

Chapter 2

Just MIP it!

Matteo Fischetti, Andrea Lodi, and Domenico Salvagnin

Abstract Modern Mixed-Integer Programming (MIP) solvers exploit a rich arsenal of tools to attack hard problems. It is widely accepted by the OR community that the solution of very hard MIPs can take advantage from the solution of a series of time-consuming auxiliary Linear Programs (LPs) intended to enhance the performance of the overall MIP solver. For instance, auxiliary LPs may be solved to generate powerful disjunctive cuts, or to implement a strong branching policy. Also well established is the fact that finding good-quality heuristic MIP solutions often requires a computing time that is just comparable to that needed to solve the LP relaxations. So, it makes sense to think of a new generation of MIP solvers where auxiliary MIPs (as opposed to LPs) are heuristically solved on the fly, with the aim of bringing the MIP technology under the chest of the MIP solver itself. This leads to the idea of “translating into a MIP model” (*MIPping*) some crucial decisions to be taken within a MIP algorithm (How to cut? How to improve the incumbent solution? Is the current node dominated?). In this paper we survey a number of successful applications of the above approach.

Matteo Fischetti
DEI, Università di Padova, Padua, Italy
e-mail: matteo.fischetti@unipd.it

Andrea Lodi
DEIS, Università di Bologna, Bologna, Italy
e-mail: andrea.lodi@unibo.it

Domenico Salvagnin
DMPA, Università di Padova, Padua, Italy
e-mail: dominiqs@gmail.com

2.1 Introduction

Modern MIP solvers exploit a rich arsenal of tools to attack hard problems. Some successful examples involve the solution of LP models to control the branching strategy (strong branching), the cut generation (lift-and-project), and the heuristics (reduced costs). As a matter of fact, it is well known by the OR community that the solution of very hard MIPs can take advantage of the solution of a series of auxiliary LPs intended to guide the main steps of the MIP solver.

Also well known is the fact that finding good-quality heuristic MIP solutions often requires a computing time that is just comparable to that needed to solve the LP relaxation of the problem at hand. This leads to the idea of “translating into a MIP model” (*MIPping*) some crucial decisions to be taken within a MIP algorithm (in particular: How to cut? How to improve the incumbent solution? Is the current node dominated?), with the aim of bringing the MIP technology well inside the MIP solver.

The present paper gives a survey of three successful applications of the MIPping approach. In Section 2.2 we address the generation of strong cutting planes. In this context, the MIPping approach has been extensively applied to modeling and solving (possibly in a heuristic way) the NP-hard separation problems of famous classes of valid inequalities for mixed integer linear programs. Besides the theoretical interest in evaluating the strength of these classes of cuts computationally, the approach proved successful also in practice, and allowed the solution of very hard MIPLIB instances [2] that could not be solved before.

In Section 2.3 we address enhanced (primal) heuristic approaches for the solution of hard MIP models. An example of the benefits deriving from the use of a black-box MIP solver to produce heuristic primal solutions for a generic MIP is the recently-proposed *local branching* paradigm that uses a general-purpose MIP solver to explore large solution neighborhoods defined through the introduction in the MIP model of invalid linear inequalities called *local branching cuts* [25]. More recently, a different heuristic approach called *Feasibility Pump* has been proposed to address the problem of finding an initial feasible solution and of improving it. In Section 2.3 we describe a hybrid algorithm that uses the feasibility pump method to provide, at very low computational cost, an initial (possibly infeasible) solution to the local branching procedure.

In Section 2.4 we finally address the general-purpose dominance procedure proposed in the late 80’s by Fischetti and Toth [30], that overcomes some of the drawbacks of the classical dominance definition. Given the current node α of the search tree, let J^α be the set of variables fixed to some value. Following the MIPping paradigm, we construct an auxiliary problem XP^α that looks for a new partial assignment involving the variables in J^α and such that (i) the objective function value is not worse than the one associated with the original assignment, and (ii) every completion of the old partial assignment

is also a valid completion of the new one. If such a new partial assignment is found (and a certain tie-breaking rule is satisfied), one is allowed to fathom node α .

The present survey is based on previous published work; in particular, Sections 2.2, 2.3 and 2.4 are largely based on [26], [28] and [54], respectively.

2.2 MIPping Cut Separation

In this section we first introduce our basic notation and definitions and review some classical results on cutting planes for pure and mixed integer problems. Then, we discuss in Section 2.2.1 the separation of pure integer cuts, i.e., those cuts in which (i) all coefficients are integer and (ii) continuous variables (if any) have null coefficients. In Section 2.2.2 we address the more general (and powerful) family of split cuts which are instead mixed integer inequalities because the two conditions above do not apply. Finally, in Subsection 2.2.3 we discuss computational aspects of these models and we report results on the strength of the addressed cuts.

Consider first the pure integer linear programming problem $\min\{c^T x : Ax \leq b, x \geq 0, x \text{ integral}\}$ where A is an $m \times n$ rational matrix, $b \in \mathbb{Q}^m$, and $c \in \mathbb{Q}^n$, along with the two associated polyhedra $P := \{x \in \mathbb{R}_+^n : Ax \leq b\}$ and $P_I := \text{conv}\{x \in \mathbb{Z}_+^n : Ax \leq b\} = \text{conv}(P \cap \mathbb{Z}^n)$.

A *Chvátal-Gomory (CG) cut* (also known as *Gomory fractional cut*) [35, 13] is an inequality of the form $\lfloor u^T A \rfloor x \leq \lfloor u^T b \rfloor$ where $u \in \mathbb{R}_+^m$ is a vector of multipliers, and $\lfloor \cdot \rfloor$ denotes the lower integer part. Chvátal-Gomory cuts are valid inequalities for P_I . The *Chvátal closure* of P is defined as

$$P^1 := \{x \geq 0 : Ax \leq b, \lfloor u^T A \rfloor x \leq \lfloor u^T b \rfloor \text{ for all } u \in \mathbb{R}_+^m\}. \quad (2.1)$$

Thus $P_I \subseteq P^1 \subseteq P$. By the well-known equivalence between optimization and separation [37], optimizing over the first Chvátal closure is equivalent to solving the *CG separation problem* where we are given a point $x^* \in \mathbb{R}^n$ and are asked to find a hyperplane separating x^* from P^1 (if any). Without loss of generality we can assume that $x^* \in P$, since all other points can be cut by simply enumerating the members of the original inequality system $Ax \leq b, x \geq 0$. Therefore, the separation problem we are actually interested in reads:

CG-SEP: Given any point $x^* \in P$ find (if any) a CG cut that is violated by x^* , i.e., find $u \in \mathbb{R}_+^m$ such that $\lfloor u^T A \rfloor x^* > \lfloor u^T b \rfloor$, or prove that no such u exists.

It was proved by Eisenbrand [23] that CG-SEP is NP-hard, so optimizing over P^1 also is.

Moreover, Gomory [36] proposed a stronger family of cuts, the so-called *Gomory Mixed Integer (GMI) cuts*, that apply to both the pure integer and

the mixed integer case. Such a family of inequalities has been proved to be equivalent to two other families, the so-called *split cuts* defined by Cook et al. [15], and the *Mixed Integer Rounding (MIR) cuts* introduced by Nemhauser and Wolsey [50]. The reader is referred to Cornuéjols and Li [17] for formal proofs of the correspondence among those families, and to Cornuéjols [16] for a very recent survey on valid inequalities for mixed integer linear programs. Let us consider a generic MIP of the form:

$$\min\{c^T x + f^T y : Ax + Cy \leq b, x \geq 0, x \text{ integral}, y \geq 0\} \quad (2.2)$$

where A and C are $m \times n$ and $m \times r$ rational matrices respectively, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, and $f \in \mathbb{Q}^r$. We also consider the two following polyhedra in the (x, y) -space:

$$P(x, y) := \{(x, y) \in \mathbb{R}_+^n \times \mathbb{R}_+^r : Ax + Cy \leq b\}, \quad (2.3)$$

$$P_I(x, y) := \text{conv}(\{(x, y) \in P(x, y) : x \text{ integral}\}). \quad (2.4)$$

Split cuts are obtained as follows. For any $\pi \in \mathbb{Z}^n$ and $\pi_0 \in \mathbb{Z}$, the disjunction $\pi^T x \leq \pi_0$ or $\pi^T x \geq \pi_0 + 1$ is of course valid for $P_I(x, y)$, i.e., $P_I(x, y) \subseteq \text{conv}(\Pi_0 \cup \Pi_1)$ where

$$\Pi_0 := P(x, y) \cap \{(x, y) : \pi^T x \leq \pi_0\}, \quad (2.5)$$

$$\Pi_1 := P(x, y) \cap \{(x, y) : \pi^T x \geq \pi_0 + 1\}. \quad (2.6)$$

A valid inequality for $\text{conv}(\Pi_0 \cup \Pi_1)$ is called a *split cut*. The convex set obtained by intersecting $P(x, y)$ with all the split cuts is called the *split closure* of $P(x, y)$. Cook et al. proved that the split closure of $P(x, y)$ is a polyhedron.

Nemhauser and Wolsey [50] introduced the family of *MIR cuts*, whose basic (2-dimensional) version can be obtained in the following way. Let $1 < \hat{b} < 0$ and $\bar{b} \in \mathbb{Z}$, and consider the two-variable mixed integer program $T = \{(x, y) : x + y \geq \hat{b} + \bar{b}, y \geq 0\}$. Then, it is easily seen that the points in T with $x \in \mathbb{Z}$ satisfy the *basic MIR* inequality:

$$\hat{b}x + y \geq \hat{b}(\bar{b} + 1), \quad (2.7)$$

that turns out to be a split cut derived from the disjunction $x \leq \bar{b}$ and $x \geq \bar{b} + 1$. The hardness of separation of split cuts (and hence of MIR inequalities) has been established by Caprara and Letchford [11].

While Chvátal-Gomory cuts are by definition integer inequalities, split/GMI/MIR inequalities are instead mixed integer cuts in the sense that the coefficients are generally not integer and the continuous variables (if any) might have nonzero coefficients.

2.2.1 Pure Integer Cuts

As just mentioned, Chvátal-Gomory cuts are of course pure integer inequalities because of the rounding mechanism and since they do apply only to the pure integer case. In the following section we discuss their separation through a MIP model while in Section 2.2.1.2 we show that a closely related model has been used to separate a new class of pure integer cuts for mixed integer problems.

2.2.1.1 Chvátal-Gomory Cuts

Fischetti and Lodi [27] addressed the issue of evaluating the practical strength of P^1 in approximating P_I . The approach was to model the CG separation problem as a MIP, which is then solved through a general-purpose MIP solver. To be more specific, given an input point $x^* \in P$ to be separated¹, CG-SEP calls for a CG cut $\alpha^T x \leq \alpha_0$ which is (maximally) violated by x^* , where $\alpha = \lfloor u^T A \rfloor$ and $\alpha_0 = \lfloor u^T b \rfloor$ for some $u \in \mathbb{R}_+^m$. Hence, if A_j denotes the j th column of A , CG-SEP can be modeled as:

$$\max \alpha^T x^* - \alpha_0 \tag{2.8}$$

$$\alpha_j \leq u^T A_j \quad \forall j = 1, \dots, n \tag{2.9}$$

$$\alpha_0 + 1 - \epsilon \geq u^T b \tag{2.10}$$

$$u_i \geq 0 \quad \forall i = 1, \dots, m \tag{2.11}$$

$$\alpha_j \text{ integer} \quad \forall j = 0, \dots, n, \tag{2.12}$$

where ϵ is a small positive value. In the model above, the integer variables α_j ($j = 1, \dots, n$) and α_0 play the role of coefficients $\lfloor u^T A_j \rfloor$ and $\lfloor u^T b \rfloor$ in the CG cut, respectively. Hence the objective function (2.8) gives the amount of violation of the CG cut evaluated for $x = x^*$, that is what has to be maximized. Because of the sign of the objective function coefficients, the rounding conditions $\alpha_j = \lfloor u^T A_j \rfloor$ can be imposed through upper bound conditions on variables α_j ($j = 1, \dots, n$), as in (2.9), and with a lower bound condition on α_0 , as in (2.10). Note that this latter constraint requires the introduction of a small value ϵ so as to prevent an integer $u^T b$ being rounded to $u^T b - 1$.

Model (2.8)-(2.12) can also be explained by observing that $\alpha^T x \leq \alpha_0$ is a CG cut if and only if (α, α_0) is an integral vector, as stated in (2.12), and $\alpha^T x \leq \alpha_0 + 1 - \epsilon$ is a valid inequality for P , as stated in (2.9)-(2.11) by using the well-known characterization of valid inequalities for a polyhedron due to Farkas.

¹ Recall that Gomory's work [35] implies that CG-SEP is easy when x^* is an extreme point of P .

2.2.1.2 Projected Chvátal-Gomory Cuts

Bonami et al. [10] extended the concept of Chvátal-Gomory cuts to the mixed integer case. Such an extension is interesting in itself and has the advantage of identifying a large class of cutting planes whose resulting separation problem retains the simple structure of model (2.8)-(2.12) above. One can define the projection of $P(x, y)$ onto the space of the x variables as:

$$P(x) := \{x \in \mathbb{R}_+^n : \text{there exists } y \in \mathbb{R}_+^r \text{ s.t. } Ax + Cy \leq b\} \quad (2.13)$$

$$= \{x \in \mathbb{R}_+^n : u^k A \leq u^k b, \ k = 1, \dots, K\} \quad (2.14)$$

$$=: \{x \in \mathbb{R}_+^n : \bar{A}x \leq \bar{b}\}, \quad (2.15)$$

where u^1, \dots, u^K are the (finitely many) extreme rays of the projection cone $\{u \in \mathbb{R}_+^m : u^T C \geq 0^T\}$. Note that the rows of the linear system $\bar{A}x \leq \bar{b}$ are of Chvátal rank 0 with respect to $P(x, y)$, i.e., no rounding argument is needed to prove their validity.

We define a *projected Chvátal-Gomory (pro-CG) cut* as a CG cut derived from the system $\bar{A}x \leq \bar{b}$, $x \geq 0$, i.e., an inequality of the form $\lfloor w^T \bar{A} \rfloor x \leq \lfloor w^T \bar{b} \rfloor$ for some $w \geq 0$. Since any row of $\bar{A}x \leq \bar{b}$ can be obtained as a linear combination of the rows of $Ax \leq b$ with multipliers $\bar{u} \geq 0$ such that $\bar{u}^T C \geq 0^T$, it follows that a pro-CG cut can equivalently (and more directly) be defined as an inequality of the form:

$$\lfloor u^T A \rfloor x \leq \lfloor u^T b \rfloor \quad \text{for any } u \geq 0 \text{ such that } u^T C \geq 0^T. \quad (2.16)$$

As such, its associated separation problem can be modeled as a simple extension of the system (2.8)-(2.12) by amending it through the following set of inequalities:

$$u^T C_j \geq 0 \quad \forall j = 1, \dots, r. \quad (2.17)$$

Projected Chvátal-Gomory cuts are dominated by split cuts, and therefore $P^1(x, y)$ contains the split closure of $P(x, y)$. More precisely, $P^1(x, y)$ is the intersection of $P(x, y)$ with all the split cuts where one of the sets Π_0 , Π_1 defined in (2.5) and (2.6) is empty (see [10]).

2.2.2 Mixed Integer Cuts

The computational results reported in [27] and [10] showed that P^1 often gives a surprisingly tight approximation of P_I , thus triggering research in the attempt of extending the approach to (more powerful) mixed integer cuts.

Unfortunately, model (2.8)-(2.12) does not extend immediately to the mixed integer case if one wants to concentrate on split/MIR/GMI cuts where coefficients are not necessarily integer and the continuous variables might as-

sume nonzero coefficients in the cut. A natural mixed integer nonlinear model has been suggested in [11]. Variants of such a model have been solved with two different approaches: by solving either a parametric mixed integer problem [7] (Section 2.2.2.1) or a nonlinear mixed integer problem [19, 20] (Section 2.2.2.2).

Finally, it is not difficult to see that one can use the multipliers u computed as in (2.8)-(2.12) or (2.8)-(2.12),(2.17) and write a GMI inequality instead of a CG or pro-CG cut. However, such an *a posteriori* strengthening did not turn out to be very effective (see [10]).

2.2.2.1 Split Cuts Solving a Parametric MIP

Balas and Saxena [7] directly addressed the separation problem of the most violated split cut of the form $\alpha^T x + \gamma^T y \geq \beta$ by looking at the union of the two polyhedra (2.5) and (2.6). In particular, they addressed a generic MIP of the form:

$$\min\{c^T x + f^T y : Ax + Cy \geq b, x \text{ integral}\}, \quad (2.18)$$

where the variable bounds are included among the explicit constraints, and wrote a first nonlinear separation model for split cuts as follows:

$$\min \quad \alpha^T x^* + \gamma^T y^* - \beta \quad (2.19)$$

$$\alpha_j = u^T A_j - u_0 \pi_j \quad \forall j = 1, \dots, n \quad (2.20)$$

$$\gamma_j = u^T C_j \quad \forall j = 1, \dots, r \quad (2.21)$$

$$\alpha_j = v^T A_j + v_0 \pi_j \quad \forall j = 1, \dots, n \quad (2.22)$$

$$\gamma_j = v^T C_j \quad \forall j = 1, \dots, r \quad (2.23)$$

$$\beta = u^T b - u_0 \pi_0 \quad (2.24)$$

$$\beta = v^T b + v_0(\pi_0 + 1) \quad (2.25)$$

$$1 = u_0 + v_0 \quad (2.26)$$

$$u, v, u_0, v_0 \geq 0 \quad (2.27)$$

$$\pi, \pi_0 \quad \text{integer.} \quad (2.28)$$

Normalization constraint (2.26) allows one to simplify the model to the form below:

$$\min u^T (Ax^* + Cy^* - b) - u_0(\pi^T x^* - \pi_0) \quad (2.29)$$

$$u^T A_j - v^T A_j - \pi_j = 0 \quad \forall j = 1, \dots, n \quad (2.30)$$

$$u^T C_j - v^T C_j = 0 \quad \forall j = 1, \dots, r \quad (2.31)$$

$$-u^T b + v^T b + \pi_0 = u_0 - 1 \quad (2.32)$$

$$0 < u_0 < 1, \quad u, v \geq 0 \quad (2.33)$$

$$\pi, \pi_0 \quad \text{integer,} \quad (2.34)$$

where v_0 has been removed by using constraint (2.26), and one explicitly uses the fact that any nontrivial cut has $u_0 < 1$ and $v_0 < 1$ (see Balas and Perregaard [6]). Note that the nonlinearity only arises in the objective function. Moreover, for any fixed value of parameter u_0 the model becomes a regular MIP.

The continuous relaxation of the above model yields a parametric linear program which can be solved by a variant of the simplex algorithm (see, e.g., Nazareth [49]). Balas and Saxena [7] however avoided solving the parametric MIP through a specialized algorithm, and considered a grid of possible values for parameter u_0 , say $u_0^1 < u_0^2 < \dots < u_0^k$. The grid is initialized by means of the set $\{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$ and then is enriched, on the fly, by bisecting a certain interval $[u_0^t, u_0^{t+1}]$ through the insertion of the new grid point $u_0' := (u_0^t + u_0^{t+1})/2$.

2.2.2.2 Split Cuts Solving a Nonlinear MIP

Dash et al. [19, 20] addressed the optimization over the split closure by looking at the corresponding MIR inequalities and, more precisely, developed a mixed integer nonlinear model and linearized it in an effective way.

For the ease of writing the model, we slightly change the definition of polyhedron $P(x, y)$ by putting the constraints in equality form as:

$$P(x, y) = \{(x, y) \in \mathbb{R}_+^n \times \mathbb{R}_+^r : Ax + Cy + Is = b, s \geq 0\}, \quad (2.35)$$

through the addition of nonnegative slack variables s .

One is looking for an MIR inequality in the form:

$$u^+ s + \hat{\gamma}^T y + (\hat{\alpha}^T + \hat{\beta} \bar{\alpha}^T) x \geq \hat{\beta}(\bar{\beta} + 1), \quad (2.36)$$

where $\bar{\alpha}$ and $\bar{\beta}$ are vectors of integer variables, u^+ , $\hat{\alpha}$ and $\hat{\gamma}$ are vectors of nonnegative variables, and $0 < \hat{\beta} < 1$.

Let $\sum_{k \in K} \epsilon_k < 1$ (e.g., $\epsilon_k = 2^{-k}$). We approximate $\hat{\beta}$ with $\sum_{k \in \bar{K}} \epsilon_k$ for some $\bar{K} \subset K$ and write the RHS of the MIR inequality as $\sum_{k \in \bar{K}} \epsilon_k \Delta$ where $\Delta = \bar{\beta} + 1 - \bar{\alpha}^T x^*$. Using the fact that there is a violated MIR inequality if and only if there is one with $\Delta < 1$, we have the following formulation for the separation of the most violated MIR inequality, where for each $k \in K$ we set $\pi_k = 1$ if $k \in \bar{K}$, and $\pi_k = 0$ otherwise.

$$\min u^+ s^* - \epsilon^T \Phi + \hat{\gamma}^T y^* + \hat{\alpha}^T x^* \quad (2.37)$$

$$\hat{\gamma}_j \geq u^T C_j \quad \forall j = 1, \dots, r \quad (2.38)$$

$$\hat{\alpha}_j + \bar{\alpha}_j \geq u^T A_j \quad \forall j = 1, \dots, n \quad (2.39)$$

$$\hat{\beta} + \bar{\beta} \leq u^T b \quad (2.40)$$

$$\hat{\beta} = \sum_{k \in K} \epsilon_k \pi_k \quad (2.41)$$

$$\Delta = (\bar{\beta} + 1) - \bar{\alpha}^T x^* \quad (2.42)$$

$$\Phi_k \leq \Delta \quad \forall k \in K \quad (2.43)$$

$$\Phi_k \leq \pi_k \quad \forall k \in K \quad (2.44)$$

$$u_i^+ \geq u_i \quad \forall i = 1, \dots, M \quad (2.45)$$

$$u^+, \hat{\alpha}, \hat{\beta}, \hat{\gamma} \geq 0 \quad (2.46)$$

$$\bar{\alpha}, \bar{\beta} \text{ integer}, \quad \pi \in \{0, 1\}^{|K|} \quad (2.47)$$

where $u_i^+ = \max\{u_i, 0\}$ and $M := \{i : s_i^* > 0, i = 1, \dots, m\}$, i.e., we define a variable u_i^+ only if the corresponding constraint i written in ‘less or equal form’ is not tight.

The validity of inequality (2.36) can be easily shown. Inequality $u^T s + (\bar{\alpha}^T + \hat{\alpha}^T)x + \hat{\gamma}^T y \geq u^T b$ is valid by Farkas derivation. It remains of course valid by replacing $u_i, \forall i \in M$ with u_i^+ and then one can use the basic MIR inequality (2.7) to obtain the MIR form (2.36) by having as a continuous (nonnegative) part the term $u^+ s + \hat{\alpha}^T x + \hat{\gamma}^T y$.

The approximate model (2.37)–(2.47) turns out to be an exact model if K is chosen appropriately (see [19, 20]).

2.2.3 A Computational Overview

In this section we discuss some simple issues that turn out to be crucial to make the presented models solvable and the MIPping approach successful. Moreover, we show their strength by reporting computational results on MIPs included in the MIPLIB 3.0 [9].

2.2.3.1 Making the Models Solvable

All papers discussed in the previous sections implement pure cutting plane approaches in which (as usual) the following steps are iteratively repeated:

1. the continuous relaxation of the mixed integer program at hand is solved;
2. the separation problem is (heuristically) solved and a set of violated constraints is eventually found;
3. the constraints are added to the original formulation.

Of course, the original formulation becomes larger and larger but in order to provide cuts of rank 1, the separation problem solved at step 2 above only uses the original constraints in the cut derivation. For what concerns the solution of those separation problems, it is important that state-of-the-art MIP solvers such as **ILOG-Cplex** or **Xpress Optimizer** are used, as they incorporate very powerful heuristics that are able to find (and then improve) feasible solutions in short computing times. Indeed, good heuristic solutions are enough for step 2 above, where the NP-hard separation problem does not need to be solved to optimality² since any feasible solution provides a valid inequality cutting off the current solution of step 1 above.

In order to make these MIPs solvable, a few issues have to be addressed.

All authors noted that only integer variables in the support of the fractional solution of step 1 above have to be considered, e.g., a constraint $\alpha_j \leq u^T A_j$ for j such that $x_j^* = 0$ is redundant because α_j (times x_j^*) does not contribute to the violation of the cut, while it can be computed a posteriori by an efficient post-processing procedure. It is easy to see that this is also the case of integer variables whose value is at the upper bound, as these variables can be complemented before separation.

The ultimate goal of the cutting plane sketched above is to find, for each fractional point (x^*, y^*) to be cut off, a “round” of cuts that are significantly violated and whose overall effect is as strong as possible in improving the current LP relaxation. A major practical issue for accomplishing such a goal is the strength of the returned cuts. As a matter of fact, several equivalent solutions of the separation problems typically exist, some of which produce very weak cuts for the MIP model. This is because the separation problem actually considers the face $F(x^*, y^*)$ of P_I where all the constraints that are tight at (x^*, y^*) (including the variable bounds) are imposed as equalities. Hence, for this face there exist several formulations of each cut, which are equivalent for $F(x^*, y^*)$ but not for P_I .

The computational experiments in [27] have shown a relation between the strength of a cut and the sparsity of the vector of multipliers u generating it. In particular, the introduction of a penalty term $-\sum_i w_i u_i$ (where i denotes the index of a constraint) in the objective function (2.8), has the effect of making the cut itself sparser. The sparser the cuts the better for the LP problems solved on step 1 of the cutting plane procedure.³ The importance of making the cuts as sparse as possible has been also documented by Balas and Saxena [7], who noticed that split disjunctions with sparse support tend to give rise to sparse split cuts.

Another important issue in order to accelerate the cutting plane procedure is the cut selection, i.e., finding a set of cuts whose overall behavior is as effective as possible. Cut selection is somehow related to finding a set of cuts which are “as diverse as possible”, possibly more effective together. One can

² Except eventually in the last step, in which one needs a proof that no additional violated cut exists.

³ The same sparsification trick is also used in Bonami et al. [10].

Table 2.1 Results for 25 *pure* integer linear programs in the MIPLIB 3.0.

		Split closure	CG closure
% Gap closed	Average	71.71	62.59
% Gap closed	98-100	9 instances	9 instances
% Gap closed	75-98	4 instances	2 instances
% Gap closed	25-75	6 instances	7 instances
% Gap closed	< 25	6 instances	7 instances

expect that such kind of diversification can be strongly improved with cuts obtained by heuristically solving two or more of the discussed separation models; promising results in this direction have been obtained by combining either CG or pro-CG cuts with MIR inequalities [19, 20].

2.2.3.2 Strength of the Closures

The strength of the closures, namely CG, pro-CG and split (or MIR), have been evaluated by running cutting plane algorithms for large (sometimes huge) computing times. Indeed, the goal of the investigation was in all cases to show the tightness of the closures, rather than investigating the practical relevance of the separation MIPping idea when used within a MIP solver. On the other hand, as discussed in the previous section, several techniques can be implemented to speed up the computation and, even in the current status, the MIPping separation approach is not totally impractical. Indeed, one can easily implement a hybrid approach in which the MIP-based separation procedures are applied (for a fixed amount of time) in a preprocessing phase, resulting in a tighter MIP formulation to be solved at a later time by a standard MIP solver. Using this idea, two unsolved MIPLIB 2003 [2] instances, namely `nsrand-ipx` and `arki001`, have been solved to proven optimality for the first time by Fischetti and Lodi [27] and by Balas and Saxena [7], respectively. In other words, for very difficult and challenging problems it does pay to improve the formulation by adding cuts in these closures before switching to either general- or special-purpose solution algorithms.

In Tables 2.1 and 2.2 we report, in an aggregated fashion, the tightness of the closures for MIPLIB 3.0 [9] instances, in terms of percentage of gap closed⁴ for pure integer and mixed integer linear programs, respectively.

Most of the results reported in the previous tables give a lower approximation of the exact value of the closures⁵, due to the time limits imposed on the cutting plane algorithms. Nevertheless, the picture is pretty clear and

⁴ Computed as $100 - 100(\text{opt_value}(P_I) - \text{opt_value}(P^1)) / (\text{opt_value}(P_I) - \text{opt_value}(P))$.

⁵ In particular, the time limit in [10] to compute a bound of the pro-CG closure is rather short, 20 CPU minutes, and there are pathological instances for which such a closure is ineffective, see [10] for details.

Table 2.2 Results for 33 *mixed* integer linear programs in the MIPLIB 3.0.

		Split closure	pro-CG closure
% Gap closed	Average	84.34	36.38
% Gap closed	98-100	16 instances	3 instances
% Gap closed	75-98	10 instances	3 instances
% Gap closed	25-75	2 instances	11 instances
% Gap closed	< 25	5 instances	17 instances

shows that, although one can construct examples in which the rank of the facets for a polyhedron is very large, in most practical cases the inequalities of rank 1 already give a very tight approximation of the convex hull of integer and mixed integer programs.

2.3 MIPping Heuristics

In this section we consider the problem of finding a feasible (primal) solution to a generic mixed-integer linear program with 0-1 variables of the form:

$$(P) \quad \min c^T x \quad (2.48)$$

$$\text{s.t. } Ax \leq b \quad (2.49)$$

$$x_j \in \{0, 1\} \quad \forall j \in \mathcal{B} \neq \emptyset \quad (2.50)$$

$$x_j \geq 0, \text{ integer } \forall j \in \mathcal{G} \quad (2.51)$$

$$x_j \geq 0 \quad \forall j \in \mathcal{C}, \quad (2.52)$$

where A is an $m \times n$ input matrix, and b and c are input vectors of dimension m and n , respectively. Here, the variable index set $\mathcal{N} := \{1, \dots, n\}$ is partitioned into $(\mathcal{B}, \mathcal{G}, \mathcal{C})$, where $\mathcal{B} \neq \emptyset$ is the index set of the 0-1 variables, while the possibly empty sets \mathcal{G} and \mathcal{C} index the general integer and the continuous variables, respectively. Note that we assume the existence of 0-1 variables, as one of the components of the method we actually implemented (namely, the local branching heuristic) is based on this assumption. Our approach can, however, be extended to remove this limitation, as outlined in the concluding remarks of [25]. Also note that constraints (2.49), though stated as inequalities, can involve equalities as well. Let $\mathcal{I} := \mathcal{B} \cup \mathcal{G}$ denote the index set of all integer-constrained variables.

Heuristics for general-purpose MIPs include [4, 5, 8, 18, 22, 29, 32, 33, 34, 38, 39, 40, 43, 44], among others. Recently, we proposed in [25] a heuristic approach, called *Local Branching* (LB), to improve the quality of a given feasible solution. This method, as well as other refining heuristics (including the recently-proposed RINS approach [18]), requires the availability of a starting

feasible solution, which is an issue for some difficult MIPs. This topic was investigated by Fischetti et al. [24], who introduced the so-called *Feasibility Pump* (FP) scheme for finding a feasible (or, at least, an “almost feasible”) solution to general MIPs through a clever sequence of roundings.

We analyze computationally a simple variant of the original LB method that allows one to deal with infeasible reference solutions, such as those returned by the FP method. Our approach is to start with an “almost feasible” reference solution \bar{x} , as available at small computational cost through the FP method. We then relax the MIP model by introducing for each violated constraint: (i) an artificial continuous variable in the constraint itself, (ii) a binary (also artificial) variable, and (iii) a constraint stating that, if the artificial variable has to be used to satisfy the constraint, then the binary variable must be set to 1. Finally, the objective function is replaced, in the spirit of the first phase of the primal simplex algorithm, by the sum of the artificial binary variables. The initial solution turns out now to be feasible for the relaxed model and its value coincides with the number of initial violated constraints. We then apply the standard LB framework to reduce the value of the objective function, i.e., the number of infeasibilities and a solution of value 0 turns out to be feasible for the initial problem. Note that, although a continuous artificial variable for each violated constraint could be enough, binary variables are better exploited by LB as it will be made clear in Section 2.3.1 and discussed in detail in Section 2.3.2.

Our approach also produces, as a byproduct, a small-cardinality set of constraints whose relaxation (removal) converts a given MIP into a feasible one—a very important piece of information in the analysis of infeasible MIPs. In other words, our method can be viewed as a tool for repairing infeasible MIP *models*, and not just as a heuristic for repairing infeasible MIP *solutions*. This is in the spirit of the widely-studied approaches to find maximum feasible (or minimum infeasible) subsystems of LP models, as addressed e.g. in [3, 12, 31], but is applied here to MIP models. This may be a useful technique in practice.

The section is organized as follows. In Subsection 2.3.1 we review the LB and FP methods. In Subsection 2.3.2 we describe the LB extension we propose to deal with infeasible reference solutions. Computational results are presented in Subsection 2.3.3, where we compare the LB performance with that of the commercial software **ILOG-Cplex** on two sets of hard 0-1 MIPs, specifically 44 problems taken from the MIPLIB 2003 library [2] and 39 additional instances already considered in [24].

2.3.1 Local Branching and Feasibility Pump

We next review the LB and FP methods. The reader is referred to [25] and [24] for more details.

2.3.1.1 Local Branching

The Local Branching approach works as follows. Suppose a feasible *reference solution* \bar{x} of the MIP is given, and one aims at finding an improved solution that is “not too far” from \bar{x} . Let $\bar{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of \bar{x} . For a given positive integer parameter k , we define the k -OPT neighborhood $\mathcal{N}(\bar{x}, k)$ of \bar{x} as the set of the feasible solutions of the MIP satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \bar{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \bar{S}} x_j \leq k, \quad (2.53)$$

where the two terms in the left-hand side count the number of binary variables flipping their value (with respect to \bar{x}) either from 1 to 0 or from 0 to 1, respectively. As its name suggests, the local branching constraint (2.53) can be used as a branching criterion within an enumerative scheme for the MIP. Indeed, given the incumbent solution \bar{x} , the solution space associated with the current branching node can be partitioned by means of the disjunction:

$$\Delta(x, \bar{x}) \leq k \quad (\text{left branch}) \quad \text{or} \quad \Delta(x, \bar{x}) \geq k + 1 \quad (\text{right branch}), \quad (2.54)$$

where the neighborhood-size parameter k is chosen so as to make the neighborhood $\mathcal{N}(\bar{x}, k)$ “sufficiently small” to be optimized within short computing time, but still “large enough” to likely contain better solutions than \bar{x} (typically, $k = 10$ or $k = 20$).

In [25], we investigated the use of a general-purpose MIP solver as a black-box “tactical” tool to explore effectively suitable solution subspaces defined and controlled at a “strategic” level by a simple external branching framework. The procedure is in the spirit of well-known local search metaheuristics, but the neighborhoods are obtained through the introduction in the MIP model of the local branching constraints (2.53). This allows one to work within a perfectly general MIP framework, and to take advantage of the impressive research and implementation effort that nowadays are devoted to the design of MIP solvers. The new solution strategy is exact in nature, though it is designed to improve the heuristic behavior of the MIP solver at hand. It alternates high-level strategic branchings to define solution neighborhoods, and low-level tactical branchings (performed within the MIP solver) to explore them. The result can then be viewed as a two-level branching strategy aimed at favoring early updatings of the incumbent solution, hence producing improved solutions at early stages of the computation. The computational results reported in [25] show the effectiveness of the LB approach. These have also been confirmed by the recent works of Hansen et al. [38] (where LB is used within a Variable Neighborhood Search metaheuristic [48]) and of Fischetti et al. [29] (where MIPs with a special structure are investigated).

2.3.1.2 Feasibility Pump

Let $P := \{x \in \mathbb{R}^n : Ax \leq b\}$ denote the polyhedron associated with the LP relaxation of the given MIP, and assume without loss of generality that system $Ax \leq b$ includes the variable bounds:

$$l_j \leq x_j \leq u_j, \quad \forall j \in \mathcal{I},$$

where $l_j = 0$ and $u_j = 1$ for all $j \in \mathcal{B}$. With a little abuse of terminology, we say that a point x is *integer* if $x_j \in \mathbb{Z}^n$ for all $j \in \mathcal{I}$ (no matter the value of the other components). Analogously, the rounding \tilde{x} of a given x is obtained by setting $\tilde{x}_j := [x_j]$ if $j \in \mathcal{I}$ and $\tilde{x}_j := x_j$ otherwise, where $[\cdot]$ represents scalar rounding to the nearest integer. The (L_1 -norm) distance between a generic point $x \in P$ and a given integer vector \tilde{x} is defined as

$$\Phi(x, \tilde{x}) = \sum_{j \in \mathcal{I}} |x_j - \tilde{x}_j|,$$

(notice that continuous variables x_j , $j \notin \mathcal{I}$, if any, are immaterial) and can be modeled as:

$$\Phi(x, \tilde{x}) := \sum_{j \in \mathcal{I}: \tilde{x}_j = l_j} (x_j - l_j) + \sum_{j \in \mathcal{I}: \tilde{x}_j = u_j} (u_j - x_j) + \sum_{j \in \mathcal{I}: l_j < \tilde{x}_j < u_j} (x_j^+ + x_j^-),$$

where the additional variables x_j^+ and x_j^- require the introduction into the MIP model of the additional constraints:

$$x_j = \tilde{x}_j + x_j^+ - x_j^-, \quad x_j^+ \geq 0, \quad x_j^- \geq 0, \quad \forall j \in \mathcal{I} : l_j < \tilde{x}_j < u_j. \quad (2.55)$$

It follows that the closest point $x^* \in P$ to \tilde{x} can easily be determined by solving the LP:

$$\min\{\Phi(x, \tilde{x}) : Ax \leq b\}. \quad (2.56)$$

If $\Phi(x^*, \tilde{x}) = 0$, then x_j^* ($= \tilde{x}_j$) is integer for all $j \in \mathcal{I}$, so x^* is a feasible MIP solution. Conversely, given a point $x^* \in P$, the integer point \tilde{x} closest to x^* is easily determined by just rounding x^* .

The FP heuristic works with a pair of points (x^*, \tilde{x}) with $x^* \in P$ and \tilde{x} integer, that are iteratively updated with the aim of reducing as much as possible their distance $\Phi(x^*, \tilde{x})$. To be more specific, one starts with any $x^* \in P$, and initializes a (typically infeasible) integer \tilde{x} as the rounding of x^* . At each FP iteration, called a *pumping cycle*, \tilde{x} is fixed and one finds through linear programming the point $x^* \in P$ which is as close as possible to \tilde{x} . If $\Phi(x^*, \tilde{x}) = 0$, then x^* is a MIP feasible solution, and the heuristic stops. Otherwise, \tilde{x} is replaced by the rounding of x^* so as to further reduce $\Phi(x^*, \tilde{x})$, and the process is iterated.

The basic FP scheme above tends to stall and stop prematurely. This happens whenever $\Phi(x^*, \tilde{x}) > 0$ is not reduced when replacing \tilde{x} by the rounding

of x^* , meaning that all the integer-constrained components of \tilde{x} remained unchanged in this iteration. In the original FP approach [24], this situation is dealt with by heuristically choosing a few components \tilde{x}_j to be modified, even if this operation increases the current value of $\Phi(x^*, \tilde{x})$. A different approach, to be elaborated in the next section, is to switch to a method based on enumeration, in the attempt to explore a small neighborhood of the current “almost feasible” \tilde{x} (that typically has a very small distance $\Phi(x^*, \tilde{x})$ from P).

2.3.2 LB with Infeasible Reference Solutions

The basic idea of the method presented in this section is that the LB algorithm does not necessarily need to start with a feasible solution—a partially feasible one can be a valid warm start for the method. Indeed, by relaxing the model in a suitable way, it is always possible to consider any infeasible solution, say \hat{x} , to be “feasible”, and penalize its cost so the LB heuristic can drive it to feasibility.

The most natural way to implement this idea is to add a continuous artificial variable for each constraint violated by \hat{x} , and then penalize the use of such variables in the objective function by means of a very large cost M . We tested this approach and found it performs reasonably well on most of the problems. However, it has the drawback that finding a proper value for M may not be easy in practice. Indeed, for a substantial set of problems in the MIPLIB 2003 [2] collection, the value of the objective function is so large that it is difficult to define a value for M that makes any infeasible solution worse than any feasible one. Moreover, the way the LB method works suggests the use of the following, more combinatorial, framework.

Let T be the set of the indices of the constraints $a_i^T x \leq b_i$ that are violated by \hat{x} . For each $i \in T$, we relax the original constraint $a_i^T x \leq b_i$ into $a_i^T x - \sigma_i \leq b_i$, where $\sigma_i \geq 0$ is a nonnegative continuous artificial variable, and add the constraint:

$$\sigma_i \leq \delta_i y_i, \quad y_i \in \{0, 1\}, \quad (2.57)$$

where δ_i is a sufficiently large value, and y_i is a binary artificial variable attaining value 1 for each violated constraint.⁶ Finally, we replace the original objective function $c^T x$ by $\sum_{i \in T} y_i$, so as to count the number of violated constraints. It has to be noted that the set of binary variables in the relaxed model is $\mathcal{B} \cup \mathcal{Y}$, where $\mathcal{Y} := \{y_i : i \in T\}$, hence the structure of the relaxation turns out to be particularly suited for the LB approach, where the

⁶ Note that when the violated constraint is in equality form two nonnegative artificial variables, σ_i^+ and σ_i^- , are added with opposite signs and the corresponding constraint (2.57) becomes $\sigma_i^+ + \sigma_i^- \leq \delta_i y_i$.

local branching constraint affects precisely the binary variables (including the artificial ones).

An obvious drawback of the method above is that the original objective function is completely disregarded, thus the feasible solution obtained can be arbitrarily bad. A way of avoiding this situation could be to put a term in the artificial objective function that takes the original costs into account. However, a proper balancing of the two contributions (original cost and infeasibility penalty) may not be easy to achieve although promising results in this direction have been reported very recently by Achterberg and Berthold [1]. As a matter of fact, the outcome of a preliminary computational study is that a better overall performance is obtained by using the artificial objective function (alone) until feasibility is reached, and then improving the quality of this solution by using a standard LB or RINS approach. This can be done by recovering the original objective function and simply using the computed feasible solution in the usual LB way. In other words, the overall algorithm remains in principle exact (see [25] for details) and the proposed scheme is used to provide an initial solution.

2.3.3 Computational Results

In this section, we report on computational results comparing the proposed method with both the FP heuristic and the commercial software **IL0G-Cplex** 9.0.3. In our experiments, we used the “asymmetric” version of the local branching constraint (2.53), namely:

$$\Delta(x, \bar{x}) := \sum_{j \in \bar{S}} (1 - x_j). \quad (2.58)$$

Indeed, as discussed in [25], this version of the constraint seems to be particularly suited for set covering problems where LB aims at finding solutions with a small binary support—which is precisely the case of interest in our context.

Our testbed is made up of 33 among the 45 0-1 MIP instances from MIP-LIB 2003 [2] and described in Table 2.3, plus an additional set of 39 hard 0-1 MIPs described in Table 2.4. (The 0-1 MIPLIB instance **stp3d** was not considered since the computing time required for the first LP relaxation is larger than one hour, while 11 instances, namely **fixnet6**, **markshare1**, **markshare2**, **mas74**, **mas76**, **modglob**, **pk1**, **pp08a**, **pp08aCUTS**, **set1ch** and **vpm2** have been removed because all tested algorithms found a feasible solution within 0.0 CPU seconds.) The two tables report the name, total number of variables (n), number of 0-1 variables ($|\mathcal{B}|$), and number of constraints (m) for each instance.

Table 2.3 Set of 33 among the 45 0-1 MIP instances collected in MIPLIB 2003 [2].

Name	n	$ \mathcal{B} $	m	Name	n	$ \mathcal{B} $	m
10teams	2025	1800	230	mod011	10958	96	4480
A1C1S1	3648	192	3312	momentum1	5174	2349	42680
afflow30a	842	421	479	net12	14115	1603	14021
afflow40b	2728	1364	1442	nsrand_ipx	6621	6620	735
air04	8904	8904	823	nw04	87482	87482	36
air05	7195	7195	426	opt1217	769	768	64
cap6000	6000	6000	2176	p2756	2756	2756	755
dano3mip	13873	552	3202	protfold	1835	1835	2112
danoint	521	56	664	qiu	840	48	1192
ds	67732	67732	656	rd-rplusc-21	622	457	125899
fast0507	63009	63009	507	seymour	1372	1372	4944
fiber	1298	1254	363	sp97ar	14101	14101	1761
glass4	322	302	396	swath	6805	6724	884
harp2	2993	2993	112	t1717	73885	73885	551
liu	1156	1089	2178	tr12-30	1080	360	750
misc07	260	259	212	van	12481	192	27331
mkc	5325	5323	3411				

Table 2.4 The additional set of 39 0-1 MIP instances.

Name	n	$ \mathcal{B} $	m source	Name	n	$ \mathcal{B} $	m source
biella1	7328	6110	1203 [25]	blp-ar98	16021	15806	1128 [43]
NSR8K	38356	32040	6284 [25]	blp-ic97	9845	9753	923 [43]
dc1c	10039	8380	1649 [21]	blp-ic98	13640	13550	717 [43]
dc1l	37297	35638	1653 [21]	blp-ir98	6097	6031	486 [43]
dolom1	11612	9720	1803 [21]	CMS750_4	11697	7196	16381 [42]
siena1	13741	11775	2220 [21]	berlin_5_8_0	1083	794	1532 [42]
trento1	7687	6415	1265 [21]	railway_8.1.0	1796	1177	2527 [42]
rail507	63019	63009	509 [25]	usAbbrev.8.25_70	2312	1681	3291 [42]
rail2536c	15293	15284	2539 [25]	manpower1	10565	10564	25199 [53]
rail2586c	13226	13215	2589 [25]	manpower2	10009	10008	23881 [53]
rail4284c	21714	21705	4284 [25]	manpower3	10009	10008	23915 [53]
rail4872c	24656	24645	4875 [25]	manpower3a	10009	10008	23865 [53]
A2C1S1	3648	192	3312 [25]	manpower4	10009	10008	23914 [53]
B1C1S1	3872	288	3904 [25]	manpower4a	10009	10008	23866 [53]
B2C1S1	3872	288	3904 [25]	ljb2	771	681	1482 [18]
sp97ic	12497	12497	1033 [25]	ljb7	4163	3920	8133 [18]
sp98ar	15085	15085	1435 [25]	ljb9	4721	4460	9231 [18]
sp98ic	10894	10894	825 [25]	ljb10	5496	5196	10742 [18]
bg512142	792	240	1307 [47]	ljb12	4913	4633	9596 [18]
dg012142	2080	640	6310 [47]				

The framework described in the previous section has been tested by using different starting solutions \hat{x} provided by FP. In particular, we wanted to test the sensitivity of our modified LB algorithm with respect to the degree of infeasibility of the starting solution, as well as its capability for improving it. Thus, we executed the FP code for 0, 10 and 100 iterations and passed to LB the integer (infeasible) solution \hat{x} with minimum distance $\Phi(x^*, \hat{x})$ from P . (The case with 0 iterations actually corresponds to starting from the solution

of the continuous relaxation, rounded to the nearest integer.) The resulting three versions of the modified LB are called LB_0 , LB_{10} , and LB_{100} , respectively.

In our experiments, we avoided any parameter tuning; FP was implemented exactly as in [24], and for the modified LB code we used a time limit of 30 CPU seconds for the exploration of each local branching neighborhood. As to the value of the neighborhood-size parameter k in LB, we implemented an adaptive procedure: at each neighborhood exploration, we try to reduce the number of violated constraints in the current solution by half, i.e., we set $k = \lfloor |T'|/2 \rfloor$, where $|T'|$ is the value of the current solution. (Since the support of the solution also takes into account non-artificial binary variables, when the number of violated constraints becomes less than 20 we fix $k = 10$, i.e., we use the value suggested in [25] for the asymmetric version of the local branching constraint.) The motivation for this choice is that the number of violated constraints in an initial solution can be extremely large, in which case the use of a small value of k would result in a very slow convergence. A possible drawback is that, in some cases, some of the neighborhoods in the LB sequence can contain no feasible solutions (with respect to the original model) because we do not allow enough artificial variables y to change. The approach can therefore appear counterintuitive, but the idea is that of reducing the neighborhood size iteratively so as to eventually converge.

All codes are written in ANSI C and use the `ILOG-Cplex` callable libraries. The three modified LB codes (LB_0 , LB_{10} , and LB_{100}) are compared with FP and `ILOG-Cplex` 9.0.3 in Table 2.5 for the MIPLIB 2003 instances, and in Table 2.6 for the additional set of instances. Computing times are expressed in CPU seconds, and refer to a Pentium M 1.6 GHz notebook with 512 MByte of main memory. A time limit of 1,800 CPU seconds was provided for each instance with each algorithm and the computation was halted as soon as a first feasible solution was found.

For each instance, we report in both tables: for `ILOG-Cplex`, the number of nodes (nodes) needed to find an initial solution and the corresponding computing time (time); for FP, the number of iterations (FPit) and its computing time (time); for each of the three variants of LB, the computing time spent in the FP preprocessing phase (FP time), the initial number of violated constraints ($|T|$), the number of LB iterations (LBit), and the overall computing time (time). Note that we define an LB iteration as the exploration, generally within a time limit, of the neighborhood of the current solution. Moreover, the time reported is the sum of the time of the FP initialization plus the LB time, thus it can be larger than 1,800 CPU seconds. When one of the algorithms was not able to find a feasible solution in the given time limit, we wrote (*) in column “nodes” (for `ILOG-Cplex`) or “FPit” (for FP), or wrote (μ) in column “ $|T|$ ” near the number of initial infeasible constraints (for LB), where μ is the number of violated constraints in the final solution.

As expected, the degree of infeasibility of the starting solution plays an important role in the LB methods—the better the initial solution, the faster the method. In this view, the FP approach seems to fit particularly well in

Table 2.5 Convergence to a first feasible solution on the MIPLIB 2003 instances.

name	IL0G-Cplex 9.0.3 nodes	time	FP FPit	time	FPtime	LB ₀ T	LBit	time	FPtime	LB ₁₀ T	LBit	time	FPtime	LB ₁₀₀ T	LBit	time
10teams	335	8.4	70	11.7	0.1	75	29	667.7	1.1	18	11	177.4	11.7	—	—	11.7
A1C1S1	150	4.1	8	3.8	0.1	63	5	0.8	3.8	—	—	—	3.8	—	—	3.8
aflow30a	0	0.1	18	0.1	0.0	29	4	3.0	0.1	29	4	0.3	0.1	—	—	0.1
aflow40b	370	5.9	6	0.3	0.1	40	5	57.6	0.3	—	—	0.3	0.3	—	—	0.3
air04	40	8.6	6	74.7	3.4	125	24	671.8	74.7	—	—	74.7	74.7	—	—	74.7
air05	70	3.4	25	83.8	0.8	208	12	135.0	22.8	14	3	25.0	83.8	—	—	83.8
cap6000	0	0.2	2	0.2	0.1	1	1	0.2	0.2	—	—	0.2	0.2	—	—	0.2
dano3mip	0	67.7	2	86.3	65.0	946 (105)	31	1,865.0	86.3	—	—	86.3	86.3	—	—	86.3
danooint	40	1.7	23	1.5	0.1	125	6	16.9	0.6	120	5	3.7	1.5	—	—	1.5
ds	0	55.0	133 (*)	1,800.0	54.5	656	16	582.8	229.9	350	8	302.1	1,358.6	133	6	1,385.0
fast0507	0	39.0	3	46.7	43.4	148	1	45.8	46.7	—	—	46.7	46.7	—	—	46.7
fiber	0	0.1	2	0.0	0.0	41	5	0.5	0.0	—	—	0.0	0.0	—	—	0.0
glass4	5389	1.6	124	0.3	0.0	52	5	0.9	0.0	45	4	0.1	0.2	45	4	0.3
harp2	0	0.0	654	5.0	0.0	9	3	0.9	0.1	6	1	0.1	0.8	6	1	0.8
liu	0	0.1	0	0.1	0.1	—	—	0.1	0.1	—	—	0.1	0.1	—	—	0.1
misc07	67	0.2	78	0.4	0.0	135	7	1.7	0.1	81	6	0.6	0.4	—	—	0.4
mkc	0	0.2	2	0.2	0.1	9	3	2.2	0.2	—	—	0.2	0.2	—	—	0.2
mod011	0	0.2	0	0.1	0.1	—	—	0.1	0.1	—	—	0.1	0.1	—	—	0.1
momentum1	314 (*)	1,800.0	502	1,329.6	1.8	697 (106)	18	1,801.8	42.6	895 (15)	58	1,842.6	178.8	895 (15)	58	1,978.8
net12	203 (*)	1,800.0	1507	225.0	1.8	406	14	246.2	12.9	239	7	16.8	21.8	239	7	25.5
nsrand_ipx	0	0.5	4	0.9	11.3	390	8	14.1	0.9	—	—	—	0.9	—	—	—
nw04	1	4.9	1	4.6	0.3	6	2	6.8	4.6	—	—	4.6	4.6	—	—	4.6
opt1217	117	0.1	0	0.0	0.0	—	—	0.0	0.0	—	—	0.0	0.0	—	—	0.0
p2756	0	0.1	150023 (*)	1,800.0	0.0	41	6	0.8	0.1	19	1	0.2	1.2	19	1	1.3
proftold	190	640.9	367	502.0	2.7	37 (37)	7	1,802.7	16.1	13 (1)	50	1,816.1	125.6	7 (1)	55	1,925.6
qu	0	0.2	5	0.3	0.1	132	1	0.2	0.3	—	—	—	0.3	—	—	0.3
rd-rplusc-21	10978 (*)	1,800.0	401 (*)	1,800.0	3.9	119021 (7094)	23	1,803.9	36.8	119017 (1)	71	1,836.8	449.5	119017 (2)	75	2,249.5
seymour	0	3.5	7	3.6	3.0	921	1	3.8	3.6	—	—	—	3.6	—	—	—
sp97ar	0	3.4	4	4.2	2.9	222	1	3.8	4.2	20	6	70.8	2.9	—	—	2.9
swath	0	0.2	49	2.9	0.1	20	6	124.6	1.0	20	6	70.8	2.9	—	—	2.9
tl1717	710	301.0	40	814.8	10.7	445 (50)	25	1,810.7	133.2	108 (5)	35	1,933.2	814.8	—	—	814.8
tr12-30	179	0.9	8	0.1	0.0	348	8	0.6	0.1	—	—	0.1	0.1	—	—	0.1
van	0	872.8	10	300.5	27.4	192 (128)	9	1,827.4	300.5	—	—	300.5	300.5	—	—	300.5

Table 2.6 Convergence to a first feasible solution on the additional set of 0-1 MIP instances.

name	ILoG-Cplex 9.0.3		FP		LB ₀		LB ₁₀		LB ₁₀₀			
	nodes	time	FPit	time	FPtime	[T]	LBit	time	FPtime	[T]	LBit	time
biella1	594	108.4	4	2.8	2.3	1193	9	18.2	2.8	—	—	2.8
NSR8K	5 (*)	1,800.0	3	195.5	185.8	5488 (5488)	1	1,985.8	195.5	—	—	195.5
dcllc	4749	474.0	2	12.7	11.6	1483	11	81.6	12.7	—	—	12.7
dcll	0	80.8	2	16.2	14.0	1567	1	14.8	16.2	—	—	16.2
dolom1	367	504.4	22	22.6	11.9	1410	12	277.1	17.7	632	8	49.4
sienal	600	1,371.5	3	43.7	40.6	1750	12	271.2	43.7	—	—	43.7
trento1	340	276.8	7	11.0	9.3	603	8	22.6	11.0	—	—	11.0
rail507	0	32.8	2	8.7	6.5	218	1	7.4	8.7	—	—	8.7
rail2536c	0	16.8	1	15.2	14.3	2008	1	14.9	15.2	—	—	15.2
rail2586c	0	63.9	1	8.3	7.6	1871	1	7.9	8.3	—	—	8.3
rail4284c	0	204.9	2	56.7	53.5	3305	1	54.2	56.7	—	—	56.7
rail4872c	0	186.4	2	19.3	17.5	3254	1	18.3	19.3	—	—	19.3
A2C1S1	0	0.1	5	4.7	0.1	60	1	0.2	4.7	—	—	4.7
B1C1S1	0	0.1	6	5.0	0.1	208	1	0.2	5.0	—	—	5.0
B2C1S1	0	0.1	7	4.7	0.1	217	1	0.3	4.7	—	—	4.7
sp97ic	0	2.4	3	3.1	1.7	173	1	2.4	3.1	—	—	3.1
sp98ar	0	3.8	3	5.2	3.5	260	7	23.6	5.2	—	—	5.2
sp98ic	0	2.1	2	2.6	1.8	147	6	6.0	2.6	—	—	2.6
blp-ar98	8300	158.3	835	122.9	0.5	212	8	31.7	2.5	204	7	16.4
blp-ic97	1120	16.2	8	1.3	0.3	59	5	5.5	1.3	—	—	1.3
blp-ic98	1570	33.6	3	1.5	0.9	76	5	5.0	1.5	—	—	1.5
blp-ir98	1230	8.1	4	0.4	0.1	37	4	1.3	0.4	—	—	0.4
CM5750.4	940	27.2	16	6.5	0.7	2446	1	9.2	3.3	2441	1	11.7
berlin_5.8.0	152	0.4	13	0.2	0.0	170	20	275.4	0.1	167	1	0.2
railway_8.1.0	350	1.3	12	0.3	0.1	374	17	358.4	0.2	373	1	0.5
usAbbrv.8.25_70	274581	1,371.5	31	0.7	0.1	400	1	0.6	0.3	376	1	0.8
bg512142	0	0.3	0	0.2	0.2	—	—	0.2	0.2	—	—	0.2
dq012142	0	1.0	0	0.8	0.8	—	—	0.8	0.8	—	—	0.8
manpower1	154 (*)	1,800.0	30	18.8	8.4	1142	15	108.7	13.4	336	9	52.0
manpower2	150	364.6	92	137.5	39.5	1181	30	774.2	73.6	309	13	394.8
manpower3	181	326.9	42	76.2	27.1	1160	23	534.7	55.5	427	17	363.3
manpower3a	181	925.1	293	294.1	30.6	1327 (7)	57	1,830.6	53.3	369	18	491.2
manpower4	185	671.0	208	138.9	14.3	1105	34	1,010.8	41.8	604	19	427.7
manpower4a	194	1,039.9	308	289.2	36.4	1226	37	814.8	69.1	483	18	440.0
lib2	30	0.2	0	0.0	0.0	—	—	0.0	0.0	—	—	0.0
lib7	100	3.8	0	0.6	0.6	—	—	0.6	0.6	—	—	0.6
lib9	180	7.0	0	0.8	0.8	—	—	0.8	0.8	—	—	0.8
lib10	90	5.9	0	1.1	1.1	—	—	1.1	1.1	—	—	1.1
lib12	110	5.8	0	0.7	0.7	—	—	0.7	0.7	—	—	0.7

our context, in that it is able to provide very good solutions (as far as the degree of infeasibility is concerned) in very short computing times. Among the three LB implementations, LB_0 failed eight times in finding a feasible solution within the time limit, LB_{10} four times, and LB_{100} only three times. Among the 64 instances for which the three LB implementations found a feasible solution within the time limit, LB_0 was at least as fast as the other two in 26 cases, LB_{10} in 34 cases, and LB_{100} in 42 cases. Overall, LB_{100} qualifies as the most effective (and stable) of the three methods.

A comparison between **ILOG-Cplex** and LB_{100} shows that:

1. **ILOG-Cplex** was not able to find any feasible solution (within the 1,800 second time limit) in five cases, whereas LB_{100} was unsuccessful three times;
2. among the 66 instances for which both algorithms found a feasible solution within the time limit, **ILOG-Cplex** was strictly faster in 21 cases, while the opposite holds in 41 cases;
3. among the same 66 instances, the average computing time for finding a feasible solution for **ILOG-Cplex** was 146.7 CPU seconds, while for LB_{100} it was 65.0 CPU seconds.

As expected, the quality of the initial **ILOG-Cplex** solution (not reported in the tables) is typically better than that provided by the LB methods. More precisely, the geometric mean of the ratio between the solution found by an algorithm and best solution is 2.28, 2.22, 2.11 and 1.13 for the three LB implementations and **ILOG-Cplex**, respectively. As noted in Section 2.3.2, however, the first solution can be easily improved by standard use of the LB algorithm. As an example, on instance **dc1c** the ratio of the solution obtained by algorithm LB_0 with respect to the first solution computed by **ILOG-Cplex** is 12.16. However, the ratio reduces significantly by applying LB, and becomes 4.83, 2.74, and 1.02 in the first three iterations, respectively, and reaches value 0.77 (i.e., the **ILOG-Cplex** solution is eventually improved) in the fourth one. Those four iterations take 138.1 CPU seconds, plus 81.6 seconds to find the first solution, thus the overall computing time of 219.7 CPU seconds is less than a half of the time spent by **ILOG-Cplex** to find its first solution, namely 474.0 CPU seconds.

This very satisfactory behavior is unfortunately not confirmed on other instances, though local search heuristics such as LB or RINS [18] generally improve the first solution considerably. On the other hand, an effective exploitation of a first feasible solution within an enumerative algorithm is by itself a relevant (and difficult) research topic, that recently started to receive considerable attention in the field.

2.4 MIPping the Dominance Test

In the standard B&B (or B&C) framework, a node is fathomed in two situations:

1. the LP relaxation of the node is infeasible;
2. the LP relaxation optimum is not better than the value of the incumbent optimal solution.

There is, however, a third way of pruning a node, by using dominances. According to [51], a dominance relation is defined as follows: if we can show that the best descendant of a node β is at least as good as the best descendant of a node α , then we say that node β *dominates* node α , meaning that the latter can be fathomed (in case of ties, an appropriate rule has to be taken into account in order to avoid fathoming cycles). Unfortunately, this definition may become useless in the context of general MIPs, where we do not actually know how to perform the dominance test without storing huge amounts of information for all the previously-generated nodes — which is often impractical.

Fischetti and Toth [30] proposed a different (and more “local”) dominance procedure which overcomes many of the drawbacks of the classical definition, and resembles somehow the *isomorphic-pruning* introduced recently by Margot [45]. Here is how the procedure works.

Let the MIP problem at hand denoted as:

$$\min\{c^T x : x \in \mathbb{R}_+^n, Ax \leq b, x_j \text{ integer for all } j \in J\}, \quad (2.59)$$

where $J \subseteq I := \{1, \dots, n\}$ is the index-set of the integer variables. For any $J' \subseteq J$ and for any $x' \in \mathbb{R}_+^n$, we denote as:

$$c(J', x') := \sum_{j \in J'} c_j x'_j,$$

the contribution of the variables in J' to the overall cost $c^T x'$. Now, let us suppose to solve problem (2.59) by an enumerative (B&B or B&C) algorithm whose branching rule fixes some of the integer-constrained variables to some values. For every node k of the search tree, let $J^k \subseteq J$ denote the set of indices of the variables x_j fixed to a certain value x_j^k (say). Every solution x such that $x_j = x_j^k$ for all $j \in J^k$ (i.e., belonging to the subtree rooted at node k) is called a *completion* of the partial solution associated at node k .

Definition 1. Let α and β be two nodes of the search tree. Node β dominates node α if:

1. $J^\beta = J^\alpha$;
2. $c(J^\beta, x^\beta) \leq c(J^\alpha, x^\alpha)$, i.e., the cost of the partial solution at node β is not worse than that at node α ;

3. every completion of the partial solution associated with node α is also a completion of the partial solution associated with node β .

Clearly, according to the classical dominance theory, the existence of a node β unfathomed that dominates node α is a sufficient condition to fathom node α . A key question at this point is: Given the current node α , how can we check the existence of a dominating node β ? Fischetti and Toth answered this question by modeling the search of dominating nodes as a structured optimization problem, to be solved exactly or heuristically. For generic MIP models, this leads to the following *auxiliary problem*:

$$\begin{cases} XP^\alpha : & \min \sum_{j \in J^\alpha} c_j x_j \\ \text{s.t.} & \sum_{j \in J^\alpha} A_j x_j \leq b^\alpha := \sum_{j \in J^\alpha} A_j x_j^\alpha \\ & x_j \text{ integer for all } j \in J^\alpha \end{cases} \quad (2.60)$$

If a solution x^β (say) of the auxiliary problem having a cost strictly smaller than $c(J^\alpha, x^\alpha)$ is found, then it defines a dominating node β and the current node α can be fathomed.

It is worth noting that the auxiliary problem is of the same nature as the original MIP problem, but with a smaller size and thus it is often easily solved (possibly in a heuristic way) by a general-purpose MIP solver, so we are indeed “MIPping the dominance test”.

The Fischetti-Toth dominance procedure, called *Local Dominance* (LD) procedure in the sequel to stress its local nature, has several useful properties:

- there is no need to store any information about the set of previously generated nodes;
- there is no need to make any time-consuming comparison of the current node with other nodes;
- a node can be fathomed even if the corresponding dominating one has not been generated yet;
- the correctness of the enumerative algorithm does not depend on the branching rule; this is a valuable property since it imposes no constraints on the B&B parameters (though an inappropriate branching strategy could prevent several dominated nodes to be fathomed).

In addition, the LD test needs not be applied at every node. This is a crucial property from the practical point of view, as the dominance test introduces some overhead and it would make the algorithm uncompetitive if applied at every node. Note that skipping a LD test at a given node is not likely to induce a great pruning loss, since the following *inheritance* property holds (see [54] for the proof):

Proposition 1. *Let α and β be two nodes of the search tree and let β dominate α . Then for every α' successor of α there exists a node β' such that β' dominates α' .*

As a consequence, if β dominates α and α is not fathomed because the corresponding dominance test was skipped, we still have the possibility to prune some descendant nodes of α .

An important issue to be addressed when implementing the LD test is to avoid fathoming cycles arising when the auxiliary problem actually has a solution x^β different from x^α but of the same cost, in which case one is allowed to fathom node α only if a tie-break rule is used to guarantee that node β itself is not fathomed for the same reason. In order to prevent these “tautological” fathoming cycles the following criterion (among others) has been proposed in [30]: In case of cost equivalence, define as unfathomed the node β corresponding to the solution found by a *deterministic*⁷ exact or heuristic algorithm used to solve the auxiliary problem. Unfortunately, this criterion can be misleading for two important reasons. First of all, it is not easy to define a “deterministic” algorithm for MIP. In fact, besides the possible effects of randomized steps, the output of the MIP solver typically depends, e.g., on the order in which the variables are listed on input, that can affect the choice of the branching variables as well as the internal heuristics.

In view of the considerations above, in our implementation we used a different tie-break rule, also described in [30], that consists in ranking cost-equivalent solutions in lexicographical order. To be more specific, in case of cost ties we fathom node α if and only if the partial solution x^β associated with the dominating node β is lexicographically smaller⁸ than x^α . Using this tie-breaking rule, it is easy to prove [54] the correctness of the overall enumerative method.

2.4.1 *Borrowing Nogoods from Constraint Programming*

The computational overhead related to the LD test can be reduced considerably if we exploit the notion of nogoods taken from Constraint Programming. A *nogood* is a partial assignment of the problem variables such that every completion is either infeasible (for constraint satisfaction problems) or nonoptimal (for constraint optimization problems). The key observation here is that whenever we discover (through the solution of the auxiliary problem) that the current node α is dominated, we have indeed found a *nogood configuration* $[J^\alpha, x^\alpha]$ that we want to exclude from being re-analyzed at a later time.

There are two possible ways of exploiting nogoods in the context of MIP solvers:

⁷ In our context, an algorithm is said to be deterministic if it always provides the same output solution for the same input set.

⁸ We use the standard definition of lexicographic order on vectors of fixed size over a totally ordered set.

- Generate a constraint $\alpha^T x \leq \alpha_0$ cutting the nogood configuration off, so as to prevent it appears again in a later fractional solution. This is always possible (for both binary and general-integer linear problems) through a local branching constraint [25], and leads to the so-called *combinatorial Benders cuts* studied by Codato and Fischetti [14].
- Maintain explicitly a pool of previously found nogood configurations and solve the following problem (akin to separation) at each node α to be tested: Find, if any, a nogood configuration $[J', x']$ stored in the pool, such that $J' \subseteq J^\alpha$ and $x'_j = x_j^\alpha$ for all $j \in J'$. If the test is successful, we can of course fathom node α without the need of constructing and solving the auxiliary problem XP^α .

In our implementation we use the nogood-pool option, that according to our computational experience outperforms the cut options. It is worth noting that we are interested in minimal (with respect to set inclusion) nogoods, so as to improve both for efficiency and effectiveness of the method. Indeed, if node β dominates node α and $J' := \{j \in J^\alpha : x_j^\alpha \neq x_j^\beta\}$, then clearly the restriction of x^β onto J' dominates the restriction of x^α onto J' . If applied at every node, our procedure guarantees automatically the minimality of the nogood configurations found. If this is not the case, instead, minimality is no longer guaranteed, and is enforced by a simple post-processing step before storing any new nogood in the pool.

2.4.2 Improving the Auxiliary Problem

The effectiveness of the LD test presented in the previous section heavily depends on the auxiliary problem that is constructed at a given node α . In particular, it is crucial for its solution set to be as large as possible, so as to increase the chances of finding a dominating partial solution. Moreover, we aim at finding a partial solution different from (and hopefully lexicographically better than) the one associated with the current node; finding the same solution x^α is of no use within the LD context. For these reasons, several improvements of the original auxiliary-problem formulation have been proposed in [54], as outlined below.

The auxiliary problem XP^α constructed at node α is always feasible, as the partial assignment x^α corresponding to node α itself is always feasible. This is not a desired behavior, for two main reasons:

- Often x^α turns out to be the only feasible solution to XP^α —for our purposes, it would be better to consider it as infeasible, meaning that the node cannot be fathomed by our procedure.
- When solving the auxiliary problem, the solver often finds solution x^α (even if it is not provided explicitly on input for initializing the incum-

bent) and proves its optimality without looking for alternative (hopefully lexicographically better) optimal solutions.

Moreover, as the depth of the nodes in the B&B increases, the auxiliary problem grows in size and becomes heavier to solve. In addition, the resulting nogood (if any) may be of little applicability in the remaining part of the search because it may involve too many variables.

For these reasons one can heuristically limit the search space of the auxiliary problem to alternative assignments that are not too far from the current one, but different from it. This can be achieved again with two local branching [25] constraints, which however, in the most general case, could need the introduction of complicating auxiliary variables. According to our computational experience, a good compromise is to consider local branching constraints involving only the (binary or general integer) variables fixed to their lower or upper bound, namely:

$$\sum_{j \in U} (u_j - x_j) + \sum_{j \in L} (x_j - l_j) \leq k, \quad (2.61)$$

$$\sum_{j \in U} (u_j - x_j) + \sum_{j \in L} (x_j - l_j) \geq 1, \quad (2.62)$$

where

$$U = \{j \in J^\alpha \mid x_j^\alpha = u_j\} \quad \text{and} \quad L = \{j \in J^\alpha \mid x_j^\alpha = l_j\}.$$

It is worth noting that the above constraint may exclude some feasible solutions that differ from x^α with respect to variables fixed to values different from a lower or upper bound. In this case, our fathoming test can become less powerful, but the overall method remains correct. Finally, we found it useful to add the following *optimality* constraint

$$\sum_{j \in J^\alpha} c_j x_j \leq \sum_{j \in J^\alpha} c_j x_j^\alpha.$$

2.4.3 Computational Results

The enhanced dominance procedure presented in the previous section has been implemented in C++ within the ILOG-Cplex [41] framework on a Linux platform. Here are some implementation issues that deserve further description.

One of the main drawbacks of LD tests is that they postpone finding a better incumbent solution, thus increasing the number of nodes needed to solve the problem. This behavior is quite undesirable, especially in the first phase of the algorithm, when we have no incumbent and no nodes can be

fathomed through bounding criteria. A practical solution to this problem is to skip the dominance test until the first feasible solution is found.

The systematic application of the dominance test to every node of the search tree can become too heavy to be worthwhile in practice. A first consideration is that we should skip the dominance test on nodes near the top or the bottom of the search tree. Indeed, in the first case only a few variables have been fixed, hence there are little chances of finding a dominating partial assignment. In the latter case, instead, it is likely that the node would be pruned anyway by standard bounding tests; moreover, at the bottom of the tree the number of fixed variables is large and the auxiliary problem may be quite heavy to solve. In our implementation, we provide two thresholds on the tree depth of the nodes, namely $depth_{\min}$ and $depth_{\max}$, a node α being tested for dominance only if $depth_{\min} \leq \text{depth}(\alpha) \leq depth_{\max}$. Moreover, we decided to test for dominance a node only if its depth is a multiple of a given parameter, $depth_interval$. The three parameters above have been set as relative percentages on the number of variables. Finally, we set a limit on the computing time spent by the black-box MIP solver used for solving each auxiliary problem.

In our computational experiments we tested ILOG-Cplex 9.0 [41] commercial code with and without our LD test. All runs were performed on a AMD Athlon64 3500+ PC with 4GB of RAM, under Linux. The ILOG-Cplex code was run with its default options, and the overall time limit for processing each instance was set to 2,000 CPU seconds. As to LD tests, we used the following parameters:

- $depth_min = 0.3$ times the total number of variables;
- $depth_max = 0.8$ times the total number of variables;
- $depth_interval = 0.1$ times the total number of variables.

Moreover, in this implementation we did not use local branching constraint (2.61). The definition of the test-bed for testing the potentiality of our approach is a delicate issue. As a matter of fact, one cannot realistically expect any dominance relationship to be effective on all types of MIPs. Therefore, we looked for classes of problems whose structure can trigger the dominance relationship, and measured the speedup that can be achieved by using our specific LD procedure. In particular, we next give results on single and multiple knapsack problems [46]. We generated hard single knapsack instances according to the so-called *spanner instances* method in combination with the *almost strongly correlated* profit generation technique; see Pisinger [52] for details. Multiple knapsack instances were generated in a similar way, by choosing a same capacity for all the containers.

The results on hard single knapsack instances with 60 to 90 items are given in Table 2.7.

According to the table, LD tests are very effective on this class of problems, yielding consistently a large speedup and solving to optimality three instances where the standard code reached the time limit. It is worth noting that little

Table 2.7 Computational results for hard single knapsack instances.

Problem	Standard Cplex			Dominance Code			Ratio	
	Nodes	Time (s)	Gap	Nodes	Time (s)	Gap	Nodes	Time
kp60.1	286,056	9.60	0	725	0.04	0	394.56	240.00
kp60.2	27,108,819	2,050.82	0.016	773,890	2,067.44	0.028	35.03	1.00
kp60.3	718,887	24.89	0	1330	0.34	0	540.52	73.21
kp60.4	804,304	26.32	0	28,947	12.17	0	27.79	2.16
kp60.5	688,122	24.36	0	48,895	4.87	0	14.07	5.00
kp70.1	23,671,129	2,050.52	0.406	1,638,641	2,047.53	0.406	14.45	1.00
kp70.2	1,060,259	35.43	0	153,552	61.54	0	6.90	0.58
kp70.3	665,668	23.12	0	147,899	28.00	0	4.50	0.83
kp70.4	23,037,172	2,048.61	0.399	935,986	2,065.34	0.216	24.16	1.00
kp70.5	424,815	15.17	0	19,685	0.89	0	21.58	17.04
kp80.1	413,489	13.98	0	249,582	10.39	0	1.66	1.35
kp80.2	587,456	22.54	0	140,191	7.25	0	4.19	3.11
kp80.3	673,318	22.61	0	26,803	2.13	0	25.12	10.62
kp80.4	529,026	17.56	0	5,274	0.24	0	100.31	73.17
kp80.5	32,604,432	2,050.79	0.328	460,908	109.26	0	70.74	18.77
kp90.1	25,409,911	2,047.65	0.065	928,586	2,034.52	0.065	27.36	1.00
kp90.2	37,650,100	2,041.93	0.137	3,957,332	167.81	0	9.51	12.17
kp90.3	3,024,346	126.59	0	266,137	16.82	0	11.36	7.53
kp90.4	1,926,498	81.39	0	134,385	10.25	0	14.34	7.94
kp90.5	26,510,264	2,052.64	0.263	1,047,483	551.93	0	25.31	3.72
Total	207,794,071	14,787.34	-	10,966,231	9,198.76	-	18.95	1.61

Table 2.8 Parameter tuning on specific hard single knapsack instances.

Problem	Nodes	Time (s)	Gap	Depth Min	Depth Max
kp60.2	1,653,691	68.24	0	0.3	0.6
kp70.2	7,763	1.02	0	0.3	0.8
kp80.2	11,171	0.57	0	0.3	0.8
kp80.3	2,955	0.30	0	0.3	0.6

parameter tuning would have produced better results in terms of elapsed time and/or final gap for four instances, as in Table 2.8.

As to hard multiple knapsack problems, we have generated instances with a number of items ranging from 20 to 40 and a number of knapsacks ranging from 3 to 5. The LD parameters were set to:

- *depth_min* = 0.3 times the total number of variables;
- *depth_max* = 0.5 times the total number of variables;
- *depth_interval* = 0.1 times the total number of variables.

For these problems, the time limit was increased to one hour.

The results on multiple knapsack problems are available in Table 2.9.

In the multiple knapsack case, LD was not as effective as in the single case, yielding some improvements only on the smallest instances. It is how-

Table 2.9 Computational results for hard multiple knapsack problems.

Problem	Standard Cplex			Dominance Code			Ratio	
	Nodes	Time (s)	Gap	Nodes	Time (s)	Gap	Nodes	Time
mkp20.3.lp	187,713	16.13	0	129,516	12.18	0	1.45	1.32
mkp20.4.lp	80,845	11.20	0	79,650	10.20	0	1.02	1.10
mkp20.5.lp	1,639,251	171.07	0	1,563,925	181.96	0	1.05	0.94
mkp30.3.lp	33,205,268	3,652.63	0.627	31,141,581	3,654.65	0.632	1.07	1.00
mkp30.4.lp	5,414,529	650.24	0	26,707,752	3,649.55	0.439	0.20	0.18
mkp30.5.lp	28,159,141	3,644.38	0.515	25,653,218	3,649.69	0.515	1.10	1.00
mkp40.3.lp	40,576,080	3,654.31	0.515	22,963,265	3,699.83	0.515	1.77	0.99
mkp40.4.lp	25,354,639	3,645.51	0.437	21,250,331	3,652.06	0.437	1.19	1.00
mkp40.5.lp	777,810	160.81	0	389,427	128.98	0	2.00	1.25
Total	135,395,276	15,606.28	-	129,878,665	18,639.10	-	1.04	0.84

ever worth mentioning that the ratio *dominated_nodes/dominance_tests* was quite good also for these problems (though lower than in the single knapsack case) and that the effectiveness of LD could have been hidden by the limited computational time given to the solvers.

Acknowledgements This work was supported by the Future and Emerging Technologies unit of the EC (IST priority), under contract no. FP6-021235-2 (project “ARRIVAL”) and by MiUR, Italy.

References

1. T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4:77–86, 2007.
2. T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34:361–372, 2006. Problems available at <http://miplib.zib.de>.
3. E. Amaldi, M.E. Pfetsch, and L.E. Trotter Jr. On the maximum feasible subsystem problem, IISs and IIS-hypergraphs. *Mathematical Programming*, 95:533–554, 2003.
4. E. Balas, S. Ceria, M. Dawande, F. Margot, and G. Pataki. OCTANE: A new heuristic for pure 0–1 programs. *Operations Research*, 49:207–225, 2001.
5. E. Balas and C.H. Martin. Pivot-and-complement: A heuristic for 0-1 programming. *Management Science*, 26:86–96, 1980.
6. E. Balas and M. Perregaard. Lift-and-project for mixed 0-1 programming: Recent progress. *Discrete Applied Mathematics*, 123:129–154, 2002.
7. E. Balas and A. Saxena. Optimizing over the split closure. *Mathematical Programming*, 113:219–240, 2008.
8. E. Balas, S. Schmieta, and C. Wallace. Pivot and shift — a mixed integer programming heuristic. *Discrete Optimization*, 1:3–12, 2004.
9. R.E. Bixby, S. Ceria, C.M. McZeal, and M.W.P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.
10. P. Bonami, G. Cornuéjols, S. Dash, M. Fischetti, and A. Lodi. Projected Chvátal–Gomory cuts for mixed integer linear programs. *Mathematical Programming*, 113:241–257, 2008.

11. A. Caprara and A.N. Letchford. On the separation of split cuts and related inequalities. *Mathematical Programming*, 94:279–294, 2002.
12. J.W. Chinneck. Fast heuristics for the maximum feasible subsystem problem. *INFORMS Journal on Computing*, 13:210–223, 2001.
13. V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 5:305–337, 1973.
14. G. Codato and M. Fischetti. Combinatorial Benders cuts. In D. Bienstock and G. Nemhauser, editors, *Integer Programming and Combinatorial Optimization, IPCO X*, volume 3064 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2004.
15. W. Cook, R. Kannan, and A. Schrijver. Chvatal closures for mixed integer programming problems. *Mathematical Programming*, 47:155–174, 1990.
16. G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112:3–44, 2008.
17. G. Cornuéjols and Y. Li. On the rank of mixed 0,1 polyhedra. *Mathematical Programming*, 91:391–397, 2002.
18. E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90, 2005.
19. S. Dash, O. Günlük, and A. Lodi. On the MIR closure of polyhedra. In M. Fischetti and D.P. Williamson, editors, *Integer Programming and Combinatorial Optimization, IPCO XII*, volume 4513 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2007.
20. S. Dash, O. Günlük, and A. Lodi. MIR closures of polyhedral sets. *Mathematical Programming*, DOI 10.1007/s10107-008-0225-x, 2008.
21. Double-Click sas. personal communication, 2001.
22. J. Eckstein and M. Nediak. Pivot, cut, and dive: a heuristic for 0-1 mixed integer programming. *Journal of Heuristics*, 13:471–503, 2007.
23. F. Eisenbrand. On the membership problem for the elementary closure of a polyhedron. *Combinatorica*, 19:297–300, 1999.
24. M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005.
25. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.
26. M. Fischetti and A. Lodi. MIPping Closures: An instant survey. *Graphs and Combinatorics*, 23:233–243, 2007.
27. M. Fischetti and A. Lodi. Optimizing over the first Chvátal closure. *Mathematical Programming*, 110:3–20, 2007.
28. M. Fischetti and A. Lodi. Repairing MIP infeasibility through local branching. *Computers & Operations Research*, 35:1436–1445, 2008.
29. M. Fischetti, C. Polo, and M. Scantamburlo. A local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks*, 44:61–72, 2004.
30. M. Fischetti and P. Toth. A New Dominance Procedure for Combinatorial Optimization Problems. *Operations Research Letters*, 7:181–187, 1988.
31. J. Gleeson and J. Ryan. Identifying minimally infeasible subsystems of inequalities. *ORSA Journal on Computing*, 2:61–63, 1990.
32. F. Glover and M. Laguna. General purpose heuristics for integer programming – part I. *Journal of Heuristics*, 2:343–358, 1997.
33. F. Glover and M. Laguna. General purpose heuristics for integer programming – part II. *Journal of Heuristics*, 3:161–179, 1997.
34. F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
35. R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.
36. R.E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The Rand Corporation, 1960.

37. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.
38. P. Hansen, N. Mladenović, and D. Urošević. Variable neighborhood search and local branching. *Computers & Operations Research*, 33:3034–3045, 2006.
39. F.S. Hillier. Efficient heuristic procedures for integer linear programming with an interior. *Operations Research*, 17:600–637, 1969.
40. T. Ibaraki, T. Ohashi, and F. Mine. A heuristic algorithm for mixed-integer programming problems. *Mathematical Programming Study*, 2:115–136, 1974.
41. ILOG S.A. *CPLEX: ILOG CPLEX 11.0 User's Manual and Reference Manual*, 2007. <http://www.ilog.com>.
42. G.W. Klau. personal communication, 2002.
43. A. Løkketangen. Heuristics for 0-1 mixed-integer programming. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 474–477. Oxford University Press, 2002.
44. A. Løkketangen and F. Glover. Solving zero/one mixed integer programming problems using tabu search. *European Journal of Operational Research*, 106:624–658, 1998.
45. F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94:71–90, 2002.
46. S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.
47. A.J. Miller. personal communication, 2003.
48. N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.
49. J.L. Nazareth. The homotopy principle and algorithms for linear programming. *SIAM Journal on Optimization*, 1:316–332, 1991.
50. G. Nemhauser and L. Wolsey. A recursive procedure to generate all cuts for 0-1 mixed integer programs. *Mathematical Programming*, 46:379–390, 1990.
51. C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
52. D. Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32:2271–2284, 2005.
53. E. Rothberg. personal communication, 2002.
54. D. Salvagnin. A dominance procedure for integer programming. Master's thesis, University of Padua, October 2005.

Matheuristics

Hybridizing Metaheuristics and Mathematical
Programming

Maniezzo, V.; Stützle, Th.; Voß, S. (Eds.)

2010, XVIII, 270 p. 19 illus., Softcover

ISBN: 978-1-4419-1305-0