$ implies $l_i \leq x_i - k_i \leq u_i$.

$k_j \geq 0$ and $x_j \geq l_j + k_i$ implies $l_j \leq x_j - k_j \leq u_j$.

$k_j \leq 0$ and $x_j \leq u_j + k_j$ implies $l_j \leq x_j - k_j \leq u_j$.

Thus, $l \leq x' \leq u$.

Also, observe that

$a_i x'_i + a_j x'_j = a_i x_i - a_i k_i + a_j x_j - a_j k_j = a_i x_i + a_j x_j$

and,

$c_i x'_i + c_j x'_j = c_i x_i - c_i k_i + c_j x_j - c_j k_j = c_i x_i + c_j x_j$.

Observe that $x'_i < x_i$. In addition, since $j > i$, we have $1 \leq k < i$ implies $x'_k = x_k$.

Thus, x' is feasible for PP-MIKP, and x' lexicographically-dominates x. ∎

The following proposition follows directly.

Proposition 3.8. Consider an integer lexicographic-domination tuple (i, j, k_i, k_j). There exists an optimal solution x of PP-MIKP such that:

- If $k_j \geq 0$,

$$x_i \geq l_i + k_i \Rightarrow x_j \leq l_j + k_j - 1$$

and

$$x_j \geq l_j + k_j \Rightarrow x_i \leq l_j + k_j - 1.$$

- If $k_j \leq 0$,

$$x_i \geq l_i + k_i \Rightarrow x_j \geq u_j + k_j + 1$$

and

$$x_j \leq u_j + k_j \Rightarrow x_i \leq l_i + k_i - 1.$$

Proof. Follows from the fact that PP-MIKP admits a non-lexicographically dominated optimal solution, and Proposition 3.7. ∎

3.6.3 Domination tuples: Important properties

In this section we establish some fundamental properties of domination tuples. These properties will allow us to effectively use the derived implications that originate from these in the context of a branch and bound algorithm.

Observe that all non-zero rational numbers satisfy $|q| = q \cdot sign(q)$. Throughout this section we will make use of the fact that instances of PP-MIKP satisfy the following properties: For all $i, j \in P \cup N$ we have a_i, a_j, c_i, c_j are all non-zero, $sign(a_i) = sign(c_i)$, and $sign(a_j) = sign(c_j)$.

Cost-Domination Tuples

Here we address some basic questions: Do cost-domination tuples exist? How many can exist? What can we say about them?

Proposition 3.9. Consider $c_i, a_i, c_j, a_j \in \mathbb{Q} \setminus \{0\}$. If (c_j, a_j) and (c_i, a_i) are linearly independent, there exist non-zero integers k_i and k_j satisfying $a_i k_i + a_j k_j = 0$ and $c_i k_i + c_j k_j < 0$.

Proof. Observe that $\frac{a_i}{a_j}$ is a rational number, hence there exist integers k'_i, k'_j such that $-\frac{a_i}{a_j} = \frac{k'_j}{k'_i}$. If $c_i k'_i + c_j k'_j \leq 0$, let $k_i = k'_i$ and $k_j = k'_j$. Otherwise, let $k_i = -k'_i$ and $k_j = -k'_j$. Clearly, $a_i k_i + a_j k_j = 0$ and $c_i k_i + c_j k_j \leq 0$. It is easy to see that the latter inequality is strict if and only if the linear independence holds. ∎

What this result highlights is that, given $i, j \in I$, it is easy to know if there exist k_i and k_j such that (3.2) holds. Moreover, observe that if such a pair exists, then there exist an infinite amount of other such pairs (k_i, k_j). In fact, if (k_i^1, k_j^1) and (k_i^2, k_j^2) satisfy the condition, then so does $(k_i^1 + k_i^2, k_j^2 + k_j^2)$. Thus, taking integer multiples of any one pair we immediately get an infinite amount of other pairs.

At first glance, this may seem a problem. After all, this could imply that there exist an infinite amount of cost-domination tuples to consider. However, as we will see shortly, we really need consider at most one cost-domination tuple for each pair $i, j \in I$. In order to see this we must first go through some preliminary proofs.

First, a simple observation which will simplify the coming case-analysis.

Observation 3.2. Consider $i, j \in I$. Then, $k_i, k_j \in \mathbb{Q} \setminus \{0\}$ satisfy (3.2) and (3.3) if and only if,

$$|a_i|k_i sign(a_i) + |a_j|k_j sign(a_j) \geq 0 \text{ and } |c_i|k_i sign(a_i) + |c_j|k_j sign(a_j) < 0$$

and

$$l_i - u_i \leq k_i sign(a_i) \leq u_i - l_i \text{ and } l_j - u_j \leq k_j sign(a_j) \leq u_j - l_j.$$

The importance of this observation is as follows: If we determine k_i' and k_j' such that,

$$|a_i|k_i' + |a_j|k_j' \geq 0 \text{ and } |c_i|k_i' + |c_j|k_j' < 0$$

and,

$$l_i - u_i \leq k_i' \leq u_i - l_i \text{ and } l_j - u_j \leq k_j' \leq u_j - l_j$$

then, $k_i = sign(a_i)k_i'$ and $k_j = sign(a_j)k_j'$ will satisfy (3.2) and (3.3), and thus they will define a cost-domination tuple. As we will see, this means we can restrict our attention to the cases where a_i, c_i, a_j, c_j are all strictly positive.

Observation 3.3. Assume a_i, c_i, a_j, c_j are all strictly positive.
Integers k_i, k_j satisfy (3.2) if and only if,

$$-\frac{a_j}{a_i}k_j \leq k_i < -\frac{c_j}{c_i}k_j.$$

Analogously, k_i, k_j satisfy (3.2) if and only if,

$$-\frac{a_i}{a_j}k_i \leq k_j < -\frac{c_i}{c_j}k_i.$$

Observation 3.4. Assume a_i, c_i, a_j, c_j are all strictly positive. If k_i, k_j satisfy (3.2), then $sign(k_i) \neq sign(k_j)$.

Observation 3.5. Assume a_i, c_i, a_j, c_j are all strictly positive. If k_i, k_j satisfy (3.2), then $sign(k_j) = 1$ if and only if $\frac{c_j}{a_j} < \frac{c_i}{a_i}$.

Proof. Observation 3.3 is trivial, and Observation 3.4 follows directly from Observation 3.3. Thus, we focus on Observation 3.5.

Note that Observation 3.3 implies $-\frac{a_j}{a_i}k_j < -\frac{c_j}{c_i}k_j$. However,

$$-\frac{a_j}{a_i}k_j < -\frac{c_j}{c_i}k_j \iff \frac{c_j}{a_j}k_j < \frac{c_i}{a_i}k_j.$$

Where from the result follows. ∎

Proposition 3.10. Assume a_i, c_i, a_j, c_j are all strictly positive, and consider $i, j \in I$. Consider two distinct integer pairs (k_i^1, k_j^2) and (k_i^2, k_j^2) satisfying condition (3.2). If $|k_i^1| \leq |k_i^2|$, let $k_i = k_i^1$. Otherwise, let $k_i = k_i^2$. Analogously, if $|k_j^1| \leq |k_j^2|$, let $k_j = k_j^1$. Otherwise, let $k_j = k_j^2$. Then, it follows that the pair (k_i, k_j) satisfies condition (3.2).

Proof. From the previous lemmas we know that $sign(k_i^1) = sign(k_i^2)$, $sign(k_j^1) = sign(k_j^2)$, and $sign(k_i^1) \neq sign(k_j^1)$. Without loss of generality, assume $sign(k_i^1) = 1$ and $sign(k_j^2) = -1$, as the reverse case will be analogous.

If $(k_i, k_j) = (k_i^1, k_j^1)$ or $(k_i, k_j) = (k_i^2, k_j^2)$, the result follows trivially. Moreover, we may assume without loss of generality that $|k_i^1| \leq |k_i^2|$, thus $(k_i, k_j) = (k_i^1, k_j^2)$.

Observe $a_j > 0$ and $k_j^1 \leq k_j < 0$ implies $a_j k_j^1 \leq a_j k_j$. Thus $a_i k_i + a_j k_j \geq a_i k_i^1 + a_j k_j^1 \geq 0$.

On the other hand, $c_i > 0$ and $0 \leq k_i \leq k_i^2$ imply $c_i k_i \leq c_i k_i^2$. Thus $c_i k_i + c_j k_j \leq c_i k_i^2 + c_j k_j^2 < 0$.

∎

Theorem 3.2. Consider an instance of PP-MIKP, and $i, j \in I$ such that (a_i, c_i) and (a_j, c_j) are linearly independent. Exactly one of the two following conditions hold:

- There exists no cost-domination tuple (i, j, k_i, k_j).
- There exists a cost-domination tuple (i, j, k_i^o, k_j^o) such that all other cost-domination tuples (i, j, k_i, k_j) satisfy: $|k_i| \geq |k_i^o|$, $|k_j| \geq |k_j^o|$, $sign(k_i) = sign(k_i^o)$, and $sign(k_j) = sign(k_j^o)$.

If there exists a cost-domination tuple for $i, j \in I$ we say that (i, j, k_i^o, k_j^o) is the *minimal* cost-domination tuple for i, j.

Proof. Assume that there exists some cost-domination tuple for $i, j \in I$.

First, assume that a_i, c_i, a_j, c_j are all positive.

Let (i, j, k_i^1, k_j^1) be a cost-domination tuple such that $|k_i^1|$ is minimized. Also, let (i, j, k_i^2, k_j^2) be a cost-domination tuple such that $|k_j^2|$ is minimized. Let $k_i^o = k_i^1$ and $k_j^o = k_j^2$. From Proposition 3.10 it follows $a_i k_i^o + a_j k_j^o \geq 0$ and $c_i k_i^o + c_j k_j^o < 0$. In addition, $k_i^o = k_i^1$ implies k_i^o non-zero, and $l_i - u_i \leq k_i^o \leq u_i - l_i$. Analogously, $k_j^o = k_j^1$ implies k_j^o non-zero, and $l_j - u_j \leq k_j^o \leq u_j - l_j$. Thus, (i, j, k_i^o, k_j^o) is a cost-domination tuple, and the minimality condition hold.

From Observation 3.5 if follows that if $c_j/a_j < c_i/a_i$ then all cost-domination pairs must satisfy $sign(k_j) = 1$. From this, and Observation 3.4, it follows that in addition they must satisfy $sign(k_i) = -1$. Analogously, if $c_j/a_j \geq c_i/a_i$ then all cost-domination pairs must satisfy $sign(k_j) = -1$ and $sign(k_i) = 1$. Thus the sign equality condition holds.

If a_i, c_i, a_j, c_j are not all positive, the result directly follows from the previous analysis and Observation 3.2. ∎

Finally, an important proposition which will be used in the next section:

Proposition 3.11. Consider $i, j \in I$ such that $u_i = u_j = +\infty$, and, such that (c_i, a_i) and (c_j, a_j) are linearly independent. There exists a minimal cost-domination tuple for (i, j).

Proof. From Proposition 3.9 we know there exist non-zero integers k_i, k_j such that $a_i k_i + a_j k_j = 0$ and $c_i k_i + c_j k_j < 0$. Further, given that $u_i = u_j = +\infty$, it is clear that $l_i - u_i \leq k_i \leq u_i - l_i$ and $l_j - u_j \leq k_j \leq u_j - l_j$. Thus, (i, j, k_i, k_j) defines a cost-domination tuple. The existence of a minimal cost-domination tuple follows from Theorem 3.2. ∎

Lexicographic Domination-Tuples Here we address the same basic questions we tackled concerning Cost-Domination Tuples.

Proposition 3.12. Consider $c_i, a_i, c_j, a_j \in \mathbb{Q} \setminus \{0\}$. If (c_j, a_j) and (c_i, a_i) are linearly dependent, there exist non-zero integers k_i^o and k_j^o such that (i) $a_i k_i^o + a_j k_j^o = 0$ and $c_i k_i^o + c_j k_j^o = 0$, (ii) For all other pairs k_i, k_j such that $a_i k_i + a_j k_j = 0$ and $c_i k_i + c_j k_j = 0$ hold, we have that $|k_i^o| \leq |k_i|$ and $|k_j^o| \leq |k_j|$, and (iii) $k_i^o > 0$.

Proof. Observe that $\frac{a_i}{a_j}$ is a rational number, hence there exist integers k_i, k_j such that $-\frac{a_i}{a_j} = \frac{k_j}{k_i}$. Clearly, this implies $a_i k_i + a_j k_j = 0$. That $c_i k_i + c_j k_j = 0$ follows from the linear dependence. Thus (i) follows. Further, we can select k_i and k_j to be relative prime (by factorizing $-a_i/a_j$), and thus obtain a minimal pair, as desired in (ii). Part (iii) follows immediately as we can multiply both k_i^o and k_j^o by minus one if it does not hold, and still satisfy (i) and (ii). ∎

Theorem 3.3. Consider an instance of PP-MIKP, and $i, j \in I$ such that (a_i, c_i) and (a_j, c_j) are linearly dependent. Exactly one of the two following conditions hold:

- There exists no lexicographic domination tuple (i, j, k_i, k_j).
- There exists a lexicographic domination tuple (i, j, k_i^o, k_j^o) such that all other lexicographic domination tuples (i, j, k_i, k_j) satisfy: $|k_i| \geq |k_i^o|$, $|k_j| \geq |k_j^o|$, $sign(k_i) = 1$, and $sign(k_j) = sign(k_j^o)$.

If there exists a lexicographic domination tuple for $i, j \in I$ we say that (i, j, k_i^o, k_j^o) is the *minimal* lexicographic domination tuple for i, j.

Proof. Follows directly from Proposition 3.12. ∎

Finally, an important Proposition which will be used in the next section:

Proposition 3.13. Consider $i, j \in I$ such that $u_i = u_j = +\infty$, and, such that (c_i, a_i) and (c_j, a_j) are linearly dependent. There exists a minimal lexicographic-domination tuple for (i, j).

Proof. From Proposition 3.12 we know there exist non-zero integers k_i, k_j such that $a_i k_i + a_j k_j = 0$, $c_i k_i + c_j k_j = 0$, and $k_i > 0$. Further, given that $u_i = u_j = +\infty$, it is clear that $1 \leq k_i \leq u_i - l_i$ and $l_j - u_j \leq k_j \leq u_j - l_j$. Thus, (i, j, k_i, k_j) defines a lexicographic-domination tuple. The existence of a minimal lexicographic-domination tuple follows from Theorem 3.3. ∎

Domination Tables

Definition 3.10. Consider the set D comprised of all minimal cost-domination tuples and all minimal lexicographic-domination tuples for PP-MIKP. We say that D is the *domination table* of PP-MIKP.

From Theorem 3.2 and Theorem 3.3 we know that such a domination table is well defined, and has a polynomial number of entries (in terms of the number of variables). Domination Tables summarize the most important information associated to domination tuples, in terms of the implications which can be derived from them. In fact, by observing Proposition 3.6 and Proposition 3.8 we can see that the strongest implications follow from minimal domination tuples.

3.6.4 An improved branch and bound algorithm: Using domination

In this section we focus on how to utilize a domination table in order to speed up the branch and bound algorithm described in Section 3.5.

Branching A domination table can be used to reduce the number of nodes in a branch and bound tree by changing variable bounds at every node of the tree.

Assume that at some node in the branch and bound tree we impose the upper bound constraint $x_i \leq \alpha_i$:

- If there exists $(i, j, k_i, k_j) \in D$ such that $k_i \leq 0,\ k_j \geq 0$ and such that $\alpha_i \leq u_i + k_i$, then we can impose the constraint $x_j \leq \min\{l_j + k_j - 1, u_j\}$.

  In fact, imposing $x_i \leq \alpha_i$ implies $x_i \leq u_i + k_i$ in that sub-tree, hence that $x_j \leq l_j + k_j - 1$ can be imposed follows from the previous section.

- If there exists $(i, j, k_i, k_j) \in D$ such that $k_i \leq 0,\ k_j \leq 0$ and such that $\alpha_i \leq u_i + k_i$, then we can impose the constraint $x_j \geq \max\{u_j + k_j + 1, l_j\}$.

Analogously, if at some node we impose the constraint $x_i \geq \beta_i$:

- If there exists $(i, j, k_i, k_j) \in D$ such that $k_i \geq 0,\ k_j \geq 0$ and such that $\beta_i \geq l_i + k_i$, then we can impose the constraint $x_j \leq \min\{l_j + k_j - 1, u_j\}$.

  In fact, imposing $x_i \geq \beta_i$ implies $x_i \geq l_i + k_i$ in that sub-tree, hence that $x_j \leq l_j + k_j - 1$ can be imposed follows from the previous section.

- If there exists $(i, j, k_i, k_j) \in D$ such that $k_i \geq 0,\ k_j \leq 0$ and such that $\beta_i \geq l_i + k_i$, then, we can impose the constraint $x_j \geq \max\{u_j + k_j + 1, l_j\}$.

Reduced-Cost Bound Improvements A domination table can also be used to more effectively improve variable bounds during the reduced-cost bound improvement phase.

To see this, consider a variable x_j, with $j \in I$, such that $\bar{c}^i_j > 0$.

Consider $\delta_j \in \mathbb{Z}_+$, and suppose we want to know if the bound $x_j \geq u_j - \delta_j$ is valid for the problem. As we saw before, if $\bar{c}^i_j \delta_j \geq \Delta$, then the answer is yes. Using domination, however, we might be able to determine if said bound is valid even if $\bar{c}^i_j \delta_j < \Delta$.

As before, consider LP-RPP-MIKP(i), and re-define in this problem $u_j = u_j - \delta_j - 1$.

Consider each variable x_q with $q \in I \setminus \{j, i\}$.

If $\bar{c}^i_q > 0$ and from a domination table we know that $x_j \geq \beta_j$ implies $x_q \geq \beta_q$, and in addition, $\beta_j \leq u_j - \delta_j - 1$, then re-define $u_q = \beta_q - 1$.

If $\bar{c}^i_q < 0$ and from a domination table we know that $x_j \geq \beta_j$ implies $x_q \leq \alpha_q$, and in addition, $\beta_j \leq u_j - \delta_j - 1$, then re-define $l_q = \alpha_q + 1$.

If, after these bound changes, we have $z^*_i \leq z_{lb}$ then the proposed bound change, for example $x_j \geq u_j - \delta_j$, is valid for PP-MIKP.

Analogously, consider a variable x_j, with $j \in I$, such that $\bar{c}_j^i < 0$.

Consider $\delta_j \in \mathbb{Z}_+$, and suppose we want to know if the bound $x_j \leq l_j + \delta_j$ is valid for the problem.

As before, consider LP-RPP-MIKP(i), and re-define $l_j = l_j + \delta_j + 1$.

Consider each variable x_q with $q \in I \setminus \{j, i\}$.

If $\bar{c}_q^i > 0$ and from a domination table we know that $x_j \leq \alpha_j$ implies $x_q \geq \beta_q$, and in addition, $\alpha_j \geq l_j + \delta_j + 1$, then re-define $u_q = \beta_q - 1$.

If $\bar{c}_q^i < 0$ and from a domination table we know that $x_j \leq \alpha_j$ implies $x_q \leq \alpha_q$, and in addition, $\alpha_j \geq l_j - \delta_j + 1$, then re-define $l_q = \alpha_q + 1$.

If, after these bound changes, we have $z_i^* \leq z_{lb}$ then the proposed bound change, e.g. $x_j \leq l_j + \delta_j$, is valid for PP-MIKP.

Fixing bounds a-priori An interesting application of the domination concept is that it allows us to fix bounds even before we initiate the branching process in the presence of unbounded variables. This is interesting not only because it can help eliminate an infinite amount of feasible solutions from consideration, but also, because once every variable is bounded, it is possible to complement variables and obtain a problem where all coefficients are positive. This can greatly ease the burden of programming and case-analysis, and, in case that all of the variables are integer, this might even allow for the use of more specialized algorithms, such as those mentioned in Section 3.1.1 for KP and BKP.

In order to show this we proceed in two steps:

Step 1: Given any two indices $i, j \in I$ such that $u_i = u_j = +\infty$ and such that $sign(a_i) = sign(a_j)$, it is possible to fix one one of these upper bounds to a finite value.

If (c_i, a_i) and (c_j, a_j) are linearly independent, then Proposition 3.11 assures us that there exists a domination tuple (i, j). Likewise, if If (c_i, a_i) and (c_j, a_j) are linearly dependent, Proposition 3.13 assures us the same.

Now consider the minimal domination tuple (i, j, k_i, k_j). First, note that either $k_i > 0$ or $k_j > 0$. In fact, if $k_i < 0$ and $k_j < 0$ we know that (i, j, k_i, k_j) is a cost-domination tuple. In this case Proposition 3.5 implies that every finite solution to PP-MIKP is cost-dominated. However, this is only possible if the problem is unbounded, which we know not to be the

case from our initial hypothesis regarding PP-MIKP.

Without loss of generality assume $k_i > 0$. Note that $k_j < 0$. In fact, if $k_i > 0$ and $k_j > 0$, given that $sign(a_i) = sign(a_j) = sign(c_i) = sign(c_j)$, it would be impossible that $a_i k_i + a_j k_j \leq 0$ and $c_i k_i + c_j k_j \geq 0$.

Thus we have $k_i > 0$ and $k_j < 0$. However, from Proposition 3.5 and Proposition 3.7 we know that: every solution x such that $x_i \geq l_i + k_i$ and $x_j \leq u_j + k_j$ is dominated. Given that $u_j = +\infty$ we conclude that all non-dominated solutions must satisfy $x_i \leq l_i + k_i - 1$.

Step 2: After completing the previous step, we know that there can be at most two unbounded variables: one of each sign.

First, assume that there is only one unbounded variable x_k and assume that $k \in P$. Observe that the smallest left hand side achievable by the rest of the variables is:

$$L_k = \sum_{i \in P \setminus \{k\}} l_i a_i + \sum_{i \in N} u_i a_i.$$

Then, because $c_k > 0$ and $a_k > 0$ we know that in an optimal solution, x_k will never be greater than $(b - L_k)/a_k$ because otherwise the solution would be infeasible. Hence we can impose $x_k \leq (b - L_k)/a_k$ if x_k is continuous, and $x_k \leq \lfloor (b - L_k)/a_k \rfloor$ if x_k is integer.

Second, assume that there is only one unbounded variable x_k and assume that $k \in N$. Observe that the greatest left hand side achievable by the rest of the variables is:

$$U_k = \sum_{i \in P} u_i a_i + \sum_{i \in N \setminus \{k\}} l_i a_i.$$

Then, because $c_k < 0$ and $a_k < 0$ we know that in an optimal solution, x_k will never take a value greater than $(b - U_k)/a_k$ because if it does, we can decrease x_k while preserving feasibility and increasing the objective function value.

Third, assume that there are two unbounded variables: x_p and x_q with $p \in P$ and $q \in N$. From the previous analysis we know that there exists $k_p > 0$ and $k_q > 0$ such that (p, q, k_p, k_q) define a cost-domination pair. Thus, from Proposition 3.5 it follows that all optimal solutions satisfy $x_p \leq l_p + k_p - 1$ or $x_q \leq l_q + k_q - 1$. Define,

$$L_{k,p} = \sum_{i \in P \setminus \{k\}} l_i a_i + \sum_{i \in N \setminus \{q\}} u_i a_i.$$

If $x_q \geq l_q + k_q$ we know that $x_p \leq l_p + k_p - 1$. Otherwise, from the previous analysis, we know that if $x_q \leq l_q + k_q - 1$, then $x_p \leq [b - L_{k,p} - a_q(l_q + k_q - 1)]/a_p$. Thus, we can impose:

$$x_p \leq \max\{l_p + k_p - 1, [b - L_{k,p} - a_q(l_q + k_q - 1)]/a_p\}.$$

Analogously, we can impose:

$$x_q \leq \max\{l_q + k_q - 1, [b - U_{k,p} - a_p(l_p + k_p - 1)]/a_q\}$$

where,

$$U_{k,p} = \sum_{i \in P \setminus \{k\}} u_i a_i + \sum_{i \in N \setminus \{q\}} l_i a_i.$$

3.6.5 Examples: Using domination tables

In this section we illustrate via some simple (albeit pathological) examples the practical importance of using domination in the context of a branch and bound algorithm.

Cost-Domination Consider an odd positive integer n, and rational values $0 < \varepsilon_2 < \varepsilon_3 < \ldots < \varepsilon_n << 1$. Consider the following instance of a binary knapsack problem:

$$\begin{aligned} \max \quad & x_1 + (1 - \varepsilon_2)x_2 + \ldots + (1 - \varepsilon_n)x_n \\ s.t. \quad & \\ & x_1 + (1 + \varepsilon_2)x_2 + \ldots + (1 + \varepsilon_n)x_n \leq n/2 \\ & x_i \in \{0, 1\} \quad \forall i \in 1, \ldots, n. \end{aligned}$$

Let $m = (n-1)/2$. It is easy to see that when ε_n is very small, the optimal answer is given by $x_1 = x_2 = \ldots = x_m = 1$ and $x_{m+1} = x_{m+2} = \ldots = x_n = 0$.

Observe that for $1 \leq i < j \leq n$ we have that for $k_i = -1, k_j = 1$, the tuple (i, j, k_i, k_j) is cost-dominating. Thus, if we use a cost-domination branching scheme, every time we branch up on a variable x_i (e.g., imposing $x_i = 1$), we will be fixing $x_k = 1$ for every $k < i$. In addition, every time we branch down on a variable x_i (e.g., imposing $x_i = 0$), we will be fixing $x_k = 0$ for all $k > i$. In this way, it is easy to see that a branch and bound algorithm will only need to branch once in order to optimally solve the problem.

Assume, instead, that we do not use a cost-domination branching scheme. Observe that if we fix m variables or less to their bounds, regardless of the variables fixed, and regardless

of the bounds to which we fix them, the linear-programming relaxation bound obtained will be strictly greater than the optimal value. Thus, in a normal branch and bound scheme, it will be necessary to explore (at least) a full tree of depth m. However, this tree size is exponential!

Lexicographic-Domination Consider the following trivial integer knapsack problem:

$$\begin{aligned} \max \quad & x_1 - 2x_2 \\ s.t. \quad & \\ & x_1 - 2x_2 \leq 1.5 \\ & x_1, x_2 \in \mathbb{Z} \\ & x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

It is easy to see that the solutions given by $x_1 = 2k + 1$ and $x_2 = k$ are optimal for all positive integers k, and that the optimal solution value is 1.

Observe that for $k_1 = 2,\ k_2 = 1$ the tuple $(1, 2, k_1, k_2)$ is lexicographically dominating. Thus, a domination-based branch and bound algorithm will solve this problem in five nodes. In fact, the optimal solution that will be obtained by applying the afore-described LP relaxation algorithm will be $x_1^* = 1.5$ and $x_2^* = 0$. In the down-branch, we will impose $x_1 \leq 1$, which will give us the optimal integer solution to the problem. Then, in the up-branch we will impose $x_1 \geq 2$. However, because of the domination-tuple, in this branch we will also impose $x_2 \leq 1$. This will result in an optimal value of $x_1^* = 2$ and $x_2 = 0.25$. However, after branching but once more it easy to see that the branching will conclude (setting $x_2 = 0$ results in an infeasible node, and setting $x_2 = 1$ results in the optimal solution once again).

Now, note that if we modify the lower bounds so that $x_1 \geq r_1$ and $x_2 \geq r_2$, where $r_1, r_2 \in \mathbb{Z}_+$, then, the LP-relaxation of the problem has value 1.5. This means that a normal branch and bound algorithm will never terminate! In fact, it is easy to see that if we branch normally, there will always be a node in which only the lower bounds of x_1 and x_2 have been changed. This node will never be pruned, because the upper bound it provides is always 1.5.

3.6.6 Building a domination table

In this section we are concerned with describing an algorithm for actually constructing a domination table. Throughout this section we will consider $i, j \in I$, and make use of Observation 3.2 assuming c_i, c_j, a_i, a_j are all positive rationals. In addition, we will assume that (c_j, a_j) and (c_i, a_i) are linearly independent, as the linear dependent case is trivial. Our goal will be to find a minimal cost-domination tuple (i, j, k_i, k_j) or prove no such pair exists.

First, note that from Observation 3.4 and Observation 3.5, it follows that if $\frac{c_j}{a_j} \leq \frac{c_i}{a_i}$, then $k_j > 0$ and $k_i < 0$. Likewise, if $\frac{c_j}{a_j} > \frac{c_i}{a_i}$, then $k_j < 0$ and $k_i > 0$. Without loss of generality we will assume $k_j > 0$ and $k_i < 0$, since the other case is strictly analogous.

Next, observe that given a value k_i^o it is easy to determine if there exists k_j such that (k_i^o, k_j) satisfies (3.2). In fact, from Observation 3.3 it is easy to see that k_j exists if and only if,

$$\left\lceil -\frac{a_i}{a_j} k_i^o \right\rceil < -\frac{c_i}{c_j} k_i^o$$

Furthermore, then the smallest such k_j has value $\left\lceil -\frac{a_i}{a_j} k_i^o \right\rceil$.

Analogously, observe that given a value k_j^o it easy to determine if there exists k_i such that (k_i, k_j^o) satisfies (3.2). In fact, Observation 3.3 implies that such a k_i exists if and only if,

$$-\frac{a_j}{a_i} k_j^o \leq \left\lceil -\frac{c_j}{c_i} k_j^o \right\rceil - 1$$

Furthermore, if the inequality holds, then the largest such k_i has value $\left\lceil -\frac{c_j}{c_i} k_j^o \right\rceil - 1$. Recall that we are interested in the k_i minimizing $|k_i|$, hence, since $k_i < 0$, we look for the largest possible value.

Now, given k_i we know how to get k_j if such exists. Likewise, given k_j we know how to get k_i, provided such exists. What we do next is simply enumerate over possible values of k_i or k_j until we find a pair satisfying the desired conditions.

Let (k_i^a, k_j^a) be such that $a_i k_i^a + a_j k_j^a = 0$ and $c_i k_i^a + c_j k_j^a < 0$. Such a pair of integers is easy to compute. Observe that any minimal pair (k_i, k_j) will satisfy

- $1 \leq k_j \leq \min\{k_j^a, u_j - l_j\}$,

- $\max\{k_i^a, l_i - u_i\} \leq k_i \leq -1$.

In order to minimize the amount of effort then, it seems wisest to enumerate over k_i or k_j depending on which has a smaller feasibility interval, always starting from the value closest to 0.

The entire procedure is summarized in Algorithm 6, where it is assumed that a_i, a_j, c_i, c_j are all positive rational numbers. Note that the algorithm incorporates an auxiliary constant $LOOP\_MAX$. Setting this to a finite value may lead to the algorithm returning a failure code, even in the case where (c_i, a_i) and (c_j, a_j) are linearly independent. However, incorporating this constant may, in cases where numbers are very badly conditioned, speed up the process of filling up a domination table considerably.

3.7 Computational results

In this section, computational results obtained after testing the ideas mentioned in this chapter are described. All of the implementations were written in the "C" and "C++" programming languages, and compiled with the gcc and g++ compilers, version 3.2, on a Linux operating system, version 2.4.27. The computers used to run the implementations were all Intel Xeon dual-processor machines, each with 2GB of RAM, running at 2.66GHz per processor. All of the knapsack algorithms were implemented with templates, thus allowing for them to use different types of numerical precision. However, in this study we just focus on using the standard double floating point arithmetic implemented in the "C++" programming language.

We shall henceforth refer to our algorithm as KBB, for Knapsack Branch and Bound. All of the techniques in this chapter have been implemented, with the exception of the domination-induced reduced-cost bound improvement technique which is a later development. Whenever we say that KBB is ran with its full functionality, we mean that all of the improvement techniques described in the chapter except this one. The KBB algorithm, by default, is set to work with a precision of 10^{-6}.

In order to assess the performance of our algorithms we use as a benchmark the CPLEX mixed-integer programming solver, version 9.0. We change the settings of CPLEX so that

Algorithm 6 Obtaining k_i and k_j

Input: $a_i, c_i, l_i, u_i, a_j, c_j, l_j, u_j$
Output: $status, k_i, k_j$

if (c_i, a_i) and (c_j, a_j) are linearly dependent **then**
  $status \leftarrow$ failure
  return
else if $c_j/a_j > c_i/a_i$ **then**
  Swap (c_i, a_i, l_i, u_i) and (c_j, a_j, l_j, u_j)
  Compute $k_i,\ k_j$ (recursion) and obtain $status$
  Swap k_i and k_j
  return
end if
Compute k_i^a and k_j^a such that $a_i k_i^a + a_j k_j^a = 0$ and $c_i k_i^a + c_j k_j^a < 0$
$L_i \leftarrow \max\{l_i - u_i,\ k_i^a,\ -1 * LOOP\_MAX\}$
$U_j \leftarrow \min\{u_j - l_j,\ k_j^a,\ LOOP\_MAX\}$
if $|L_i| \leq |U_j|$ **then**
  for $k_i = -1$ to L_i **do**
    $k_j \leftarrow \lceil -k_i(a_i/a_j) \rceil$
    if $k_j < -k_i(c_i/c_j)$ **then**
      $status \leftarrow$ success
      return
    end if
  end for
else
  for $k_j = 1$ to U_j **do**
    $k_i \leftarrow \lceil -k_j(c_j/c_i) \rceil - 1$
    if $k_i \geq -k_j(a_j/a_i)$ **then**
      $status \leftarrow$ success
      return
    end if
  end for
end if
$status \leftarrow$ failure
return

optimality is ensured by a tolerance of 10^{-6}. This is the only change of settings we impose on CPLEX, otherwise all other settings are set to their default values.

All tests in this chapter will be done on a library of problems which we call `mikp_hard` consisting of 1,556 problems. These problems make up the most difficult set of individual knapsack instances encountered in the applications described in Chapter 4.

In order to summarize results we shall present some tables. In these tables we make use of the following notation:

- time: Represents the amount of time, in seconds, to solve the problem using branch and bound. This includes pre-processing time, and in cases where domination is employed, the computation of the domination table.
- bbnodes: Represents the total number of branch and bound nodes explored in order to solve the problem to optimality by the algorithm.
- nvars: Represents the total number of variables in the original instance.
- ppnvars: Represents the total number of variables in the pre-processed instance.

Whenever we use the prefix "avg-" before any of these terms, we mean that the corresponding values have been averaged over the total.

In order to present additional information we will also use cumulative histogram curves. In these curves, time will be plotted in the x-axis, and the number of instances in the y-axis. In this way, a point (t_o, n_o) on a curve will mean that n_o of the problem instances were solved in t_o seconds or less. These graphs will always be plotted using a logarithmic scale in the x axis in order to account for the large range of different solution times observed. These curves provide a wealth of information that is often missed when just looking at averages. Full tables are not presented due to the large number of problems being considered.

Performance of the algorithm We first test the KBB algorithm with all of its features on the problem instances of `mikp_hard`. For comparison, we also solve each of these instances with CPLEX. A summary of the computations is presented in Table 3.1, and the histogram curves in Figure 3.1.

Table 3.1: Comparing KBB and CPLEX: Summary

	cpx	kbb
avg-time	15.50	1.66
avg-bbnodes	176,699	110,396
errors	49	0

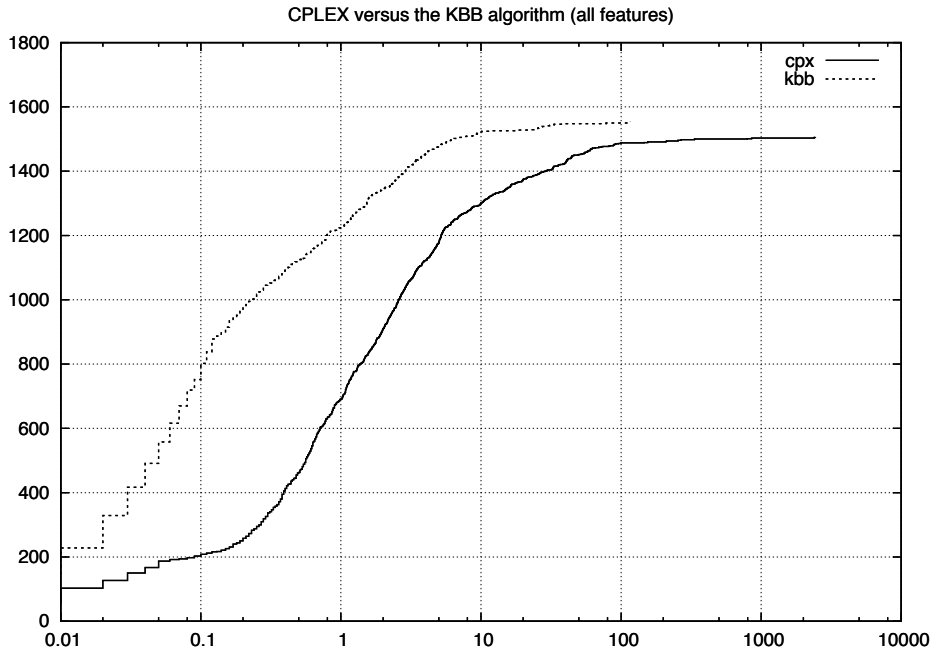


Figure 3.1: Comparing KBB and CPLEX: Histogram

As can be seen from Table 3.1 and Figure 3.1 the KBB algorithm significantly outperforms the CPLEX algorithm. This is not surprising, perhaps, if we consider that KBB is a problem-specific algorithm. However, it is important to note that KBB is the only alternative to date for solving these problems. Observe that most of the instances are solved in less than a second by KBB, yet CPLEX takes just less than 10 seconds to solve the same amount of instances. Finally, observe that CPLEX has a particularly difficult time solving some of the problems. In fact, there are three instances which take CPLEX over 2,000 seconds to solve, whereas the most difficult problem for KBB takes just over 100 seconds. It would seem from the numbers that the large reduction in the number of branch and nodes afforded by the KBB algorithm is key in obtaining this quality of solution speed.

The pre-processor In order to best understand what it is that makes KBB work so much better than CPLEX, the first thing we wanted to ensure was that this difference is not due to simply to the pre-processing algorithm. In order to answer this, all of the instances in `mikp_hard` were ran twice with CPLEX; once with our pre-processor (*pp-cpx*), and once without (*cpx*). Note that in pp-cpx the CPLEX pre-processor was not deactivated. That is, those instances are pre-processed twice; once by the KBB pre-processor, and once by CPLEX. A summary of the computations is presented in Table 3.2, and the histogram curves in Figure 3.2.

Table 3.2: Using CPLEX with the KBB pre-processor: Summary

	cpx	pp-cpx
avg-time	15.50	20.77
avg-bbnodes	176,699	227,781
errors	49	55

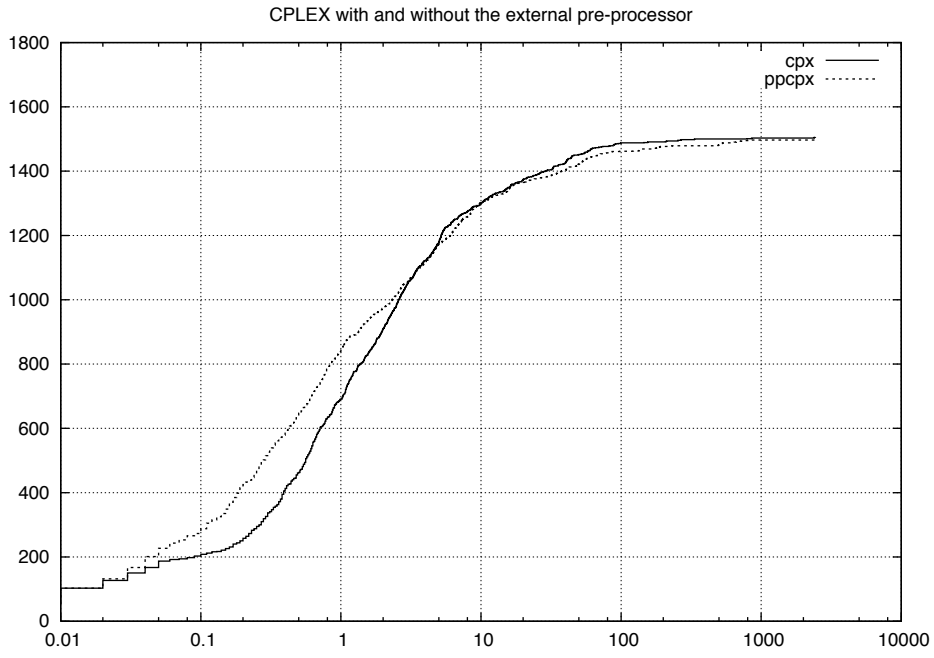


Figure 3.2: Using CPLEX with the KBB pre-processor: Histogram

As can be seen from Table 3.2 and Figure 3.2 the KBB pre-processor has little impact on the performance of CPLEX. This is not surprising, perhaps, if we consider that CPLEX

already counts with its own, very effective set of pre-processing routines. It is curious to note that CPLEX is often hampered by the KBB pre-processing. It can be seen in Figure 3.2 that on the more difficult instances CPLEX performs better if the problems are not pre-processed first.

Diving strategies When programming the KBB algorithm, a question which came up was: How should the branch-and-bound node-selection be designed? While it seemed natural to adopt a Depth-First-Search strategy, the criteria by which to select the node down which to dive was not clear. Preliminary runs on a small subset of the problems in `mikp_hard` had shown that the optimization process was very sensitive to the decision, so four different strategies were compared over all the instances. These strategies are: *up*, where diving is always done first on the branch in which the lower bound was increased, *down*, the reverse of the up strategy, *least-infeas*, where diving is always done in the direction of least infeasibility (e.g. where the rounding was less), and *most-infeas*, the reverse of least-infeas. A summary of the computations is presented in Table 3.3, and the histogram curves in Figure 3.3.

Table 3.3: Comparing different diving strategies: Summary

	up	least-inf	most-infeas	down
avg-time	1.66	2.04	2.14	2.51
avg-bbnodes	110,396	117,838	122,186	128,331

As can be observed in Figure 3.3 all four diving strategies performed similarly well when considering the entire test set. This does not mean, however, that the instances performed similarly well on a one-to-one basis. In fact, it was observed that some particular instances were very sensitive to the diving strategy employed. This is partially reflected in Table 3.3 where it can be seen how the average time and the average number of branch and bound nodes varied with the diving strategies. At first glance the results summarized in Table 3.3 might seem a bit contradictory with the graph observed in Figure 3.3. However, it is perfectly explainable by observing the tail of the plots in Figure 3.3, where it can be seen that a few particular instances took a very long time to solve with some diving strategies -

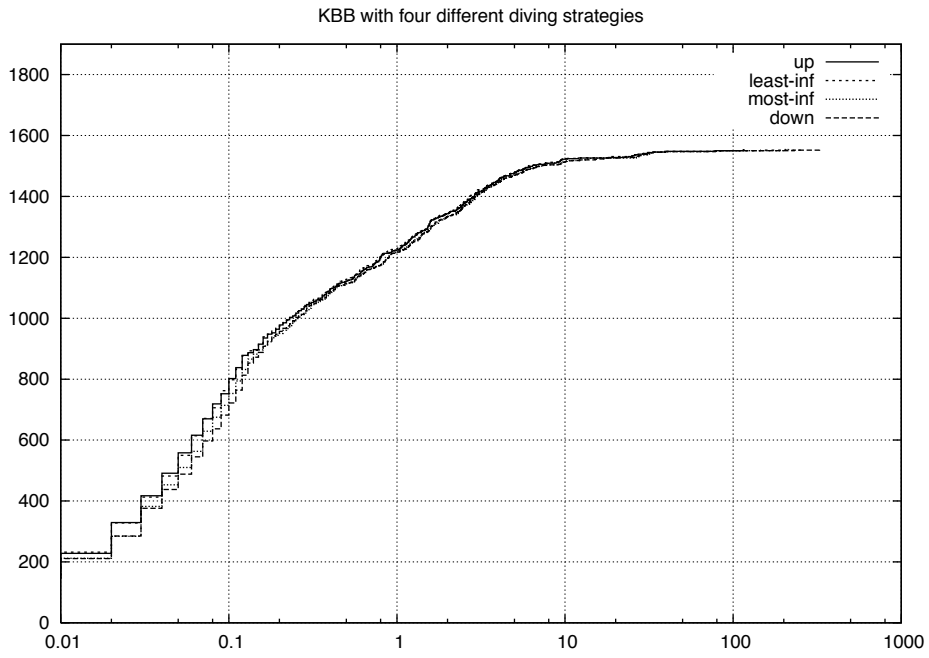


Figure 3.3: Comparing different diving strategies: Histogram

thus greatly affecting the overall averages.

Using heuristics A potential short-coming of the KBB algorithm as presented is that it does not include a heuristic for quickly generating feasible solutions in the branch and bound tree. In order to determine the importance which this concern should be given, a simple experiment was done: Each of the problems in `mikp_hard` was solved twice by the KBB algorithm. The second time, however, the optimal solution of each problem was provided as input in order that pruning might speed up the solve. A summary of the computations is presented in Table 3.4, and the histogram curves in Figure 3.4.

Table 3.4: Using the optimal solution as an upper bound: Summary

	kbb	kbb w/ bounds
avg-time	1.66	1.47
avg-bbnodes	110,396	105,511

As can be observed from Table 3.4 and Figure 3.4 there was little if any impact from using the bound obtained from the optimal solution. This is explained by the fact that

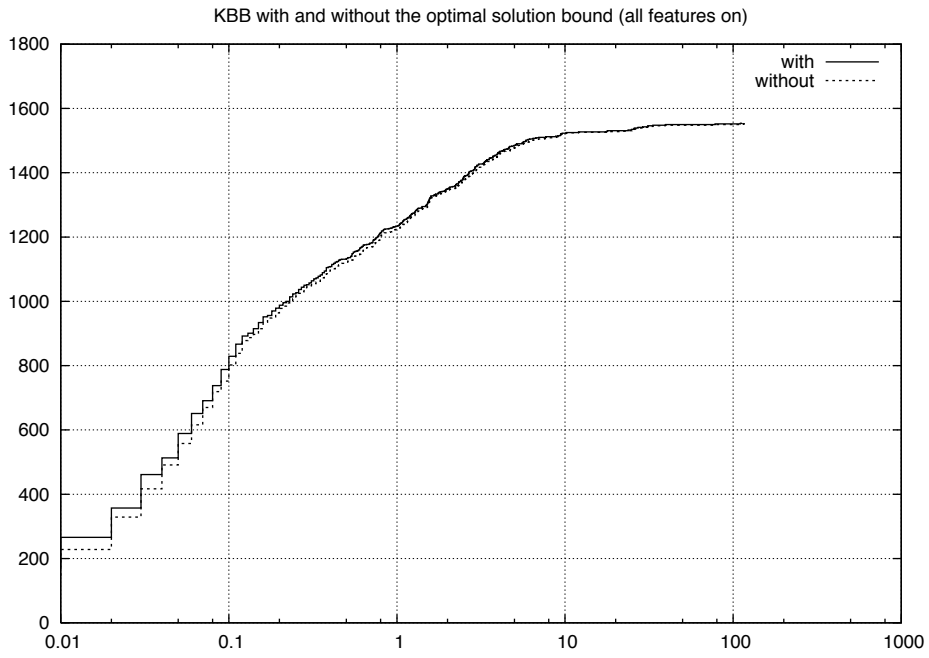


Figure 3.4: Using the optimal solution as an upper bound: Histogram

KBB seems to find optimal solutions rather early in the branch and bound algorithm, and that most of the time is actually spent in proving optimality.

The importance of reduced-cost bound improvements As mentioned before, the domination-induced reduced-cost bound improvement techniques described in Section 3.6.4 have not been implemented. The motivation for wanting to do so follows from the following experiment, in which KBB is ran twice, once with its full set of features, and once without any reduced-cost bound improvement methods at all (e.g., the methodology described in Section 3.5.2 is turned off). A summary of the computations is presented in Table 3.5, and the histogram curves in Figure 3.5.

Table 3.5: The importance of reduced-cost bound improvements: Summary

	with	without
avg-time	1.66	3.72
avg-bbnodes	110,396	300,070

As can be seen from Table 3.5 and Figure 3.5 the reduced cost bound improvement

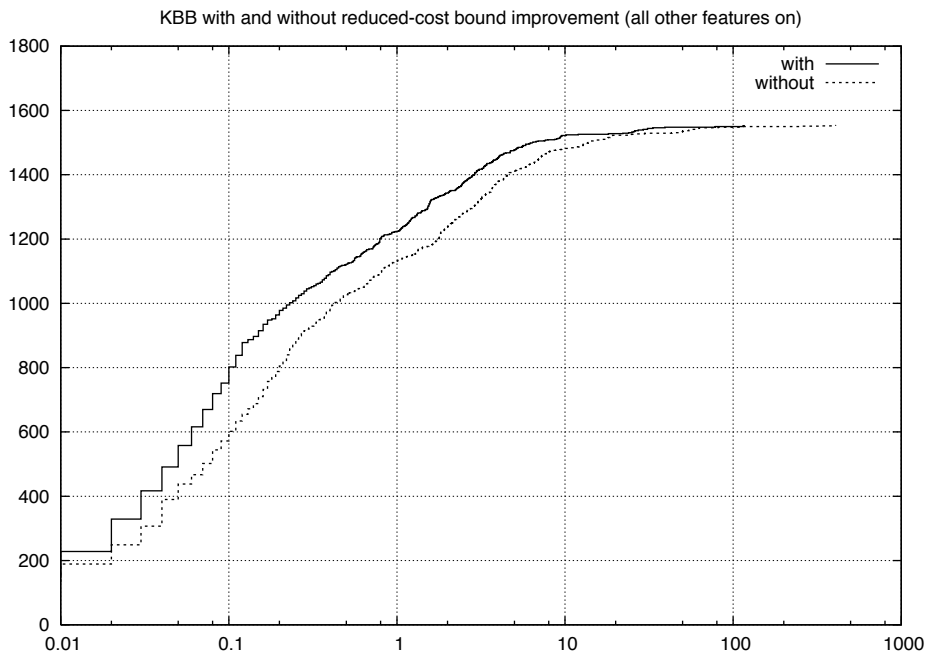


Figure 3.5: The importance of reduced-cost bound improvements: Histogram

technique had a big impact in overall solution times. This can be observed in several different aspects of the computations. For starters, by observing the plot in Figure 3.5 it can be seen that the longest solution time with the reduced-cost bound enabled took approximately 100 seconds. On the other hand, the longest solution time without the reduced cost bounds enabled took near 1000 seconds to solve. While this alone might explain the difference in averages observed in Table 3.5, it can be seen that it is not just a few instances that are greatly affected. In fact, it can be seen in Figure 3.5 that the behaviour of the algorithm with the reduced-cost bounds enabled is systematically better. For example, 600 instances are solved to optimality in 0.1 seconds without the bounds enabled, whereas 800 instances are solved in the same amount of time with the bounds enabled. Analogously, around 1100 instances are solved to optimality in 1 second without the bounds enabled, whereas just over 1200 instances are solved to optimality in the same amount of time using the bounds.

This result suggests that an implementation of the dominance-assisted reduced cost bound improvement technique could lead to important additional improvements.

3.8 Domination and general integer programming

Motivated by the success of domination-based branching in the context of MIKP, in this section we extend the ideas developed so that they can applied to general integer programming problems. Though we do not discuss the case of general mixed integer programming problems, it is our hope that this discussion will illustrate the key concepts which are employed in the procedure, and that further extensions to mixed integer programming problems will not be difficult. Further, it is important to remark that the observations made in this section correspond to very basic concepts, and are the focus of ongoing research.

3.8.1 Integral generating sets, integer programming, and domination

We begin by giving a brief overview of integral generating sets. For a more thorough review and for complete proofs, see Bertsimas and Weismantel [21].

A subset C of $\mathbb{R}^n$ is called a *cone* if it closed under taking conic combinations, i.e., for all $x, y \in C$ and $\lambda, \mu \geq 0$ we have that $\mu x + \lambda y \in C$. A cone C is called *polyhedral* if there exists $A \in \mathbb{Q}^{m \times n}$ such that $C = \{x \in \mathbb{R}^n : Ax \leq 0\}$.

Let $\mathcal{F} \subseteq \mathbb{Z}^n$. A set $H \subseteq \mathcal{F}$ is an *integral generating set* of $\mathcal{F}$ if for every $x \in \mathcal{F}$ there exist $\{h_1, \ldots, h_k\} \subseteq H$ and multipliers $\lambda_1, \ldots, \lambda_k \in \mathbb{Z}_+$ such that $x = \sum_{i=1}^{k} \lambda_i h_i$. An integral generating set H of $\mathcal{F}$ is called an *integral basis* if it is minimal with respect to inclusion.

An important theorem regarding the relationship of polyhedral cones and integral generating sets is as follows:

Theorem 3.4 (Giles and Pulleyblank [53])**.** The set of all integer points in a rational polyhedral cone has a finite integral generating set.

We now relate the integral generating sets and integer programming.

Consider $A \in \mathbb{Q}^{m \times n}$, and $b \in \mathbb{Q}^m$. Let $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, and let $P_I = P \cap \mathbb{Z}^n$. Consider the integer programming problem,

$$\max\{c^t x : x \in P_I\}. \tag{3.7}$$

Let $O_1, \ldots, O_{2^m}$ represent the orthants in $\mathbb{R}^m$. For every $j \in \{1, \ldots, 2^m\}$ let $C_j = \{x \in$

$\mathbb{R}^n | Ax \in O_j\}$, and let H_j be an integral generating set of $C_j \cap \mathbb{Z}^n$. Let $H = \bigcup_{j=1}^{2^m} H_j$. Observe that Theorem 3.4 implies that there exists a finite set H.

Theorem 3.5. [Graver [63]] $x \in P_I$ is optimal for problem (3.7) if and only for every $h \in H$ one of the two conditions holds:

- $c^t h \leq 0$
- $c^t h > 0$ and $x + h \notin P$.

Say that $x \in P_I$ is *cost-dominated* if there exists $h \in H$ such that $c^t h > 0$ and $x+h \in P_I$. Theorem 3.5 clearly implies that $x \in P_I$ is optimal if and only if x is not cost-dominated. Say that $S \subseteq P_I$ is *cost-dominated* if there exists $h \in H$ such that $c^t h > 0$ and such that for every $x \in S$, $x + h \in P_I$. If a set S is cost-dominated it is straight-forward that S does not contain an optimal solution of (3.7).

Now let $Z(d) = \{x \in P_I : c^t x = d\}$, and consider any complete order relationship $<_L$ in $Z(d)$. For example, let $x_1, x_2 \in Z(d)$. Say that $x_1 <_L x_2$ if there exists $i \in 1, \ldots, n$ such that $x_i^1 < x_i^2$ and $\forall k \in 1, \ldots, (i-1)$, $x_k^1 = x_k^2$.

Say that $x \in P_I$ is *lexicographically-dominated* if there exists $h \in H$ such that $c^t h = 0$, $x+h \in P_I$ and $(x+h) <_L x$. Analogously, say that $S \subseteq P_I$ is *lexicographically-dominated* if there exists $h \in H$ such that $c^t h = 0$ and such that every $x \in S$ satisfies $x + h \in P_I$ and $(x+h) <_L x$. If a set S is lexicographically-dominated it is straight-forward that there exists an optimal solution of (3.7) outside of S.

Say that a set $S \subseteq P_I$ is *dominated* if S is either cost-dominated or lexicographically-dominated.

Finally, consider a branch-and-bound tree with nodes $\{\mathcal{N}_1, \ldots, \mathcal{N}_q\}$. Let $\{S_1, \ldots, S_q\}$ represent the feasible regions defined at each node. Clearly, $S_i \subseteq P_I$ for all $i = 1, \ldots, q$. The idea of domination-branching is to incorporate domination information in the branch-and-bound exploration in one of two different ways. The first way consists in using a-posteriori information. That is, if a branch-and-bound node $\mathcal{N}_i$ is such that S_i dominated, then node $\mathcal{N}_i$ can be pruned from the search tree. The other way consists in using a-priori information.

That is, branching rules can be constructed in such a way as to ensure than no set S_i will be dominated.

3.8.2 The KBB algorithm and domination branching

The KBB domination-branching technique described in Section 3.6 uses a simplified version of the framework outlined in Section 3.8.1. The idea is to exploit the fact that typically a branch is accomplished by imposing variable bound changes. In order to explain how this idea fits in with the more general scheme of domination-branching, consider $A' \in \mathbb{Q}^{m' \times n}$, $b' \in \mathbb{Q}^{m'}$, and $l, u \in \mathbb{Q}^n$ such that $P = \{x \in \mathbb{R}^n : A'x \leq b',\ l \leq x \leq u\}$, where P is assumed equivalent the set defined in Section 3.8.1.

Let H' be a generating set for $\{x \in \mathbb{Z}^n : A'x \leq 0\}$, and consider $x \in P_I$. Observe that if $h \in H'$ is such that $l \leq (x + h) \leq u$, then, $(x + h) \in P_I$. Thus, if $h \in H'$ is such that $c^t h > 0$ and $l \leq (x + h) \leq u$, then x is cost-dominated. Likewise, if $h \in H'$ is such that $c^t h = 0$, $l \leq (x + h) \leq u$, and $(x + h) <_L x$, then x is lexicographically-dominated.

Let $D^c = \{x \in H' : c^t x > 0\}$, and let $D^l = \{x \in H' : c^t x = 0$, and $(x + y) <_L x$ for all $y \in \mathbb{R}^n\}$. Observe that for many order relationships $<_L$, including the definition given in Section 3.8.1, the set D^l is well-defined. Further, observe that if $x \in P_I$ is such that $l \leq (x + h) \leq u$ for some $h \in D^c$, then x is cost-dominated. Likewise, if $x \in P_I$ is such that $l \leq (x + h) \leq u$ for some $h \in D^l$, then x is lexicographically-dominated. Finally, observe that the cost-domination tuples defined for MIKP in Section 3.6.1 correspond to a subset of D^c, and that the lexicographic-domination tuples defined for MIKP in Section 3.6.2 correspond to a subset of D^l.

Now consider a branch-and-bound tree in which branching is performed by imposing upper and lower bounds on variables. If $\{\mathcal{N}_1, \ldots, \mathcal{N}_q\}$ correspond to the nodes of the branch-and-bound tree and $\{S_1, \ldots, S_q\}$ represent the feasible regions defined at each node, then there exist $l^1, \ldots, l^q$ and $u^1, \ldots, u^q$ such that $l \leq l^i$, $u^i \leq u$ and $S_i = P_I \cap \{x \in \mathbb{R}^n : l^i \leq x \leq u^i\}$ for each $i = 1, \ldots, q$.

Observe that if for some $i \in \{1, \ldots, q\}$ and some $h \in H'$ we have that $l^i - l \leq h \leq u - u^i$, then, for every $x \in S_i$ we have that $l \leq (x + h) \leq u$. Thus, if at some node $\mathcal{N}_i$ we have

that there exists $h \in D^c \cup D^l$ such that $l^i - l \leq h \leq u - u^i$, then S_i is dominated. Hence, at every node one could check if such a vector $h \in D^c \cup D^l$ exists, and if so, prune the respective node. This would correspond to using domination information in an aposteriori way. Though this idea has not been tested in the context of MIKP, it is a topic of ongoing research.

Another way of making use of this information is as follows: Consider a node $\mathcal{N}_i$ of the branch-and-bound tree and $h \in D^c \cup D^l$. Assume that there exists an index $j_o \in \{1, \ldots, n\}$ such that (a) $(h_{j_o} > u_{j_o} - u^i_{j_o})$ or $(h_{j_o} < l^i_{j_o} - l_{j_o})$, and, (b) for all $j \in \{1, \ldots, n\} \backslash \{j_o\}$ we have that $l^i_j - l_j \leq h_j \leq u_j - u^i_j$. Then, if $(h_{j_o} > u_{j_o} - u^i_{j_o})$, at node $\mathcal{N}_i$ we could add constraint $x \leq u_{j_o} - h_{j_o}$ (thus, effectively strengthening the value of $u^i_{j_o}$). Likewise, if $(h_i < l^i_{j_o} - l_{j_o})$, we could add constraint $x \geq l_{j_o} + h_{j_o}$ (thus, effectively strengthening the value of $l^i_{j_o}$). Observe that this inequalities are valid for the current node of the branch-and-bound tree. In fact, if these new bounds are not satisfied we will be considering a dominated solution. This in fact is exactly the same strategy pursued in the KBB domination-branching scheme, where a subset of $D^u \cup D^l$ is kept in memory at all times, and checks are performed at every node in order to improve bounds as much as possible.

3.8.3 Final remarks

During the last few decades, it can be seen that solving general integer programming problems has proved to be much harder than solving their binary counterparts. Though much progress has been recently made in terms of understanding valid inequalities for general integer sets, little work has been done in better understanding the branching process. Branching, however, seems a natural place to look for improvements. After all, branching is very prone to the difficulties arising from symmetry and problems in which there are many solutions having variable values similar to each other. This seems especially true in the case of general integer programming.

While it is not clear yet that domination branching will prove a successful means to tackling these type of difficulties, there are several reasons which make it an interesting methodology to consider. On the one hand, lexicographic dominance is a very general way

of dealing with certain types of symmetry. On the other hand, cost dominance may help deal with problems having many solutions which can easily be obtained from another by simply shifting values from some variables to others.

In any case, there is yet much research to be conducted in order to better understand domination, both the context of general integer programming, and in the context of MIKP. For instance, in the context of MIKP it would be very interesting to extend the notion of domination-pairs to more complex domination vectors. In addition, it would be interesting to consider mixed-integer domination pairs. That is, domination pairs in which one index corresponds to that of an integer variable, and the other that of a continuous variable. Yet another important direction to pursue would consist in, instead of storing domination vectors in memory which are computed a-priori, somehow find these vectors as the algorithm goes along. Perhaps, by solving a sort of "domination" problem, as opposed to a "separation" or "pricing" problem. What the experience of MIKP shows is that it might not be necessary to actually search the entire space of possible domination vectors. Rather, if one could identify certain classes of such vectors which frequently hinder the branching process, it may be enough to obtain important gains. This separation process could be used in the aposteriori way, or perhaps together with alternative branching schemes.

CHAPTER 4

The Mixed Integer Rounding Cut

4.1 Introduction

Consider $a \in \mathbb{Q}^n,\ b \in \mathbb{Q},\ I \subseteq \{1,\ldots,n\},\ C = \{1,\ldots,n\} \setminus I$, and the following mixed integer polyhedron:

$$P = \{x \in \mathbb{Q}^n : \sum_{i=1}^{n} a_i x_i \geq b,\ x \geq 0,\ x_i \in \mathbb{Z}\ \forall i \in I\}. \tag{4.1}$$

For every $q \in \mathbb{Q}$ define:

- $(q)^+ = \max\{q, 0\}$.
- $\lfloor q \rfloor$: the largest integer less-than-or-equal to q.
- $\lceil q \rceil$: the smallest integer greater-than-or-equal to q.
- $\hat{q} = q - \lfloor q \rfloor$, e.g. the "fractional" part of q.
- $\lceil q \rfloor = \lceil q \rceil$ if $\hat{q} \geq 1/2$, and $\lceil q \rfloor = \lfloor q \rfloor$ if $\hat{q} < 1/2$, e.g. the "nearest" integer to q.

Let $C^+ = \{i \in C : a_i > 0\}$ and assume $\hat{b} \neq 0$. The inequality:

$$\sum_{i \in I} \Big(\min\{\hat{a}_i, \hat{b}\} + \hat{b} \lfloor a_i \rfloor \Big) x_i + \sum_{i \in C^+} a_i x_i \geq \hat{b} \lceil b \rceil,$$

is known as the Mixed Integer Rounding (MIR) cut, and is known to be valid for P.

4.1.1 Background

The Mixed Integer Rounding cut dates back to the early sixties. More precisely, to the fractional cut of Gomory [56], [57]. More than just deriving an important valid inequality for general integer programming, Gomory realized that by repeatedly applying his fractional cuts, one could theoretically solve any integer programming problem to optimality in a finite number of iterations. This idea was very exciting at the time, though unfortunately, it met with little success in practice, as computational results were largely disappointing.

Despite this, much theoretical work ensued up through the 1980s extending these inequalities. Gomory [58] first characterized his inequalities in a broader framework by introducing the master cyclic group polyhedra, Gomory and Johnson [59], [60] extended this framework to mixed integer programming problems, Balas [14] introduced the disjunctive cut, Nemhauser and Wolsey [89] introduced the Mixed Integer Rounding cut as we know it today, Cook, Kannan and Schrijver [33] introduced the split cut, and Balas, Ceria and Cornuéjols [15] introduced the lift-and-project cut. All these authors, as well as many others, generalized the work of Gomory in some way or another.

During the mid-nineties things took an interesting turn, with an important shift towards successful computational experiments. Balas, Ceria, Cornuéjols, and Natraj [17] obtained remarkable results applying the MIR cuts to general mixed integer programming problems (MIPs). These results were quickly echoed by Bixby, Fenelon, Gu, Rothberg, Wunderling [23] and many others. Cornuéjols [35] speculates the following regarding the sudden success of these cuts:

> "The truth of the matter is that, although many authors commented on how bad Gomory cuts worked in practice, very few had actually tried themselves. Those who tried them made poor implementation decisions, such as (i) adopting a pure cutting plane approach and (ii) adding cuts one at a time, re-optimizing the LP after each cut addition."

Up to this day, Bixby, Gu, Rothberg, Wunderling [24] still report that the MIR cuts are the most important cutting plane used in commercial MIP solver, CPLEX.

During the last decade much work has gone into further pursuing an understanding of MIR inequalities, and their importance as a tool for solving general MIPs. Marchand and Wolsey [74] show that many well-known cuts are MIRs, and present an algorithm that applies the MIR procedure to cuts derived by aggregating original formulation rows and by complementing variables. Andersen, Cornuéjols, and Li [3], [4], and also Balas and Perregaard [18] relate the MIR inequalities to split cuts and lift-and-project cuts, and show how these relationships can help in better applying these cuts to solve difficult MIPs. Cornuéjols, Li, and Vanderbussche [36] obtain the class of k-cuts by scaling MIR inequalities, Caprara and Fischetti [29] and Fischetti and Lodi [48] apply MIR inequalities to an aggregation of rows derived by solving an integer programming problem. And the list goes on...

What is remarkable about the MIR is that it is a very simple "template" cut which can be derived from any single valid inequality of an MIP. What seems clear from all this recent research is that when derived from the "correct" valid inequality, the MIR inequality can be very effective.

A fundamental question concerning MIR cuts is the following: What other inequalities derived from single valid inequalities are there? If the MIR is so effective, is there perhaps another class of inequalities which can easily be derived from single valid inequalities which is comparable?

Various researchers have tried to address these questions. These attempts have followed two main directions: The first direction starts from the observation that MIR inequalities are facets of the master cyclic group polyhedron (see Gomory and Johnson, [59], [60]). The work of Aaroz, Evans, Gomory and Johnson [9], Gomory, Johnson, and Evans [61], Dash and Günluk [42], [41], Dash, Goycoolea, and Günluk [38], and Fischetti and Saturni [49] attempt, both theoretically and computationally, to derive other facets of the cyclic group polyhedron. The second direction starts from the observation that MIR inequalities can be derived by lifting simpler systems with less variables. The work of Atamtürk [10], [11], Atamtürk and Rajan [12], Agra and Constantino [2], and Richard, de Farias, and Nemhauser [99] are examples that follow this approach. Though many new families of cuts have been derived through these approaches, and much insight has been gained regarding

the underlying structure of mixed integer cuts, little success has been achieved in being able to improve upon the performance of the MIR computationally.

Define as a *group cut* any valid inequality of the master cyclic group polyhedron. Recently, Dash and Günluk [40] made a startling computational observation: Noting that MIRs were the only group cuts to consistently yield positive results in separation algorithms, they devised an experiment which after adding MIR inequalities derived from tableau rows of a problem, tested if any other group cuts derived from those same tableau rows could possibly be violated. After performing this experiment on a very large set of MIP instances, they found that in a significant number of them, a notable 35% of the problems, no group cuts were violated in any of the tableau rows after the MIRs were added. This experiment, of course, would largely explain many of the unsuccessful attempts at adding additional classes of cuts.

4.1.2 In this chapter

In this Chapter we follow up on the computational experiments of Dash and Günluk [40]. For this we proceed in two steps. First we discuss the MIR inequalities, and describe the implementation of a simple heuristic by which MIR inequalities can be separated. When designing the algorithm we will focus on two issues: Dealing with numerical problems and selecting heuristic rules for complementing variables and scaling inequalities. We accompany this discussion with some computational tests illustrating the effect of the different algorithmic parameters.

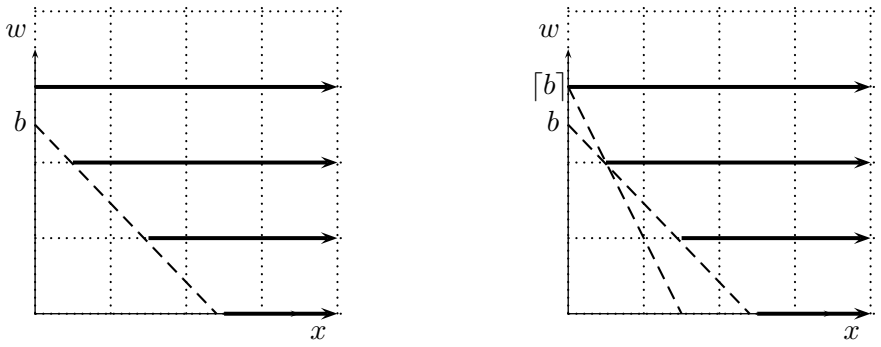
Second, we discuss an experiment similar to that performed by Dash and Günluk. Define as a *knapsack cut* any valid inequality of a mixed integer knapsack problem (MIKP) polyhedron (See Chapter 3). Our goal is to determine if, like in the master cyclic group polyhedron, after adding MIRs all other knapsack cuts are satisfied.

4.2 A simple inequality system

Consider the mixed integer set,

$$Q = \{x \in \mathbb{R}, w \in \mathbb{Z} : w + x \geq b,\ x \geq 0\}.$$

Consider the example in Figure 4.1(a), where the dashes represents the sub-space $w+x = b$, and the dark lines represent Q.



(a) Feasible region (b) A valid inequality

Figure 4.1: A simple mixed integer set

From Figure 4.1(b) it is easy to see that the constraint:

$$x \geq \hat{b}\ (\lceil b \rceil - w), \tag{4.2}$$

is valid for Q. A simple disjunctive proof now follows.

Proof.

First, assume $w \geq \lceil b \rceil$. This implies that $\lceil b \rceil - w \leq 0$, and hence $\hat{b}(\lceil b \rceil - w) \leq 0$. Since $0 \leq x$ the result follows.

Second, assume $w \leq \lceil b \rceil - 1$. This implies that $-(\lceil b \rceil - w) \leq -1$, and hence

$$-(1 - \hat{b})(\lceil b \rceil - w) \leq -(1 - \hat{b}). \tag{4.3}$$

On the other hand, $x + w \geq b$ implies $x + w \geq \lceil b \rceil - (1 - \hat{b})$, and hence,

$$(\lceil b \rceil - w) \leq x + (1 - \hat{b}). \tag{4.4}$$

Adding (4.3) and (4.4) the result follows. ∎

4.3 The MIR inequality

Consider $a \in \mathbb{Q}^n,\ b \in \mathbb{Q}$ and disjoint sets I, C such that $I \cup C = \{1, \ldots, n\}$. Define $C^+ = \{j \in C : a_j > 0\}$ and consider the polyhedron

$$P^= = \{x \in \mathbb{Q}^n : \sum_{j=1}^{n} a_j x_j = b,\ x \geq 0,\ x_j \in \mathbb{Z}\ \forall j \in I\}. \tag{4.5}$$

Let $S \subseteq I$ and note that for all $x \in P^=$:

$$\underbrace{\left(\sum_S \lfloor a_j \rfloor x_j + \sum_{I \setminus S} \lceil a_j \rceil x_j\right)}_{w} + \left(\sum_S \hat{a_j} x_j - \sum_{I \setminus S} (1 - \hat{a_j}) x_j + \sum_C a_j x_j\right) = b.$$

Since $-(1 - \hat{a_j})x_j\ \forall j \in I \setminus S$ and $\sum_{C^-} a_j x_j$ are non-positive it follows that

$$w + \left(\sum_S \hat{a_j} x_j + \sum_{C^+} a_j x_j\right) \geq b. \tag{4.6}$$

Given that $w \in \mathbb{Z}$ and $\left(\sum_S \hat{a_j} x_j + \sum_{C^+} a_j x_j\right) \in \mathbb{R}_+$, from Section 4.2 we have:

$$\left(\sum_S \hat{a_j} x_j + \sum_{C^+} a_j x_j\right) \geq \hat{b}(\lceil b \rceil - w). \tag{4.7}$$

From here we can derive two different forms of the MIR cut, depending on how we substitute w. If we substitute $w = \sum(\lfloor a_j \rfloor x_j : j \in S) + \sum(\lceil a_j \rceil x_j : j \in J \setminus S)$ and do the algebra, we obtain *The Simple-Form of the MIR Inequality* (MIR^1):

$$\sum_S (\hat{a_j} + \hat{b}\lfloor a_j \rfloor) x_j + \sum_{I \setminus S} (\hat{b} + \hat{b}\lfloor a_j \rfloor) x_j + \sum_{C^+} a_j x_j + \geq \hat{b}\lceil b \rceil. \tag{4.8}$$

If we substitute $w = b - \sum(\hat{a_j} x_j : j \in S) + \sum((1 - \hat{a_j}) x_j : j \in I \setminus S) - \sum_{C^+} a_j x_j + \sum_{C^-} a_j x_j$, we obtain *The Normalized-Form of the MIR Inequality* (MIR^2):

$$\sum_S \frac{\hat{a_j}}{\hat{b}} x_j + \sum_{I \setminus S} \frac{(1 - \hat{a_j})}{(1 - \hat{b})} x_j + \frac{1}{\hat{b}} \sum_{C^+} a_j x_j + \frac{1}{(1 - \hat{b})} \sum_{C^-} a_j x_j \geq 1. \tag{4.9}$$

How to choose S? Note that $\hat{a_j} \leq \hat{b}$ if and only if $(\hat{a_j} + \hat{b}\lfloor a_j \rfloor) \leq (\hat{b} + \hat{b}\lfloor a_j \rfloor)$. Likewise, $\hat{a_j} \leq \hat{b}$ iff $\frac{\hat{a_j}}{\hat{b}} \leq \frac{(1 - \hat{a_j})}{(1 - \hat{b})}$. Hence, it is convenient to choose $S = \{j \in I : \hat{a_j} \leq \hat{b}\}$ in order to get the strongest cut possible.

In summary, define $f(q_1, q_2) = \min\{\hat{q_1}, \hat{q_2}\} + \hat{q_2}\lfloor q_1 \rfloor$ and $g(q_1, q_2) = \min\{\frac{\hat{q_1}}{\hat{q_2}}, \frac{1 - \hat{q_1}}{1 - \hat{q_2}}\}$.

Constraint MIR^1 is equivalent to,

$$\sum_{i\in I} f(a_i,b)x_i + \sum_{i\in C^+} a_i x_i \geq \hat{b}\lceil b\rceil. \tag{4.10}$$

Constraint MIR^2 is equivalent to,

$$\sum_{i\in I} g(a_i,b)x_i + \frac{1}{\hat{b}}\sum_{i\in C^+} a_i x_i + \frac{1}{1-\hat{b}}\sum_{i\in C^-} a_i x_i \geq 1. \tag{4.11}$$

Note that constraint MIR^1 is also valid for P (see (4.1)), since in the derivation the equality condition was never explicitly used. This does not hold for MIR^2.

In the rest of this chapter we will limit ourselves to using the Simple-Form of the MIR (e.g., MIR^1) for several reasons:

- The analysis will always be strictly analogous if we consider MIR^2.
- Inequality MIR^1 is valid and well-defined even if $\hat{b} = 0$.
- MIR^1 is numerically convenient as it is not subject to division by very small numbers.

4.4 The complemented-MIR (c-MIR) inequality

For each $i \in C$ let $l_j \in \mathbb{Q},\ u_j \in \mathbb{Q}\cup\{+\infty\}$ and for each $j \in I$ let $l_j \in \mathbb{Z},\ u_j \in \mathbb{Z}\cup\{+\infty\}$. Assume $l_j \leq u_j$ for each $j = 1,\ldots,n$ and consider the more general polyhedron

$$P_B^{=} = \{x \in \mathbb{Q}^n : \sum_{i=1}^{n} a_j x_j = b,\ l_j \leq x_j \leq u_j\ \forall j \in \{1,\ldots,n\} \text{ and } x_j \in \mathbb{Z}\ \forall j \in I\}. \tag{4.12}$$

The introduction of lower and upper bounds allows us to introduce a slightly-more general form of the MIR inequality called the Complemented-MIR (c-MIR) inequalities (see Marchand and Wolsey [74]), because they can be derived by complementing variables.

For each $j = 1,\ldots,n$, if $u_j < \infty$, define $x_j^u = (u_j - x_j)$. Also, define $x_j^l = (x_j - l_j)$. Observe that x_j^u and x_j^l are both bounded below by zero. Furthermore, if x_j takes an integer value, so will x_j^u and x_j^l.

Let sets U, L define a partition of $\{1,\ldots,n\}$ such that every $j \in U$ satisfies $u_j < \infty$. By substituting variables x_j with $j \in U$ by $(u_j - x_j^u)$ and variables x_j with $j \in L$ by $(x_j^l + l_j)$, we obtain the following:

$$\sum_{j\in U}(-a_j)x_j^u + \sum_{j\in L} a_j x_j^l = b - \sum_{j\in U} a_j u_j - \sum_{j\in L} a_j l_j. \tag{4.13}$$

Observe that $x_j^u, x_j^l \geq 0$ for all $j \in \{1,\ldots,n\}$, hence we can apply the MIR procedure to obtain a valid inequality for (4.13) subject to the corresponding non-negativity constraints. Let $r = b - \sum_{j\in U} a_j u_j - \sum_{j\in L} a_j l_j$. If we apply MIR^1 we obtain:

$$\sum_{U\cap I} f(-a_j, r)x_j^u + \sum_{L\cap I} f(a_j, r)x_j^l - \sum_{U\cap C^+} a_j x_j^u + \sum_{L\cap C^+} a_j x_j^l \geq \hat{r}\lceil r\rceil.$$

If we substitute back the variables we get the following valid inequality in terms of our original variables:

$$-\sum_{U\cap I} f(-a_j, r)x_j + \sum_{L\cap I} f(a_j, r)x_j + \sum_{U\cap C^+} a_j x_j + \sum_{L\cap C^+} a_j x_j \geq R, \tag{4.14}$$

where,

$$R = \hat{r}\lceil r\rceil - \sum_{U\cap I} f(-a_j, r)u_j + \sum_{L\cap I} f(-a_j, r)l_j - \sum_{U\cap C^+} a_j u_j + \sum_{L\cap C+} a_j l_j.$$

Constraint (4.14) will henceforth be referred to as the *Simple Complemented-MIR inequality* (c-MIR), or simply, c-MIR^1. Observe that deriving the exact form of the *Normal Complemented-MIR inequality*, or c-MIR^2, is analogous.

4.5 Making use of the MIR inequalities

4.5.1 Before generating any cuts

We now discuss several steps that can be performed before MIR cuts are actually generated. These steps are designed to make the MIR separation more effective and less time consuming.

Ranking integer variables It helps to generate an apriori-ranking system for integer variables having fractional values. A natural, though perhaps naive approach, would consist in ranking variables according to how "fractional" they are. That is, to every integer variable x_i assign weight $w_i = |\hat{x}_i^* - 0.5|$, and say that a variable x_i is more important that x_j if $w_i < w_j$. Another method would be to use the so-called "strong-branching" information.

For every integer variable x_i^* having a fractional value, branch once and compute the LP relaxation bound obtained for each of the branches (or do a limited number of pivots for each branch). Then, assign to w_i the "worst" (the largest in case our problem is a maximization one, or the smallest if minimization) of the two bounds obtained. Then, say that x_i is more important than x_j if the bound w_i is stronger than bound w_j. Note that most integer programming software packages have a function which provides this strong-branching information (or some variant thereof) for all variables through a single function call. Bixby, Fenelon, Gu, Rothberg and Wunderling [24] suggest ranking the variables using Driebeek penalties [43].

Defining slack variables Working with MIR inequalities requires properly defining slack variables in order to convert inequality constraints to equality constraints. One way of defining slack variables is as follows:

- Ranged constraints: Consider a constraint of the form $b_1 \leq ax \leq b_2$. Define an alternative constraint of the form $ax + s = b_2$ and where the lower and upper bounds of continuous variable s are 0 and $b_2 - b_1$ respectively. Observe that this new constraint is equivalent to the ranged constraint, because $ax - b_1 \geq 0$ is equivalent to $s \leq b_2 - b_1$, and $b_2 - ax \geq 0$ is equivalent to $s \geq 0$.

- Less-than-or-equal-to constraints: Consider a constraint of the form $ax \leq b$. Define an alternative constraint of the form $ax + s = b$, where continuous variable s has lower bound 0.

- Greater-than-or-equal-to constraints: Consider a constraint of the form $ax \geq b$. Define an alternative constraint of the form $ax - s = b$, where continuous variable s has lower bound 0.

When defining slack-variables as described above, the slack variables are treated in a very general way. They are defined as being continuous and only the most trivial bounds are defined for them. Consider a less-than-or-equal-to constraint $ax \leq b$. If all of the variables which participate in the constraint are of integer type, define t to be the smallest integer

such that ta and tb are integer (i.e., t is the least common multiple of the denominators of the coefficients and right-hand-side). It is easy to see that an alternative way of converting the above constraint to equality form consists in defining an integer non-negative variable s, and defining the equality $ax + (1/t)s = b$. This method naturally extends to ranged constraints and greater-than-or-equal-to constraints. In practice, it is desirable to do this extension only if the value t is not too large. For this, it is advisable to use a parameter MAX_SLACK_SCALE indicating the largest admissible value of t. In addition, by observing the bounds of the variables participating in a constraint, it is possible to derive stronger upper and lower bounds for the slack variables.

Selecting rows An MIR cut can be applied to any valid constraint of an MIP. However, there are far too many such constraints, and an important issue consists in selecting a reasonably sized subset. Several different approaches have been used for this. We describe the most important of these rules.

- Formulation rows. The simplest option consists in selecting rows from the original formulation rows as supplied by the user. When problems are poorly formulated, this can lead to important MIR constraints. When selecting formulation rows, it is helpful to limit the selection to rows which are not trivially facets (for example, clique constraints and bound constraints), and which involve involve fractional integer variables.

- Tableau rows. The most typical method consists in deriving MIR cuts from rows of the current tableau (see Gomory [56], [57], Balas, Ceria, Cornuéjols, and Natraj [17], and Bixby, Gu, Rothberg and Wunderling [24]). Tableau rows have the advantage that if the corresponding basic variable is fractional and integer, then it can be proven that a violated MIR cut exists (these are the well-known *Gomory Mixed-Integer Cuts*, or GMIs). Tableau rows are very dense, and are typically poorly conditioned, so it is not desirable to add MIR inequalities on all of them. Because of this, a possible selection scheme would be to select those tableau rows corresponding to the most "important" fractional integer variables, as described above. Here it is advisable to use

a parameter MAX_TABROW_CUTS, indicating the maximum number of tableau rows to consider. Bixby, Fenelon, Gu, Rothberg, and Wunderling [24] propose setting this value to 200. Another approach would be to generate all possible MIR cuts from the tableau rows, and in a subsequent stage select the "best" of these, by considering the violation, or euclidean distance of the point separated to the cut. This latter approach is sure to yield the best results, at the cost of taking a longer amount of time.

- Aggregated formulation rows. Several authors have pursued an approach consisting of aggregating formulation rows by means of taking linear and conic combinations of them, and then deriving an MIR inequality for the resulting valid constraint. Some very successful experiences have been those of Marchand and Wolsey [74], who propose a heuristic for aggregating rows, Caprara and Fischetti [29], who focus their attention on using multipliers 0 and 1/2 to combine rows, and Fischetti and Lodi [48], who solve an IP in order to determine which rows to aggregate, and afterwards apply the MIR cut.

- Extending the information derived from tableau rows. Andersen, Cornuéjols, and Li [3] exploit the fact that MIR cuts are in fact split-cuts (see Balas [13] and Cook, Kannan, and Schrijver [33]). Their "reduce-and-split" algorithm consists in adding together integer multiples of tableau-rows and applying the MIR template to this new row. As a counter-part to this idea, Balas and Perregaard [18] exploit the relationship between MIR cuts and lift-and-project cuts (see Balas, Ceria, and Cornuéjols [16]). Their algorithms provides a way of pivoting away from the current basis, in such a way as to obtain a new tableau row to which the MIR template can be applied.

- Delayed tableau rows. Instead of using the current tableau rows, an alternative approach is to consider the tableau rows derived in some previous iteration of the cut-separation algorithm. By doing this, one can effectively limit the Chvátal-Gomory rank (if we allow the notion to be extended to MIPs) of the derived constraints. Dash, Goycoolea, and Günluk [38], as well as Fischetti and Saturni [49] have used

these delayed rows in different ways.

A heuristic for improving numerical stability Consider a valid inequality of the form $ax \geq b$. We describe a procedure which seeks to slightly modify the coefficients and right-hand-side in such a way as to ensure no feasible solutions are lost, and such that the obtained cut is numerically "nicer".

The purpose of these relaxations is to improve the numerical stability of a constraint, weakening it a bit in hopes of repairing slight infeasibilities, and making the coefficients a bit nicer in order to help the LP solver. Most of these ideas originate from the MIR implementation of the Cut Generation Library (CGL), programmed by John Forest, in the COIN-OR [32] optimization package. Note that in Section 4.5.2.1 we will discuss other ways of dealing with numerical stability.

- (ROUNDING) If $|\hat{a_j}| < 10^{-12}$ set $a_j = \lfloor a_j \rceil$.

  Here it assumed that if the fractional part is very small, then the coefficient may as well be assumed to have been integer. Hence, the coefficient is rounded.

- (INTEGER-PADDING) For each integer variable x_j:

  - If $\hat{a_j} < 10^{-6}$ and $|\hat{a_j} u_j| < 10^{-6}$ then set $a_j = \lfloor a_j \rfloor$ and $b = b - \hat{a_j} u_j$.
  - Else, if $\hat{a_j} < 10^{-6}$ set $a_j = \lfloor a_j \rfloor + 10^{-6}$.
  - Else, if $(1 - \hat{a_j}) < 10^{-6}$ then set $a_j = \lceil a_j \rceil$.

  Here, the point is to deal with coefficients that are near-integer but not near enough that we can just round.

- (CONTINUOUS-PADDING) For each continuous variable x_j:

  - If $a_j u_j < 10^{-6}$ then $a_j = 0$ and $b = b - a_j u_j$.
  - Else, if $a_j < 10^{-6}$ set $a_j = 10^{-6}$.

  With continuous variables it makes little difference if the coefficients are integer or not (in terms of cut generation). However, if coefficients are zero things can change.

- (RATIO-TEST) If the ratio between the highest and lowest coefficient is more than 10000 then do not generate a cut from this row.

This procedure may be easily customized by replacing all values 10^{-6} by a parameter PADDING_THRESHOLD, all values 10^{-12} by INTEGER_THRESH.

Note that this procedure will convert an equality into an inequality. This means that this procedure can only be used with MIR[1].

Integer scaling One of the most unusual characteristics of MIR cuts is that not only does their effectiveness depend on the feasible region defined by the constraint from which they are derived, but also, on the *form* of the constraint from which they are derived. An interesting observation in this regard is made by Cornuéjols, Li, and Vanderbussche [36], who define the k-cuts as the MIR inequalities obtained after scaling the originating row by an integer k. As they show, these cuts capture information that could completely be missed by a normal MIR inequality. So far, no publications have appeared indicating how rows should be scaled in order to obtain the best inequalities. As a simple heuristic, we define parameters T_MIN and T_MAX, and scale rows by all integers in between these values.

4.5.2 Generating cuts

We classify these rows into two groups: Tableau rows corresponding to an optimal basis will be called Type-I rows, and all other rows will be said to be of Type-II. Observe that Type-II rows include original formulation rows, delayed tableau rows, and rows obtained by aggregation.

4.5.2.1 Type-I rows (current-basis tableau rows)

Finding violated inequalities: Consider the constraint:

$$x_0 + \sum_{j=1}^{n} a_j x_j = b$$

We will assume that this equality corresponds to a tableau row of an MIP, that x_o is a basic variable of integer type, and that $\hat{b} \neq 0$. If we assume that every variable x_j with $j = 0, \ldots, n$ satisfies $u_j = \infty$ and $l_j = 0$, then, as we have seen in Section 4.3, the corresponding MIR[2]

inequality has form:

$$\sum_{S} \frac{\hat{a_j}}{\hat{b}} x_j + \sum_{I \setminus S} \frac{(1-\hat{a_j})}{(1-\hat{b})} x_j + \frac{1}{\hat{b}} \sum_{C^+} a_j x_j + \frac{1}{(1-\hat{b})} \sum_{C^-} a_j x_j \geq 1.$$

where $S = \{j \in I : \hat{a_j} \leq \hat{b}\}$. Now, observe that since every variable x_j with $j = 1, \ldots, n$ is non-basic, then each of these variables has value zero. Further, since $a_0 = 1$ (where a_0 is coefficient of variable x_0 in the tableau row), it follows that $0 \in I$, and hence variable x_0 has coefficient zero in the MIR inequality. This implies that the constraint will necessarily be violated by the current basic solution, and that it will have slack equal to one.

This well-known observation indicates that in MIPs where variables only have non-negativity bounds, one can always obtain violated MIR inequalities from the tableau row when the solution is fractional. In fact, these obtained inequalities are the Gomory Mixed Integer cuts which are commonly employed by commercial mixed integer programming solvers.

Unfortunately, by further observing MIR^2, and considering the conditions under which the MIR^{2}was derived, it becomes evident that if the lower bound is non-zero, or the upper-bound is not infinite for some variables, the argument just used does not hold. In order to remedy this situation the c-MIR is employed.

Again, without loss of generality, assume that the basic variable is x_1, and that this is an integer variable having a fractional value. Define $U = \{j : x_j = u_j\}$ and $L = \{j : x_j = l_j\} \cup \{0\}$. Then, substitute variables as indicated in Section 4.4 to obtain constraint c-MIR^1. By using above argument, it is easy to see that if $r = b - \sum_{j \in U} a_j u_j - \sum_{j \in L} a_j l_j$ is fractional (that is, $\hat{r} \neq 0$), then the cut in the transformed space is violated. Thus, the corresponding c-MIR is also violated. We call this variable-complementation scheme COMPLEMENT_RULE_I.

Obtaining "safe" tableau rows: An important issue concerning the use of MIR inequalities based on tableau-rows is assuring that one obtains valid inequalities. In practice, this can be very difficult due to lack of numerical precision. There are two places where it is easy to make numerical mistakes: The first of these is in obtaining the actual tableau rows from which cuts will be derived. The second of these is in generating the MIR inequality.

In fact, it is natural to expect that tableau row generation can go wrong. After all, tableau rows are obtained by using the explicit inverse of a matrix, and it is well-known that taking matrix inverses is a numerically unstable operation. We now address the issue in more detail by considering an approach proposed by Sanjeeb Dash [39].

Consider an integer programming problem defined by the constraints $Ax = b$. Assume that the rows of A are the vectors $a^1, \ldots, a^m$ and let $b = (b_1, \ldots, b_m)$. Consider a basic optimal solution of the LP relaxation and a corresponding simplex dictionary. Without loss of generality, assume that variables $1, \ldots, m$ are basic, and let B be the basis matrix. Consider the k-th tableau row, $\pi^k x = \pi^k_o$, and the k-th row of the matrix B^{-1}, $\alpha^k = (\alpha^k_1, \alpha^k_2, \ldots, \alpha^k_m)$. We know that π and π_o can be obtained by taking a linear combination of the rows of A and b, using the multipliers in α^k. More precisely,

$$\pi^k = \sum_{i=1}^{m} \alpha^k_i a^i \quad \text{and} \quad \pi^k_o = \sum_{i=1}^{m} \alpha^k_i b_o. \tag{4.15}$$

Moreover, we know that tableau rows have the special form,

$$x_k + \sum_{j \geq m+1} \pi^k_j x_j = \pi^k_o.$$

That is, $\pi^k_k = 1$ and $\pi^k_i = 0 \; \forall i \in \{1, \ldots, m\} \setminus \{k\}$.

The first important observation to keep in mind when generating a tableau row, is that it is not strictly necessary to get the exact vector α^k in order to generate the k-th tableau row. In fact, let $\tilde{\alpha}^k$ is an approximation of α^k, and define:

$$\tilde{\pi}^k = \sum_{i=1}^{m} \tilde{\alpha}^k_i a^i \quad \text{and} \quad \tilde{\pi}^k_o = \sum_{i=1}^{m} \tilde{\alpha}^k_i b_o.$$

Though constraint $\tilde{\pi}^k x = \tilde{\pi}_o$ will only be an approximation of the k-th tableau row, it will still be a valid equality of the original system from which MIR inequalities can be derived. The only problem is that it might not have such a nice form as a tableau row. That is, the coefficients $\tilde{\pi}^k_k$ might not be exactly one, and the coefficients $\tilde{\pi}^k_i$ with $i \in \{1, \ldots, m\} \setminus \{k\}$ might not be exactly zero. Because of this, we call constraint $\tilde{\pi} x = \tilde{\pi}_o$ a *dirty tableau row*.

Second, even after recognizing that it may not be possible to get an exact vector α^k, and that instead, what we really work with is an approximation $\tilde{\alpha}^k$, there is still a risk of

errors in actually computing $\tilde{\pi}^k$ and $\tilde{\pi}^k_o$. In fact, the numbers represented in the vectors $\tilde{\alpha}^k$ may be numerically very bad. A vector $\tilde{\alpha}^k$ may have coefficients that are very large, and others very small (with several orders of magnitude difference). Furthermore, the numbers usually have a high degree of precision; that is, their floating-point representation may use a lot of decimal places. What this means is that when carrying out the multiplications and additions in order to obtain the vectors $\tilde{\pi}^k$ it is likely that many overflows will occur. Normally, in the "C "and "C++" programming languages these overflows are not reported to the user, and hence, the mistakes, though small, can exist without the user knowing. Thus leading to invalid cuts. Further, these errors greatly propagate as multiple rounds of cuts are generated. A concern is to somehow control these numerical imprecisions.

Several natural ways of dealing with these issues are possible:

- One very simple option would be to approximate the numbers α^k_i from $i = 1, \ldots, m$ by numerically tractable numbers $\tilde{\alpha_i}$ which would be "unlikely" to result in many overflows. This approximation may be a diaphontine approximation, or, simply the result of truncating/rounding the numbers to a certain decimal place. Unfortunately, this method is very heuristic in nature and it is hard to actually be sure when it might lead to invalid equalities.

- Another option might consist in using an exact arithmetic software package (or a higher precision floating point arithmetic package), for example GMP [62], during this intermediate stage. Most mixed integer programming solvers work in finite precision floating point arithmetic. Hence, this would require converting the approximation of the row α^k provided by the solver to an exact arithmetic data structure, converting the original rows of the system to this same data structure, and carrying out the operations with the enhanced precision structures. Then, after deriving the MIR inequality in this numerically safe environment, it could be converted back to finite precision floating point arithmetic. This final conversion, of course, would also have to be very carefully done. For instance, suppose that in the infinite precision environment we obtain an inequality $\beta x \geq \beta_o$. The numbers β_j and β_o might not be representable in the finite

precision environment used by the solver. Thus, we would have to approximate every coefficient β_j with a representable coefficient β_j^+ such that $\beta_j^+ \geq \beta_j$, and the coefficient β_o with a representable coefficient β_o^- such that $\beta_o^- \leq \beta_o$. Since the inequality $\beta^+ x \geq \beta^-$ is a relaxation of the actual cut obtained, and since it is representable in the less precise arithmetic, it could be added to the MIP solver with the certainty that it is valid.

- Yet another method might be to change the way the finite precision floating point arithmetic is performed. The way finite precision floating point arithmetic is typically carried out in modern computers is that the result of arithmetic operations is rounded to nearest representable number. This rounding could involve either rounding up or rounding down. Modern computer processors, however, are typically IEEE 754-1985 compliant [68] (see also Goldberg [55]). This means that the way they deal with unrepresentable results to arithmetic operations can be changed, so that instead of approximating to the nearest representable number, results could instead be approximated up to the nearest representable number, or down. This change in rounding modes can be used to generate valid MIR inequalities. In fact, let $\tilde{\alpha}^k$ be defined as before. Set the rounding mode to "up" and compute $\pi^{k,+} = \sum_{i=1}^{m} \tilde{\alpha}_i^k a^i$. Set the rounding mode to "down" and compute $\pi_o^{k,-} = \sum_{i=1}^{m} \tilde{\alpha}_i^k b_o$. We now have a valid inequality of form $\pi^{k,+} x \geq \pi_o^{k,-}$. Since we no longer have an equality, but rather, we have an inequality, we have to use the simple form of the MIR (i.e., MIR^1) to obtain a valid constraint from this inequality. Again, by being careful to round "up" or "down" at the appropriate times, one can generate a valid inequality which is sure to be safe. It is important to note that when applying this procedure one can obtain cuts which are weaker than the previous "safe" procedure. However, since floating point operations are much faster than using exact arithmetic operations, further exploring this idea might be of great practical use in generating safe MIR inequalities for more than just one or two rounds as is typically done in commercial MIP solvers.

4.5.2.2 Type II rows: non-tableau rows.

As observed in Section 4.5.2.1, when deriving MIR inequalities from optimal basis tableau rows it is easy to find violated constraints. This is not the case when deriving MIR inequalities from other valid inequalities. Hence, the techniques which must be applied are slightly different.

The main observation which is exploited when deriving MIR constraints from optimal basis tableau rows is that these have exactly of one basic variable, and all other variables are non-basic. This allowed us to use a very simple variable complementation scheme which almost always assured that generated constraints would be violated. In the case of more general rows, there may be several fractional basic variables.

This suggests that variable complementation should be done in a more sophisticated manner. Before, when complementing non-basic variables we were assured that in the transformed space these would take value zero. This can no longer be achieved by complementation if variables are not at their bounds. An alternative rule, which we call COMPLEMENT_RULE_II, consists in complementing variables so that in the transformed space they are as close to zero as possible. For this, define the sets $U = \{j : |x_j - u_j| < |x_j - l_j|\}$ and $L = \{j : |x_j - l_j| \leq |x_j - u_j|\}$ and then apply c-MIR template (see Marchand and Wolsey [74]).

Another important condition which was exploited when deriving MIR constraints was the fact that the right hand side was fractional. This was always the case in tableau rows corresponding to fractional basic variables of integer type, since the right hand side corresponded to the value held by the variable. However, in formulation rows, for example, the right-hand side is often defined to take integer values. Also, observe that in tableau rows, the coefficient of the basic variable always has a coefficient of one, thus leading to its having a zero coefficient in the derived cut. Heuristically speaking, this helps achieve the violation of the cut, as all the other variables are at zero after complementation. Considering these two observations, it seems a reasonable heuristic to scale rows from which we wish to derive inequalities so that the coefficient of basic variables becomes integer and so that

the right hand side becomes fractional. The heuristic we employ for this uses a parameter MAX_COEFFICIENT_SCALE to generate a set A of coefficients by which to scale a generic inequality $ax \leq b$ in order to generate c-MIR constraints.

1. Let $A = \emptyset$.
2. Select the most important variable x_i participating in the constraint which has not been selected so far, and add a_i to A.
3. If $|A|$ =MAX_COEFFICIENT_SCALE, or there are no further important variables, go to Step 4. Else, go back to Step 3.
4. For each $a \in A$, scale the cut by multiplying throughout by $1/a$, and apply the c-MIR template.

4.5.3 After generating cuts

After the cut has been generated, it should be decided if the cut will be added to the LP. For this the following tests can be performed.

- The Violation Test: If the cut is not violated by more than 10^{-4}, do not add the cut. This value can be customized by means of a MIN_VIOLATION parameter.
- The Ratio Test: If the ratio between the highest and lowest coefficient is more than 100000 then do not add the cut. This value can be customized by means of a MAX_RATIO parameter.
- The Density Test: If the cut has too many non-zeroes (more than 500), do not add the cut. This value can be customized by means of a MAX_NON_ZEROES parameter.

In addition, after a cut has been generated, an attempt can be made to "improve" the coefficients of the cut. One way of doing this could be by applying the numerical stability heuristic. Another, used by the COIN-OR software [32], consists in doing the following when all of the lower bounds are non-negative:

- Estimate each coefficient (and right-hand side) of the constraint by a rational number with a bounded denominator, which is greater than or equal to it.

- Find the least common multiple of the denominators of these approximations, and multiply through to obtain a cut consisting of only integers.

4.6 Group cuts, knapsack cuts, and the MIR

4.6.1 Group cuts and the MIR

In this section we briefly introduce the work of Dash and Günluk [40]. For a more thorough description of the methodology and background, the reader is advised to consult Dash and Günluk [40], [41], and [42], in addition to Gomory [58], and Gomory and Johnson [59] from which we summarize below. Note that throughout this section we employ the notation of Dash and Günluk.

For integers a, b, c say that $a \equiv b \mod c$, when $(a - b)$ is divisible by c.

Consider $r, k \in \mathbb{Z}$ where $k > r > 0$. Gomory [58] defined the *master cyclic group polyhedron* of size k and right-hand side r as follows:

$$P(k,r) = \text{conv}\left\{ w \in \mathbb{Z}^{k-1} : \sum_{i=1}^{k-1} i w_i \equiv r \mod k,\ w \geq 0 \right\}. \tag{4.16}$$

Gomory completely characterized the non-trivial facets (i.e., not a non-negativity constraint) of $P(k,r)$ by means of the following theorem:

Theorem 4.1 (Gomory [58])**.** If $r \neq 0$, then $\sum_{j=1}^{k-1} \eta_j w_j \geq 1$ is a non-trivial facet of $P(k,r)$ if and only if $\eta = (\eta_j)$ is an extreme point of the polyhedron:

$$\eta_i + \eta_j \geq \eta_{(i+j) \bmod k} \qquad \forall i, j \in \{1, \ldots, k-1\} \tag{4.17}$$

$$\eta_i + \eta_j = \eta_r \qquad \forall i, j \text{ such that } r = (i+j) \mod k \tag{4.18}$$

$$\eta_j \geq 0 \qquad \forall j \in \{1, \ldots, k-1\} \tag{4.19}$$

$$\eta_r = 1. \tag{4.20}$$

Observe that $w \in P(k,r)$ if and only if there exists $z \in \mathbb{Z}$ such that $(\sum_{i=1}^{k-1} i w_i - r) = zk$.

Thus, dividing by k and re-arranging the terms we get

$$P(k,r) = \text{conv}\left\{ w \in \mathbb{Z}^{k-1} : z + \sum_{i=1}^{k-1} \frac{i}{k} w_i = r,\ w \geq 0,\ z \in \mathbb{Z} \right\}. \tag{4.21}$$

Now consider the slightly more general mixed-integer set

$$G(k,r) = \text{conv}\left\{ v_1, v_2 \in \mathbb{R},\ w \in \mathbb{Z}^{k-1} : v_1 - v_2 + z + \sum_{i=1}^{k-1} \frac{i}{k} w_i = r,\ w, v_1, v_2 \geq 0,\ z \in \mathbb{Z} \right\}.$$

The set $G(k,r)$ is called the *mixed-integer extension of the cyclic group polyhedron*. Gomory and Johnson [59] showed that the facets of $P(k,r)$ and $G(k,r)$ are related in the following way.

Theorem 4.2 (Gomory and Johnson [59])**.** $\sum_{i=1}^{k-1} \eta_i w_i \geq 1$ is a facet of $P(k,r)$ if and only if $k\eta_1 v_1 + k\eta_{k-1} v_2 + \sum_{i=1}^{k-1} \eta_i w_i \geq 1$ is a facet of $G(k,r)$.

This means that given a non-negative vector $w^* \in \mathbb{Z}^{k-1}$ and non-negative values $v_1^*, v_2^* \in \mathbb{Q}$ it is possible to obtain a non-trivial facet maximizing the violation at (w^*, v_1^*, v_2^*) by solving the problem:

$$\min\{(kv_1^*)\eta_1 + (kv_2^*)\eta_{k-1} + \sum_{i=1}^{k-1} w_i^* \eta_i : \eta \text{ satisfies (4.17)} - \text{(4.20)}\}. \tag{4.22}$$

Let us now see how the work of Gomory and Johnson can be theoretically put to use in order to obtain cutting planes for general MIPs.

Consider a matrix $A \in \mathbb{Q}^{m \times n}$, a vector $d \in \mathbb{Q}^m$, and disjoint sets I, C such that $I \cup C = \{1, \ldots, n\}$. Assume A has full rank, and $A = [B, N]$ where B is a feasible basis. Let,

$$P^M = \{x \in \mathbb{Q} : Ax = d,\ x \geq 0,\ x_i \in \mathbb{Z}\ \forall i \in I\}.$$

Now consider a system

$$Q = \{x \in \mathbb{Q} : \sum_{i \in I} a_i x_i + \sum_{i \in C} a_i x_i = b,\ x \geq 0,\ x_i \in \mathbb{Z},\ \forall i \in I\}.$$

We will assume that system Q is implied by P^M. That is, there exists $\lambda \in \mathbb{Q}^m$ such that $a = \lambda^t A$ and $b = \lambda^t d$. Hence, $P^M \subseteq Q$, and so, any inequality valid for Q will be valid for P^M. Observe that by re-arranging terms we have

$$Q = \{x \in \mathbb{Q} : (\sum_{i \in I} \lfloor a_i \rfloor x_i - \lfloor b \rfloor) + \sum_{i \in C} a_i x_i + \sum_{i \in I} \hat{a}_i x_i = b,\ x \geq 0,\ x_i \in \mathbb{Z},\ \forall i \in I\}.$$

Now assume that $\hat{a}_i$ and $\hat{b}$ are all multiples of $1/k$ for some integer k. If k is the smallest such integer we say that k is the *scaling factor* of Q. Let $C^+ = \{i \in C : a_i > 0\}$, $C^- = \{i \in C : a_i < 0\}$, and $I_j = \{j \in I : \hat{a}_i = j/k\}$. We can correspond system Q to system $G(k,r)$ by means of the following mapping:

$$\begin{aligned} w_j &= \sum_{i \in I_j} x_i \qquad \forall j = 1, \ldots, k-1. \\ v_1 &= \sum_{i \in C^+} a_i x_i. \\ v_2 &= -\sum_{i \in C^-} a_i x_i. \\ z &= \sum_{i \in I} \lfloor a_i \rfloor x_i - \lfloor b \rfloor. \end{aligned}$$

Note that if any $I_j = \emptyset$ then we set $w_j = 0$.

Thus, if $k\eta_1 v_1 + k\eta_{k-1} v_2 + \sum_{i=1}^{k-1} \eta_i w_i \geq 1$ is a valid inequality for $G(k,r)$, then,

$$k\eta_1 (\sum_{i \in C^+} a_i x_i) - k\eta_{k-1} (\sum_{i \in C^-} a_i x_i) + \sum_{j=1}^{k-1} \sum_{i \in I_j} \eta_j x_i \geq 1$$

defines a valid inequality for Q. Dash and Günluk [40] call these inequalities *group cuts* for Q. What the preceding discussion highlights is that given a fractional vector $x^* \in \mathbb{Q}^n$ such that $x^* \geq 0$ and $a^t x^* = b$, we can apply the mapping $x^* \longrightarrow (w^*, v_1^*, v_2^*, z^*)$, solve problem (4.22), find (if any) a maximally violated inequality for $G(k,r)$ in the transformed space, and then convert it, obtaining a maximally violated (if any) group cut. Note that in particular, the MIR is a group cut.

While in theory this would seem a promising way of deriving cuts for Q, in practice it is hard to do so due to the fact that problem (4.22) can involve a very large number of variables and constraints. In addition, Gomory and Johnson [59] proved that $G(k,r)$ admits an exponential number of extreme points (as a function of n). Thus, the original idea, as conceived by Gomory and Johnson, was to identify the most "important" extreme points of $G(k,r)$, and use *these* to derive valid inequalities for Q.

Following up on this idea, Gomory, Johnson and Evans [61] describe a computational approach to identify these "important" facets. Basically, for small values of k, what they do is a *shooting experiment*, consisting in randomly choosing directions d and identifying

the first facet encountered along the ray λd. Sampling many times, they found (for small k) that only a small number of facets were "important". They also found that, by far, the most important facets correspond, when transformed, to the MIR inequality.

Evans [46], and Dash and Günluk [42], [41] use the information obtained from these experiments to identify the "second-most-important" group cuts after the MIR, and perform computational tests in order to determine the impact of these on general MIPs (see Evans [46] and Dash, Goycoolea, and Günluk [40]). However, on a large number of problems it was found that these other group cuts had very little effect in further improving the performance of the MIR inequalities. This brings us to the main point we wish to address in this chapter.

Dash and Günluk [40] set up a computational experiment to answer the following question: After adding the MIR cuts to all of the tableau rows in a fractional solution and resolving, are there any additional violated group cuts for these same tableau rows? Note that they did not seek to actually identify those potentially violated cuts. As remarked earlier identifying cuts is an extremely demanding computational task. Instead, Dash and Günluk simply sought to know if there *existed* a violated group cut after the MIRs were added. For this, instead of actually solving (4.22) they used a number of clever bounds to determine if the optimal value was positive or negative.

Their computational experiment, thus, was as follows. They collected a large set of test instances including problems from MIPLIB 3.0 [22], MIPLIB 2003 [1], MIPLPLib [82] and a set of instances collected by Fischetti and Lodi [47]. They then proceeded in five steps. First, they solved the LP relaxation of each problem. Second, for each tableau row corresponding to a fractional integer variable they added the corresponding MIR inequality. Third, they stored each tableau row they used to generate a cut. Fourth, they resolved the LP. Fifth, they went back to the stored cuts and tested if there were any violated group cuts. Their results were surprising. Of the 156 problems they tested, they found that in 55 (i.e., 35%), no group cuts were violated after adding the MIRs and resolving.

As a final remark concerning the computational tests of Dash and Günluk [40], note the

following: Instead of using the system Q as previously described, they used a system,

$$Q_B = \{x \in \mathbb{Q} : \sum_{i \in I} a_i x_i + \sum_{i \in C} a_i x_i = b,\ l \leq x \leq u,\ x_i \in \mathbb{Z},\ \forall i \in I\}$$

where a and b are again assumed implied by a system $Ax = b$, but where the additional use of lower and upper bounds is considered ($u_i \in \mathbb{Q} \cup \{+\infty\}$ and $l_i \in \mathbb{Q}$ for $i = 1, \ldots, n$). Given that the theory developed in this section does not account for such use of bounds, they used COMPLEMENT_RULE_II, as described earlier, in order to convert the system to one in which the lower bound is zero, and then they relax the upper bound. It is important to observe that the complementation rule provides a single instance of type Q. However, there are up to 2^n different possible ways in which the bounds could have been complemented. Regardless, their result shows that once having chosen such a complementation scheme, the MIRs suffice to push the fractional solution inside the intersection of the group relaxation polyhedra.

4.6.2 Knapsack cuts and the MIR

In this section we follow up on the experiments of Dash and Günluk [40]. Instead of using a complementation rule and relaxing bounds, we work directly with system Q_B, calling an inequality which is valid for Q_B a *knapsack cut*. Observe that group cuts are also knapsack cuts. Further, all of the group cuts, for all of the possible variable complementation/relaxation schemes are also knapsack cuts; thus the class of knapsack cuts is considerably larger than that of group cuts. The question we seek to answer, given the result of Dash and Günluk, is as follows: After adding the MIR cuts to all of the tableau rows in a fractional solution and resolving, are there any additional violated knapsack cuts for these same tableau rows?

The approach we follow is very similar to that used by Applegate, Bixby, Chvátal and Cook [7] - henceforth ABCC - and Espinoza [45] in the context of local cuts, and that of Boyd [25], [26] in the context of Fenchel cuts. Technically, the definition of Fenchel cuts [25] and knapsack cuts is almost identical. Because Boyd [25], [26] uses a very different methodology to separate Fenchel cuts than we do to separate knapsack cuts, we will persist with the name distinction.

Consider a fractional value x^* in the LP relaxation of Q_B. Observe that if there exists a knapsack cut which is violated by x^*, because of re-scaling, we may assume that it is violated by one. Thus, there exists a violated knapsack cut if and only if the following polyhedron is non-empty (see ABCC [7]):

$$\begin{aligned} \pi x^* &= \pi_o + 1 \\ \pi x^k &\leq \pi_i \qquad \forall k = 1, \ldots, q \\ \pi r^k &\leq 0 \qquad \forall k = 1, \ldots, t \end{aligned} \tag{4.23}$$

where $\{x_k\}_{k=1}^q$ corresponds to the set of all extreme points and $\{\mu_k\}_{k=1}^t$ the set of all extreme rays of Q_B, and any feasible solution π corresponds to such a cut. Observe that if we consider (4.23) as a minimization problem with an objective function value of all zeroes, testing infeasibility is equivalent to testing if the dual of the optimization problem is unbounded. This dual problem consists in solving

$$\begin{aligned} \max \quad & s \\ \text{s.t.} \quad & \\ & sx^* - \textstyle\sum_{k=1}^q \lambda_k x^k - \sum_{k=1}^t \mu_k r^k = 0 \qquad \text{(C1)} \\ & -s + \textstyle\sum_{k=1}^q \lambda_k = 0 \qquad \text{(C2)} \\ & \lambda \geq 0, \mu \geq 0. \end{aligned} \tag{4.24}$$

Observe that if (4.23), or equivalently (4.24), is solved to optimality, and further, the problem is feasible, then the actual solution yields a knapsack cut π which is violated, and which in turn could be added to the LP. In fact, this is (essentially) what Boyd [25], [26] does, and the cuts he obtains are what he calls "Fenchel Cuts". It is important to note, however, that Boyd limits his studies to systems Q_R which are exactly a row of the original LP formulation. He does not consider tableau rows nor rows obtained in any other way (such as aggregation, etc.). ABCC [7] also do something similar, but in the context of the Traveling Salesman Problem. They use a framework similar to the one described above, where Q_R, instead of originating from a single row, originates from a multiple row system consisting of several variables. Espinoza [45] extends this work on local cuts to general MIPs, and incorporates exact arithmetic into the computations. The systems Q_R used

by Espinoza [45] also originate from tableau rows, but instead, by using a combination of projections and lifting, Espinoza focuses his attention on systems with several rows and a small number of variables. It is important to note that in these last two studies, a slightly different formulation is used for (4.23), in which the objective function is modified so as to guide the cut selection to prefer cuts which are "deeper" (i.e., such that the distance from the point separated to the cutting hyperplane is maximized).

Solving problem (4.24) or (4.23) by explicitly formulating it, however, is extremely difficult. This is because it is not clear how all of the extreme points and rays could be generated a-priori, and because, even if there were some way of generating them, the problem would be far too large for modern LP solvers to tackle. We now explain how this problem can be successfully solved in practice.

First note that (4.24) is unbounded if and only if $x^* \in \text{conv}(Q_B)$. For this, observe that (4.24) is unbounded if and only if there exists a feasible solution such that $s \neq 0$. In fact, if such a solution exists, it can be scaled, thus increasing the objective function arbitrarily.

Second, note that if a solution is feasible for $s \neq 0$, again by scaling, a solution must be feasible for $s = 1$. However, any feasible λ, μ which is feasible for $s = 1$ is such that the conditions of Minkowski's Theorem hold (see Nemhauser and Wolsey [90]), or equivalently, $x^* \in \text{conv}(Q_B)$.

Now, assume that (4.24) is feasible and consider a solution (λ, μ) such that $s = 1$. It is easy to see that if for some variable x_i we have that $x_i^* = l_i$, then every k such that $\lambda_k \neq 0$ will be such hat $x_i^k = l_i$. Analogously, if for some variable x_i we have that $x_i^* = u_i$, then every k such that $\lambda_k \neq 0$ will be such hat $x_i^k = u_i$. This allows us to eliminate a great number of variables. In fact, if $x_i^* = l_i$ we need not consider the columns for λ_k, where $x_i^k \neq l_i$. Likewise, if $x_i^* = u_i$ we need not consider the columns for λ_k, where $x_i^k \neq u_i$.

The way the resulting problem is solved is by column generation (see Nemhauser and Wolsey [90] for an introduction to this method and examples). Starting with a few trivial columns, at each iteration we solve (4.24) with a restricted number of columns. If the problem turns out to be unbounded in the restricted set, we conclude that the entire problem is unbounded and stop. If not, we collect the dual values associated to the rows of (4.24)

and solve the *oracle problem* to determine if by adding any new columns to the problem the simplex algorithm will pivot. Observe that the problem has one dual variable associated to each variable in Q_B corresponding to the constraints (C1), we identify these by $\pi \in \mathbb{Q}^n$, and one additional dual variable corresponding to constraint (C2), which we identify by π_0. The oracle problem consists in determining if there exists a column λ_k such that $\pi x^k > \pi_0$ or a column μ_k such that $\pi r^k > 0$. More formally, the oracle problem consists in performing the following steps.

First, solve the problem:

$$\max\{\pi x : x \in Q_B\}. \tag{4.25}$$

Second, if problem (4.25) is bounded, then the optimal solution x' will correspond to an extreme point of Q_B. If $\pi x' > \pi_0$ we add the corresponding column x' to the restricted version of (4.24). If $\pi x' \leq \pi_0$ we conclude that there are no columns which need be added to the restricted version of (4.24). In fact, this clearly shows that there are no other extreme points x'' of Q_B such that $\pi x'' > \pi_0$. In addition, because the problem is bounded, it also shows there are no extreme rays r' of Q_B such that $\pi r' > 0$. So, in this latter case, we conclude that the current solution (μ, λ, s) is optimal for (4.24), that $x^* \in \text{conv}(Q_B)$, and hence, that there are no violated knapsack cuts separating x^*. Third, if problem (4.25) is unbounded, then we find a ray r' indicating the unbounded direction. This will be an extreme ray of Q_B satisfying $\pi r' > 0$, so we add it to (4.24). Note that we can assume the problem is always feasible. Were this not the case, then the original MIP which we are solving would not be infeasible!

If, after solving the oracle problem, we have not concluded that $x^* \in \text{conv}(Q_B)$, then, we resolve the new restricted (4.24) and iterate.

Note that problem (4.25) is in fact a Mixed Integer Knapsack Problem, and can be solved by the methodology described in Chapter 3. Also, the fact that we are only interested in solutions having value greater than π_0 means we can use this information in the branch and bound tree for early pruning of solutions. Finally, note that because we can restrict our attention to only some of the columns of (4.24) when some variables are at their bounds, we can solve a simpler version of Q_B. Let $I^U = \{i = 1, \ldots, n : x_i^* = u_i\}$, and $I^L = \{i =$

$1, \ldots, n : x_i^* = l_i\}$. When solving Q_B we can fix all of the variables in $I^U \cup I^L$ to their respective bounds before solving. Because x^* will correspond to a basic feasible solution of an LP, it will normally be the case that many variables will be at some bound. Hence this a-priori bound fixing can greatly reduce the number of variables to consider in the oracle.

4.7 Computations

In this section, computational results obtained after testing some ideas mentioned in this chapter are described. All of the implementations were written in the "C" and "C++" programming languages, and compiled with the gcc and g++ compilers, version 3.2, on a Linux operating system, version 2.4.27. The computers used to run the implementations were all Intel Xeon dual-processor machines, each with 2GB of RAM, running at 2.66GHz per processor. All of the algorithms were implemented with templates, thus allowing for them to use different types of numerical precision. Tests are conducted using two distinct types of numerical precision: "doubles" and "mpq". The first of these corresponds to the standard 32 bit implementation of floating point arithmetic employed in the "C" and "C++" programming languages. The second of these corresponds to using the exact rational arithmetic library GMP [62], which can be used to represent and perform arithmetic with any rational number.

We implemented two main algorithms. The first of these is a tool for generating MIR inequalities. The implementation of this tool follows the guidelines described in the chapter. We test the performance of this separation algorithm on the 65 instances of MIPLIB 3.0 (see Bixby, Ceria, McZeal and Savelsbergh [22]) by generating MIR inequalities over tableau rows and over formulation rows. We assess the impact of several different parameters discussed in the chapter for the generation of violated MIR inequalities. The second algorithm implemented is a tool for detecting the existence of violated knapsack cuts. Using the KBB algorithm described in Chapter 3 as an oracle, and following the guidelines given in this chapter, we implemented a testing algorithm which works both with "doubles" and "mpq" arithmetic. The tests performed aim to determine whether or not there are violated knapsack cuts after the completion of the MIR separation algorithm over tableau rows.

Table 4.1: Tableau-MIRs: Settings of separation algorithm.

Form of the inequality	Standard (MIR[1])
Integer scaling	No scaling
Cut elimination	Whenever a cut has positive slack
Arithmetic	Doubles (32 bit floating point representation)
Number of rounds	10
Pre-selection rule	No pre-selection
Complementation rule	Complementation rule I

It is important to note that when implementing these algorithms, running time was not a priority. Implementing a generic MIR separation tool which can work with many different settings and which can operate with different arithmetics is not very compatible with implementing a quick tool with very specific parameters to be used in practice. The same holds for the violated knapsack cut test. Because of this, no running times are reported, though it is our next goal to develop a fast tool which can be used in more practical settings.

In order to solve linear programming problems we use the CPLEX LP solver, version 9.0. We use all the default settings of CPLEX for these tests.

In order to present information we will use cumulative histogram curves. In these curves, gap will be plotted in the x-axis, and number of instances in the y-axis. In this way, a point (g_o, n_o) on a curve will mean that in n_o of the problem instances a gap of g_o or more has been closed by the corresponding algorithm.

4.7.1 Tableau rows

We begin by analyzing the performance of the MIR separation algorithm on tableau rows. The set-up is as follows: after solving the LP relaxation of each problem, we proceed to add MIR inequalities derived from the tableau rows, and repeat ten times. The goal is to evaluate the performance of the MIR separation algorithm, and to determine how to set some of the parameters discussed in this chapter. After much testing, we concluded that the following parameters worked very well:

In what remains of this section, we will justify the use of these parameters, and analyze the performance of the algorithm.

Convergence of the MIR separation procedure We begin by observing the impact of the MIR inequalities as a function of the number of rounds. This effect is illustrated in Figure 4.2, where the gap closed after 1,2,4 and 8 rounds is plotted.

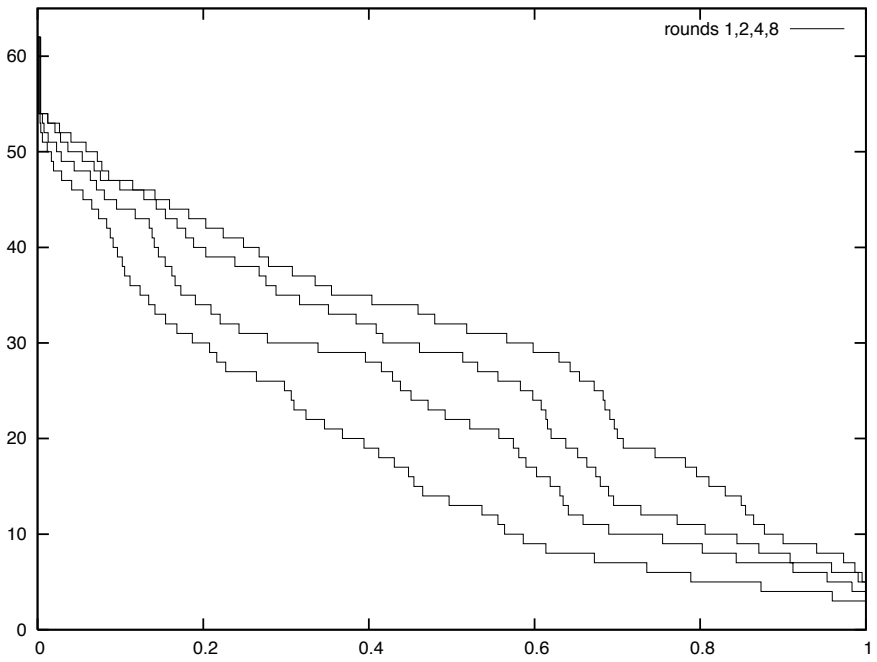


Figure 4.2: Convergence of tableau-MIRs after eight rounds.

As can be seen in the graph, the gap-closure improvements tend to decrease logarithmically with the number of rounds. However, the improvement is steady. For instance, observe how the number of problems solved to within 60% of the gap evolves with the number of rounds. It can be seen that after one round, the number of such instances is just under 10. However, after eight rounds this number goes up to about 30. Considering that the sample set is of 62 instances, this is quite an improvement.

Unfortunately, it is questionable whether or not the cuts added after 8 rounds are valid. Though we have not explicitly tested if they are, it is agreed that they often are not. Most commercial IP solvers (including CPLEX) add at most two rounds of MIR inequalities derived from tableau rows (Gomory Mixed Integer inequalities) to avoid the risk of inadvertently cutting off optimal problem solutions. However, as Figure 4.2 shows, this means that there is a very large potential gap improvement which is being lost.

Use of rational arithmetic A natural question which arises is whether or not being more careful with regards to the validity of the added inequality results in a gap-closure improvement which is worse. After all, one might argue that the high improvements in gap-closure observed in Figure 4.2 might mostly be due to invalid inequalities. In order to tackle this question, we implemented a "safe" version of the MIR separation heuristic which works by using exact arithmetic in order to generate the inequalities (See Section 4.5.2.1). Using the same parameters as those used in the previous experiment, we compare the effect of using exact arithmetic (mpq) versus the effect of using 32 bit floating point arithmetic (dbl) on the same set of instances. The results are shown in Figure 4.3 for the first round, and in Figure 4.4 for the tenth round.

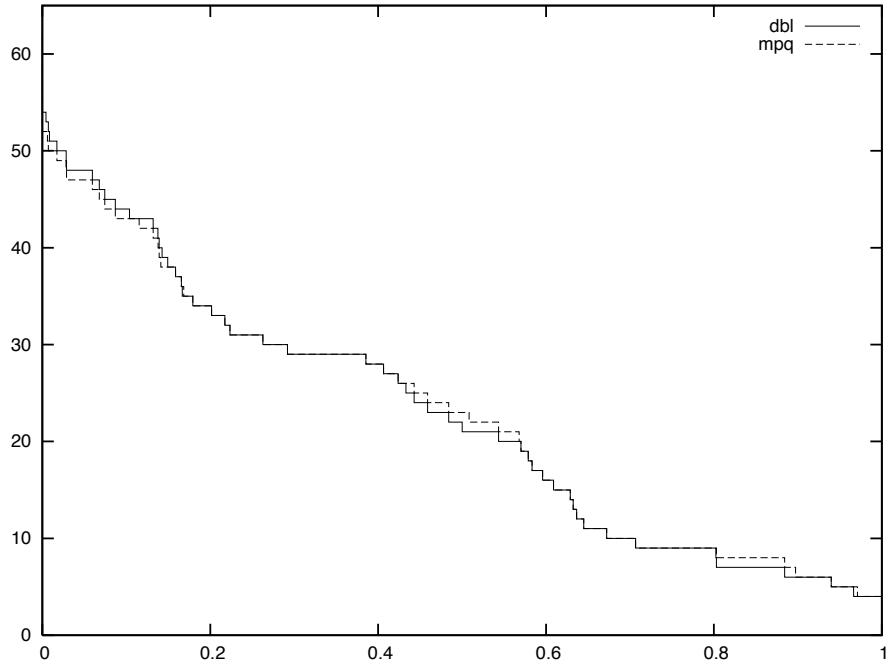


Figure 4.3: Effect of using exact arithmetic in order to generate safe tableau-MIRs (1st round).

As can be observed from Figure 4.3 and Figure 4.4, the impact on gap improvement had by using exact arithmetic to ensure the generation of valid cuts is almost none. That is, the bound improvements are nearly the same, and it is difficult to say if any of the two arithmetics actually performs better than the other.

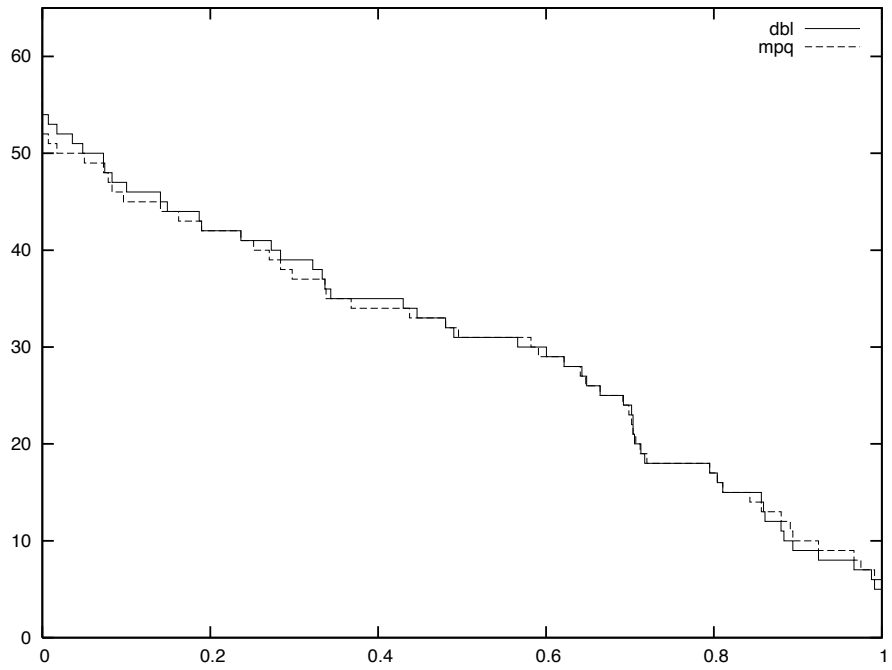


Figure 4.4: Effect of using exact arithmetic in order to generate safe tableau-MIRs (10th round).

This suggests that further efforts to generate safe MIRs derived from tableau rows after the initial rounds might well be a fruitful investment. However, it is important to note that the exact arithmetic routines implemented are very slow and inadequate for practical problems. In fact, tableau rows tend to be very dense - especially after a few rounds of the MIR procedure - hence the number of additions and multiplications required is very large. Furthermore, operations in exact arithmetic take 100 to 1000 times more time than they do with 32 bit floating point arithmetic. Instead of using exact arithmetic, an interesting direction to pursue would consist in changing the rounding modes used by the floating point arithmetic (see Section 4.5.2.1). Other ways of using generating safe MIR inequalities quickly could be of great value in solving difficult problems.

Form of the MIR As mentioned in Section 4.3, we have chosen to use the Simple-Form (MIR^1) of the MIR inequality. So far, all tests have been conducted using inequalities in this form. However, it is important to note that despite the fact that these two inequalities are equivalent, their use might result in different gap-improvements after repeated iterations

of the procedure. In fact, different inequalities might lead the underlying simplex algorithm to pivot differently, thus leading to different tableau rows from which subsequent cuts are added. In order to make sure that using MIR^1instead of $\text{MIR}^2$was not a bad decision, we re-solved all of the instances using MIR^2(the Normalized-Form). The comparison is illustrated in Figure 4.7.1 for the first round, and Figure 4.7.1 for the tenth round.

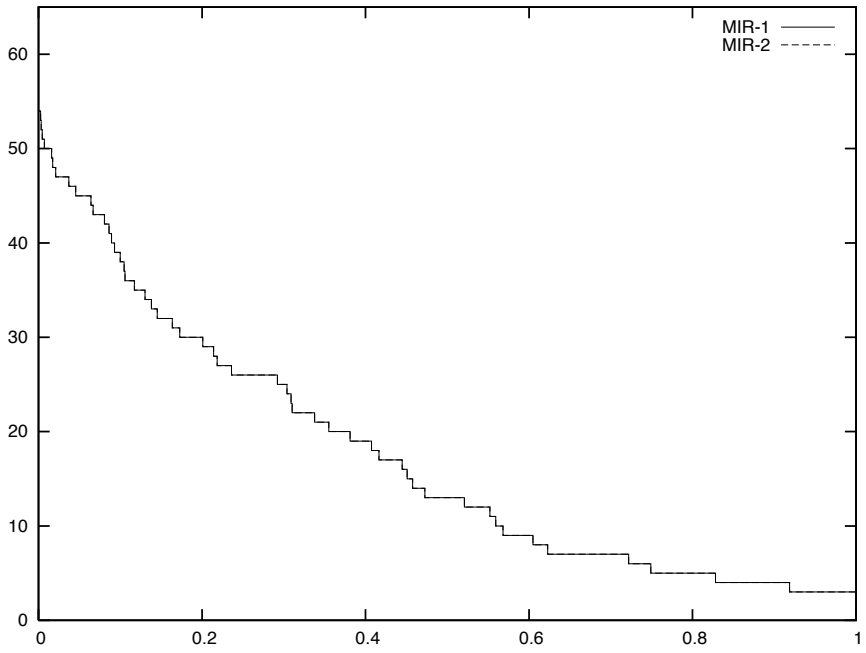


Figure 4.5: Form of the tableau-MIR: Round 1.

As should be expected, after a single round of adding MIR inequalities, both $\text{MIR}^1$and MIR^2 behave exactly alike. After the 10th round, however, the same thing cannot be said. In fact, it is clear that some problems ended up with better gaps using MIR^1, while others ended up with a better gap using MIR^2. However, the overall impact is very comparable and it can be seen that both forms of the MIR perform equivalently well in practice.

Elimination of inactive MIR inequalities In order to speed up the time taken to perform the computational tests, we decided to eliminate from the system any MIR inequality having positive slack. In order to make sure that this did not have a detrimental effect on the performance of the algorithm, we also solved all of the instances once using the default settings, except for the fact that cuts were never removed from the system. The comparison

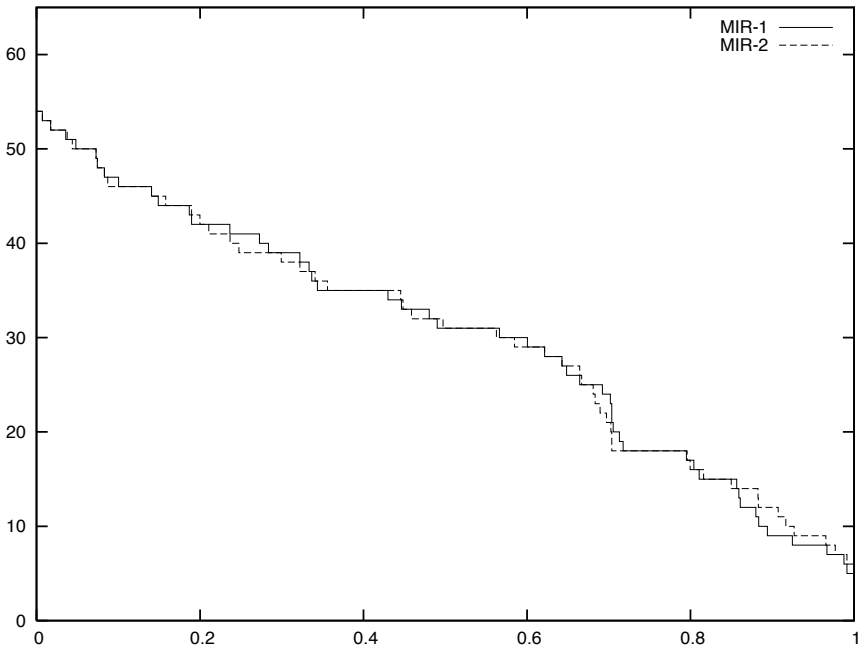


Figure 4.6: Form of the tableau-MIR: Round 10.

of the runs with and without cut elimination is summarized in Figure 4.7.

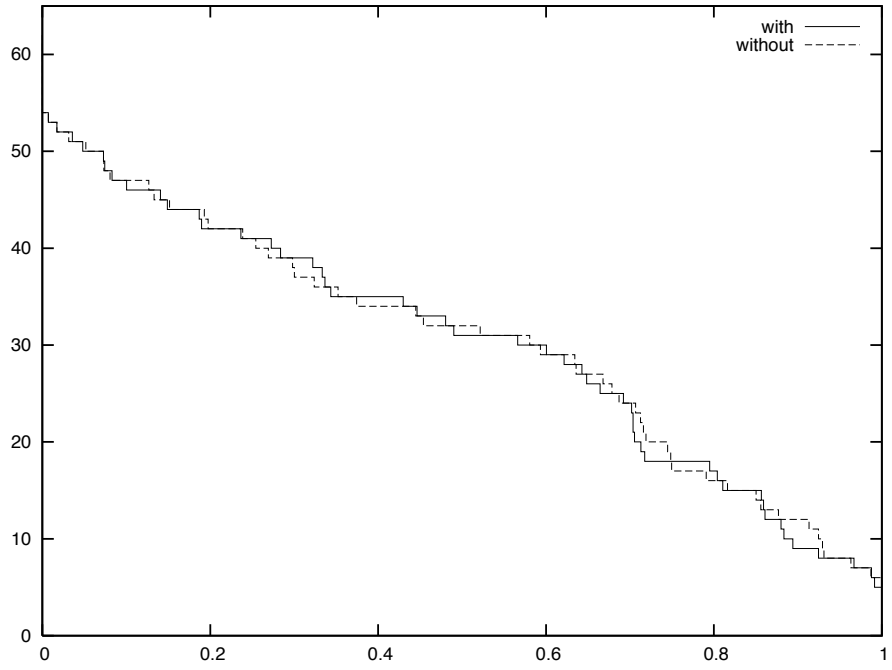


Figure 4.7: Effectiveness of tableau-MIRs with and without cut elimination.

As can be see in Figure 4.7, the effect of eliminating inactive MIR inequalities is marginal. It is important to note that the amount of time required to solve the problem without cut elimination is considerably more than the amount of time required to solve the problem with.

Integer scaling The next experiment consists in evaluation the effect of integer scaling. The procedure is as follows: Instead of generating a single MIR inequality from each tableau row, T_MAX- T_MIN different MIR inequalities are generated; one for each integer value $t \in$[T_MIN, T_MAX]. For each of the tests, T_MIN was set to 1. Then, T_MAX was tested with values $1, 3$ and 5. The results are summarized in Figure 4.8 for the first round, and in Figure 4.9 for the tenth round.

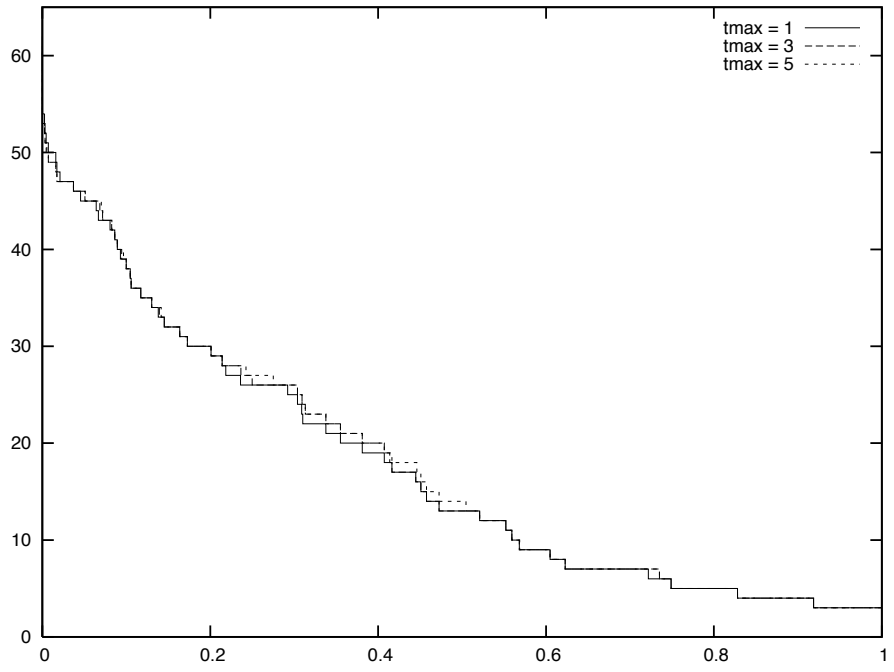


Figure 4.8: Effect of integer scaling on tableau-MIRs after one round.

The results of this experiment are a bit surprising. In fact, consider first Figure 4.8. As would be expected, the best results are obtained by setting T_MAX= 5, followed by T_MAX= 3, with the worst results being for T_MAX= 1. This is only natural, considering that for T_MAX= 5 we are adding five times as many cuts as for T_MAX= 1. However, the first surprise is that the improvements result to be very marginal. Next, consider Figure 4.9.

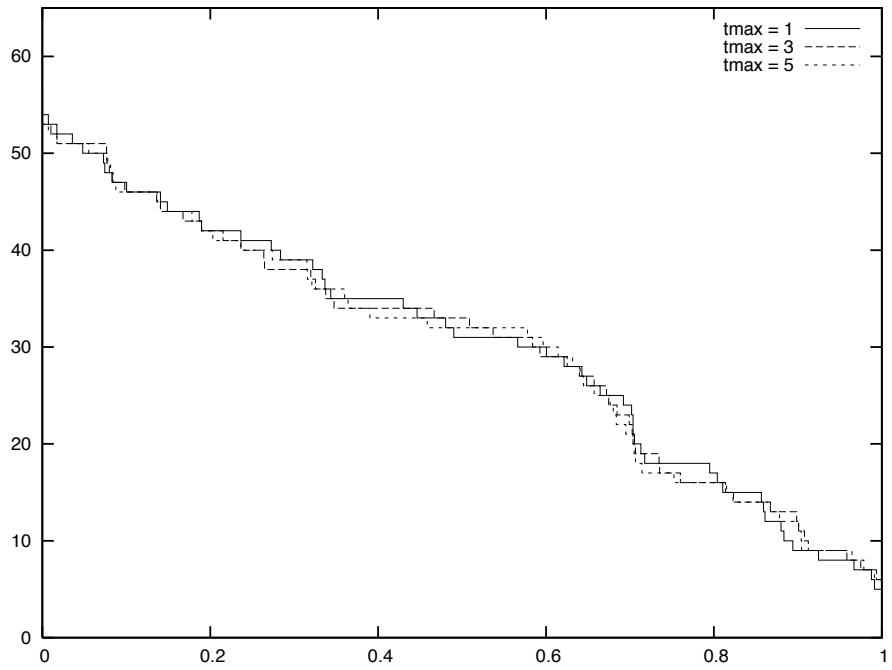


Figure 4.9: Effect of integer scaling on tableau-MIRs after ten rounds.

This is the really surprising observation: Regardless of the integer scaling parameters used, the final gaps obtained after 10 rounds are essentially the same. A naive conclusion from this analysis might be that scaled MIRs are simply not very effective. However, consider Table 4.2, where a selection of the runs plotted in Figure 4.9 are detailed. In this table, the gap closed in eight different instances is shown for each of the three different scaling options.

Table 4.2: Effect of integer scaling on tableau-MIRs after ten rounds.

	T_MAX		
Instance	1	3	5
arki001	0.49	0.58	0.64
l152lav	0.33	0.26	0.20
misc06	0.80	0.70	0.69
misc07	0.007	0.01	0.007
mkc	0.48	0.50	0.39
nw04	0.85	1.00	1.00
p0282	0.33	0.21	0.36
rgn	0.42	0.32	0.36

It can be seen in Table 4.2 that the effect depicted in Figure 4.9 does not imply that the scaled MIRs behave poorly for all instances. In fact, it shows that in some problems they can help a lot, and that in some problems the addition of of those extra cuts leads to a worse final gap. The conclusion that must be made is that the effect of adding the additional scaled cuts is less important than the effect of following different sequences of basis which determine the tableau rows from which cuts are derived.

Complementing variables Next we analyze the impact of the variable complementation scheme COMPLEMENT_RULE_I, as discussed in Section 4.5.2.1. For this we re-solve all of the problem instances, this time without complementing any variables, and compare the results to those obtained with the standard settings after the first and tenth rounds. The results can be seen in Figure 4.10 and Figure 4.11.

As can be seen from the tables, the variable complementation technique is very important.

Row pre-selection rules As a final experiment we explore three different tableau row pre-selection rules (see Section 4.5.1, "Selecting rows"). The idea is that in large problems it is often too slow to derive MIR inequalities from every single tableau row. Hence, it is

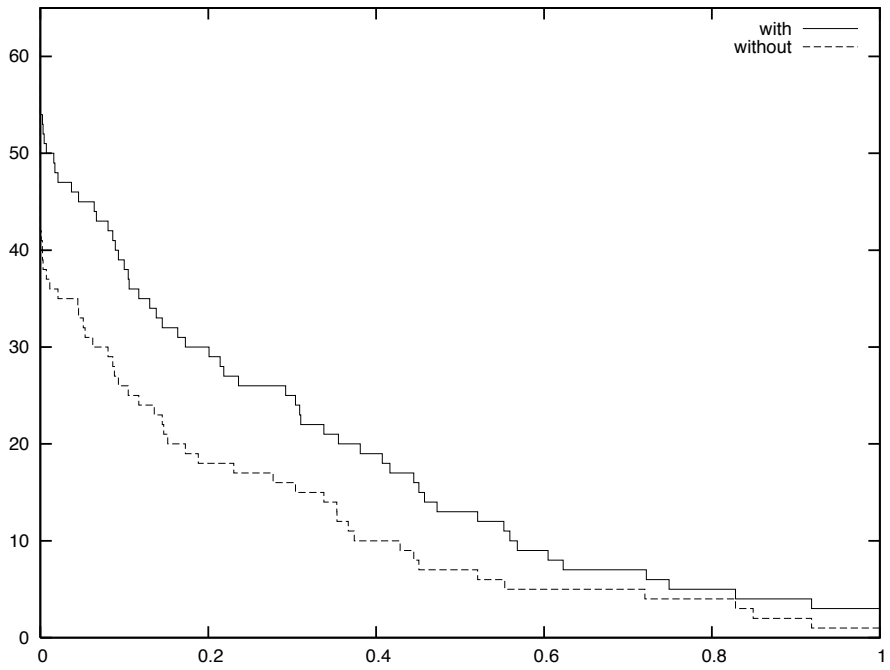


Figure 4.10: The effect of variable complementation on tableau-MIRs after one round.

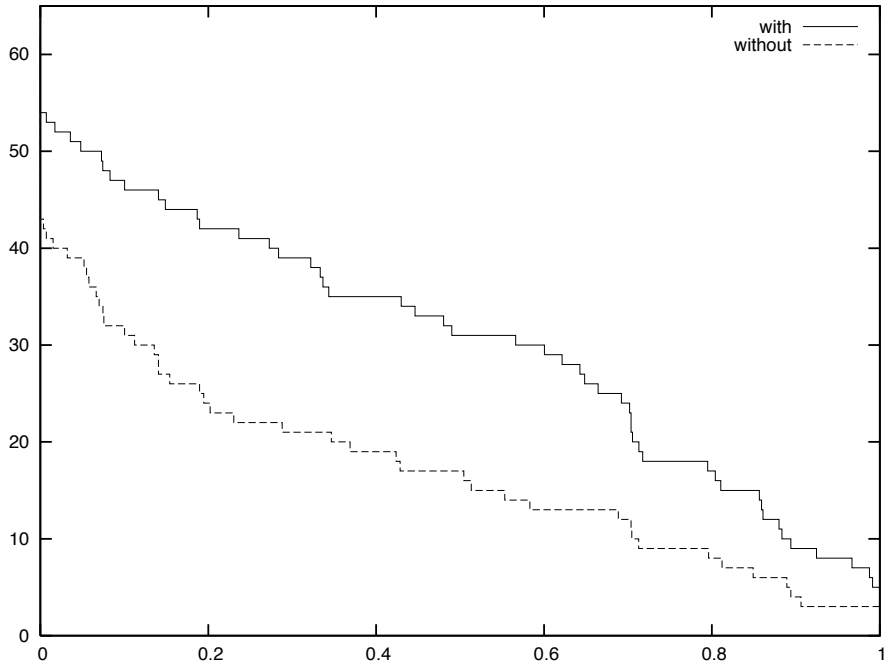


Figure 4.11: The effect of variable complementation on tableau-MIRs after ten rounds.

important to know what the impact of selecting a small subset of rows might be. For the purposes of this test we will choose fifty tableau rows at each iteration and only derive MIRs from those rows. Our set up is as follows: We first rank all of the basic integer variables having fractional variables by one of three possible criteria: In the first, which we call the "random" criteria, the ranking of variables corresponds simply to the order in which the variables have been defined by the user. In the second, which we call the "most fractional" criteria, variables are ranked depending on how close their value is to 0.5. In the third, which we call the "strong branching" criteria, we rank variables according to the quality of the bound given by the strong branching information (see Section 4.5.1, "Ranking integer variables"). After variables have been ranked, we select the fifty tableau rows corresponding to the highest-ranked basic integer variables having fractional values. We finally proceed to derive MIRs only from those tableau rows. The results of each of these three runs is compared to the results obtained without any row pre-selection in Figure 4.12 after one round, and in Figure 4.13 after ten rounds.

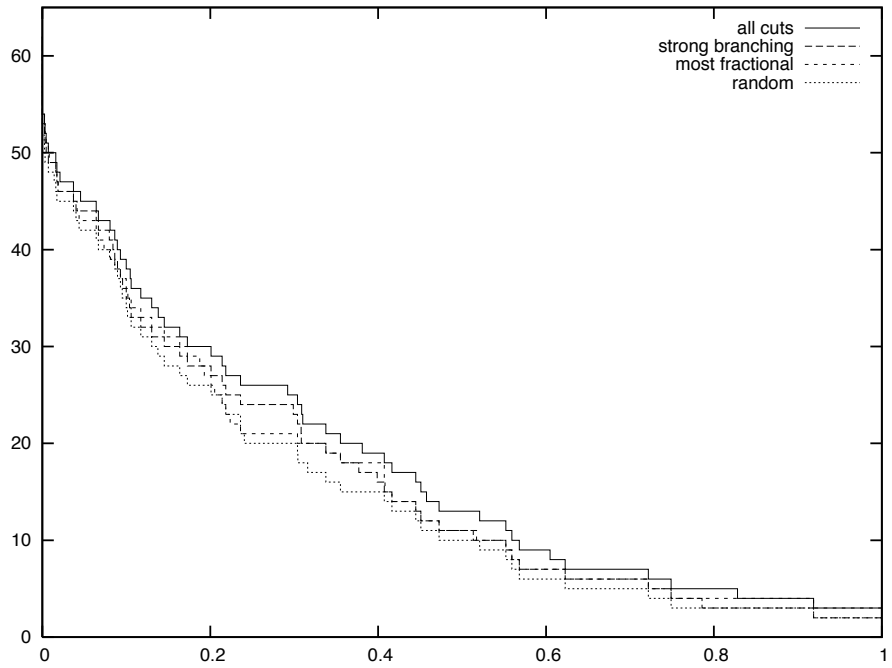


Figure 4.12: The effect of selecting a subset of tableau-MIRs after one round

As can be seen in Figure 4.12 and Figure 4.13 there is a non-trivial difference between

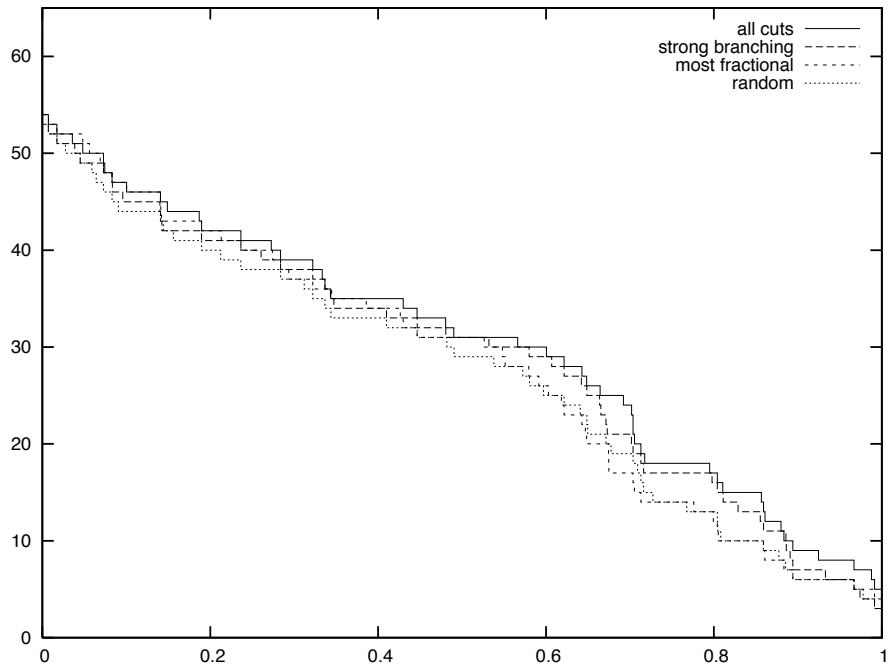


Figure 4.13: The effect of selecting a subset of tableau-MIRs after ten rounds

the proposed criteria. First of all, it is clear that deriving MIRs from every row results in better gaps than deriving MIRs from a subset of rows. Also, it is clear that the "strong branching" criteria outperforms the "most fractional" criteria, and that the "most fractional" criteria outperforms the "random" criteria. Finally, it is interesting to note that the "strong branching" criteria does remarkably well considering that the selected subset size of 50 is often much smaller than the original number of tableau rows. Unfortunately, obtaining the strong branching information was very slow in some cases. This is likely due to the fact that we allowed the simplex algorithm to perform up to 1000 pivots when trying to derive the strong branching bounds. This result suggests that it is worthwhile to further explore the issue of row pre-selection. For example, how does the performance vary if we reduce the number of pivots from 1000 to a much smaller number? Is 50 the correct number of rows to select? Is the best number larger, smaller, or should it be determined dynamically? Bixby, Fenelon, Gu, Rothberg and Wunderling [24] propose setting this value to a fixed 200. They also suggest using Driebeek penalties. How well do these help rank variables as opposed to strong branching information?

Table 4.3: Formulation-MIRs: Settings of separation algorithm.

Cut elimination	Whenever a cut has positive slack
Number of rounds	As many as possible
Coefficient scaling	5 highest ranked variables
Complementation rule	Complementation rule II

4.7.2 Formulation rows

We now analyze the performance of the MIR separation algorithm on formulation rows. The set-up is as follows: after solving the LP relaxation of each problem, we proceed to add MIR inequalities derived from the original formulation rows which define the problem, and repeat until no more violated inequalities can be found. The goal is to evaluate the performance of the MIR separation algorithm, and to determine how to set some of the parameters discussed in this chapter. Our experimentation showed that most of the parameters chosen for the tableau-MIRs work well on the formulation rows as well, so we focus more on parameters which are specific to formulation-MIRs. After much testing, we concluded that the following parameters described in Table 4.3 worked very well.

Of the entire problem set defined by MIPLIB 3.0, we found that in only 26 of the problems there were violated formulation-MIRs. Thus, our study focuses on these problems.

Convergence of the MIR separation procedure We begin by observing the impact of the MIR inequalities as a function of the number of rounds. This effect is illustrated in Figure 4.14, where the gap closed after one round is compared to the gap closed upon ending the procedure.

As can be observed in Figure 4.14 formulation MIRs manage to close a sizeable gap on most of the 26 problems. Further, it can be seen that repeatedly iterating results in important further improvements. For example, in just over ten instances 40% of the gap is closed after one round. However, after completing the procedure an additional five problems close that same amount of gap.

It is important to note that though in some cases many rounds were performed (over thirty in some cases), these iterations were very quick with only a few cuts being added each time. In most problems, the entire procedure took less than a second. This is quite

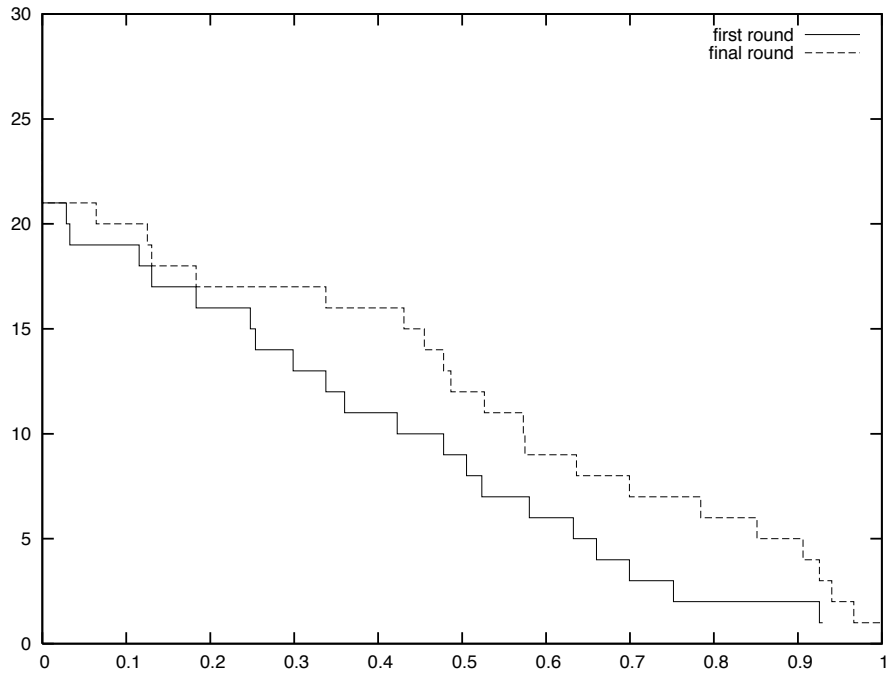


Figure 4.14: Convergence of formulation-MIRs: First and final rounds.

fast considering that little effort was put into making this code efficient.

Complementing variables Next we analyze the impact of the variable complementation schemes COMPLEMENT_RULE_I and COMPLEMENT_RULE_II, as discussed in Section 4.5.2.1 and Section 4.5.2.2. For this we re-solve all of the problem instances twice: the first time without complementing any variables, the second time complementing variables with COMPLEMENT_RULE_II. We then compare the results to those obtained with the standard settings after the first and final rounds. The results can be seen in Figure 4.15 and Figure 4.16.

As can be seen from the tables, the variable complementation technique is very important. Further, though after the first round it is not clear if COMPLEMENT_RULE_I or COMPLEMENT_RULE_II is better, after the final round it is evident that COMPLEMENT_RULE_II has been a much more effective complementation rule, resulting in more and better violated MIR inequalities.

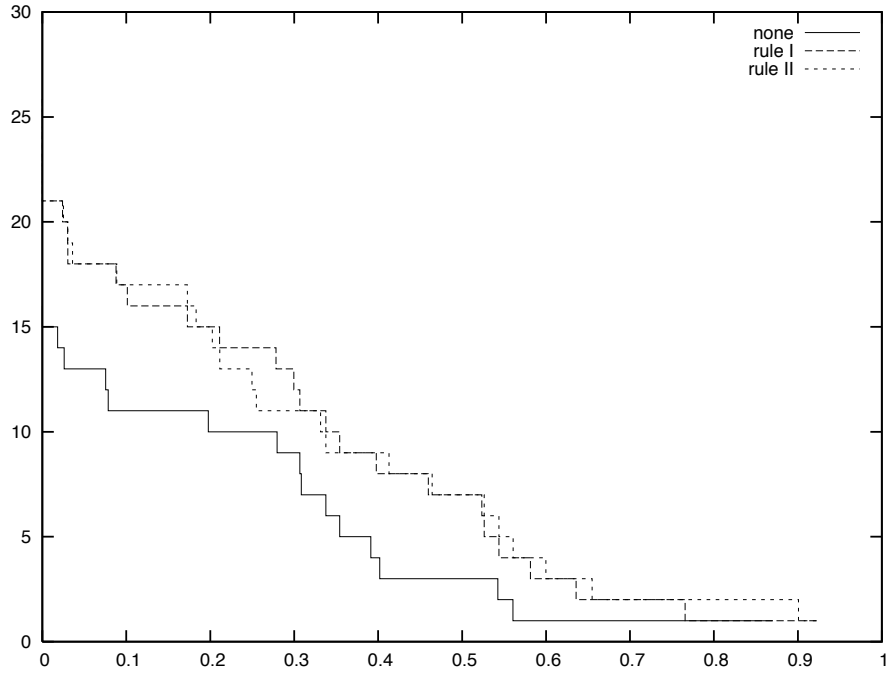


Figure 4.15: The effect of variable complementation on formulation-MIRs after one round.

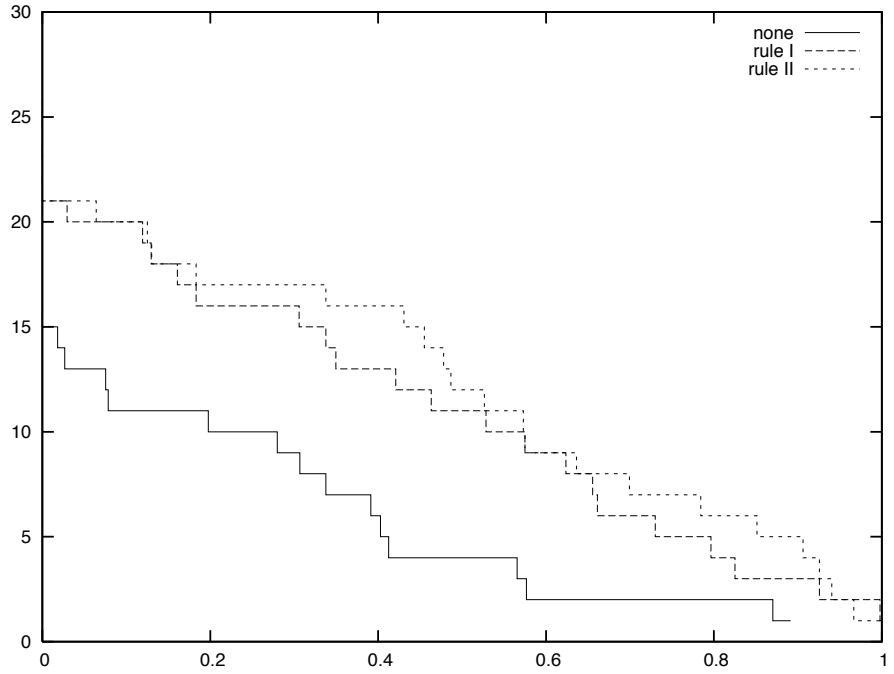


Figure 4.16: The effect of variable complementation on formulation-MIRs after the final round.

Coefficient scaling We now analyze the coefficient scaling heuristic described in Section 4.5.2.2. The parameter with which we control the experiment is the number of variables to scale. That is, the parameter MAX_COEFFICIENT_SCALE. Note that if MAX_COEFFICIENT_SCALE= k then up to k different MIRs might be derived from each row of the original system. Throughout this experiment variables are ranked according the "most fractional" criteria. The results of this experiment are summarized in Figure 4.17 after the first round, and in Figure 4.18 after the final round.

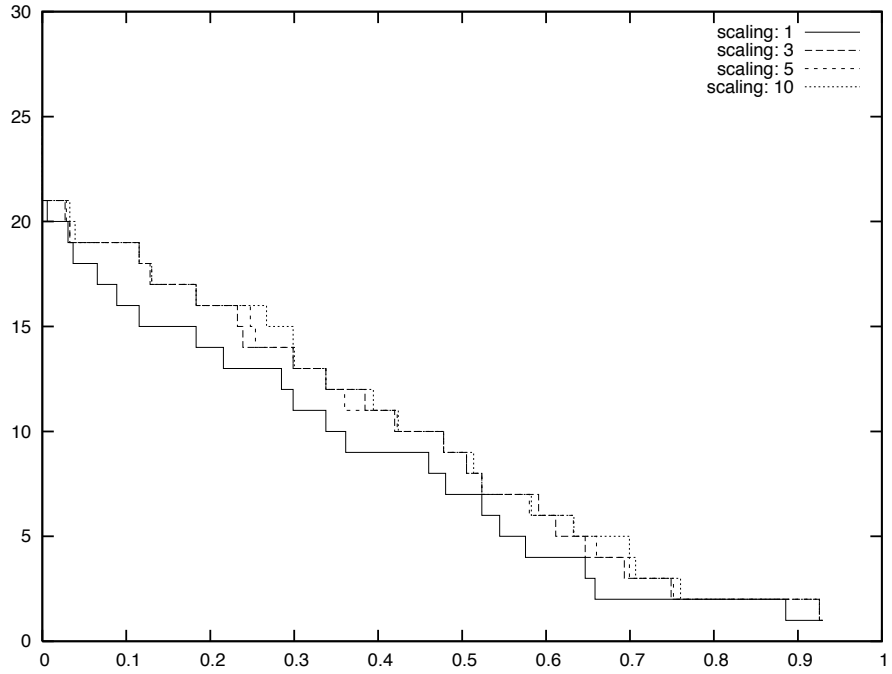


Figure 4.17: The effect of coefficient-scaling on formulation-MIRs after the first round.

As can be observed in Figure 4.17 and Figure 4.18, there is a sizeable jump from setting the scaling parameter to 1, to setting it to a larger value. Naturally, after the first round setting this value to 10 is what works best. But the difference between setting to 1 and setting it to 5 is marginal. In the final round, however, it is not entirely clear if setting the scaling parameter to 10 is better than setting it to 5. However, it seems clear that both 5 and 10 are better settings than 1 and 3. Given that 5 is computationally quicker to work with, it would seem from this small data set that 5 is the best number to use.

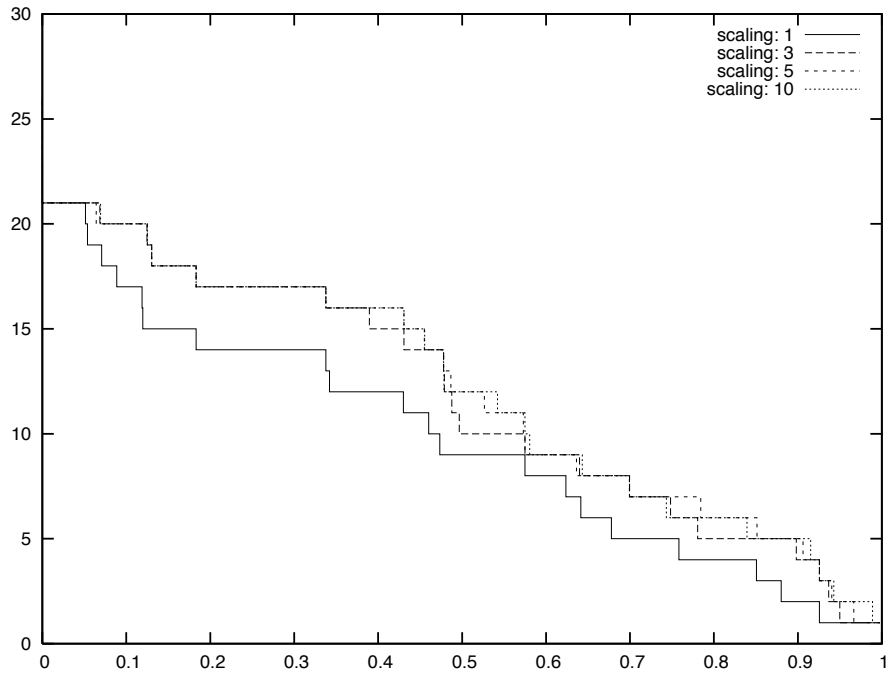


Figure 4.18: The effect of coefficient-scaling on formulation-MIRs after the final round.

4.7.3 Knapsack cuts

We now tackle the question raised in Section 4.6.2: After adding the MIR cuts to all of the tableau rows in a fractional solution and resolving, are there any additional violated knapsack cuts for these same tableau rows?

In order answer this question we could simply apply the procedure outlined in Section 4.6.2 to all of the problems in the test set. However, as this would take a very large amount of time, we instead imitate the procedure of Dash and Günluk [40], and first attempt to find violated knapsack cuts by a simpler means. More precisely, after generating the tableau-MIR inequalities, we proceed to treat the tableau rows used as formulation rows, and attempt to find formulation-MIRs from these. We call these inequalities *delayed tableau-MIRs*, following the notation of Dash, Goycoolea and Günluk [38].

The results obtained generating the tableau-MIRs, and the delayed tableau-MIRs are outlined in Table 4.4. The first two columns indicate the number of violated tableau-MIRs identified, and the gap closed by these inequalities for each instance. The second

two columns indicate the number of violated delayed tableau-MIRs identified, and the gap closed by these inequalities together with the tableau-MIRs. Those cases in which no violated delayed tableau-MIRs are identified are indicated with a hyphen. The last column of this table indicates if in the experiments of Dash and Günluk [40] the corresponding problem was found to be such that after adding all tableau-MIRs, no further group cuts were violated. Note that swath.mps and dano3mip.mps have not been solved to date. Thus it is not possible to compute the gap closed, and so the corresponding columns are filled with question marks.

Table 4.4: Effect of delayed MIRs on tableau rows and results of Dash and Günluk.

	tableau MIRs		delayed tableau MIRs		(Dash and Günluk)
Instance	cuts	gap	cuts	gap	Group closure?
10teams.mps	145	100.0 %	-	-	no
air03.mps	36	100.0 %	-	-	yes
air04.mps	292	8.1 %	78	8.9 %	no
air05.mps	224	4.6 %	76	5.1 %	no
arki001.mps	87	29.2 %	30	39.5 %	no
bell3a.mps	50	45.1 %	-	-	yes
bell5.mps	36	14.5 %	-	-	no
blend2.mps	6	16.4 %	-	-	yes
cap6000.mps	9	41.6 %	168	59.3 %	no
dano3mip	124	???	-	-	unclassified
danoint.mps	52	1.7 %	-	-	unclassified
dcmulti.mps	50	47.3 %	46	50.2 %	no
dsbmip.mps	18	0.0 %	-	-	yes
egout.mps	40	55.9 %	-	-	yes
enigma.mps	8	0.0 %	73	0.0 %	no

Table 4.4 continued.

fast0507.mps	357	2.1 %	18	2.2 %	no
fiber.mps	40	72.2 %	96	76.5 %	no
fixnet6.mps	60	10.5 %	-	-	yes
flugpl.mps	10	11.7 %	-	-	yes
gen.mps	43	55.2 %	4	57.6 %	no
gesa2.mps	91	30.9 %	-	-	no
gesa2_o.mps	121	31.0 %	-	-	unclassified
gesa3.mps	100	45.8 %	-	-	no
gesa3_o.mps	145	60.5 %	2	60.5 %	no
gt2.mps	12	91.9 %	49	92.6 %	no
harp2.mps	30	23.6 %	327	35.2 %	no
khb05250.mps	19	74.9 %	-	-	yes
l152lav.mps	53	9.3 %	122	27.3 %	no
lseu.mps	12	21.8 %	36	28.4 %	no
markshare1.mps	6	0.0 %	233	0.0 %	no
markshare2.mps	7	0.0 %	255	0.0 %	no
mas74.mps	12	6.7 %	59	7.5 %	no
mas76.mps	11	6.4 %	72	7.2 %	no
misc03.mps	38	8.6 %	-	-	no
misc06.mps	16	30.4 %	-	-	yes
misc07.mps	47	0.7 %	-	-	yes
mitre.mps	994	82.8 %	184	84.0 %	no
mkc.mps	153	13.8 %	1161	21.6 %	no
mod008.mps	5	20.1 %	30	22.5 %	no
mod010.mps	39	100.0 %	10	100.0 %	yes
mod011.mps	32	35.5 %	-	-	yes
modglob.mps	30	17.3 %	6	18.1 %	unclassified
noswot.mps	39	0.0 %	-	-	yes

Table 4.4 continued.

nw04.mps	6	62.3 %	24	97.7 %	yes
p0033.mps	7	56.8 %	27	75.6 %	no
p0201.mps	22	33.8 %	8	33.8 %	yes
p0282.mps	32	3.7 %	31	8.7 %	no
p0548.mps	49	40.7 %	570	67.5 %	no
p2756.mps	72	0.5 %	863	57.1 %	no
pk1.mps	15	0.0 %	77	0.0 %	no
pp08a.mps	51	52.1 %	-	-	yes
pp08aCUTS	46	68.4 %	1	68.4 %	unclassified
qiu.mps	36	0.3 %	-	-	yes
qnet1.mps	54	10.6 %	136	11.2 %	no
qnet1_o.mps	11	44.5 %	587	49.9 %	no
rentacar.mps	17	21.4 %	-	-	yes
rgn.mps	18	1.6 %	31	3.3 %	no
rout.mps	38	0.3 %	550	0.3 %	unclassified
set1ch.mps	138	38.1 %	-	-	yes
seymour.mps	4488	8.9 %	10	9.5 %	no
stein27.mps	84	0.0 %	1	0.0 %	no
stein45.mps	218	0.0 %	3	0.0 %	no
swath	44	???	66	???	no
vpm1.mps	13	10.0 %	27	11.1 %	yes
vpm2.mps	31	13.0 %	25	16.9 %	yes

As can be observed from Table 4.4, in 41 of the 65 instances we were able to identify violated delayed tableau-MIRs. This indicates that each of those 41 instances had violated knapsack cuts.Note also that there is a strong relationship between those instances which Dash and Günluk identified as having violated group cuts, with those instances which have violated delayed tableau-MIRs. In fact, Dash and Günluk established that in 38 of the

MIPLIB 3.0 instances there were violated group cuts. Of these 38 instances, 33 coincide with those in which there are violated delayed tableau-MIRs.

As a next step it becomes necessary to determine if in the 24 instances remaining there are violated knapsack cuts. These instances are listed in Table 4.5. For this, we employed the methodology described in Section 4.6.2 using exact arithmetic to ensure the validity of the results. Of these 24 instances we were able to determine that 23 of them did indeed have violated knapsack cuts. The one exception was dano3mip, which was much too large for the current implementation of the algorithm to work.

What the preceding result highlights is that there is a potential for identifying additional classes of cuts derived from single row systems. However, this study should be followed up by a procedure which can identify those cuts and add them to the LP relaxation. It is still possible that these cuts are weak and that they might contribute very little to improving the bounds, or the time required to solve IPs. Even if this separation procedure is slow, its value would be of a more empirical nature, providing important guidelines as to where future research should head in attempting to determine new and better classes of valid inequalities for general mixed integer programming problems.

Table 4.5: Instances for which there were no violated delayed tableau-MIRs.

Instance	knapsack closure?
10teams.mps	no
air03.mps	no
bell3a.mps	no
bell5.mps	no
blend2.mps	no
dano3mip.mps	-
danoint.mps	no
dsbmip.mps	no
Table 4.5 continued.	

egout.mps	no
fixnet6.mps	no
flugpl.mps	no
gesa2.mps	no
gesa2_o.mps	no
gesa3.mps	no
khb05250.mps	no
misc03.mps	no
misc06.mps	no
misc07.mps	no
mod011.mps	no
noswot.mps	no
pp08a.mps	no
qiu.mps	no
rentacar.mps	no
set1ch.mps	no

REFERENCES

[1] Achterberg, T., Koch, T., and Martin, A., "MIPLIB 2003," *ZIB Technical Report*, vol. 05-28, 2003.

[2] Agra, A. and Constantino, M. F., "Lifting 2-integer knapsack inequalities," *Mathematical Programming*, vol. To appear, 2006.

[3] Andersen, K., Cornuéjols, G., and Li, Y., "Reduce-and-split cuts: Improving the performance of mixed-integer gomory cuts," *Management Science*, vol. 51, 2005.

[4] Andersen, K., Cornuéjols, G., and Li, Y., "Split closure and intersection cuts," *Mathematical Programming*, vol. 102, pp. 457 – 493, 2005.

[5] Andonov, R., Poirriez, V., and Rajopadhye, S., "Unbounded knapsack problem: dynamic programming revisited," *European Journal of Operational Research*, vol. 123, pp. 394–407, 2000.

[6] Appel, K. and Haken, W., "Every planar map is four colorable," *Contemporary Math.*, vol. 98, 1989.

[7] Applegate, D., Bixby, R. E., Chvátal, V., and Cook, W., "On the solution of traveling salesman problems," *Documenta Mathematica*, vol. Extra Volume Proceedings ICM III, pp. 645–656, 1998.

[8] Applegate, D., Bixby, R. E., Chvátal, V., and Cook, W., "Tsp cuts which do not conform to the template paradigm," in *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School]*, (London, UK), pp. 261–304, Springer-Verlag GmbH, 2001.

[9] Araoz, J., Evans, L., Gomory, R. E., and Johnson, E. L., "Cyclic group and knapsack facets," *Mathematical Programming*, vol. 96, pp. 377–408, May 2003.

[10] Atamtürk, A., "On the facets of the mixed–integer knapsack polyhedron," *Mathematical Programming*, vol. 98, pp. 145–175, 2003.

[11] Atamtürk, A., "Sequence independent lifting for mixed–integer programming," *Operations Research*, vol. 52, pp. 487–490, 2004.

[12] Atamtürk, A. and Rajan, D., "Valid inequalities for mixed-integer knapsack from two-integer variable restrictions," Tech. Rep. BCOL.04.02, IEOR, University of California–Berkeley, December 2004.

[13] Balas, E., "Intersection cuts - a new type of cutting planes for integer programming.," *Operations Research*, vol. 19, pp. 19–39, 1971.

[14] Balas, E., "Facets of the knapsack polytope," *Mathematical Programming*, vol. 8, pp. 146–164, 1975.

[15] Balas, E., Ceria, S., and Cornuéjols, G., "A lift-and-project cutting plane algorithm for mixed 0-1 programs," *Mathematical Programming*, vol. 58, pp. 295–324, 1993.

[16] Balas, E., Ceria, S., and Cornuéjols, G., "Mixed 0-1 programming by lift-and-project in a branch-and-cut framework," *Management Science*, vol. 42, pp. 1229–1246, 1996.

[17] Balas, E., Ceria, S., Cornuéjols, G., and Natraj, N., "Gomory cuts revisited," *Operations Research Letters*, vol. 19, pp. 1–9, 1996.

[18] Balas, E. and Perregaard, M., "A precise correspondence between lift-and-project cuts, simple disjuntive cuts, and mixed integer gomory cuts for 0-1 programming," *Mathematical Programming*, vol. 94, pp. 221–245, 2003.

[19] Balas, E. and Zemel, E., "An algorithm for large zero-one knapsack problems," *Operations Research*, vol. 28, pp. 1130–1154, 1980.

[20] Bellman, R. E., *Dynamic Programming*. Princeton University Press, 1957.

[21] Bertsimas, D. and Weismantel, R., *Optimization over Integers*. Dynamic Ideas, 2005.

[22] Bixby, R. E., Ceria, S., McZeal, C. M., and Savelsbergh, M. W. P., "An updated mixed integer programming library: MIPLIB 3.0," *Optima*, pp. 12–15, June 1998.

[23] Bixby, R. E., Fenelon, M., Gu, Z., Rothberg, E., and Wunderling, R., "Mip: Theory and practice - closing the gap," in *System Modelling and Optimization*, pp. 19–50, 1999.

[24] Bixby, R., Gu, Z., Rothberg, E., and Wunderling, R., "Mixed integer programming: a progress report," in *The sharpest cut: the impact of Manfred Padberg and his work. MPS/SIAM Series on Optimization*, pp. 309–326, 2004.

[25] Boyd, A. E., "Fenchel cutting planes for integer programs," *Operations Research*, vol. 42, pp. 53–64, 1992.

[26] Boyd, A. E., "Generating fenchel cutting planes for knapsack polyhedra," *Siam journal on optimization*, vol. 3, pp. 734–750, 1993.

[27] Boyd, S. and Cunningham, W., "Small travelling salesman polytopes," *Mathematics of Operations Research*, vol. 16, no. 2, pp. 259–271, 1991.

[28] Boyd, S. C., Cockburn, S., and Vella, D., "On the domino-parity inequalities for the STSP," Tech. Rep. TR-2001-10, University of Ottawa, Ottawa, Canada, 2001.

[29] Caprara, A. and Fischetti, F., "0-1/2 chvátal-gomory cuts," *Mathematical Programming*, vol. 74, pp. 221–236, 1996.

[30] Carr, R., "Separating clique trees and bipartition inequalities having a fixed number of handles and teeth in polynomial time," *Mathematics of Operations Research*, vol. 22, no. 2, pp. 257–265, 1997.

[31] Chvátal, V., "Edmonds polytopes and weakly hamiltonian graphs," *Mathematical Programming*, vol. 5, pp. 29–40, 1973.

[32] COIN-OR, "Computation infrastructure for operations research." Available on-line at http://www.coin-or.org (November, 2006).

[33] Cook, W., Kannan, R., and Schrijver, A., "Chvátal closures for mixed integer programming problems," *Mathematical Programming*, vol. 47, pp. 155–174, 1990.

[34] Cook, W. J., Espinoza, D. G., and Goycoolea, M., "Computing with domino-parity inequalities for the tsp," *INFORMS Journal on Computing*, vol. To appear, 2006.

[35] Cornuéjols, G., *State-of-the-Art and Recent Advances in Integer Programming*, ch. Revival of the Gomory Cuts in the 1990's. 2005.

[36] Cornuéjols, G., Li, Y., and Vanderbussche, D., "K-cuts: A variation of gomory mixed integer cuts from the lp tableau," *Informs Journal on Computing*, vol. 15, pp. 385–396, Fal 2003.

[37] Dantzig, G., Fulkerson, R., and Johnson, S., "Solution of a large-scale traveling salesman problem," *Operations Research*, vol. 2, pp. 393–410, 1954.

[38] Dash, S., Goycoolea, M., and Gunluk, O., "Two-step mir inequalities for mixed-integer programs," *Optimization Online*, Jul 2006.

[39] Dash, S. and Gunluk, O. Internal Communication, 2005-2006.

[40] Dash, S. and Gunluk, O., "On the strength of gomory mixed-integer cuts as group cuts," *IBM research report RC23967*, 2006.

[41] Dash, S. and Gunluk, O., "Valid inequalities based on simple mixed-integer sets," *Mathematical Programming*, vol. 105, pp. 29–53, Jan 2006.

[42] Dash, S. and Günlük, O., "Valid inequalities based on the interpolation procedure," *Mathematical Programming*, vol. 106, pp. 111–136, May 2006.

[43] Driebeek, N., "An algorithm for the solution of mixed integer programming problems," *Management Science*, vol. 12, pp. 576–587, 1966.

[44] Edmonds, J., "Lehman's switching game and a theorem of Tutte and Nash-Williams," *Journal of Research of the National Bureau of Standards*, vol. 69B, pp. 73–77, 1965.

[45] Espinoza, D. G., *On Linear Programming, Integer Programming and Cutting Planes*. PhD thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, March 2006.

[46] Evans, L., *Cyclic Group and Knapsack Facets with Applications to Cutting Planes*. PhD thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, 2002.

[47] Fischetti, A. and Lodi, A., "UNIBO, mixed integer programming instances." Available on-line at http://www.or.deis.unibo.it/research_pages/ORinstances/MIPs.html (November, 2006).

[48] Fischetti, M. and Lodi, A., "Optimizing over the Chvátal closure," *IPCO, Lecture notes in computer science, M.Juenger and V. Kaibel, eds*, vol. 3509, pp. 12–22, 2005.

[49] Fischetti, M. and Saturni, C., "Mixed-integer cuts from cyclic groups," *Lecture Notes in Computer Science*, vol. 3509, pp. 1–11, 2005.

[50] Fleischer, L. and Tardos, E., "Separating maximally violated comb inequalities in planar graphs," *Mathematics of Operations Research*, vol. 24, pp. 130–148, 1999.

[51] Fleischmann, B., "A new class of cutting planes for the symmetric traveling salesman problem," *Mathematical Programming*, vol. 40, pp. 225–246, 1988.

[52] Gerards, A. and Schrijver, A., "Matrices with the Edmonds-Johnson property," *Combinatorica*, vol. 6, pp. 365–379, 1986.

[53] Giles, F. and Pulleyblank, W., "Total dual integrality and integer polyhedra," *Linear Algebra and its Applications*, vol. 25, pp. 191–196, 1979.

[54] Gilmore, P. and Gomory, R., "A linear programming approach to the cutting stock problem, part II," *Operations Research*, vol. 11, pp. 863–888, 1963.

[55] Goldberg, D., "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR)*, vol. 23, pp. 5–48, 1991.

[56] Gomory, R. E., "Solving linear programming in integers," in *Combinatorial Analysis, Proceedinigs of the Symposia in Applied Mathematics*, American Mathematical Society, 1960.

[57] Gomory, R. E., *Recent Advances in Mathematical Programming*, ch. An Algorithm for Integer Solutions to Linear Programs, pp. 269–302. McGray-Hill, New York, 1963.

[58] Gomory, R. E., "Some polyhedra related to combinatorial problems," *Journal of Linear Algebra and its Applications*, vol. 2, pp. 451–558, 1969.

[59] Gomory, R. E. and Johnson, E., "Some continuous functions related to corner polyhedra I," *Mathematical Programming*, vol. 3, pp. 23–85, 1972.

[60] Gomory, R. E. and Johnson, E., "Some continuous functions related to corner polyhedra II," *Mathematical Programming*, vol. 3, pp. 359–389, 1972.

[61] Gomory, R., Johnson, E., and Evans, L., "Corner polyhedra and their connection with cutting planes," *Mathematical Programming*, vol. 96, pp. 321–339, May 2003.

[62] Granlund, T., "The GNU multiple precision arithmetic library." Available on-line at http://www.swox.com/gmp/ (November, 2006).

[63] Graver, J., "On the foundations of linear and integer programming I," *Mathematical Programming*, vol. 8, pp. 206–226, 1975.

[64] Greenberg, H. and Hegerich, R., "A branch search algorithm for the knapsack problem," *Management Science*, vol. 16, pp. 327–332, 1970.

[65] Grötschel, M. and Padberg, M. W., "On the symmetric traveling salesman problem II: Lifting theorems and facets," *Mathematical Programming*, vol. 16, pp. 281–302, 1979.

[66] Grötschel, M. and Pulleyblank, W. R., "Clique tree inequalities and the symmetric travelling salesman problem," *Mathematics of Operations Research*, vol. 11, pp. 537–569, 1986.

[67] Horowitz, E. and Sahni, S., "Computing partitions with applications to the knapsack problem," *Journal of the ACM*, vol. 21, pp. 277–292, 1974.

[68] IEEE, "ANSI/IEEE standard 754-1985, standard for binary floating point arithmetic." Available on-line at http://standards.ieee.org/ (November, 2006).

[69] ILOG, "CPLEX, a mathematical programming optimizer." http://www.ilog.com/products/cplex (November, 2006).

[70] Jüunger, M., Reinelt, G., and Rinaldi, G., "The traveling salesman problem," pp. 225–330, North Holland, Amsterdam, The Netherlands, 1995.

[71] Karger, D., "A new approach to the minimum cut problem," *Journal of the ACM*, vol. 43, pp. 601–640, 1996.

[72] Kellerer, H., Pferschy, U., and Pisinger, D., *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[73] Letchford, A. N., "Separating a superclass of comb inequalities in planar graphs," *Mathematics of Operations Research*, vol. 25, pp. 443–454, 2000.

[74] Marchand, H. and Wolsey, L., "Aggregation and mixed integer rounding to solve mips," *Operations Research*, vol. 49, pp. 363–371, 2001.

[75] Martello, S., , Pisinger, D., and Toth, P., "Dynamic programming and strong bounds for the 0-1 knapsack problem," *Management Science*, vol. 45, pp. 414–424, 1999.

[76] Martello, S. and Toth, P., *Advances in Operations Research*, ch. Branch and bound algorithms for the solution of general unidimensional knapsack problems, pp. 295–301. North Holland, 1977.

[77] Martello, S. and Toth, P., "An upper bound for the zero-one knapsack problem and a branch and bound algorithm," *European Journal of Operations Research*, vol. 1, pp. 169–175, 1977.

[78] Martello, S. and Toth, P., "A new algorithm for the 0-1 knapsack problem," *Management Science*, vol. 34, pp. 633–644, 1988.

[79] Martello, S. and Toth, P., "An exact algorithm for larged unbounded knapsack problems," *Operations Research Lettrs*, vol. 9, pp. 15–20, 1990.

[80] Martello, S. and Toth, P., *Knapsack Problems: Algorithms and Computer Implementations*. J. Wiley, New York, 1990.

[81] Martello, S. and Toth, P., "An exact algorithm for hard 0-1 knapsack problems," *Operations Research*, vol. 45, pp. 768–778, 1997.

[82] Mittelman, H., "MIPLBLIB." Available online at http://plato.asu.edu/topics/testcases.html (November, 2006).

[83] Mohar, B. and Thomassen, C., *Graphs on Surfaces*. Johns Hopkins University Press, 2001.

[84] Naddef, D., "Polyhedral theory and branch-and-cut algorithms for the symmetric traveling salesman problem," pp. 29–116, Kluwer, Dordrecht, 2002.

[85] Naddef, D. and Thienel, S., "Efficient separation routines for the symmetric traveling salesman problem I: General tools and comb separation," *Mathematical Programming*, vol. 92, pp. 237–255, 2002.

[86] Naddef, D. and Thienel, S., "Efficient separation routines for the symmetric traveling salesman problem II: Separating multi handle inequalities," *Mathematical Programming*, vol. 92, pp. 257–283, 2002.

[87] Naddef, D. and Wild, E., "The domino inequalities: Facets for the symmetric traveling salesman polytope," *Mathematical Programming*, vol. 98, pp. 223–251, Sep 2003.

[88] Nagamochi, H., Nishimura, K., and Ibaraki, T., "Computing all small cuts in undirected networks," *SIAM Journal on Discrete Mathematics*, pp. 469–481, 1997.

[89] Nemhauser, G. L. and Wolsey, L. A., "A recursive procedure for generating all cuts for 0-1 mixed integer programs," *Mathematical Programming*, vol. 46, pp. 379–390, 1990.

[90] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization*. Discrete Mathematics and Optimization, Wiley-Interscience, 1999.

[91] Padberg, M. W. and Rao, M. R., "Odd minimum cut-sets and b-matchings," *Mathematics of Operations Research*, vol. 7, pp. 67–80, 1982.

[92] Padberg, M. W. and Rinaldi, G., "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM Review*, vol. 33, pp. 60–100, 1991.

[93] Pisinger, D., "An expanding-core algorithm for the 0-1 knapsack problem," *European Journal of Operations Research*, vol. 87, pp. 175–187, 1995.

[94] Pisinger, D., "A minimal algorithm for the 0-1 knapsack problem," *Operations Research*, vol. 45, pp. 758–767, 1997.

[95] Pisinger, D., "Core problems in knapsack algorithms," *Operations Research*, vol. 47, pp. 570–575, 1999.

[96] Pisinger, D., "Linear time algorithms for knapsack problems with bounded weights," *Journal of Algorithms*, vol. 34, pp. 1–14, 1999.

[97] Pisinger, D., "A minimal algorithm for the bounded knapsack problem," *INFORMS Journal on Computing*, vol. 34, pp. 75–84, 2000.

[98] Reinelt, G., "TSPLIB - a traveling salesman library," *ORSA Journal on Computing*, vol. 3, pp. 376–384, 1991.

[99] Richard, J., de Farias, I., and Nemhauser, G., "Lifted inequalities for 0-1 mixed integer programming: Basic theory and algorithms," *Mathematical Programming*, vol. 98, pp. 89–113, 2003.

[100] Robertson, N., Sanders, D., Seymour, P., and Thomas, R., "The four color theorem," *J. Combin. Theory Ser. B*, vol. 70, pp. 2–44, 1997.

[101] Savelsbergh, M., "Preprocessing and probing for mixed integer programming problems," *ORSA Journal on Computing*, vol. 6, pp. 445–454, 1994.

[102] Schrijver, A., *Combinatorial Optimization : Polyhedra and Efficiency.* Volume A: Paths, flows, matchings, Chapters 1-38, Springer, 2003.

VITA

Marcos Goycoolea was born in St. Paul, MN, on November 27 1975. In March, 1994, he enrolled at the school of Engineering of Universidad de Chile, from where he received a Bachelor in Engineering Sciences, with a major in Mathematics (June 2000), and a degree in Mathematical Engineering (August 2001). While pursuing his studies in Chile, Marcos worked with Dr. Andres Weintraub, Dr. Francisco Barahona, and Dr. Alan Murray on integer programming applications to Forest Harvest Scheduling. In August 2001, he enrolled in the School of Industrial and Systems Engineering at the Georgia Institute of technology. There, he worked on several research projects, both applied and theoretical. On the one hand, he worked with a team headed by Dr. George Nemhauser and Dr. Martin Savelsbergh on routing algorithms for air-taxi transportation services, and on the other, with Dr. William Cook on the Traveling Salesman Problem, the Mixed Integer Knapsack Problem, and general Mixed Integer Programming. This thesis covers some of this latter work.