UNIVERSITÀ DEGLI STUDI DI PADOVA

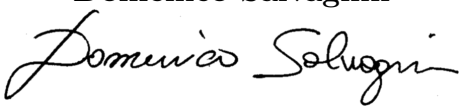
Sede Amministrativa: Università degli Studi di Padova
Dipartimento di Matematica Pura e Applicata

Scuola di Dottorato di Ricerca in Matematica Computazionale

XXI Ciclo

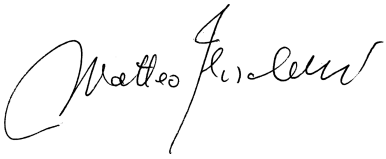
Constraint Programming Techniques for Mixed Integer Linear Programs

Tesi di Dottorato di:
Domenico Salvagnin



Il Coordinatore
Ch.mo Prof. Paolo Dai Pra

Il Supervisore
Ch.mo Prof. Matteo Fischetti



Padova, 25 Gennaio 2009

UNIVERSITÀ DI PADOVA

DIPARTIMENTO DI MATEMATICA PURA E APPLICATA

Constraint Programming Techniques for Mixed Integer Linear Programs

Ph.D. THESIS

Author: Domenico Salvagnin
Coordinator: Ch.mo Prof. Paolo Dai Pra
Supervisor: Ch.mo Prof. Matteo Fishetti

2008

Scuola di Dottorato di Ricerca in Matematica Computazionale – XXI CICLO

Abstract

Many decision problems in industry, logistics, and telecommunications can be viewed as satisfiability or optimization problems. Two paradigms have reached a high degree of sophistication from the point of view of both theory and implementation: Constraint Programming (CP) and Mixed Integer Programming (MIP). The CP and MIP paradigms have strengths and weaknesses that complement each other. On the one hand, CP, through the use of sophisticated propagation techniques, privileges primal inference. On the other hand, MIP, through the techniques of relaxation and strengthening through cutting planes, privileges dual inference.

This thesis presents several studies in Mixed Integer Programming, with emphasis on computational aspects and integration with the Constraint Programming paradigm. In particular, CP concepts and techniques, such as nogoods, minimal infeasiblity and constraint propagation, are used to improve different MIP solving components, namely, dominance detection, Benders cuts selection strategy and primal heuristics. This cross-fertilization of techniques and ideas has proven very effective.

Finally, an appendix is given covering a MIP application to robust railway timetabling.

Sommario

Molti problemi decisionali nell'industria, nella logistica e nelle telecomunicazioni possono essere formulati come problemi di soddisfacibilità o di ottimizzazione. Due paradigmi per la modellazione e la risoluzione di tali problemi hanno raggiunto un elevato grado di sviluppo, sia dal punto di vista teorico che implementativo: la Programmazione a Vincoli (Constraint Programming, CP) e la Programmazione Lineare Intera Mista (Mixed Integer Programming, MIP). I paradigmi CP e MIP hanno vantaggi e debolezze complementari. Da una parte, la CP privilegia l'inferenza primale, attraverso sofisticate tecniche di propagazione. Dall'altra, la MIP privilegia l'inferenza duale, attraverso i rilassamenti e il loro rafforzamento mediante piani di taglio.

Questa tesi presenta alcuni studi in Programmazione Lineare Intera Mista, con enfasi sugli aspetti computazionali e sull'integrazione col paradigma della Programmazione a Vincoli. In particolare, concetti e tecniche CP, quali nogood, insoddisfacibilità minimale e propagazione, sono usati per migliorare varie componenti risolutive per la MIP, quali procedure di dominanza, strategia di selezione dei tagli di Benders e euristiche primali. Questo scambio di idee e tecniche si è dimostrato molto efficace.

Infine, un'applicazione MIP alla generazione di orari robusti in ambito ferroviario è presentata in appendice.

Ringraziamenti

Molte persone hanno contribuito con il loro supporto allo sviluppo di questa tesi di dottorato che, per alcuni aspetti, porta a compimento un cammino iniziato molti anni fa e che, per altri, è un nuovo punto di partenza: a tutte va il mio più sincero ringraziamento.

In primis voglio ringraziare la mia famiglia, che mi ha supportato in tutti questi anni e che mi ha permesso di affrontare gli anni di studio (e ricerca) qui all'Università di Padova senza preoccupazione alcuna.

L'esperienza di questi anni è stata unica: ho avuto l'opportuna di conoscere e collaborare con persone eccezionali, sia dal punto di vista professionale che umano. Voglio ringraziare per questo tutto il gruppo di Ricerca Operativa del DEI (un ambiente unico), nonché molti colleghi (e amici) del Dipartimento di Matematica e del DEIS dell'Università di Bologna.

Un ringraziamento particolare va ovviamente al mio supervisor, Matteo Fischetti: i suoi consigli, le sue idee, il suo supporto e il suo modo di fare sono stati fondamentali. Un giorno si è scherzosamente autodefinito "maestro di vita": beh, Matteo, in parte lo sei stato veramente e spero che la nostra collaborazione possa continuare ancora a lungo!

Infine, ma non certo per importanza, voglio ringraziare mia moglie Susanna che, credendo di aver sposato un serio e responsabile ingegnere, si è invece ritrovata come marito un (ormai ex) studente di dottorato che vuole tentare l'incerta via della ricerca, ma non per questo mi ha supportato un ε in meno: grazie di cuore.

Padova, 25 gennaio 2009

Domenico Salvagnin

Contents

Chapter 1

Introduction

In this thesis we will deal with *optimization problems*, i.e., problems of the form

$$z = \min f(x) \tag{1.1}$$
$$\mathcal{C} \tag{1.2}$$
$$x \in D \tag{1.3}$$

Here $f(x)$ is a real valued *objective function* of the variable x to be minimized, D is the *domain* of x and $\mathcal{C}$ is a finite set of constraints that must be satisfied. Any $x \in D$ is called a *solution*; if it satisfies all constraints in $\mathcal{C}$, it is called a *feasible* solution. A feasible solution x^* is *optimal* if $f(x^*) \leq f(x)$ for all feasible solutions x.

A problem with no feasible solutions is called *infeasible*. On the other hand, if there is no lower bound on $f(x)$ over the feasible set, the problem is called *unbounded*. In this thesis we assume that an optimization problem is either infeasible, unbounded or has finite optimum value[1].

Since the key to efficiently solve an optimization problem is to exploit its particular structure, optimization methods come in a great variety of forms. Different methods with different properties have been developed in different research communities, yet the key ingredients of optimization methods are quite common among them. In this chapter we will briefly introduce three modeling and optimization paradigms, namely *Constraint Programming*, *Boolean Satisfiability* and *Mixed Integer Linear Programming*, highlighting commonalities and differences, strengths and weaknesses of all of them.

1.1 Constraint Programming

Constraint Programming (CP) is the study of computational systems based on constraints. It is an emergent paradigm to declarative model and effectively solve large, often combinatorial, optimization problems.

The basic concepts of constraint programming date back to the early studies in the sixties and seventies done by the *artificial intelligence* community [111, 118]. Further

[1] This is not always the case. For example we rule out problems like $\min e^{-x}$ subject to $x \geq 0$.

steps were achieved when it was noted [39,57] that logic programming (and declarative programming in general) was just a particular kind of constraint programming (hence the development of *Constraint Logic Programming*). However, this does not mean that constraint programming is restricted to declarative languages (like CHiP [28], CLP($\mathcal{R}$) [57] or Prolog III [22]), since constraint programming services can and are actually implemented also for imperative ones (like ILOG Solver [96] or Gecode [40]).

Constraint programming deals with *Constraint Satisfaction Problems*. A constraint satisfaction problem (CSP) is defined as a triple $(x, D, \mathcal{C})$, where

- $x = \{x_1, \ldots, x_n\}$ denotes a finite set of *variables*
- $D = D_1 \times \ldots \times D_n$ represents the *domain* of the variables
- $\mathcal{C}$ denotes a finite set of constraints restricting the values that the variables can simultaneously take.

Note that both domains and constraints can be of arbitrary type. A CSP where all domains are finite, although not necessarily numeric, is called a *finite domain* CSP, CSP(FD) for short.

We may want to find one solution, all solutions, or prove that no solution exists. If we are also given an objective function $f : D \to \mathbb{R}$ mapping each complete solution to a real value we may also want to find an optimal (or at least good) feasible solution—in this case we speak of constraint optimization problems.

From the complexity point view, since CSPs include satisfiability problems as special case, we have that CSP is $\mathcal{NP}$-hard.

1.1.1 CP Solvers

The most common algorithm to solve CSPs is *backtracking search*. This technique consists in extending a (initially empty) partial solution assigning a value from its domain to an uninstantiated variable (this is called *labeling*). If every variable is assigned a value without violating any constraint, then we have found a solution. Otherwise we need to backtrack (undo) one or more assignments and pick different values. This scheme can be organized as a search tree; every node (subproblem) corresponds to a partial assignment and is derived by its parent with the addition of a variable assignment[2].

This simple scheme is sufficient to solve CSP, or at least CSP(FD), but it is hardly a practical scheme by itself. The key element for solving CSPs in practice is the use of *constraint propagation*. Constraint propagation's task is to analyze the set of constraints and the domain of the current subproblem, and to infer additional constraints and domain reductions. Constraints discovered this way are always *implied* by the original constraint set $\mathcal{C}$, and thus unnecessary to correctly model the problem; yet they are helpful in reducing the amount of search that needs to be done to solve the problem,

[2]In practice it is possible and worthwhile to generalize the concept of variable assignment to an arbitrary split of the subproblem space.

because they make implicit information explicit. If constraint propagation is used to infer only domain reductions it is called *domain propagation*.

If a constraint optimization problem is to be solved, a constraint

$$f(x) < f(\tilde{x})$$

is added to the set of constraints whenever a new *incumbent* $\tilde{x}$—the best solution available—is found. The search terminates when the feasible set becomes empty.

1.2 Boolean Satisfiability Problems

The *boolean satisfiability problem* (SAT) is the problem of determining wheter the variables of a given formula of propositional logic can be assigned so as to satisfy the formula, or prove that no such assignment exists. It is well known that any formula in propositional logic can be expressed as a conjunction of *clauses*, where a clause is a disjunction of *literals* (atomic propositions or their negations). As such, SAT is a very special case of CSP, where all variables are constrained to be boolean, and all constraints are logical clauses. SAT has many practical applications, mainly in the context of integrated circuits design and verification [15] and logic systems [114]. Despite is simplicity, SAT is $\mathcal{NP}$-complete, indeed it was the first problem shown to be such [23].

1.2.1 SAT solvers

Although SAT problems could in principle be solved by general-purpose CSP solvers, the particular structure of the problem enables the design of more specialized—and efficient—solving schemes. In particular, modern SAT solvers (like CHAFF [85] or MINISAT [29]) exploit the following techniques:

- an implicit enumeration scheme (the DPLL algorithm [27, 73]). At every node a variable x_j is chosen and the disjunction $x_j = false \vee x_j = true$ is used to generate two child nodes. Nodes are processed in depth first order.
- *unit resolution*, a particular and efficient form of domain propagation, to be applied on the subproblems.
- *conflict analysis*, i.e., analyze infeasible subproblems in order to deduce new clauses, called conflict clauses [110], which can help prune the search tree.
- *restarts*. Restart strategies, often in conjunction with randomization, are helpful to reduce the sensitivity of backtracking search algorithms to early branching mistakes (see, for example, [44, 47]).

It is also worth noting that, in principle, SAT problems can be solved without a backtracking algorithm, applying *resolution* [97, 115], which is a complete inference method for propositional logic. Unfortunately, the worst-case complexity of resolution

is not smaller than that of backtracking search, hence the DPLL algorithm is preferred in practice.

1.3 Mixed Integer Linear Programming

A *mixed integer linear program* is defined as follows

$$z = \min c^T x \tag{1.4}$$
$$Ax \leq b \tag{1.5}$$
$$l \leq x \leq u \tag{1.6}$$
$$x \in \mathbb{R}^n \tag{1.7}$$
$$x_j \in \mathbb{Z}\ \forall j \in I \tag{1.8}$$

where $A \in \mathbb{Q}^{m \times n}$, $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$, $l \in (\mathbb{Q} \cup \{-\infty\})^m$, $u \in (\mathbb{Q} \cup \{\infty\})^m$ and $I \subseteq N = \{1, \ldots, n\}$. Here $c^T x$ is the *objective function*, $Ax \leq b$ are the linear constraints, l and u are simple *lower* and *upper bounds* on the problem variables x, and I is the subset of indices denoting the variables required to be integer. Integer variables with bounds $0 \leq x_j \leq 1$ are called *binary* variables and play a special role in MIP modeling/solving. We will denote the set of binary variables indices with $B \subseteq I$. Variables not required to be integer are called *continuous* variables.

There are several special cases of the above general model, namely:

- *linear programs* (LP) if $I = \emptyset$
- *integer programs* (IP) if $I = N$
- *binary programs* (BP) if $B = I = N$

Note that a MIP is just a special case, albeit a very important one, of CP, where all constraints and the objective function are required to be linear and we have only integer or real-valued domains. Despite these limitations, MIPs proved to be very effective in modeling and solving both theoretical and practical optimization problems.

Since SAT is a special case of binary program, BP, IP and MIP are $\mathcal{NP}$-hard. Only LPs were shown to be polynomially solvable [60], yet the algorithm used to solve them in practice, the *simplex* [26] algorithm, is non-polynomial.

1.3.1 MIP solvers

State-of-the-art MIP solvers are based, like CSP and SAT solvers, on a backtracking scheme, namely the *branch-and-bound* (B&B) scheme. This technique optimizes the search exploiting *relaxations*.

An outline of the basic scheme is given in Figure 1.1. According to this scheme, the B&B algorithm consists of the following steps:

- At the beginning of the search, we initialize a queue $\mathcal{Q}$ of unprocessed subproblems with the original problem P and set the value of the current incumbent to $+\infty$.

input : minimization problem P
output: optimal solution x^{best} of value z^{best}

Initialize $\mathcal{Q} = \{P\}$, $z^{best} = +\infty$
while $\mathcal{Q}$ *not empty* **do**
    Choose $Q \in \mathcal{Q}$ and remove it from $\mathcal{Q}$
    Solve the LP relaxation R of Q
    if R *is feasible* **then**
        Let x^* be the optimal solution of R, of value z^*
        if $z^* < z^{best}$ **then**
            if x^* *is feasible for* P **then**
                Let $z^{best} = z^*$ and $x^{best} = x^*$
            else
                Split problem Q into subproblems $Q_1, \ldots, Q_k$ and add them to $\mathcal{Q}$
            end
        end
    end
end
return x^{best} and z^{best}

Figure 1.1: Basic B&B scheme.

- We choose a subproblem Q from $\mathcal{Q}$. If the queue is empty we are done.
- We solve the LP relaxation R of Q, i.e., we solve a relaxed version of Q where we have dropped all integrality constraints. If the relaxation is feasible we denote with x^* and z^* the relaxed solution and its value, respectively.
- If $z^* \geq z^{best}$ than we can discard the current subproblem immediately, because it cannot contain a feasible solution better than the current incumbent (*bounding*).
- If x^* is feasible for the original problem, i.e., x^* satisfies all integrality requirements, then we have found a new incumbent and we can update x^{best} and z^{best}. Otherwise, we split the current subproblem Q into smaller parts and we add them to $\mathcal{Q}$. This is usually done by choosing a variable x_j that has a fractional value x_j^* in the relaxation and by imposing the disjunction $x_j \leq \lfloor x_j^* \rfloor \vee x_j \geq \lceil x_j^* \rceil$ (we say that we have *branched* on variable x_j).

The effectiveness of the B&B scheme depends on the convergence rate of primal (z^{best}) and dual (the smallest relaxation value z^* of unprocessed nodes) bounds. These bounds are affected by many factors:

- The node selection strategy used to choose the next subproblem Q to extract from $\mathcal{Q}$. There are two main possible strategies to visit the search tree. One is *best-first* search, where the node with best dual bound (lowest z^*) is chosen, while the other is *depth-first* [24, 71] search, where the deepest unprocessed node is chosen. The first strategy tries to move the dual bound as quickly as possible, thus minimizing the number of nodes of the tree, but in doing so it is quite slow

at finding new incumbents and is more memory demanding. On the other hand, depth-first has lower memory requirements and a greater node throughput. It tends to find primal solutions earlier in the search; however, a bad branching decision can make the dual bound move very slowly. State-of-the-art MIP solvers use a hybrid strategy combining the characteristics of both methods [3].

- The branching strategy, i.e., how we choose the variable x_j to branch on. We would like to choose the variable that leads to the highest dual bound change, but this may be very difficult to estimate. What is done in practice is to select a manageable list of "candidate" variables, and estimate the change of LP bound that would occur branching on them. Many methods have been studied to get a good estimate in a reasonable amount of time, among them *strong branching* [7], *pseudocost branching* [12], and *reliability branching* [3, 5].

- How fast we find an incumbent solution providing a good primal bound. Just waiting to find good feasible solutions at B&B nodes by solving LP relaxations has proven not to be effective, so MIP solvers resort to *primal heuristics* [14]. These heuristics are used both to find a first feasible solution to the model, with techniques such as rounding, partial enumeration and diving or to improve existing ones, usually through local search algorithms.

- How tight is the LP relaxation of the subproblems. There are essentially two techniques to improve the LP relaxation of a model, namely *preprocessing* and *cutting planes*. Preprocessing can alter quite significantly the formulation of a problem with techniques such as fixing, aggregation, substitution, redundancy detection, coefficient reduction, and bound strengthening, without changing the set of feasible (in case of primal reductions) or optimal (in case of dual reductions) solutions [104]. Cutting planes are linear constraints that are satisfied by all feasible solutions but not by every solution of the LP relaxation, hence the strengthening of the LP relaxation. Cutting planes can be generated "a priori" or "on the fly" in order to cut a fractional vertex of the LP relaxation (*separation*). Cutting planes implemented in MIP solvers include both generic cuts valid for any MIP problem, like Gomory mixed-integer and mixed-integer rounding cuts [76, 87], and strong polyhedral cuts studied for particular linear constraints, such as knapsack and fixed-charge flow [89,91]. The combination of cutting planes and B&B was introduced successfully in [88, 90] under the name *Branch & Cut* (B&C) and is the algorithm of choice implemented in state-of-the-art MIP solvers.

1.4 Integration

Despite their differences, the three aforementioned paradigms all share the same key ingredients, namely *search*, *inference* and *relaxation*.

Search is the enumeration of problem *restrictions*, each of which is obtained by adding constraints to the original problem P. The solution space of a given optimization

problem is almost invariably exponentially large and highly irregular, and a natural strategy is to split this search space into smallest pieces, to be solved separately, and to pick the best solution found. As we have seen, a very common search strategy is *branching search*, which recursively splits the feasible set of a given problem until the resulting subproblem is easy to solve or proven infeasible. As we have seen, branching search—combined with appropriate inference and relaxation techniques—is at the basis of the most efficient strategies for solving CP problems (backtracking algorithm), SAT problems (DPLL algorithm), and MIP problems (branch-and-bound or branch-and-cut algorithms). However, branching search is not the only option: another general approach is that of *constraint-directed search*, in which once a restriction is solved a constraint is generated that excludes that restriction and possibly others that are proven to be no better (such constraint is named *nogood*). The search continues until the nogoods collected in the process exclude all the search space. Moreover, if the nogoods are written to impose bounds on the optimal value instead of excluding solutions (in this case they are named *nogood bounds*), then we have that the nogood set can be seen as a problem relaxation. SAT solving algorithms such as *dynamic backtracking* [42], as well as decomposition techniques for MIPs, such as Benders' decompositions [11, 49] and its generalizations, can be seen as cases of constraint-directed search.

Inference is the act of revealing implicit constraints from the existing ones, in order to reduce the search space. The connection between inference and optimization is fundamental; indeed, any optimization problem can in principle be expressed as an inference problem, namely the problem of deriving from the constraints the best lower bound on the objective function valid for the solution space. Inference is used extensively in all three paradigms we have seen, although in different ways. The rich modeling language of the constraint programming paradigm allows one to fully exploit the structure of the constraints, by means of sophisticated and powerful constraint propagation algorithms. These structures are often lost when translating these constraints (if possible) into linear inequalities. The unrestricted nature of CP constraints is also its biggest weakness, because it prevents inferences for the problem as a whole. On the other hand, MIP solver have to deal only with linear inequalities, for which global inference methods, such as the Farkas' lemma and Gomory cutting planes separators, are known.

Relaxation means replacing the solution space of our problem with a larger, but more easily searchable, one. Relaxations provide information to guide (through relaxed solutions) and accelerate (through bounds) the search phase. Linear relaxations are the key ingredient of B&B and B&C algorithms for MIPs. The existence of a good (and fast) relaxation is one of the biggest advantages of the MIP paradigm over CP. Due to the very generic nature of CP constraints, a relaxation of the problem is often unavailable to CP solvers. As such, CP solvers can perform much weaker global inferences and dual reductions.

In the last years, several researchers in the CP and MIP communities have investigated the possibility to integrate the methodologies proper to these paradigms [51, 81]. Such an integration has the potential to yield important benefits, and the literature on this topic is recent but growing. From the modelling/programming point of view,

many CP languages include nowadays Operation Research techniques for the propagation of global constraints [81]. Global constraints enable the use of local relaxations of structured constraints, providing precious information for cost-based filtering [37]. Some languages enable the use of hybridization techniques, where different solvers work on distinct parts of the optimization process [49,81,113]. Several methods combine CP with linear or Lagrangian relaxation, and maintain a unique model that is used by distinct solvers that cooperate through common data structures, such as the domains of the variables and their extremes, while others combine MIP and CP through generalizations of Benders decomposition. More recently, the Constraint Integer Programming [3] and search-infer-and-relax [51] paradigms proposed a higher level of integration between CP and MIP. Both paradigms are currently implemented in experimental codes, namely SCIP and SIMPL, respectively.

Chapter 2

Pruning Moves

A concept playing a potentially relevant role in trying to keep the size of the branch-and-bound tree as small as possible is that of *dominance*. Following e.g. Papadimitriou and Steiglitz [92], a branching node α is said to be dominated by another node β if every feasible solution in the subtree of α corresponds to a solution in the subtree of β having a better cost (tie breaks being handled by a suitable lexicographic rule). This concept seems to have been studied first by Kohler and Steiglitz [63], and has been developed in the subsequent years, most notably by Ibaraki [55]. However, although known for a long time, dominance criteria are not exploited in general-purpose MILP codes, due to a number of important limitations of the classical definition. In fact, as stated, the dominance relationship is too vague to be applicable in a generic MILP context—in principle, every node not leading to an optimal solution could be declared as being dominated.

In this chapter we build on the general-purpose dominance procedure proposed in the late 80's by Fischetti and Toth [36], that overcomes some of the drawbacks of the classical dominance definition. As far as we know, no attempt to actually use the above dominance procedure within a general-purpose MILP scheme is reported in the literature. This is due to the fact that the approach tends to be excessively time consuming—the reduction in the number of nodes does not compensate for the large overhead introduced. In an attempt to find a viable way to implement the original scheme, we found that the concept of *nogood*, borrowed from Constraint Programming (CP), can play a crucial role for the practical effectiveness of the overall dominance procedure. Roughly speaking, a *nogood* is a partial assignment of the variables such that every completion is either infeasible (for constraint satisfaction problems) or nonoptimal (for constraint optimization problems). Though widely used in the CP context, the concept of nogood is seldom used in Mathematical Programming. One of the the first uses of nogoods in ILP was to solve verification problems (Hooker and Yan [52]) and fixed-charge network flow problems (Hooker et al. [61]). Further applications can be found in Codato and Fischetti [21], where nogoods are used to generate cuts for a MILP problem. Very recently, attempts to apply the concept of "nogood" to MILPs, deriving nogoods from nodes discarded by the B&B algorithm, have been presented

in [2, 103]. In the context of dominance, a nogood configuration is available, as a byproduct, whenever the auxiliary problem is solved successfully. More generally, we show how the auxiliary problem can be used to derive "improving moves" that capture in a much more effective way the presence of general integer (as opposed to binary) variables.

Improving moves are the basic elements of *test sets*. For an integer programming problem, a test set is defined as a set T of vectors such that every feasible solution x to the integer program is non-optimal if and only if there exists an element $t \in T$ (the "improving move") such that $x + t$ is a feasible solution with strictly better objective function value; see, e.g., [46, 105, 112]. Test sets are often computed for a family of integer linear programs, obtained by varying the right-hand-side vector. This makes test sets useful for sensitivity analysis, or for solving stochastic programs with a large number of scenarios. Improving moves are meant to be used to convert any feasible solution to an optimal one by a sequence of moves maintaining solution feasibility and improving in a strictly monotone way the objective value. Computing and using test sets for NP-hard problems is however by far too expensive in practice. In our approach, instead, improving moves are exploited heuristically within a node fathoming procedure—hence the name "pruning moves". More specifically, we generate pruning moves on the fly, and store them in a *move pool* that is checked in a rather inexpensive way at each node of the branch-and-bound tree. Computational results show that this approach can be very successful for problems whose structure is amenable to dominance.

2.1 The Fischetti-Toth dominance procedure

In the standard Branch-and-Bound (B&B) or Branch-and-Cut (B&C) framework, a node is fathomed in two situations:

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

There is however a third way of fathoming a node, by using the concept of *dominance*. According to [92], a dominance relation is defined as follows: if we can show that a 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 MILPs, 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 [36] proposed a different dominance procedure that overcomes many of the drawbacks of the classical definition, and resembles somehow the *logic cuts* introduced by Hooker et al. in [54] and the *isomorphic pruning* introduced recently by Margot [77, 78]. Here is how the procedure works.

Let P be the MILP problem

$$P: \quad \min\{c^T x : x \in F(P)\}$$

whose feasible solution set is defined as

$$F(P) := \{x \in \mathbb{R}^n : \ Ax \leq b, \ l \leq x \leq u, \ x_j \text{ integer for all } j \in I\} \tag{2.1}$$

where $I \subseteq N := \{1, \cdots, n\}$ is the index set of the integer variables. For any $I' \subseteq I$ and for any $x' \in \mathbb{R}^{I'}$, let

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

denote the contribution of the variables in I' to the overall cost. Now, suppose we are solving P by an enumerative (B&B or B&C) algorithm whose branching rule fixes some of the integer-constrained variables to certain values. For every node k of the branch-and-bound tree, let $I^k \subseteq I$ denote the set of indices of the variables x_j fixed to a certain value x_j^k (say). Every solution $x \in F(P)$ such that $x_j = x_j^k$ for all $j \in I^k$ (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 branch-and-bound tree. Node β dominates node α if:*

1. $I^\beta = I^\alpha$
2. $c(I^\beta, x^\beta) \leq c(I^\alpha, x^\alpha)$, *i.e., the cost of the partial solution x^β at node β is not worse than that at node α, namely x^α.*
3. *every completion of the partial solution x^α associated with node α is also a completion of the partial solution x^β associated with node β.*

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 MILP models, this leads to the following *auxiliary problem* XP^α:

$$\min \sum_{j \in I^\alpha} c_j x_j$$

$$\sum_{j \in I^\alpha} A_j x_j \leq b^\alpha := \sum_{j \in I^\alpha} A_j x_j^\alpha \tag{2.2}$$

$$l_j \leq x_j \leq u_j, \quad j \in I^\alpha \tag{2.3}$$

$$x_j \text{ integer}, \quad j \in I^\alpha \tag{2.4}$$

If a solution x^β (say) of the auxiliary problem having a cost strictly smaller than $c(I^\alpha, x^\alpha)$ 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 MILP problem, but with a smaller size and thus it is often easily solved (possibly in a heuristic way) by a general-purpose MILP solver. In a sense, we are using here the approach of "MIPping the dominance test" (i.e., of modeling it as a MILP [31]), in a vein similar to the recent approaches of Fischetti and Lodi [30] (the so-called *local-branching* heuristic, where a suitable MILP model is used to improve the incumbent solution) and of Fischetti and Lodi [35] (where an ad-hoc MILP model is used to generate violated Chvátal-Gomory cuts). Also note that, as discussed in Section 2.3, the auxiliary problem gives a sufficient but not necessary condition for the existence of a dominating node, in the sense that some of its constraints could be relaxed without affecting the validity of the approach. In addition, inequalities (2.2) could be converted to equalities in order to reduce the search space and get a simpler, although possibly less effective, auxiliary problem.

The Fischetti-Toth dominance procedure, called LD (for *Local Dominance*) in the sequel, 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;
- the LD test needs not be applied at every node; this is crucial from a practical point of view, as the dominance test introduces some overhead and it would make the algorithm too slow if applied at every node.

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 [36]: In case of cost equivalence, define as unfathomed the node β corresponding to the solution found by the *deterministic*[1]

[1]In this context, an algorithm is said to be *deterministic* if it always provides the same output solution for the same input.

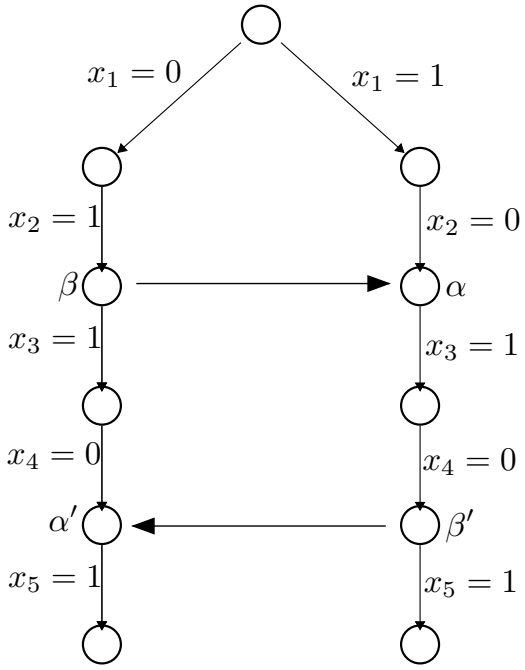


Figure 2.1: A nasty situation for LD test

(exact or heuristic) algorithm used to solve the auxiliary problem. However, even very simple "deterministic" algorithms may lead to a wrong result, as shown in the following example.

Let P be the problem:

$$\begin{cases} & \min -x_1 - x_2 - x_3 - x_4 - 99x_5 \\ \text{s.t.} & x_1 + x_2 \leq 1 \\ & x_3 + x_4 \leq 1 \\ & x_4 + x_5 \leq 1 \\ & x \in \{0,1\}^5 \end{cases}$$

whose optimal solutions are $[1, 0, 1, 0, 1]$ and $[0, 1, 1, 0, 1]$, and let us consider the branch-and-bound tree depicted in Figure 2.1. The deterministic algorithm used to perform the LD test is as follows: If the number of variables in the auxiliary problem is smaller than 3, use a greedy heuristic trying to fix variables to 1 in decreasing index order; otherwise use the same greedy heuristic, but in increasing index order.

When node α (that corresponds to the partial solution $x_1 = 1, x_2 = 0$ with cost -1) is processed, the following auxiliary model is constructed

$$\begin{cases} & \min -x_1 - x_2 \\ \text{s.t.} & x_1 + x_2 \leq 1 \\ & x_1, x_2 \in \{0, 1\} \end{cases}$$

and the deterministic heuristic returns the partial solution $x_2 = 1, x_1 = 0$ of cost -1 associated with node β, so node α is declared to be dominated by β and node α is fathomed assuming (correctly) that node β will survive the fathoming test. However, when the descendant node α' (that corresponds to the partial solution $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$ with cost -2) is processed, the following auxiliary model is constructed

$$\begin{cases} & \min -x_1 - x_2 - x_3 - x_4 \\ \text{s.t.} & x_1 + x_2 \leq 1 \\ & x_3 + x_4 \leq 1 \\ & x_4 \leq 1 \\ & x_1, x_2, x_3, x_4 \in \{0, 1\} \end{cases}$$

and our deterministic heuristic returns the partial solution $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$ of cost -2 associated with node β', so node α' is declared to be dominated by β' and node α' is fathomed as well. Therefore, in this case the enumerative algorithm cannot find any of the two optimal solutions, i.e., the LD tests produced a wrong answer.

In view of the considerations above, in our implementation we used a different tie-break rule, also described in [36], that consists in ranking cost-equivalent solutions in lexicographical order ($\prec$). To be more specific, in case of cost ties we fathom node α if and only if $x^\beta \prec x^\alpha$, meaning that the partial solution x^β associated with the dominating node β is lexicographically smaller[2] than x^α. Using this tie-break rule, it is possible to prove the correctness of the overall enumerative method.

Proposition 1. *Assuming that the projection of the feasible set $F(P)$ on the integer variable space is a bounded set, a B&B algorithm exploiting LD with the lexicographical tie-break rule returns the same optimal value as the classical B&B algorithm.*

Proof. Let x^* be the lexicographically minimal optimal solution, whose existence is guaranteed by the boundedness assumption and by the fact that $\prec$ is a well-order. We need to show that no node α having x^* among its descendants (i.e. such that $x^*_j = x^\alpha_j$ for all $j \in I^\alpha$) can be fathomed by the LD test. Assume by contradiction that a node β dominating α exists, and define

$$z_j := \begin{cases} x^\beta_j & j \in I^* \\ x^*_j & j \notin I^* \end{cases}$$

[2]We use the standard definition of lexicographic order on vectors of fixed size over a totally order set.

where $I^* := I^\beta\,(= I^\alpha)$. In other words, z is a new solution obtained from x^* by replacing its dominated part with the dominating one. Two cases can arise:

1. $c(I^*, x^\beta) < c(I^*, x^\alpha)$: we have

$$c^T z = \sum_{j \in I^*} c_j x_j^\beta + \sum_{j \notin I^*} c_j x_j^* < \sum_{j \in I^*} c_j x_j^\alpha + \sum_{j \notin I^*} c_j x_j^* = c^T x^*$$

and

$$\sum_{j=1}^{n} A_j z_j = \sum_{j \in I^*} A_j x_j^\beta + \sum_{j \notin I^*} A_j x_j^* \leq \sum_{j \in I^*} A_j x_j^\alpha + \sum_{j \notin I^*} A_j x_j^* \leq b$$

so z is a feasible solution with a cost strictly smaller than x^*, which is impossible.

2. $c(I^*, x^\beta) = c(I^*, x^\alpha)$: using the same argument as in the previous case, one can easily show that z is an alternative optimal solution with $z \prec x^*$, also impossible.

□

It is important to notice that the above proof of correctness uses just two properties of the lexicographic order, namely:

P1 *well-order*: required for the existence of a minimum optimal solution;

P2 *inheritance*: if x^α and x^β are two partial assignments such that $x^\alpha \prec x^\beta$, then the lexicographic order is not changed if we apply the same completion to both of them.

This observation will be used in section 2.3 to derive a more efficient tie-break rule.

2.2 Nogoods and pruning moves

The computational overhead related to the LD test can be reduced considerably if we exploit the notion of nogoods taken from Constraint Programming (CP). 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 in fact found a *nogood configuration* $[I^\alpha, x^\alpha]$ that we want to exclude from being re-analyzed at a later time.

Actually, when the LD test succeeds we have not just a dominated partial assignment (x^α), but also a dominating one (x^β). Combining the two we get a *pruning move* ($\Delta = [I^\alpha, x^\beta - x^\alpha]$), i.e., a list of variable changes that we can apply to a (partial) assignment to find a dominating one, provided that the new values assigned to the variables stay within the prescribed bounds. For binary MILPs, the concept of pruning move is equivalent to that of nogood, in the sense that a pruning move can be applied to a partial assignment if and only if the same assignment can be ruled out by a corresponding (single) nogood. For general-integer MILPs, however, pruning moves can

be much more effective than nogoods. This is easily seen by the following example. Suppose we have two integer variables $x_1, x_2 \in [0, U]$ and that the LD test produces the pair

$$x^\alpha = (1, 0) \quad x^\beta = (0, 1)$$

It is easy to see that, in this case, all the following (dominating, dominated)-pairs are valid:

$$\{((a, b), (a - 1, b + 1)) : a \in [1, U], b \in [0, U - 1]\}$$

The standard LD procedure would need to derive each such pair by solving a different auxiliary problem, while they can be derived all together by solving a single MILP leading to the pruning move $(-1, 1)$. As such, pruning moves can be seen as compact representations of sets of nogoods, a topic studied also in [59].

In our implementation, we maintain explicitly a pool of previously-found pruning moves and solve the following problem (akin to separation for cutting-plane methods) at each branching node α: Find, if any, a move $\Delta = [I', \delta]$ stored in the pool, such that $I' \subseteq I^\alpha$ and $l_j \leq x_j^\alpha + \delta_j \leq u_j$ for all $j \in I'$. If the test is successful, we can of course fathom node α without needing to construct and solve the corresponding auxiliary problem XP^α. In our implementation, pruning moves are stored in sparse form, while the pool is simply a list of moves sorted by length. It is worth noting that we are interested in minimal (with respect to set inclusion) pruning moves, so as to improve effectiveness of the method. To this end, before storing a pruning move in the pool we remove its components j such that $x_j^\alpha = x_j^\beta$ (if any).

It is also worth noting that a move $\Delta^1 = [I^1, \delta^1]$ implies (absorbs) a move $\Delta^2 = [I^2, \delta^2]$ in case

$$I^1 \subseteq I^2 \quad \text{and} \quad |\delta_j^1| \leq |\delta_j^2| \; \forall j \in I^1$$

This property can be exploited to keep the pool smaller without affecting its fathoming power.

At first glance, the use of a move pool can resembles classical state-based dominance tests, but this is really not the case since the amount of information stored is much smaller—actually, it could even be limited to be of polynomial size, by exploiting techniques such as relevance or length bounded nogood recording (see [58]).

2.3 Improving the auxiliary problem

The effectiveness of the dominance test presented in the previous section heavily depends on the auxiliary problem that is constructed at a given node α. In particular, it is advisable that its solution set is as large as possible, so as to increase the chances of finding a dominating partial solution. Moreover, we aim to find 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, we next propose a number of improvements over the original auxiliary problem formulation.

Objective function

The choice of the lexicographic order as a mean to resolve ties, although natural and simple, is not well suited in practice.

In the most naïve implementation, there is a good chance of not finding a lexicographically better solution even if this exists, because we do not convey in any way to the solver the information that we are interested in lexicographically minimal solutions. This is unfortunate, since we risk wasting a great computational effort.

Moreover, the lexicographic order cannot be expressed as a linear objective without resorting to huge coefficients: the only way to enforce the discovery of lexicographically better solutions is through ad-hoc branching and node selection strategies, that are quite intrusive and greatly degrade the efficiency of the solution process.

The solution we propose is to use an alternative randomly generated objective function, according to the following scheme:

- We generate an alternative random objective function at the beginning and keep it unchanged for the whole search, so as to satisfy the two properties P1 and P2 of Section 2.1 needed for the correctness of the algorithm.
- In order to guarantee that the optimal solution of the auxiliary problem will be not worse than the original partial assignment, we add the following *optimality* constraint:
$$\sum_{j \in I^\alpha} c_j x_j \leq \sum_{j \in I^\alpha} c_j x_j^\alpha$$
- We compare the original partial assignment x^α to the solution x^β found by the LD-procedure (if any) first by original objective function, then by random objective function and finally, in the unlikely case that both functions yield the same value, by lexicographic order.

Local branching constraints

As the depth of the nodes in the B&B increases, the auxiliary problem grows in size and becomes harder to solve. Moreover, we are interested in detecting moves involving only a few variables, since these are more likely to help prune the tree and are more efficient to search. For these reasons one can heuristically limit the search space of the auxiliary problem to alternative assignments not too far from the current one. To this end, we use a *local branching* [30] constraint defined as follows.

For a given node α, let $B^\alpha \subseteq I^\alpha$ be the (possibly empty) set of fixed binary variables, and define

$$U = \{j \in B^\alpha \mid x_j^\alpha = 1\} \quad \text{and} \quad L = \{j \in B^\alpha \mid x_j^\alpha = 0\}$$

Then we can guarantee a solution x of the auxiliary problem to be different from x^α in at most k binary variables through the following *local branching* constraint

$$\sum_{j \in U} (1 - x_j) + \sum_{j \in L} x_j \leq k$$

A similar reasoning could be extended to deal with general integer variables as well, although in this case the constraint is not as simple as before and requires the addition of certain auxiliary variables [30]. According to our computational experience, a good compromise is to consider a local branching constraint involving only the (binary or general integer) variables fixed to their lower or upper bound, namely

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

where

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

Right-hand side improvement

One could observe that we have been somehow over-conservative in the definition of the auxiliary problem XP^α. In particular, as noticed already in [36], in some cases the condition

$$\sum_{j \in I^\alpha} A_j x_j \leq b^\alpha$$

could be relaxed without affecting the correctness of the method.

To illustrate this possibility, consider a simple knapsack constraint $4x_1+5x_2+3x_3+2x_4 \leq 10$ and suppose we are given the partial assignment $[1, 0, 1, *]$. The corresponding constraint in the auxiliary problem then reads $4x_1 + 5x_2 + 3x_3 \leq 7$. However, since the maximum load achievable with the free variables is 2, one can safely consider the relaxed requirement $4x_1 + 5x_2 + 3x_3 \leq 10 - 2 = 8$. Notice that the feasible partial solution $[0, 1, 1, *]$ is forbidden by the original constraint but allowed by the relaxed one, i.e., the relaxation does improve the chances of finding a dominating node. Another example arises for set covering problems, where the i-th constraint reads $\sum_{j \in Q_i} x_j \geq 1$ for some $Q_i \subseteq \{1, \ldots, n\}$. Suppose we have a partial assignment x_j^α $(j \in I^\alpha)$, such that $k := \sum_{j \in I^\alpha \cap Q_i} x_j^\alpha > 1$. In this case, the corresponding constraint in the auxiliary problem would be $\sum_{j \in I^\alpha \cap Q_i} x_j \geq k$, although its relaxed version $\sum_{j \in I^\alpha \cap Q_i} x_j \geq 1$ is obviously valid as well.

The examples above show however that the improvement of the auxiliary problem may require some knowledge of the particular structure of its constraints. Even more importantly, the right-hand side strengthening procedure above can interfere and become incompatible with the post-processing procedure that we apply to improve the moves. For this reason, in our implementation we decided to avoid any right-hand side improvement.

Local search on incumbents

A drawback of the proposed scheme is that node fathoming is very unlikely at the very beginning of the search. Indeed, at the top of the tree only few variables are fixed and the LD test often fails (this is also true if short moves exist, since their detection depends on the branching strategy used), while the move pool is empty.

To mitigate this problem, each time a new incumbent is found we invoke the following local search phase aimed at feeding the move pool.

- Given the incumbent x^*, we search the neighborhood $N_k(x^*)$ defined through the following constraints:

$$\sum_{j \in I} A_j x_j = \sum_{j \in I} A_j x_j^* \tag{2.5}$$

$$\sum_{j \in I : x_j^* = u_j} (u_j - x_j) + \sum_{j \in I : x_j^* = l_j} (x_j - l_j) \leq k \tag{2.6}$$

$$\sum_{j \in I} c_j x_j \leq \sum_{j \in I} c_j x_j^* \tag{2.7}$$

$$l_j \leq x_j \leq u_j, \quad j \in I$$

$$x_j \text{ integer}, \quad j \in I$$

  In other words, we look for alternative values of the integer variables x_j ($j \in I$) using each constraint—variable bounds excluded—in the same way as x^* (constraint (2.5)), having a Hamming distance from x^* not larger than k (constraint (2.6)), and with an objective value not worse than x^* (constraint (2.7)).

- We populate a solution list by finding multiple solutions to the MILP

$$\min\{\sum_{j \in I} c_j x_j : \ x \in N_k(x^*)\}$$

  by exploiting the multiple-solution mode available in our MILP solver [102].

- Given the solution list $L = (x^1, \ldots, x^p)$, we compare the solutions pairwise and generate a pruning move accordingly, to be stored in the move pool.

- If we find a better solution during the search, we use it to update the incumbent.

It is worth noting that the use of equalities (2.5) allows us to generate a pruning move for *every* pair of distinct solutions in the list, since for every pair we have a dominating and a dominated solution whose difference produces the pruning move.

2.4 Implementation

The enhanced dominance procedure presented in the previous sections was implemented in C++ on a Linux platform, and applied within a commercial MILP solver. Here are some implementation details that deserve further description.

An important drawbacks of LD tests is that their use can postpone the finding of a better incumbent solution, thus increasing the number of nodes needed to solve the problem. This behavior is quite undesirable, particularly in the first phase of the search when we have no incumbent and no nodes can be fathomed through bounding criteria.

Our solution to this problem is to skip the dominance test until the first feasible solution is found.

The definition and solution of the auxiliary problem at every node of the branch-and-bound tree can become too expensive in practice. We face here a situation similar to that arising in B&C methods where new cuts are typically not generated at every node—though the generated cuts are exploited at each node. A specific LD consideration is that we had better skip the auxiliary problem on nodes close to the top or the bottom of the branch-and-bound tree. Indeed, in the first case only few variables have been fixed, hence there is little chance of finding dominating partial assignments. In the latter case, instead, it is likely that the node would be fathomed anyway by standard bounding tests. Moreover, at the bottom of the tree the number of fixed variables is quite large and the auxiliary problem may be quite hard to solve. In our implementation, we provide two thresholds on tree depth, namely $depth_{\min}$ and $depth_{\max}$, and solve the auxiliary problem for a node α only if $depth_{\min} \leq \text{depth}(\alpha) \leq depth_{\max}$. Moreover, we decided to solve the auxiliary problem at a node only if its depth is a multiple of a given parameter, say *depth_interval*.

In addition, as it is undesirable to spend a large computing time on the auxiliary problem for a node that would have been pruned anyway by the standard B&B rules, we decided to apply our technique just before branching—applying the LD test before the LP relaxation is solved turned out to be less effective.

In order to avoid spending too much computing time on pathologically hard auxiliary MILPs, we also set a node limit N_1 (say) for the auxiliary problem solution, and a node limit N_2 (say) for the local search on incumbents.

Finally, since the discovery of new pruning moves decreases as we proceed with the search, we set a upper bound M on the number of times the LD test is called: after this limit is reached, the pruning effect is left to the move pool only.

It is important to stress that, although the auxiliary problem is solved only at certain nodes, we check the current partial assignment against the move pool at every node, since this check is relatively cheap.

2.5 Computational Results

In our computational experiments we used the commercial solver ILOG Cplex 11.0 [102] with default options. All runs were performed on a Intel Q6600 2.4Ghz PC with 4GB of RAM, under Linux.

The definition of the test-bed for testing the potential of our approach is of course a delicate issue. As a matter of fact, one cannot realistically expect any dominance relationship to be effective on all types of MILPs. This is confirmed by a preliminary test we performed on the MIPLIB 2003 [1] testbed, where pruning moves are seldom generated. We face here a situation similar to that arising when testing techniques designed for highly-symmetric problems, such as the *isomorphic pruning* proposed recently by Margot [77, 78]—although remarkably effective on some classes of problems, the approach is clearly of no use for problems that do not exhibit any symmetry.

Therefore we looked for classes of practically relevant 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 two combinatorial problems: knapsack problems [80] and network loading problems [8, 116]. While the first class of problems is quite natural for dominance tests (and could in principle be solved much more effectively by using specialized codes [80] and problem specific dominance criteria, see for example [38]), the second one is representative of very important applications where the dominance property is hidden well inside the solution structure.

2.5.1 Knapsack problem

We generated hard single knapsack instances according to the so-called *spanner instance* method in combination with the *almost strongly correlated* profit generation technique; see Pisinger [94] for details.

The parameters of our LD procedure were set to:

$depth_{min}$**:** 5

$depth_{max}$**:** 0.8 times the number of integer variables

$depth_{interval}$**:** 6

k**:** 0.2 times the number of integer variables

N_1**:** 10

N_2**:** 5000

M**:** 1000

The results on hard single knapsack instances with 60 to 90 items are given in Table 2.1, where labels *"Dominance"* and *"Standard"* refer to the performance of the B&C scheme with and without the LD tests, and label *"Ratio"* refers to the ratios *Standard/Dominance*. For a fair comparison, the same seed was used to initialize the random generator in all runs. The performance figures used in the comparison are the number of nodes of the resulting branch-and-bound tree and the computing time (in CPU seconds). For these problems, the LD tests provided an overall speedup of 23 times and with substantially fewer nodes (the ratio being approximatively 1:33). Note that, in some cases, the ratios reported in the table are just lower bounds on the real ones, as the standard algorithm was stopped before completion due to the time limit.

Additional statistics are reported in Table 2.2 where we provide, for each instance, the final size of the move pool, the percentage of the whole solution time spent either on pool management (*Pool Time*) or LD tests (*LD time*) along with their success rates (*Pool Success* and *LD Success*, respectively). The figures indicate that the number of pruning moves stored in the pool is always manageable. In addition, the pool checks

	Standard			Dominance			Ratio	
Problem	Nodes	Time (s)	Gap	Nodes	Time (s)	Gap	Nodes	Time
kp60_1	311,490	14.70	0.00	1,793	3.45	0.00	173.73	4.26
kp60_2	831,319	43.72	0.00	3,718	3.05	0.00	223.59	14.35
kp60_3	865,469	45.32	0.00	3,995	2.15	0.00	216.64	21.11
kp60_4	1,012,287	47.54	0.00	19,720	6.42	0.00	51.33	7.41
kp70_1	>12,659,538	>1,200.00	0.41	1,634,517	138.68	0.00	7.75	8.65
kp70_2	783,092	41.21	0.00	4,466	6.43	0.00	175.35	6.41
kp70_3	830,794	41.97	0.00	6,396	4.93	0.00	129.89	8.51
kp70_4	>13,226,464	>1,200.00	0.48	27,591	4.49	0.00	479.38	267.18
kp80_1	403,396	18.71	0.00	2,599	9.41	0.00	155.21	1.99
kp80_2	559,447	28.11	0.00	3,118	1.87	0.00	179.42	15.04
kp80_3	576,885	23.46	0.00	2,962	3.40	0.00	194.76	6.90
kp80_4	277,981	13.35	0.00	5,690	3.29	0.00	48.85	4.06
kp90_1	>16,013,282	>1,200.00	0.07	803,333	65.57	0.00	19.93	18.30
kp90_2	18,330,528	863.77	0.00	5,024	3.56	0.00	3,648.59	242.74
kp90_3	>15,273,264	>1,200.00	0.27	37,017	5.61	0.00	412.60	213.78
kp90_4	2,136,389	116.21	0.00	8,056	3.82	0.00	265.19	30.46
Average	>5,255,727	>381.13	-	160,165	16.63	-	398.89	54.45
Geom. mean	>1,761,164	>98.78	-	11,625	6.00	-	151.50	16.47

Table 2.1: Computational results for hard knapsack instances

and the LD tests are rather successful as they both allow a node to be fathomed approximatively 1/3 of the times they are applied—the total effect being of fathoming approximately 2/3 of the nodes where the test is applied, which results in a dramatic reduction of the total number of branch-and-bound nodes. Although the relative overhead introduced by the LD procedure is significant (due to the fact that node relaxations are particularly cheap for knapsack problems), the overall benefit is striking, with an average speed-up of about 1–2 orders of magnitude.

2.5.2 Network loading problem

Network loading problems arise in telecommunications applications where demand for capacity for multiple commodities has to be realized by allocating capacity to the arcs of a given network. Along with a capacity plan, a routing of all commodities has to be determined and each commodity must be routed from source to destination on a single path through the network. The objective is to minimize the cost of the installed capacity in the network, ensuring that all commodities can be routed from source to destination simultaneously.

Given a directed graph $G = (V, A)$, a set of commodities K (each commodity being described by a source node s^k, a destination node t^k, and a demand size d^k), a base capacity unit C and capacity installation costs c_{ij}, our network loading problem can be formulated as:

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

Problem	Pool size	Pool Time	Pool Success	LD Time	LD Success
kp60_1	404	20.08%	38.95%	13.82%	35.00%
kp60_2	196	1.61%	34.97%	29.82%	47.06%
kp60_3	360	14.86%	37.66%	24.45%	16.67%
kp60_4	148	3.48%	38.18%	58.24%	16.80%
kp70_1	330	26.00%	29.62%	1.10%	50.50%
kp70_2	464	18.62%	39.97%	20.67%	40.59%
kp70_3	299	15.11%	40.58%	25.28%	30.43%
kp70_4	324	16.61%	29.31%	37.60%	59.90%
kp80_1	384	20.67%	42.89%	4.99%	37.05%
kp80_2	286	8.45%	37.87%	35.88%	25.45%
kp80_3	833	24.78%	43.96%	9.54%	35.50%
kp80_4	294	15.95%	43.02%	28.13%	25.13%
kp90_1	268	22.32%	37.55%	2.08%	28.70%
kp90_2	705	17.76%	38.56%	34.19%	58.15%
kp90_3	286	16.55%	34.20%	20.88%	41.20%
kp90_4	181	3.85%	33.97%	44.78%	42.20%
Average	360.13	15.42%	37.58%	24.47%	36.90%
Geom. mean	326.24	12.49%	37.33%	16.80%	34.59%

Table 2.2: Internal statistics for hard knapsack instances

$$\sum_{j\in V} x_{ij}^k - \sum_{j\in V} x_{ji}^k = \begin{cases} 1 & i = s^k \\ -1 & i = t^k \\ 0 & \text{otherwise} \end{cases} \qquad k \in K, i \in V \tag{2.8}$$

$$\sum_k d^k x_{ij}^k \le C y_{ij}, \qquad (i,j) \in A \tag{2.9}$$

$$x_{ij}^k \in \{0,1\}, y_{ij} \in \mathbb{Z}_0^+, \quad k \in K, (i,j) \in A \tag{2.10}$$

Random instances of the above network loading problem were generated as follows:

- In order to obtain a grid-like structure, we generated grid networks of dimensions 3x5 and 4x4 and randomly deleted arcs with probability 0.1. Each arc has base unit capacity of value 4 and cost 10.
- We generated a commodity of flow 10 for each pair of nodes with probability 0.2.

Some parameters of the LD procedure were changed with respect to the knapsack test-bed, due to the greatly increased size of the instances:

k**:** 0.01 times the number of integer variables

N_1**:** 100

N_2**:** 25000

The results of our experiments are given in Table 2.3. In this class of problems, the dominance procedure is still quite effective, with an overall speedup of 4 times and a node ratio of more than 5.

Typical pruning moves discovered by the LD-test for this class of problems are shown in Figure 2.2. In particular, the figure illustrates two moves taken from instance *g_15_17_43*. For each of them we present two equivalent configurations; flows of different commodities are drawn with different line styles, while the values reported by the edges (if relevant) indicate the amount y_{ij} of unit capacities installed on the arcs. In move (a) we have a simple swap of the routing of two different flows with the same value: the corresponding move involves only binary variables x_{ij}^k, whereas variables y_{ij} (not shown in the figure) are unaffected. In move (b) we have a more complex swap, involving also arc capacities (and thus general integer variables as well); here, we reroute two commodities such as to use a different arc, and move an appropriate amount of unit capacities accordingly. This last example shows the effectiveness of pruning moves: this move can be applied whenever it is possible to transfer five unit of capacity from the bottom arc to the upper arc, regardless of their actual values.

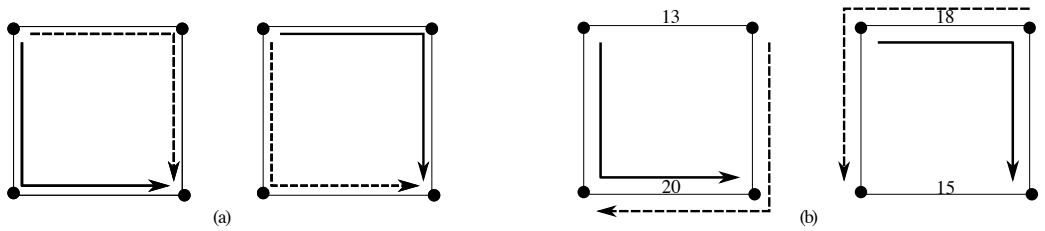


Figure 2.2: Typical moves captured by the LD-tests on network design instances.

Additional statistics are reported in Table 2.4. The format of the table is the same as for knapsack problems. As expected, both the computational overhead of the tests and their rate of success are significantly smaller than in the knapsack case, but still very satisfactory.

2.5.3 Pool effectiveness

Finally, we tested the effectiveness of the main improvements we proposed to the original Fischetti-Toth scheme. Results are reported in Table 2.5 and Table 2.6 for knapsack and network loading instances, respectively. We compared our final code (*Dominance*) against two versions of the same code obtained by disabling the use of the move pool (versions *LD1* and *LD2*) and a version without the local search on incumbents (version *LD3*). We provide the absolute performance figures for *Dominance*, while we give relative performance for the other versions—the numbers in the table give the slowdown factors of the various versions with respect to our final code (the larger the worse). According to the tables, disabling the move pool while retaining the limit M on the number of times the LD test is actually called (version *LD1*) is disastrous: not only do we lose the fathoming effect on a large part of the tree, but we waste a large computing time in solving auxiliary problems discovering a same pruning move (or even a dominated one) over and over. Better results can be obtained if we remove limit

	Standard			Dominance			Ratio	
Problem	Nodes	Time (s)	Gap	Nodes	Time (s)	Gap	Nodes	Time
g_15_17_43	1,954,292	797.01	0.00	242,693	163.80	0.00	8.05	4.87
g_15_17_45	>8,711,335	>3,600.00	0.34	1,544,646	845.00	0.00	5.64	4.26
g_15_17_51	>8,022,870	> 3,600.00	0.29	963,576	545.14	0.00	8.33	6.60
g_15_18_35	3,764,325	1,559.38	0.00	286,539	172.48	0.00	13.14	9.04
g_15_18_37	3,959,652	1,525.63	0.00	567,899	279.64	0.00	6.97	5.46
g_15_18_39	752,035	251.52	0.00	303,667	146.55	0.00	2.48	1.72
g_15_18_40	>10,156,564	>3,600.00	0.48	1,071,922	493.57	0.00	9.48	7.29
g_15_19_43	1,609,434	886.51	0.00	415,472	294.61	0.00	3.87	3.01
g_16_18_48	581,268	226.13	0.00	122,824	86.65	0.00	4.73	2.61
g_16_18_53	6,425,061	3,183.84	0.00	6,489	56.14	0.00	990.15	56.71
g_16_19_51	>7,222,780	>3,600.00	0.43	3,774,093	2,158.96	0.00	1.91	1.67
g_16_20_47	5,593,517	3,436.69	0.00	587,773	449.83	0.00	9.52	7.64
g_16_20_51	2,229,792	1,394.58	0.00	272,355	257.26	0.00	8.19	5.42
g_16_21_40	>6,187,221	>3,600.00	0.37	2,334,537	1,524.71	0.00	2.65	2.36
g_16_21_44	1,079,588	717.43	0.00	151,869	182.00	0.00	7.11	3.94
g_16_21_52	>4,565,496	>3,600.00	0.27	1,279,007	1,186.96	0.00	3.57	3.03
Average	>4,550,951.88	>2,223.67	-	870,335.06	552.71	-	67.86	7.85
Geom. mean	>3,354,264	>1,621.25	-	436,484	339.43	-	7.68	4.78

Table 2.3: Computational results for network loading problems

Problem	Pool size	Pool Time Ratio	Pool Success	LD Time Ratio	LD Success
g_15_17_43	48	0.73%	13.39%	26.20%	1.50%
g_15_17_45	61	1.08%	17.80%	8.43%	0.80%
g_15_17_51	167	2.19%	17.14%	5.79%	1.30%
g_15_18_35	55	0.94%	16.64%	15.84%	1.10%
g_15_18_37	53	1.15%	4.04%	3.27%	0.50%
g_15_18_39	108	1.52%	12.22%	16.93%	1.20%
g_15_18_40	89	1.66%	18.90%	5.17%	2.60%
g_15_19_43	135	1.68%	18.51%	8.97%	3.30%
g_16_18_48	78	0.95%	9.81%	28.68%	0.90%
g_16_18_53	84	0.17%	3.19%	59.68%	0.60%
g_16_19_51	178	2.45%	22.92%	1.49%	1.70%
g_16_20_47	56	0.92%	8.63%	8.76%	0.80%
g_16_20_51	69	0.81%	15.32%	18.28%	2.60%
g_16_21_40	67	0.92%	11.46%	2.77%	1.90%
g_16_21_44	108	0.78%	7.25%	25.27%	1.00%
g_16_21_52	71	0.77%	13.24%	5.58%	2.30%
Average	89.19	1.17%	13.15%	15.07%	1.51%
Geom. mean	82.13	1.01%	11.70%	9.89%	1.31%

Table 2.4: Internal statistics for network loading instances

M (version *LD2*): in this way we retain much of the fathoming effect, but at a much greater computational effort (*LD2* is about 5 times slower than the default version on knapsack problems, and reaches the 1-hour time limit in 11 out of 16 instances on network problems). The contribution of the local search on incumbents (version *LD3*) is more difficult to evaluate: while the number of nodes is always reduced by its use, the overall computing time is reduced only for network problems. A closer look at the individual table entries shows however that local search is not effective only for easy problems, where its overhead is not balanced by the increased fathoming power, but it turns out to be very useful on harder instances (e.g., *kp70_1* or *kp90_1*).

	Dominance		LD1 ratios		LD2 ratios		LD3 ratios	
Problem	Nodes	Time (s)	Nodes	Time	Nodes	Time	Nodes	Time
kp60_1	1,793	3.45	104.23	3.09	3.66	1.36	1.20	0.23
kp60_2	3,718	3.05	179.47	11.96	4.61	4.77	1.56	0.76
kp60_3	3,995	2.15	143.03	14.41	5.01	4.67	1.97	0.82
kp60_4	19,720	6.42	19.34	3.07	4.33	10.45	1.67	0.85
kp70_1	1,634,517	138.68	>8.16	>8.65	>1.26	>8.65	7.21	6.25
kp70_2	4,466	6.43	133.94	5.05	5.58	2.70	1.26	0.30
kp70_3	6,396	4.93	117.94	7.94	5.31	4.43	1.35	0.44
kp70_4	27,591	4.49	>500.77	>267.18	2.85	17.85	2.33	1.36
kp80_1	2,599	9.41	105.96	1.46	4.97	0.72	1.38	0.11
kp80_2	3,118	1.87	103.60	9.35	4.19	4.70	1.30	0.69
kp80_3	2,962	3.40	119.87	4.49	5.14	1.79	1.44	0.26
kp80_4	5,690	3.29	39.23	3.56	4.78	4.36	1.54	0.70
kp90_1	803,333	65.57	>20.06	>18.30	10.66	16.30	14.49	10.52
kp90_2	5,024	3.56	3,354.31	221.01	4.45	4.02	1.58	0.67
kp90_3	37,017	5.61	>432.23	>213.78	3.67	20.43	1.58	0.94
kp90_4	8,056	3.82	240.35	27.55	3.32	4.58	1.00	0.64
Average	-	-	>351.41	>51.30	>4.61	>6.99	2.68	1.60
Geom. mean	-	-	>116.83	>13.14	>4.26	>4.85	1.88	0.74

Table 2.5: Comparison of different LD versions on knapsack problems: LD1 is the version without the move pool and with the same limit M on the number of times the LD test is called, while LD2 is still without the move pool, but with no such limit. LD3 is the version without local search on the incumbents. As in the previous table, label Dominance *refers to the default LD version. > indicates a reached time limit.*

	Dominance		LD1 ratios		LD2 ratios		LD3 ratios	
Problem	Nodes	Time (s)	Nodes	Time	Nodes	Time	Nodes	Time
g_15_17_43	242,693	164	4.36	2.73	>1.97	>21.98	0.82	0.90
g_15_17_45	1,544,646	845	>5.38	>4.26	>0.22	>4.26	1.26	1.22
g_15_17_51	963,576	545	>8.38	>6.60	>0.55	>6.60	1.99	1.82
g_15_18_35	286,539	172	6.80	4.89	1.48	15.78	1.08	1.02
g_15_18_37	567,899	280	2.13	1.77	1.25	12.43	0.47	0.55
g_15_18_39	303,667	147	2.19	1.63	1.11	7.99	0.97	0.89
g_15_18_40	1,071,922	494	>9.46	>7.29	>0.65	>7.29	1.44	1.35
g_15_19_43	415,472	295	3.39	2.68	1.04	10.83	0.98	0.93
g_16_18_48	122,824	87	9.51	5.49	>4.17	>41.55	4.01	2.63
g_16_18_53	6,489	56	>1,126.00	>64.13	28.33	33.69	3.84	1.07
g_16_19_51	3,774,093	2,159	>1.93	>1.67	>0.11	>1.67	1.65	1.57
g_16_20_47	587,773	450	>10.07	>8.00	>1.05	>8.00	2.96	2.61
g_16_20_51	272,355	257	3.46	2.43	>1.83	>13.99	1.63	1.35
g_16_21_40	2,334,537	1,525	>2.66	>2.36	>0.17	>2.36	2.42	2.36
g_16_21_44	151,869	182	7.18	4.25	>3.04	>19.78	1.76	1.31
g_16_21_52	1,279,007	1,187	>3.57	>3.03	>0.23	>3.03	1.01	0.97
Average	-	-	>75.40	>7.70	>2.95	>13.20	1.77	1.41
Geom. mean	-	-	>6.49	>4.14	>1.00	>9.24	1.51	1.29

Table 2.6: Comparison of different LD versions on network problems: LD1 is the version without the move pool and the same M as the default LD version, while LD2 is still without the move pool, but with no such limit. LD3 is the version without local search on the incumbents. As in the previous table, label Dominance *refers to the default LD version. > indicates a reached time limit (for LD1 and LD2 the time limit is reached so often that the means are seriously underestimated)*

2.6 Conclusions

In this chapter we have presented a dominance procedure for general MILPs. The technique is an elaboration of an earlier proposal of Fischetti and Toth [36], with important improvements aimed at making the approach computationally more attractive in the general MILP context. In particular, the use of nogoods and of pruning moves that we propose in this chapter turned out to be crucial for an effective use of dominance tests within a general-purpose MILP code.

In our view, a main contribution of our work is the innovative use of improving moves. In a classical (yet computationally impractical) test set approach, these moves are used within a primal solution scheme leading eventually to an optimal solution. In our approach, instead, we heuristically generate improving moves on small subsets of variables by solving, on the fly, small MILPs. These moves are not used to improve the incumbent, as in the classical test set environment, but rather to fathom nodes in the branch-and-bound tree—hence the name "pruning moves". If implemented in a proper way, this approach introduces an acceptable overhead even if embedded in a highly-efficient commercial MILP solver such as ILOG Cplex 11, and may produce a drastic reduction in the number of nodes. Indeed, computational results show that the method can lead to a speedup of up to 2 orders of magnitude on hard MILPs whose structure is amenable to dominance, in the sense that it allows for local variable adjustments that produce equivalent solutions. An example of this kind of problems has been discussed, namely a network loading problem arising in telecommunication.

A drawback of our approach is of course the overhead introduced when addressing problems that turn out not to produce any pruning moves. A possible remedy is to use a conservative adaptive scheme that tries to generate those moves only at the beginning of the search, and deactivates the solution of auxiliary problem if its success rate is not satisfactory. The detailed design and implementation of this idea is however out of the scope of the present thesis, and is left to future research.

Acknowledgments

This work was supported by the University of Padova "Progetti di Ricerca di Ateneo", under contract no. CPDA051592 (project: Integrating Integer Programming and Constraint Programming) and by the Future and Emerging Technologies unit of the EC (IST priority), under contract no. FP6-021235-2 (project ARRIVAL).

Chapter 3

Minimal Infeasible Subsystems and Benders cuts

Benders decomposition is a particular kind of constraint-directed search, where all problem restrictions are defined by fixing the same subset of variables, i.e., the variable set is partitioned into a set of "search" variables and a set of "subproblem" variables. It was originally proposed in [11] as a machinery to convert a generic mixed-integer program involving integer variables x and continuous variable y into an integer program involving the x variables only, possibly plus a single continuous variable η taking into account the overall contribution to the objective function of the continuous variables. However, the basic idea of the Benders method can be extended to more general settings, provided that we are able to analyze the problem restrictions in order to infer appropriate nogoods and nogood bounds (called Benders cuts) involving the search variables. Indeed, generalized Benders decomposition is one of the most successful frameworks for the integration of CP and MIP, with speedups up to three orders of magnitude reported in various applications [50, 53, 98, 113].

In practice, the ability to infer strong Benders cuts from problem restrictions is a key ingredient for the success of a Benders scheme. In many applications, the strength of a nogood is closely related to the concept of minimality; intuitively, a minimal nogood (or nogood bound) affects a bigger part of the solution space, and so is more effective. However, if we look at the classical Benders scheme, as it is applied to MIPs, we can notice that the issue of minimality is not taken into account.

Our approach to introduce minimality-reasoning in a MIP Benders scheme is based on the correspondence between minimal infeasible subsystems of an infeasible LP and the vertices of the so-called *alternative polyhedron*. The choice of the "most effective" violated Benders cut then corresponds to the selection of a suitable vertex of the alternative polyhedron.

Computational results on a testbed of MIPLIB instances are presented, where the quality of Benders cuts is measured in terms of "percentage of gap closed" at the root node, as customary in cutting plane methods. We show that the proposed methods provide a speedup of 1 to 2 orders of magnitude with respect to the classical one.

3.1 Benders cuts: theory ...

Suppose we are given a MIP problem

$$\begin{aligned} \min c^T x + d^T y \\ Ax \geq b \\ Tx + Qy \geq r \\ x \geq 0,\ x \text{ integer} \\ y \geq 0 \end{aligned} \tag{3.1}$$

where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^t$, and matrix Q has m rows.

Classical Benders decomposition states that solving such a problem is equivalent to solving

$$\begin{aligned} \min c^T x + \eta \\ Ax \geq b \\ \eta \geq u^T(r - Tx), \quad u \in \text{VERT} \\ v^T(r - Tx) \leq 0, \quad v \in \text{RAY} \\ x \geq 0,\ x \text{ integer} \end{aligned} \tag{3.2}$$

where the additional variable η takes into account the objective function term $d^T y$, while sets VERT and RAY contain the vertices and extreme rays (respectively) of the polyhedron D defined by:

$$\begin{aligned} \pi^T Q \leq d^T \\ \pi \geq 0 \end{aligned} \tag{3.3}$$

The above formulation has exponentially many inequalities, so an iterative solution approach based on cutting planes is needed, that can be outlined as follows.

1. Solve the so-called *master problem*:

$$\begin{aligned} \min c^T x + \eta \\ Ax \geq b \\ \{\text{previously generated Benders cuts}\} \\ x \geq 0,\ x \text{ integer} \end{aligned} \tag{3.4}$$

   including (some of) the Benders cuts generated so far (none at the very beginning). Let (x^*, η^*) be an optimal solution of the master problem.

2. Solve the so-called *dual slave problem*:

$$\begin{aligned} \max \pi^T (r - Tx^*) \\ \pi^T Q \leq d^T \\ \pi \geq 0 \end{aligned} \tag{3.5}$$

3. If the dual slave problem is unbounded, choose any unbounded extreme ray $\overline{v}$, and add the so-called *Benders feasibility cut*

$$\overline{v}^T (r - Tx) \leq 0$$

to the master and go to Step 1. Otherwise, let the optimal value and an optimal vertex be z^* and $\overline{u}$ respectively. If $z^* \leq \eta^*$ then stop. Otherwise, add the so-called *Benders optimality cut*

$$\eta \geq \overline{u}^T (r - Tx)$$

to the master problem, and go to Step 1.

The distinction between *optimality cuts* (involving the η variable) and *feasibility cuts* (that assert some property of the feasible x vector) is very important in practice, and will be analyzed in greater detail in the sequel.

As already noted by other authors, but seldom applied in practice, Benders cuts can be generated to separate any solution (integer or not) of the master problem. As a consequence, these cuts can easily be embedded into a modern branch-and-cut scheme where Benders cuts (among others) are generated at each node of the branching tree.

Note that:

- Although presented for the MIP case, the Benders framework is by no means limited to it. In particular, any problem of the form

$$\begin{aligned} \min c(x) + d^T y \\ g(x) \geq 0 \\ F(x) + Qy \geq r \\ y \geq 0 \end{aligned} \tag{3.6}$$

with arbitrary $c()$, $g()$ and $F()$ is suitable to be solved with this method, provided that we have a solver for the master problem (see [41]). This also means that, given any arbitrary partition of the variables, any linear programming problem can be casted into the Benders framework, by projecting away a subset of the variables. This is indeed done in practice with problems that simplify considerably (e.g., decompose) after fixing a subset of their decision variables—this is the case, e.g., in Stochastic Linear Programs (SLPs).

- The Benders method is in fact a pure cutting plane approach in which, given a solution (x^*, η^*) of a problem relaxation (the master), we look for a violated valid

inequality. In particular, the search for such an inequality is done by solving an LP problem (the dual slave), which acts as a *Cut Generating LP* akin to the one used in disjunctive programming (as a matter of fact, disjunctive cuts can be viewed as Benders cuts derived from a compact extended formulation).

- The set of Benders cuts corresponds to the vertices and extreme rays of D and is independent of the current master solution (x^*, η^*), which is used only to decide which is next cut to add. For this purpose a suboptimal (or even infeasible) master solution can be used as well, as e.g. in the recent proposals by Rei e al. [99] and by Poojari and Beasley [95].

Given the considerations above, in the following we focus on a generic LP of the form

$$\begin{aligned} \min c^T x + d^T y \\ Ax \geq b \\ Tx + Qy \geq r \\ x \geq 0 \\ y \geq 0 \end{aligned} \tag{3.7}$$

This LP may be the root relaxation of a MIP problem, or just a large-scale LP problem suitable for Benders decomposition (e.g., a SLP problem).

3.2 ... and practice

The first question we asked ourselves was: What can be considered a modern, yet classical, implementation of Benders decomposition to be used for benchmarking purposes? As a matter of fact, any implementation of the Benders approach has to face a number of implementation issues that affect heavily the overall performance of the method, and many authors using Benders cuts tend to classify their methods as just "standard implementations" without giving sufficient details.

A first issue is how to obtain a good, yet easily computable, initial lower bound on η, so as to prevent the generation of several dominated (and thus useless) optimality cuts. From a theoretical point of view, we are interested in the best-possible optimality cut of the form

$$\eta \geq \pi^T r - 0^T x$$

so $\pi^T r$ can be obtained by just solving the LP:

$$\begin{aligned} \max \pi^T r \\ \pi^T Q \leq d^T \\ \pi^T T = 0^T \\ \pi \geq 0 \end{aligned} \tag{3.8}$$

However, if the slave problem does not have a special structure (i.e., if it does not decompose nicely), the introduction of the coupling matrix T yields an LP problem of the same size as the original LP, so this approach is not always viable computationally. Therefore, in our tests we prefer to calculate a trivial bound on $d^T y$ based only on the lower and upper bounds on the y variables (if no bounds are given, we just write $\eta \geq -M$ for a suitably large M).

Then we addressed the relative contribution of optimality and feasibility cuts to the convergence of the method. Indeed, according to our computational experience these two classes of cuts behave quite differently in many important respects:

- For many problems where term $d^T y$ gives a significant contribution to the overall optimal value, optimality cuts can be much more effective in moving the bound than feasibility cuts, because they involve the η variable explicitly.

- Optimality cuts are typically quite bad from a numerical point view. In particular, optimality cuts tend to exhibit an higher *dynamism* than feasibility cuts, i.e., a higher ratio between the maximum and minimum absolute value of the cut coefficients. This was somewhat expectable, because optimality cuts have to take into account the objective function, which may be of a completely different magnitude (and precision) with respect to the constraints.

- Optimality cuts tend to be much denser than the feasibility ones. Again, this is not surprising since the role of optimality cuts is to provide a lower bound on the objective function term η that is based on the value of the variables x of the master problem, and it is unlikely that just a few master variables can succeed in producing a tight bound.

As a consequence, it is important to have some control on the kind (and quality) of Benders cuts generated at each iteration. Unfortunately, Benders decomposition—as it is typically implemented in the literature—is heavily biased toward feasibility cuts. As a matter of fact, as long as a violated feasibility cut exists, the dual slave is unbounded and hence no optimality cut is generated. As noted by Benders himself [11], however, if we solve the dual slave with the primal simplex method, then when we discover an unbounded ray we are "sitting on a vertex" of polyhedron D, and thus we can generate also an optimality cut with no additional computational effort. A main drawback of this approach is that optimality cut produced is not guaranteed to be violated, and in any case its discovery was quite "random" as the corresponding vertex is by no mean a one maximizing a certain quality index such as cut violation, depth, etc.

The lack of control on the *quality* of the Benders cuts is even more striking when feasibility cuts are generated, since the textbook method does not give any rule to choose among the unbounded rays. To illustrate this important (and often underestimated) point, suppose that we want to apply a textbook Benders decomposition approach to the well-known Asymmetric Traveling Salesman Problem (ATSP). Our compact MIP formulation then involves binary variables x_{ij} associated with the arcs of digraph $G = (V, A)$, and continuous flow variables y^k_{ij} that describe a flow of value 1 from a fixed

source node (say node 1) to sink node k, for all $k \in V \setminus \{1\}$. In this example, system $Ax \geq b$ corresponds to in- and out-degree restrictions, whereas system $Tx + Qy \geq r$ is made by $|V| - 1$ independent blocks corresponding to the flow-conservation equations for each k, plus the coupling constraints $y^k_{ij} \leq x_{ij}$ for all $k \in V \setminus \{1\}$ and $(i, j) \in A$. It is not hard to see that, in this case, Benders cuts are of the feasibility type only, and correspond to the classical Subtour Elimination Constraints (SECs) of the form $\sum_{(i,j)\in\delta^+(S)} x_{ij} \geq 1$. These cuts are known to be facet-defining (assuming G is complete digraph), hence they are very strong in practice—so we can conclude that "Benders cuts make a wonderful job". What is clearly inefficient is instead the way these cuts would be handled by the standard Benders method. First of all, SECs would be generated only after having solved to proven optimality the current master, and used to cut integer points only. This is clearly inefficient, since SECs should be generated at each node of the branching tree, or at least whenever the incumbent solution is updated (as in the old-day method by Miliotis [82,83]). But even if SECs were generated within a modern branch-and-cut framework, what is completely missing in the Benders method is a sensible cut selection criterion—once a violated SEC exists, the dual slave becomes unbounded and *any* violated SEC can be returned by the separation procedure—whereas we know that SEC density (among other characteristics) plays a crucial role in speeding-up convergence.

The considerations above prompted us to introduce an effective criterion for choosing among violated (optimality or feasibility) Benders cuts, very much in the spirit of disjunctive cut generation that is also based on CGLPs (see Balas, Ceria, and Cornuéjols [9], and also Fischetti, Lodi and Tramontani [32]). As far as we know, no research effort was devoted to this particular topic in the literature, with one notable exception—the acceleration procedure by Magnanti and Wong [74]. This procedure provides a criterion to choose, among equivalent optimal vertices of the dual slave polyhedron, a "Pareto-optimal" one that corresponds to a maximally-violated optimality cut that is not strictly dominated (within the master feasible solution set) by any other maximally-violated cut. The procedure has however some drawbacks:

- According to its original definition, the procedure would require the dual slave to have a bounded optimal value, hence it could not be applied in a completely general context involving feasibility cuts—this drawback can however be partially overcome by introducing artificial dual bounds.

- The user has to provide a point in the relative interior of the master feasible set. This is quite a simple task if the the master has a very special structure, as in the cases addressed by Magnanti and Wong in their study, but is NP-hard in general if the master is a MIP, since we need a point in the relative interior of the convex hull of the integer feasible points, which is usually not known. Moreover, the outcome of the procedure depends on the choice of the interior point.

- The method may be computationally heavy, as it requires to solve two LPs to generate a single cut, the second LP being often quite time-consuming due to

the presence of an additional equation that fixes the degree of violation to the cut—this equation is in fact very dense and numerically unstable.

- The Magnanti-Wong criterion benefits from the existence of several equivalent optimal solutions of the dual slave problem (i.e., several maximally-violated optimality cuts), which is however not very frequent when fractional (as opposed to integer) points of the master are cut.

3.3 Benders cuts and Minimal Infeasible Subsystems

The CGLP of a Benders cut can always be seen as a feasibility problem: given a master solution (x^*, η^*), it is possible to generate a violated cut if and only if the following primal slave problem is infeasible:

$$\begin{aligned} d^T y &\leq \eta^* \\ Qy &\geq r - Tx^* \\ y &\geq 0 \end{aligned} \tag{3.9}$$

or equivalently, by LP duality, if the following dual slave problem is unbounded:

$$\begin{aligned} \max \pi^T (r - Tx^*) - \pi_0 \eta^* \\ \pi^T Q &\leq \pi_0 d^T \\ \pi, \pi_0 &\geq 0 \end{aligned} \tag{3.10}$$

If the separation is successful, given the dual solution (extreme ray) $(\overline{\pi}, \overline{\pi}_0)$ the generated cut is

$$\overline{\pi}^T (r - Tx) - \overline{\pi}_0 \eta \leq 0$$

In practice, one is interested in detecting a "minimal source of infeasibility" of (3.9), so as to detect a small set of rows that allows one to cut the master solution. According to Gleeson and Ryan [43], the rows of any *Minimal* (with respect to set inclusion) *Infeasible Subsystem* (MIS) of (3.9) are indexed by the support of the vertices of the following polyhedron, sometimes called the *alternative polyhedron*:

$$\begin{aligned} \pi^T Q &\leq \pi_0 d^T \\ \pi^T (r - Tx^*) - \pi_0 \eta^* &= 1 \\ \pi, \pi_0 &\geq 0 \end{aligned} \tag{3.11}$$

where the unbounded objective function—namely, the cut violation to be maximized—has been fixed to a normalization positive value (if the alternative polyhedron is empty, we are done). By choosing an appropriate objective function it is therefore possible to optimize over the alternative polyhedron, thus selecting a violated cut corresponding to a MIS of (3.9) with certain useful properties. Therefore, the choice of the objective function is a main issue to be addressed when designing a separation procedure based

on a CGLP, as in the Benders method.

A natural objective function whose purpose is to try to minimize the cardinality of the support of the optimal vertex (and hence to find a small-cardinality MIS [1]) is

$$\min \sum_{i=1}^{m} \pi_i + \pi_0 \tag{3.12}$$

As we are only interested in solutions with a positive cut violation, and since $\{(\pi, \pi_0) \geq 0 : \pi^T Q \leq \pi_0 d^T\}$ is a cone, we can swap the role of the objective function (3.12) and of the normalization condition in (3.11), yielding the following equivalent CGLP akin to the one used for disjunctive cuts by Balas, Ceria, and Cornuéjols [9]:

$$\begin{aligned} \max \pi^T (r - Tx^*) - \pi_0 \eta^* \\ \pi^T Q \leq \pi_0 d^T \\ \sum_{i=1}^{m} \pi_i + \pi_0 = 1 \\ \pi, \pi_0 \geq 0 \end{aligned} \tag{3.13}$$

It is worth noting that the feasible solution set of the above CGLP is never empty nor unbounded, so a violated cut can be generated if and only if the CGLP has a strictly positive optimal value. The latter formulation is preferable from a computational point because the normalization constraint $\sum_{i=1}^{m} \pi_i + \pi_0 = 1$, though very dense, is numerically more stable than its "cut violation" counterpart $\pi^T(r - Tx^*) - \pi_0 \eta^* = 1$. Moreover, at each iteration only the CGLP objective function is affected by the change in the master solution, hence its re-optimization with the primal simplex method is usually quite fast.

A geometric interpretation of (3.13) is as follows. The CGLP feasible set is now defined as the intersection of the homogenization of the dual polyhedron D with the normalization hyperplane $\sum_{i=1}^{m} \pi_i + \pi_0 = 1$. It is not difficult to see that there is a one-to-one correspondence between the vertices of this feasible set and the extreme rays (if $\pi_0 = 0$) and vertices (if $\pi_0 \neq 0$) of D. Therefore, the reformulation does not actually change the set of Benders cuts that can be generated, but it is nevertheless useful in that it allows one to use a more clever choice of the violated cut to be separated.

3.4 Computational results

The effectiveness of our CGLP formulation has been tested on a collection of problems from the MIPLIB 2003 library [1]. Among the instances in this testbed, we have chosen the mixed-integer cases with a meaningful number of integer and continuous variables. Moreover, we discarded some instances with numerical instability and which, after the variables were partitioned, were too easy to solve even by the classical Benders

[1] Finding a minimum-cardinality MIS is an NP-hard problem in general; see, e.g., Amaldi et al. [6]

Problem	# variables	# integer	# continuous	# constraints
10teams	2025	1800	225	230
a1c1s1	3648	192	3456	3312
aflow40b	2728	1364	1364	1442
danoint	521	56	465	664
fixnet6	878	378	500	478
modglob	422	98	324	291
momentum1	5174	2349	2825	42680
pp08a	240	64	176	136
timtab1	397	171	226	171
timtab2	675	294	381	294
tr12-30	1080	360	720	750

Table 3.1: Testbed characteristics

method [2]. Table 3.1 shows our final testbed with the main characteristics of each instance.

Standard variable partitioning has been applied—integer (and binary) variables are viewed as master variables x, and the continuous variables are viewed as slave variables y.

We implemented two variants of the classical (textbook) Benders method, as well as two variants of our MIS-based CGLP, namely:

tb: This is the original method as proposed by Benders [11]. If the dual slave problem is bounded, we generate one optimality cut, otherwise we generate both a feasibility and an optimality cut (the optimality cut being added to the master problem only if it is violated by the current master solution).

tb_noopt: This is a standard Benders implementation method as often seen on textbooks. This method always generate only one cut per iteration—in case of unboundedness, only the feasibility cut associated with the unbounded dual-slave ray detected by the LP solver is added to the master.

mis: This is our basic MIS-based method. It uses the CGLP (3.13) to solve the separation problem, hence it generates only one cut per iteration.

mis2: This is a modified version of *mis*: after having solved the CGLP, if the generated cut is an optimality one, we enforce the generation of an additional feasibility cut by imposing the condition $\pi_0 = 0$.

In our experiments, we handled the equations in the MIP model (if any) explicitly, without replacing them with pairs of inequalities; this implies the presence of free dual multipliers and the use of their absolute value in the normalization condition.

[2] A couple of instances exhibit a block structure of the slave problem and just a few iterations where enough to terminate the method.

The implementation was done in C++ on a Linux 2.6 platform and all tests were performed on an Intel Core2 Quad CPU Q6600 with 4GB of RAM. We used ILOG Cplex 11.0 as the black-box LP solver; we disabled the LP presolver and forced the use of the primal simplex method for the solution of the dual slaves so as to be able to get a meaningful output even in case of unbounded problems. Before solving an instance, we performed a standard bound shifting in order to reduce the number of slave variable bounds to dualize. For this reason, the optimal LP value reported in our tables may differ from the value reported in the literature.

The quality of the generated Benders cuts is measured in terms of "percentage gap closed" at the root node, as customary in cutting plane methods. The results are shown in Tables 3.2 and 3.3. In Table 3.2 we report the computing time and number of iterations needed to reach 80%, 90%, 95% and 99% of the optimal root relaxation value, as well as the total running times and number of iterations needed for convergence (within a time limit of 2,000 seconds). In Table 3.3 we report the number of generated (optimality and feasibility) cuts, their average density (column AvgD), the rate of growth of the master solution time (column MRate) as a function of the number of iterations (standard linear regression on the master-problem running times vs. iterations), and the average separation time in CPU seconds (column AvgT). In both tables a (*) indicates failed cut generation due to numerical problems. Results with *tb_noopt* are not reported since this method was never better (and often much worse) than *tb*: a typical behavior is illustrated in Figures 3.1 and 3.2.

		Time (seconds)				Iterations							
Problem	Method	80%	90%	95%	99%	80%	90%	95%	99%	bestBound	optimum	totTime	totIter
	tb	0.08	0.21	0.22	0.26	24	35	38	43	897.00	897.00	0.51	71
10teams	mis	0.04	0.05	0.06	0.07	9	11	14	19	897.00	897.00	0.21	66
	mis2	0.04	0.05	0.05	0.07	9	11	14	19	897.00	897.00	0.22	66
	tb	-	-	-	-	-	-	-	-	707.61	997.53	1000.45	3714
a1c1s1	mis	3.68	6.01	10.62	19.39	144	218	296	482	997.53	997.53	39.95	914
	mis2	86.58	189.93	280.78	-	62	118	173	-	982.38	997.53	451.98	296
	tb	0.03	0.03	0.04	0.09	1	2	5	13	1005.66	1005.66	0.24	44
aflow40b	mis	0.05	0.05	0.07	0.16	1	1	2	7	1005.66	1005.66	0.89	44
	mis2	0.04	0.04	0.07	0.17	1	1	2	7	1005.66	1005.66	0.90	44
	tb	21.22	24.30	29.16	29.16	595	654	766	766	62.64	62.64	36.15	1251
danoint	mis	0.18	0.18	0.18	1.03	43	43	43	87	62.64	62.64	1.74	186
	mis2	3.00	3.00	3.00	5.42	43	43	43	72	62.64	62.64	12.46	167
	tb	0.75	1.19	1.61	2.14	183	254	310	368	1200.88	1200.88	3.20	523
fixnet6	mis	0.05	0.12	0.16	0.34	39	65	83	139	1200.88	1200.88	0.70	230
	mis2	0.28	0.43	0.64	1.02	26	39	56	87	1200.88	1200.88	1.79	161
	tb	-	-	-	-	-	-	-	-	-	-	-	-
modglob*	mis	0.34	0.34	1.38	2.01	62	62	303	473	20430900.00	20430947.62	50.31	3573
	mis2	0.58	0.87	3.00	6.06	34	61	274	613	20430900.00	20430947.62	44.83	3079
	tb	-	-	-	-	-	-	-	-	-	-	-	-
momentum1*	mis	0.35	0.59	0.73	1.60	0	3	5	18	72793.30	72793.35	26.21	207
	mis2	-	-	-	-	-	-	-	-	-	-	-	-
	tb	0.01	0.01	0.01	1.03	9	14	16	339	2748.35	2748.35	4.11	825
pp08a	mis	0.13	0.28	0.33	0.52	125	195	213	280	2748.35	2748.35	1.71	696
	mis2	0.03	0.04	0.04	0.68	9	13	14	179	2748.35	2748.35	2.20	540
	tb	60.15	61.70	67.09	77.27	676	705	778	963	28655.10	28694.00	83.70	1046
timtab1	mis	1.51	2.33	3.08	5.03	601	831	978	1294	28694.00	28694.00	6.13	1431
	mis2	2.81	3.48	4.13	5.09	362	433	494	575	28694.00	28694.00	5.83	635
	tb	444.26	517.35	663.03	898.50	1162	1388	1731	2165	83269.00	83592.00	1003.04	2327
timtab2	mis	17.96	35.51	52.70	119.58	1091	1493	1812	2965	83592.00	83592.00	204.90	4080
	mis2	14.36	21.33	29.74	46.28	536	682	827	1131	83592.00	83592.00	64.14	1395
	tb	1.22	1.95	3.14	19.14	146	188	222	254	14210.43	14210.43	357.80	518
tr12-30	mis	18.83	36.41	59.49	88.98	547	669	768	860	14210.43	14210.43	123.18	1015
	mis2	0.63	0.66	0.66	12.69	17	18	18	249	14210.43	14210.43	13.42	272

Table 3.2: Comparison of the effectiveness of various separation methods in moving the lower bound at the root node.

As reported in Table 3.2 *tb* is the most efficient method only in 1 out of 11 instances, namely `aflow40`, and only with little advantage over the competitors. On the other hand, *mis* and *mis2* are much more effective on 10 out of 11 instances, with speedups of 1 to 2 orders of magnitude. As expected, the average density of the cuts generated by *mis* and *mis2* is considerably smaller than *tb*, see Table 3.3. This has a positive effect on the rate of growth of the master solution time as a function of the number of iterations, as reported in column *Master Rate* in the table.

A closer analysis of instance `a1c1s1` provides some insights on the strength of the proposed methods: at each iteration, while *tb* generates weak feasibility and optimality cuts, with no selection criteria for both, *mis* is able to cut the current master solution with just a good optimality cut. This is however not always the best strategy: for example, in `timtab1`, `timtab2` and `tr12-30`, feasibility cuts are really crucial for the effectiveness of the method and should be preferred—hence *mis2* becomes the leading method.

A comparison between *mis* and *mis2* shows that *mis* candidates as the method of choice, as it is usually faster due to the extra computing time that *mis2* spends in generating the additional feasibility cut (at least, in our present implementation); see Table 3.3. Nevertheless, as already mentioned, there are instances such that `timtab2` and `tr12-30` where the extra separation effort is rewarded by a significant improvement of the overall performance.

3.5 Conclusions

We have investigated alternative cut selection criteria for Benders cuts. By using the correspondence between minimal infeasible subsystems of an infeasible LP and the vertices of a so-called *alternative polyhedron*, we were able to define a simple yet effective cut-generation LP allowing for the selection of strong Benders cuts. Computational results on a set of MIPLIB instances show that the proposed method provide a speedup of 1 to 2 orders of magnitude with respect to the textbook one.

Acknowledgments

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 (PRIN 2006 project “Models and algorithms for robust network optimization”).

Problem	Method	# cuts	# opt.	# feas.	AvgD	MRate	AvgT
	tb	107	36	71	383	1.14E-04	1.94E-04
10teams	mis	66	0	66	53	3.97E-05	1.94E-04
	mis2	66	0	66	53	3.44E-05	2.39E-04
	tb	5893	3714	2179	76	1.65E-04	6.01E-03
a1c1s1	mis	914	906	8	26	2.63E-05	3.05E-02
	mis2	577	296	281	14	4.13E-05	1.52E+00
	tb	44	0	44	252	4.74E-06	4.24E-03
aflow40b	mis	44	0	44	242	-5.93E-06	1.86E-02
	mis2	44	0	44	242	1.04E-05	1.89E-02
	tb	1412	1251	161	48	3.09E-06	2.02E-02
danoint	mis	186	186	0	37	5.25E-06	8.72E-03
	mis2	180	167	13	35	4.99E-06	7.38E-02
	tb	806	523	283	46	1.05E-05	1.24E-03
fixnet6	mis	230	210	20	22	9.38E-06	2.01E-03
	mis2	321	160	161	24	1.60E-05	9.59E-03
	tb	-	-	-	-	-	-
modglob*	mis	3573	3557	16	31	6.74E-06	2.30E-03
	mis2	3088	3077	11	29	5.84E-06	6.25E-03
	tb	-	-	-	-	-	-
momentum1*	mis	414	383	31	143	3.18E-04	2.61E-02
	mis2	-	-	-	-	-	-
	tb	901	825	76	40	7.97E-06	3.14E-04
pp08a	mis	696	688	8	17	3.52E-06	5.91E-04
	mis2	613	540	73	16	3.45E-06	2.46E-03
	tb	2083	1042	1041	56	2.52E-06	4.37E-04
timtab1	mis	1431	1354	77	17	4.31E-06	1.10E-03
	mis2	1268	633	635	10	8.82E-06	4.88E-03
	tb	4609	2316	2293	103	8.98E-05	5.98E-04
timtab2	mis	4080	3918	162	45	1.90E-05	3.29E-03
	mis2	2783	1388	1395	23	3.79E-05	1.31E-02
	tb	1026	513	513	144	4.13E-03	7.69E-04
tr12-30	mis	1015	999	16	44	3.20E-04	4.55E-03
	mis2	544	272	272	19	6.75E-05	4.02E-02

Table 3.3: Statistics on the Benders cuts generated by the different methods.

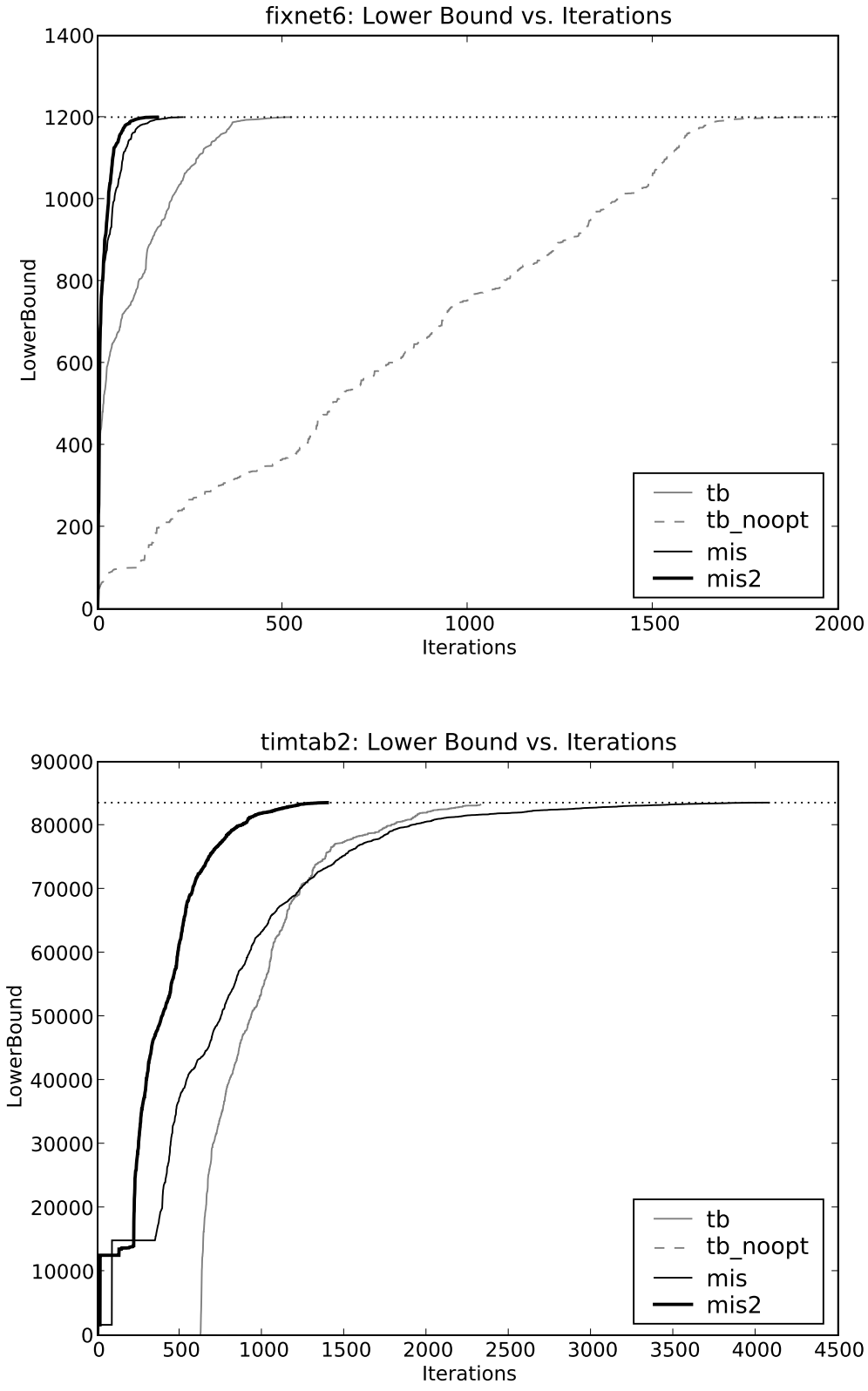


Figure 3.1: Lower bound growth vs. iterations with different separation methods. The dotted line is the known optimal value. For timtab2, tb_noopt *was not able to improve its initial null lower bound.*

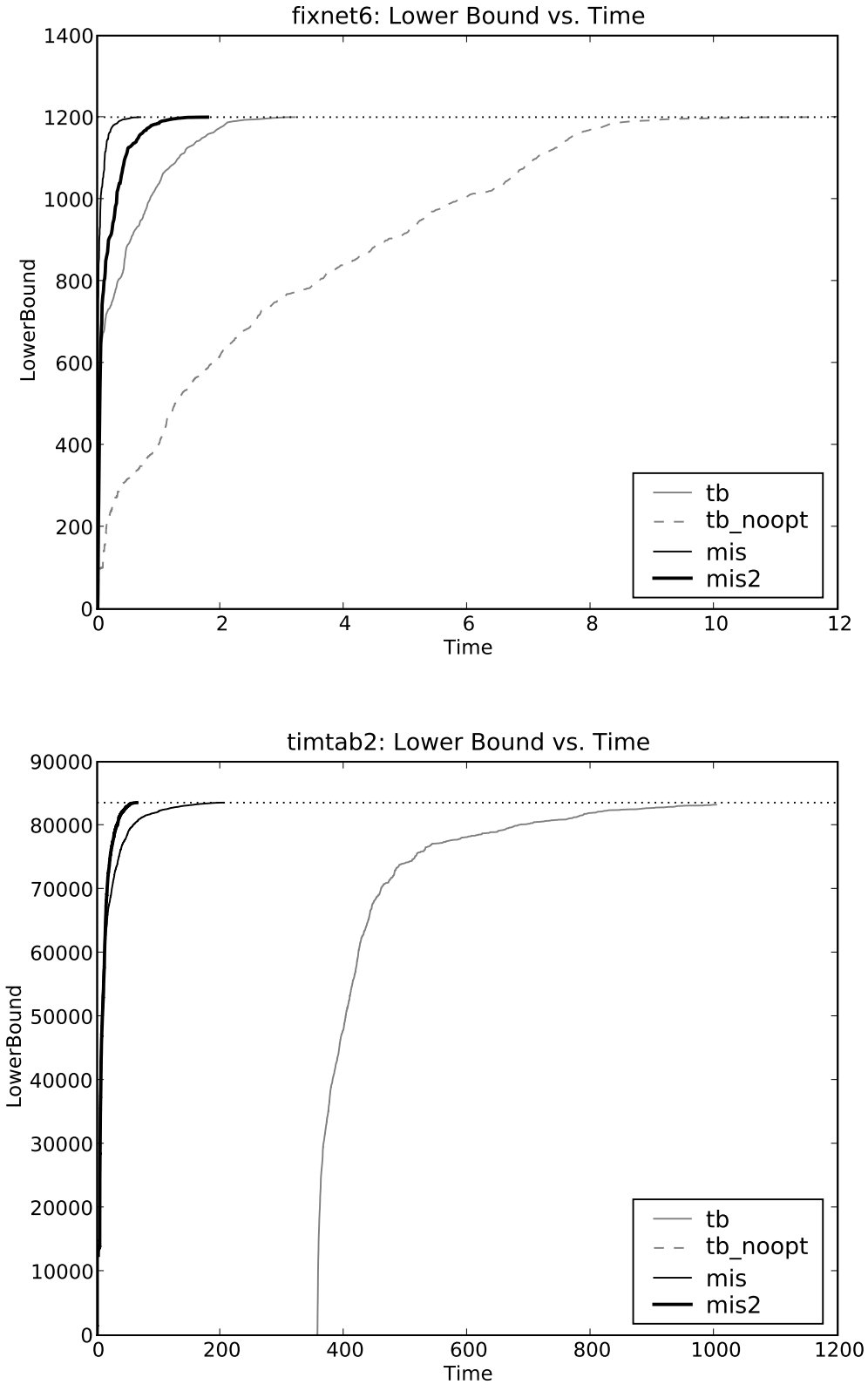


Figure 3.2: Lower bound growth vs. time with different separation methods. The dotted line is the known optimal value. For timtab2, tb_noopt *was not able to improve its initial null lower bound.*

Chapter 4

Feasibility Pump 2.0

Finding a feasible solution of a given Mixed-Integer Programming (MIP) model is a very important $\mathcal{NP}$-complete problem that can be extremely hard in practice.

Feasibility Pump (FP) is a heuristic scheme for finding a feasible solution to general MIPs recently proposed by Fischetti, Glover, and Lodi [34] and further improved by Fischetti, Bertacco and Lodi [13] and Achterberg and Berthold [4]. The FP heuristic turns out to be quite successful in finding feasible solutions even for hard MIP instances, and is currently implemented in many optimization solvers, both commercial and open-source.

FP can be viewed as a clever way of rounding a sequence of fractional solutions of the LP relaxation, until a feasible one is eventually found. In the attempt of improving the FP success rate, one can therefore try to replace the original rounding operations (which are fast and simple, but somehow blind) with more clever rounding heuristics.

In this chapter we investigate the use of a diving-like procedure based on rounding and constraint propagation. This is a basic tool from Constraint Programming, which is used in modern Branch&Cut (B&C) codes for node preprocessing.

Extensive computational results on a large testbed of both binary and general integer MIPs from the literature show that this is a promising direction for improving the FP success rate significantly, with a typically better quality of the feasible solutions found.

4.1 The Feasibility Pump

Suppose we are given a MIP:

$$\min\{c^T x : Ax \leq b,\ x_j \text{ integer } \forall j \in I\}$$

and let $P = \{x : Ax \leq b\}$ be the corresponding LP relaxation polyhedron. Here, A is an $m \times n$ matrix, and $I \subseteq \{1, 2, \cdots, n\}$ is the index set of the variables constrained to be integer. We assume that system $Ax \leq b$ includes finite lower and upper bound constraints of the form

$$l_j \leq x_j \leq u_j \ \forall j \in I$$

With a little abuse of terminology, we say that a point x is "integer" if x_j is integer for all $j \in I$ (thus ignoring the continuous components), and fractional otherwise. Moreover, we call LP-feasible a point $x \in P$. Finally, we define the distance between two given points x and $\tilde{x}$, with $\tilde{x}$ integer, as

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

again ignoring the contribution of the continuous components.

Starting from an LP-feasible point, the basic FP scheme generates two (hopefully convergent) trajectories of points x^* and $\tilde{x}$, which satisfy feasibility in a complementary way: points x^* are LP-feasible, but may not be integer, whereas points $\tilde{x}$ are integer, but may not be LP-feasible.

input : MIP $\equiv \min\{c^T x : x \in P,\ x_j \text{ integer } \forall j \in I\}$
output: a feasible MIP solution x^* (if found)

```
1 x* = arg min{c^T x : x ∈ P}
2 while not termination condition do
3     if x* is integer then return x*
4     x̃ = Round (x*)
5     if cycle detected then Perturb (x̃)
6     x* = LinearProj (x̃)
7 end
```

Figure 4.1: Feasibility Pump—the basic scheme

The two sequences of points are obtained as follows: at each iteration (called *pumping cycle*), a new integer point $\tilde{x}$ is obtained from the fractional x^* by simply rounding its integer-constrained components to the nearest integer, while a new fractional point x^* is obtained as a point of the LP relaxation that minimizes $\Delta(x, \tilde{x})$. The procedure stops if the new x^* is integer or if we have reached a termination condition—usually, a time or iteration limit. Some care must be taken in order to avoid cycling, usually through random perturbations. An outline of this basic scheme is given in Figure 4.1.

According to this scheme, there are three essential FP ingredients:

Round This is the function called to transform an LP-feasible point into an integer one. The standard choice is to simply round each component x_j^* with $j \in I$ to the nearest integer $[x_j^*]$ (say), while leaving the continuous components unchanged.

LinearProj This function is somehow the inverse of the previous one, and is responsible for calculating an LP-feasible point x^* from the current integer $\tilde{x}$. A standard choice is to solve the following LP:

$$x^* = \arg\min\{\Delta(x, \tilde{x}) :\ x \in P\}$$

In the binary case the distance function $\Delta(\cdot, \tilde{x})$ can easily be linearized as

$$\Delta(x, \tilde{x}) = \sum_{j \in I : \tilde{x}_j = 1} (1 - x_j) + \sum_{j \in I : \tilde{x}_j = 0} x_j$$

In the general integer case, however, the linearization requires the use of additional variables and constraints to deal with integer-constrained components $\tilde{x}_j$ with $l_j < \tilde{x}_j < u_j$. To be more specific, the distance function reads

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

with the addition of constraints

$$d_j \geq x_j - \tilde{x}_j \text{ and } d_j \geq \tilde{x}_j - x_j$$

Perturb This function is used to perturb an integer point when a cycle is detected. The standard choice is to apply a weak perturbation if a cycle of length one is detected, and a strong perturbation (akin to a restart) otherwise. More details can be found in [13, 34].

It is worth noting that, in absence of cycles and assuming the MIP is feasible, FP converges in a finite number of steps. Indeed, if there are no cycles, at each pumping cycle a different $\tilde{x}$ is visited, hence the finite convergence follows since finitely many choices exists due to the integer variable boundedness.

4.1.1 The general integer case

The above scheme, although applicable "as is" to the general integer case, is not particularly effective in that scenario. This is easily explained by observing that, for general integer variables, one has to decide not only the rounding direction (up or down), as for binary variables, but also the new value. For the same reasons, also the random perturbation phases must be designed much more carefully.

To overcome some of these difficulties, an extended scheme suited for the general integer case has been presented in Bertacco, Fischetti and Lodi [13] and further developed with solution quality considerations by Achterberg and Berthold in [4].

The scheme is essentially a three-staged approach. In Stage 1, the integrality constraint on the general-integer variables is relaxed, and the pumping cycles are carried out only on the binary variables. Stage 1 terminates as soon as a "binary feasible" solution is found, or some termination criterion is reached. The rationale behind this approach is trying to find quickly a solution which is feasible with respect to the binary components, in the hope that the general integer ones will be "almost integer" as well. In Stage 2, the integrality constraint on the general-integer variables is restored and the FP scheme continues. If a feasible solution is not found in Stage 2, a local search phase (Stage 3) around the rounding of a "best" $\tilde{x}$ is triggered as last resort, using a MIP solver as a black box heuristic.

4.2 Constraint propagation

Constraint propagation is a very general concept that appears under different names in different fields of computer science and mathematical programming. It is essentially a form of inference which consists in explicitly forbidding values—or combinations of values—for some problem variables.

In order to get a practical constraint propagation system, two questions need to be answered:

- What does it mean to propagate a single constraint? In our particular case, this means understanding how to propagate a general linear constraint with both integer and continuous variables. The logic behind this goes under the name of bound strengthening [79, 104] (a form of preprocessing) in the integer programming community.

- How do we coordinate the propagation of the whole set of constraints defining our problem?

In the remaining part of this section we will first describe bound strengthening (Section 4.2.1) and then we will describe our constraint propagation system (Section 4.2.2), following the propagator-based approach given by Schulte and Stuckey in [107].

4.2.1 Bound strengthening

Bound strengthening [3, 45, 51, 79, 104] is a preprocessing technique that, given the original domain of a set of variables and a linear constraint on them, tries to infer tighter bounds on the variables. We will now describe the logic behind this technique in the case of a linear inequality of the form:

$$\sum_{j \in C^+} a_j x_j + \sum_{j \in C^-} a_j x_j \leq b$$

where C^+ and C^- denote the index set of the variables with positive and negative coefficients, respectively. We will assume that *all* variables are bounded, with lower and upper bounds denoted by l_j and u_j, respectively. Simple extensions can be made to deal with unbounded (continuous) variables and equality constraints.

The first step to propagate the constraint above is to compute the minimum ($L_{\min}$) and maximum ($L_{\max}$) "activity level" of the constraint:

$$L_{\min} = \sum_{j \in C^+} a_j l_j + \sum_{j \in C^-} a_j u_j$$

$$L_{\max} = \sum_{j \in C^+} a_j u_j + \sum_{j \in C^-} a_j l_j$$

Now we can compute updated upper bounds for variables in C^+ as

$$\overline{u}_j = l_j + \frac{b - L_{\min}}{a_j} \tag{4.1}$$

and updated lower bounds for variables in C^- as

$$\overline{l}_j = u_j + \frac{b - L_{\min}}{a_j} \tag{4.2}$$

Moreover, for variables constrained to be integer we can also apply the floor $\lfloor \cdot \rfloor$ and ceiling $\lceil \cdot \rceil$ operators to the new upper and lower bounds, respectively.

It is worth noting that no propagation is possible in case the maximum potential activity change due to a single variable, computed as

$$\max_j \{|a_j(u_j - l_j)|\}$$

is not greater than the quantity $b - L_{\min}$. This observation is very important for the efficiency of the propagation algorithm, since it can save several useless propagator calls. Finally, equations (4.1) and (4.2) greatly simplify in case of binary variables, and the simplified versions should be used in the propagation code for the sake of efficiency.

4.2.2 Propagation algorithm

Constraint propagation systems [100, 106, 107] are built upon the basic concepts of *domain*, *constraint* and *propagator*.

A *domain* D is the set of values a solution x can possibly take. In general, x is a vector $(x_1, \ldots, x_n)$ and D is a Cartesian product $D_1 \times \ldots \times D_n$, where each D_i is the domain of variable x_i. We will denote the set of variables as X.

A *constraint* c is a relation among a subset $var(c) \subseteq X$ of variables, listing the tuples allowed by the constraint itself. This definition is of little use from the computational point of view; in practice, constraint propagation systems implement constraints through *propagators*.

A propagator p implementing[1] a constraint c is a function that maps domains to domains and that satisfies the following conditions:

- p is a *decreasing* function, i.e., $p(D) \subseteq D$ for all domains. This guarantees that propagators only remove values.
- p is a *monotonic* function, i.e., if $D_1 \subseteq D_2$, then $p(D_1) \subseteq p(D_2)$.
- p is *correct* for c, i.e., it does not remove any tuple allowed by c.
- p is *checking* for c, i.e., all domains D corresponding to solutions of c are *fixpoints* for p, i.e., $p(D) = D$. In other words, for every domain D in which all variables

[1] In general, a constraint c is implemented by a collection of propagators; we will consider only the case where a single propagator suffices.

involved in the constraint are fixed and the corresponding tuple is valid for c, we must have $p(D) = D$.

A *propagation solver* for a set of propagators R and some initial domain D finds a fixpoint for propagators $p \in R$.

input : a domain D
input : a set P_f of (fixpoint) propagators p with $p(D) = D$
input : a set P_n of (non-fixpoint) propagators p with $p(D) \subseteq D$
output: an updated domain D

```
1 Q = P_n
2 R = P_f ∪ P_n
3 while Q not empty do
4     p = Pop (Q)
5     D = p(D)
6     K = set of variables whose domain was changed by p
7     Q = Q ∪ {q ∈ R : var(q) ∩ K ≠ ∅}
8 end
9 return D
```

Figure 4.2: Basic propagation engine

A basic propagation algorithm is outlined in Figure 4.2. The propagators in R are assigned to the sets P_f and P_n, depending on their known fixpoint status for domain D—this feature is essential for implementing efficient incremental propagation. The algorithm maintains a queue Q of pending propagators (initially P_n). At each iteration, a propagator p is popped from the queue and executed. At the same time the set K of variables whose domains have been modified is computed and all propagators that share variables with K are added to Q (hence they are *scheduled* for execution).

The complexity of the above algorithm is highly dependent on the domain of the variables. For integer (finite domain) variables, the algorithm terminates in a finite number of steps, although the complexity is exponential in the size of the domain (it is however polynomial in the pure binary case, provided that the propagators are also polynomial, which is usually the case). For continuous variables, this algorithm may not converge in a finite number of steps, as shown in the following example (Hooker [51]):

$$\begin{cases} \alpha x_1 - x_2 \geq 0 \\ -x_1 + x_2 \geq 0 \end{cases} \tag{4.3}$$

where $0 < \alpha < 1$ and the initial domain is $[0, 1]$ for both variables: it can be easily seen that the upper bound on x_1 converges only asymptotically to zero.

4.3 The new FP scheme

As already observed, the rounding function described in the original feasibility pump scheme has the advantage of being extremely fast and simple, but it has also the drawback of completely ignoring the linear constraints of the model. While it is true that the linear part is taken into account in the LinearProj phase, still the "blind" rounding operation can let the scheme fail (or take more iterations than necessary) to reach a feasible solution, even on trivial instances.

Suppose for example we are given a simple binary knapsack problem. It is well known that an LP optimal solution has only one fractional component, corresponding to the so-called critical item. Unfortunately, a value greater than 0.5 for this component will be rounded up to one, thus resulting into an infeasible integer solution (all other components will retain their value, being already zero or one). A similar reasoning can also be done for set covering instances, where finding a feasible solution is also a trivial matter but can require several pumping cycles to an FP scheme.

Our proposal is to merge constraint propagation with the rounding phase, in order to have a more clever strategy that better exploits information about the linear constraints. This integration is based on the following observation: rounding a variable means temporarily fixing it to a given value, so we can in principle propagate this temporary fixing before rounding the remaining variables. This is very similar to what is done in modern MIP solvers during diving phases, but without the overhead of solving the linear relaxations. A sketch of the new rounding procedure is given in Figure 4.3. The procedure works as follows. At each iteration, a free variable $x_j \in I$ is selected and rounded to $[x_j^*]$. The new fixing $x_j = [x_j^*]$ is then propagated by the propagation engine, together with propagators in R. When there are no free variables left, the domain D_j of every variable x_j with $j \in I$ has been reduced to a singleton and we can "read" the corresponding $\tilde{x}$ from D.

input : a domain D and a set R of propagators
input : a fractional vector x^*
output: an integer vector $\tilde{x}$

1 I = index set of integer variables
2 **while** I *not empty* **do**
3 $\quad j$ = `Choose` (I)
4 $\quad D$ = `Propagate` $(D, R, x_j = [x_j^*])$
5 $\quad I$ = index set of integer non-fixed variables
6 **end**
7 read $\tilde{x}$ from D
8 **return** $\tilde{x}$

Figure 4.3: New rounding procedure

With respect to the original "simple" rounding scheme, it is worth noting that:

- When rounding a general integer variable, one can exploit the reduced domain

derived by the current propagation. For example, if the domain of a variable y with fractional value 6.3 is reduced to $[8, 10]$, then it is not clever to round it to values 6 or 7 (as simple rounding would do), but value 8 should be chosen instead.

- On a single iteration, the new rounding scheme strictly dominates the original one, because a feasible simple rounding cannot be ruled out because of constraint propagation. On the other hand, as outlined above, there are cases where the new scheme (but not the original one) can find a feasible solution in just one iteration.

- There is no dominance relation between the two rounding schemes as far as the "feasibility degree" of the resulting $\tilde{x}$ is concerned. This is because constraint propagation can only try to enforce feasibility, but there is no guarantee that the resulting rounded vector will be "more feasible" than the one we would have obtained through simple rounding.

- There is also no dominance relation at the feasibility pump level, i.e., we are not guaranteed to find a feasible solution in less iterations.

- As in diving, but differently from standard rounding, the final $\tilde{x}$ depends on the order in which we choose the next variable to round (and also on the order in which we execute propagators); this ordering can have a great impact on the effectiveness of the overall scheme.

Finally, while in a constraint propagation system it is essential to be able to detect as soon as possible the infeasibility of the final solution (mainly to reduce propagation overhead, but also to get smaller search trees or to infer smaller infeasibility proofs), in the FP application we always need to bring propagation to the end, because we need to choose a value for all integer variables. In our implementation, failed propagators leading to infeasible solutions are simply ignored for the rest of the current rounding, but the propagation continues in the attempt of reducing the degree of infeasibility of the final solution found.

4.4 Implementation

While for binary MIPs the implementation of an FP scheme is quite straightforward, in the general integer case some care must be taken in order to get a satisfactory behavior of the algorithm. Moreover, since the overhead of constraint propagation can be quite large (mainly if compared with the extremely fast simple rounding operation) we need a very efficient implementation of the whole constraint propagation system, that can be achieved through a careful implementation of both the single propagators and of the overall propagation engine. In the rest of the section we will describe important implementation details concerning these issues.

4.4.1 Optimizing FP restarts

Random perturbations, and in particular restarts, are a key ingredient of the original FP scheme even for binary MIPs. They are even more important in the general integer case, where the FP is more prone to cycling. As a matter of fact, our computational experience shows that just adapting the original restart procedure for binary MIPs to the general integer case results in a seriously worsening of the overall behavior.

The FP implementation of Bertacco, Fischetti and Lodi [13] uses a quite elaborated restart scheme to decide how much a single variable has to be perturbed, and how many variables have to be changed. According to this scheme, a single variable x_j is perturbed taking into account the size of its domain: if $u_j - l_j < M$ with a suitable large coefficient M, then the new value is picked randomly within the domain. Otherwise, the new value is picked uniformly in a large neighborhood around the lower or upper bound (if the old value is sufficiently close to one of them), or around the old value.

The number of variables to be perturbed, say RP, is also very important and has to be defined in a conservative way. In [13] , RP is changed dynamically according to the frequency of restarts. In particular, RP decreases geometrically with constant 0.85 at every iteration in which restarts are not performed, while it increases linearly (with a factor of 40) on the others. Finally, RP is bounded by a small percentage (10%) of the number of general integer variables, and the variables to be changed are picked at random at every restart.

4.4.2 Optimizing propagators

The default linear propagator is, by design, general purpose: it must deal with all possible combinations of variable bounds and variable types, and can make no assumption on the kind of coefficients (sign, distribution, etc.).

While it is true that constraints of this type are really used in practice, nevertheless they are often only a small part of the MIP model. As a matter of fact, most of the linear constraints used in MIP modeling have some specific structure that can be exploited to provide a more efficient propagation [3].

For the reasons above, we implemented specialized propagators for several classes of constraints, and an automated analyzer for assigning each constraint to the appropriate class. In particular, we implemented specialized propagators for

- *Knapsack constraints*, i.e., constraints with positive coefficients involving only bounded variables. These assumptions enable several optimizations to be performed. Note that this class of constraints includes both covering and packing constraints.

- *Cardinality constraints*, i.e., constraints providing lower and/or upper bounds on the sum of binary variables. Propagation can be greatly simplified in this case, since we just need to count the number of variables fixed to zero or one. This class includes classical set covering/packing/partioning constraints.

- *Binary logic constraints*, i.e., linear constraints expressing logical conditions between two binary variables, such as implications and equivalences.

According to our computational experience, the above specializations can lead to a speedup of up to one order of magnitude in the propagation phase.

4.4.3 Optimizing constraint propagation

A propagation engine should exploit all available knowledge in order to avoid unnecessary propagator execution [107]. Moreover, when the engine invokes the execution of a propagator, it should provide enough information to allow the most efficient algorithms to be used. Thus the constraint system must support

- *Propagator states*, needed to store data structures for incremental propagation.
- *Modification information* (which variables have been modified and how), needed to implement sub-linear propagators and to implement more accurate fixpoint considerations (see [100, 106, 107]).

A simple and efficient way for supporting these services is to use the so-called *advisors*. Advisors were introduced by Lagerkvist and Schulte in [65] and are used in the open-source constraint programming solver Gecode [40]. They are responsible for updating the propagator state in response to domain changes, and to decide whether a propagator needs to be executed. Each advisor is tied to a (variable, propagator) pair, so that the most specialized behavior can be implemented. Modification information is provided by *propagation events*, see [107]. More details about advised propagation are available in [65].

Finally, since the presence of continuous variables or integer variables with huge domains can make the complexity of propagation too high, we impose a small bound on the number of times the domain of a single variable can be tightened (e.g., 10): when this bound is reached, the domain at hand stops triggering propagator executions within the scope of the current rounding.

4.5 Computational results

In this section we report computational results to compare the two rounding schemes (with and without propagation). Our testbed is made by 43 binary MIP instances from MIPLIB 2003 [1] and by 29 general integer MIP instances from MIPLIB 2003 and [25, 84]. One instance (`stp3d`) was left out because even the first LP relaxation was very computationally demanding, while instances `momentum*` were left out because of numerical problems. Some characteristics of the instances are reported in Tables 4.1 and 4.2.

All algorithms were implemented in C++. We used a commercial LP solver (ILOG Cplex 11.0 [102]) to solve the linear relaxations and to preprocess our instances. All tests have been run on a Intel Core2 Q6600 system (2.40 GHz) with 4GB of RAM. The

Name	m	n	$\|B\|$	best
10teams	230	2,025	1,800	924.0
a1c1s1	3,312	3,648	192	11,503.4
aflow30a	479	842	421	1,158.0
aflow40b	1,442	2,728	1,364	1,168.0
air04	823	8,904	8,904	56,137.0
air05	426	7,195	7,195	26,374.0
cap6000	2,176	6,000	6,000	-2,451,380.0
dano3mip	3,202	13,873	552	686.5
danoint	664	521	56	65.7
disctom	399	10,000	10,000	-5,000
ds	656	67,732	67,732	422.3
fast0507	507	63,009	63,009	174.0
fiber	363	1,298	1,254	405,935.0
fixnet6	478	878	378	3,983.0
glass4	396	322	302	1,200,010,000.0
harp2	112	2,993	2,993	-73,899,800.0
liu	2,178	1,156	1,089	1,122.0
markshare1	6	62	50	1.0
markshare2	7	74	60	1.0
mas74	13	151	150	11,801.2
mas76	12	151	150	40,005.1
misc07	212	260	259	2,810.0
mkc	3,411	5,325	5,323	-563.8
mod011	4,480	10,958	96	-54,558,500.0
modglob	291	422	98	20,740,500.0
net12	14,021	14,115	1,603	214.0
nsrand-ipx	735	6,621	6,620	51,200
nw04	36	87,482	87,482	16,862.0
opt1217	64	769	768	-16.0
p2756	755	2,756	2,756	3,124.0
pk1	45	86	55	11.0
pp08aCUTS	246	240	64	7,350.0
pp08a	136	240	64	7,350.0
protfold	2,112	1,835	1,835	-31.0
qiu	1,192	840	48	-132.9
rd-rplusc-21	125,899	622	457	165,395
set1ch	492	712	240	54,537.8
seymour	4,944	1,372	1,372	423.0
sp97ar	1,761	14,101	14,101	661,677,000.0
swath	884	6,805	6,724	467.4
t1717	551	73,885	73,885	215,991.0
tr12-30	750	1,080	360	130,596.0
vpm2	234	378	168	13.8

Table 4.1: Binary testbed characteristics. m is the number of rows, n the number of columns, $|B|$ the number of binary variables, and `best` *is the objective value of the best known solution.*

Name	m	n	$\|B\|$	$\|I\| - \|B\|$	best
arki001	1,048	1,388	415	123	7,580,810.0
atlanta-ip	21,732	48,738	46,667	106	90.0
gesa2	1,392	1,224	240	168	25,779,900.0
gesa2-o	1,248	1,224	384	336	25,779,900.0
manna81	6,480	3,321	18	3,303	-13,164.0
msc98-ip	15,850	21,143	20,237	53	19,839,500.0
mzzv11	9,499	10,240	9,989	251	-21,718.0
mzzv42z	10,460	11,717	11,482	235	-20,540.0
noswot	182	128	75	25	-41.0
roll300	2,295	1,166	246	492	12,890.0
rout	291	556	300	15	1,077.6
timtab1	171	397	64	107	764,772.0
timtab2	294	675	113	181	1,161,963.0
neos7	1,994	1,556	434	20	721,934.0
neos8	46,324	23,228	23,224	4	-3,719.0
neos10	46,793	23,489	23,484	5	-1,135.0
neos16	1,018	377	336	41	454.0
neos20	2,446	1,165	937	30	-434.0
rococoB10-011000	1,667	4,456	4,320	136	19,772.0
rococoB10-011001	1,677	4,456	4,320	136	22,894.0
rococoB11-010000	3,792	12,376	12,210	166	34,830.0
rococoB11-110001	8,148	12,431	12,265	166	45,593.0
rococoB12-111111	8,978	9,109	8,778	331	40,877.0
rococoC10-001000	1,293	3,117	2,993	124	11,475.0
rococoC10-100001	7,596	5,864	5,740	124	19,251.0
rococoC11-010100	4,010	12,321	12,155	166	30,643.0
rococoC11-011100	2,367	6,491	6,325	166	23,260.0
rococoC12-100000	21,550	17,299	17,112	187	38,607.0
rococoC12-111100	10,842	8,619	8,432	187	38,390.0

Table 4.2: General integer testbed characteristics. m is the number of rows, n the number of columns, $|B|$ the number of binary variables, $|I|-|B|$ the number of general integer variables, and `best` *is the objective value of the best known solution.*

feasibility pump was run with standard parameters, as described in [13, 34], except for `rococo*` instances where we did not force the use of the primal simplex method for reoptimizing the LPs, because it was much slower than using Cplex defaults.

Comparing the performance of different feasibility pump variants is not a simple matter: the original scheme is often modified in order to generate a sequence of solutions of increasing quality, with an increase in the number of parameters that need to be set and a consequent complication of the benchmarking task. We therefore decided to consider a slightly modified two-stages FP scheme, based on the one described in [13], with some minor simplifications, namely:

- We stop as soon as a first feasible solution is found.
- We skip Stage 3. This stage is usually quite computationally demanding (it is also skipped in other FP implementation, for example in SCIP [3]) and is somehow external to the FP scheme.
- We do not perform random perturbation if the distance function improves too slowly. This is because the iteration limits we set both for Stage 1 and for Stage 2 is much smaller than the ones in [13], so the check is not worthwhile.

However, since we still want to test the ability of FP of finding good-quality solutions, for each instance in our testbed we generated a set of instances with increasing difficulty through the addition of an optimality constraint of the type:

$$c^T x \leq z^*(1 + \alpha)$$

where z^* is the best known solution from the literature, and α is the relative allowed optimality gap.

Finally, we tested the FP algorithm with two different iteration limits (IT), namely 250 and 20[2]. The former is used to evaluate the performance of the algorithm as a pure primal heuristic, while the latter simulates the use of the FP at the root node of a generic B&C code, where it is typically not worthwhile to spend a large computing time on a single heuristic. A time limit of 500 seconds was set in all cases.

The performance figures used to compare the algorithm are:

#f Number of solutions found in the testbed. Being the feasibility pump a primal heuristic, this is the most important figure for benchmarking.

nitr Number of iterations needed to find a solution.

time Time needed to find a solution.

value Objective value of the solution found. Although a good-quality solution is very important in a B&C code, in our view this is not the most important figure. Indeed, the main purpose of the feasibility pump is to find quickly a starting

[2]For general integer instances, the Stage 1 iteration limit was set to 100 and 5, respectively.

feasible solution, whose quality can hopefully be improved at a later time through local-search procedures.

As already discussed, random perturbation is a fundamental ingredient of the FP scheme, so it is not completely fair to compare different variants on a single run. For this reasons, for each instance we ran 3 times each FP variant (using different seeds for the internal pseudo-random number generator), and report the results for all 3 runs.

We denote by `std` the original FP based on simple rounding (the one used in the standard FP) and with `prop` the one using propagation-based rounding. Within `prop`, variables are ranked in order of increasing fractionality, with binary variables always preceding the general integer ones. This corresponds to the "smallest domain first" rule in Constraint Programming, and it is also common wisdom in Integer Programming. As a matter of fact, binary variables very often model the most important decisions in the model, and their fixing usually triggers more powerful propagations. Cumulative results are reported in Table 4.3, for both binary and and general integer instances. More detailed results are reported in the Tables 4.5, 4.6, 4.7, 4.8, 4.9 and 4.10[3].

The structure of the tables is as follows: on the vertical axis we have the performance figures (#f, nitr and time), grouped by iteration limit IL, while on the horizontal axis we have the cumulative results of `std` and `prop` and the percentage improvements (impr%) achieved by `prop` with respect to `std` (a positive percentage always means that the new method based on propagation outperformed the standard one).

According to Table 4.3, both for binary and general integer instances there is a substantial improvement for all performance figures and for every value of α, even on the hardest combination of parameters (IT = 20 and $\alpha = 10\%$).

Finally, in Table 4.4 we report a measure of the quality of the solutions found by `std` and `prop`, by counting the number of times a method gave a strictly better solution compared to the other. The results are in favor of `prop` even from this point of view, in that `prop` outperformed `std` for all combinations of settings—overall, `prop` found a strictly better solution 153 times, while `std` only 65 times (214 times the solution quality was the same).

[3]Here the reported number of iterations and computing times are averages over three runs with different random seeds and Column #f gives the number of solutions found (out of three).

			No Gap			100% Gap			10% Gap			Overall		
Testbed	IT	Figure	`std`	`prop`	impr %	`std`	`prop`	impr%	`std`	`prop`	impr%	`std`	`prop`	impr%
binary	20	#f	93	107	15.05%	76	99	30.26%	37	44	18.92%	206	250	21.36%
		nitr	1,055	719	31.85%	1,310	961	26.64%	2,022	1,883	6.87%	4,387	3,563	18.78%
		time	858	813	5.26%	826	761	7.81%	885	856	3.25%	2,569	2,430	5.39%
	250	#f	116	124	6.90%	96	113	17.71%	43	54	25.58%	255	291	14.12%
		nitr	5,203	2,800	46.18%	9,943	5,524	44.44%	21,374	19,083	10.72%	36,520	27,407	24.95%
		time	2,490	1,716	31.07%	2,641	1,123	57.47%	3,200	2,880	10.01%	8,331	5,720	31.35%
general	20	#f	36	47	30.56%	17	34	100.00%	13	20	53.85%	66	101	53.03%
		nitr	1,376	1,023	25.65%	1,505	1,226	18.54%	1,535	1,443	5.99%	4,416	3,692	16.39%
		time	242	235	3.03%	318	288	9.39%	350	366	-4.72%	910	889	2.27%
	250	#f	59	67	13.56%	34	48	41.18%	19	27	42.11%	112	142	26.79%
		nitr	8,214	6,285	23.48%	13,951	10,695	23.34%	17,598	16,148	8.24%	39,763	33,128	16.69%
		time	362	349	3.53%	1,146	836	27.03%	2,076	1,602	22.86%	3,584	2,787	22.24%

Table 4.3: Binary and general integer testbed results; a positive percentage means that the new method (`prop`*) outperformed the standard one (*`std`*).*

Testbed	IT	Gap	`std` better	`prop` better	equal
binary	20	10%	4	8	31
		100%	5	19	19
		None	7	19	17
	250	10%	7	7	29
		100%	8	20	15
		None	9	21	13
general	20	10%	2	5	22
		100%	3	8	18
		None	8	12	9
	250	10%	2	6	21
		100%	5	11	13
		None	5	17	7
Overall	–	–	65	153	214

Table 4.4: Solution quality comparison of the two main variants `std` *and* `prop`*.*

4.6 Conclusions and future directions of work

In this chapter a new version of the Feasibility Pump heuristic has been proposed and evaluated computationally. The idea is to exploit a concept borrowed from Constraint Programming (namely, constraint propagation) to have a more powerful rounding operation. The computational results on both binary and general integer MIPs show that the new method finds more (and better) feasible solutions than its predecessor, typically in a shorter computing time.

Future work should address the issue of integrating in the best possible way the new rounding scheme within a most elaborated FP code for MIPs with general integer variables, such as the one proposed by Achterberg and Berthold [4].

Acknowledgments

This work was supported by the University of Padova "Progetti di Ricerca di Ateneo", under contract no. CPDA051592 (project: Integrating Integer Programming and Constraint Programming) and by the Future and Emerging Technologies unit of the EC (IST priority), under contract no. FP6-021235-2 (project ARRIVAL).

	$IL = 20$						$IL = 250$					
	std			prop			std			prop		
Instance	nitr	time	#f	nitr	time	#f	nitr	time	#f	nitr	time	#f
10teams	20	1.07	0	20	0.93	0	215	11.38	1	194	8.36	2
a1c1s1	5	0.79	3	3	0.88	3	5	0.80	3	3	0.87	3
aflow30a	20	0.03	0	5	0.02	3	39	0.03	3	5	0.02	3
aflow40b	6	0.06	3	4	0.08	3	6	0.06	3	4	0.07	3
air04	12	19.46	2	5	8.73	3	14	21.99	3	5	8.66	3
air05	15	8.85	3	1	1.68	3	15	8.87	3	1	1.69	3
cap6000	3	0.04	3	1	0.06	3	3	0.04	3	1	0.06	3
dano3mip	2	19.24	3	1	18.75	3	2	19.53	3	1	18.67	3
danoint	4	0.11	3	2	0.12	3	4	0.11	3	2	0.12	3
disctom	2	4.23	3	2	4.03	3	2	4.24	3	2	4.02	3
ds	20	98.11	0	20	122.58	0	84	500.85	0	40	240.82	3
fast0507	5	29.04	3	1	23.35	3	5	29.30	3	1	22.35	3
fiber	3	0.01	3	1	0.01	3	3	0.01	3	1	0.01	3
fixnet6	2	0.00	3	1	0.01	3	2	0.00	3	1	0.01	3
glass4	20	0.01	0	20	0.02	0	49	0.02	3	49	0.03	3
harp2	20	0.01	0	5	0.01	3	205	0.09	2	5	0.01	3
liu	1	0.04	3	1	0.06	3	1	0.04	3	1	0.06	3
markshare1	1	0.00	3	1	0.00	3	1	0.00	3	1	0.00	3
markshare2	1	0.00	3	1	0.00	3	1	0.00	3	1	0.00	3
mas74	2	0.00	3	1	0.00	3	2	0.00	3	1	0.00	3
mas76	2	0.00	3	1	0.00	3	2	0.00	3	1	0.00	3
misc07	20	0.02	0	18	0.03	1	167	0.15	3	52	0.08	3
mkc	3	0.04	3	1	0.06	3	3	0.04	3	1	0.06	3
mod011	1	0.06	3	1	0.08	3	1	0.06	3	1	0.09	3
modglob	1	0.01	3	1	0.01	3	1	0.00	3	1	0.01	3
net12	20	1.93	0	13	2.34	3	154	4.40	3	13	2.21	3
nsrand-ipx	3	0.09	3	1	0.12	3	3	0.10	3	1	0.12	3
nw04	1	0.55	3	2	1.06	3	1	0.56	3	2	1.14	3
opt1217	1	0.01	3	1	0.01	3	1	0.01	3	1	0.01	3
p2756	20	0.04	0	2	0.03	3	250	0.37	0	2	0.03	3
pk1	1	0.00	3	1	0.00	3	1	0.00	3	1	0.00	3
pp08a	4	0.00	3	4	0.00	3	4	0.00	3	4	0.00	3
pp08aCUTS	4	0.01	3	4	0.01	3	4	0.01	3	4	0.01	3
protfold	20	6.83	0	18	7.47	1	146	31.20	2	25	10.75	3
qiu	4	0.11	3	3	0.11	3	4	0.10	3	3	0.12	3
rd-rplusc-21	20	4.68	0	20	11.25	0	250	51.78	0	250	100.23	0
set1ch	3	0.01	3	1	0.01	3	3	0.01	3	1	0.01	3
seymour	3	1.36	3	1	1.30	3	3	1.38	3	1	1.29	3
sp97ar	6	0.97	3	2	0.95	3	6	0.93	3	2	0.89	3
swath	20	0.37	1	20	0.48	0	27	0.52	3	203	6.17	2
t1717	20	87.70	0	20	64.20	0	35	140.90	3	36	143.00	3
tr12-30	8	0.02	3	7	0.02	3	8	0.02	3	7	0.03	3
vpm2	2	0.00	3	1	0.00	3	2	0.00	3	1	0.00	3
Overall	352	285.92	93	240	270.88	107	1,734	829.93	116	933	572.10	124

Table 4.5: Detailed binary results with no optimality constraint.

	$IL = 20$						$IL = 250$					
	std			prop			std			prop		
Instance	nitr	time	#f	nitr	time	#f	nitr	time	#f	nitr	time	#f
10teams	20	1.00	0	7	0.39	3	250	12.09	0	7	0.39	3
a1c1s1	5	0.99	3	3	0.81	3	5	0.98	3	3	0.81	3
aflow30a	16	0.03	1	5	0.02	3	56	0.06	3	5	0.02	3
aflow40b	20	0.12	0	13	0.13	3	127	0.39	3	13	0.13	3
air04	7	16.91	3	5	10.77	3	7	16.65	3	5	10.99	3
air05	2	3.25	3	4	2.24	3	2	3.23	3	4	2.26	3
cap6000	3	0.04	3	1	0.06	3	3	0.05	3	1	0.06	3
dano3mip	2	22.01	3	1	20.70	3	2	22.12	3	1	20.86	3
danoint	4	0.11	3	2	0.12	3	4	0.11	3	2	0.13	3
disctom	2	4.24	3	2	4.04	3	2	4.24	3	2	4.04	3
ds	20	96.21	0	20	124.17	0	78	503.20	0	27	159.43	3
fast0507	5	24.84	3	1	24.14	3	5	25.22	3	1	23.36	3
fiber	20	0.02	0	20	0.04	0	250	0.15	0	250	0.34	0
fixnet6	2	0.00	3	1	0.01	3	2	0.01	3	1	0.01	3
glass4	20	0.01	0	20	0.02	0	250	0.10	0	209	0.13	2
harp2	20	0.01	0	5	0.01	3	141	0.07	2	5	0.02	3
liu	20	0.08	0	20	0.13	0	250	0.44	0	250	1.10	0
markshare1	20	0.00	0	20	0.00	0	250	0.02	0	250	0.03	0
markshare2	20	0.00	0	20	0.01	0	250	0.03	0	250	0.04	0
mas74	2	0.00	3	1	0.01	3	2	0.00	3	1	0.00	3
mas76	2	0.00	3	1	0.00	3	2	0.00	3	1	0.00	3
misc07	20	0.03	0	20	0.04	0	112	0.10	3	37	0.06	3
mkc	3	0.05	3	1	0.06	3	3	0.05	3	1	0.06	3
mod011	1	0.09	3	1	0.12	3	1	0.09	3	1	0.12	3
modglob	1	0.00	3	1	0.01	3	1	0.00	3	1	0.01	3
net12	20	1.94	0	13	2.31	3	154	4.52	3	13	2.28	3
nsrand-ipx	11	0.20	2	1	0.17	3	88	0.76	2	1	0.17	3
nw04	1	0.80	3	1	1.14	3	1	0.83	3	1	1.19	3
opt1217	1	0.01	3	1	0.01	3	1	0.01	3	1	0.01	3
p2756	20	0.03	0	2	0.02	3	250	0.28	0	2	0.02	3
pk1	1	0.00	3	1	0.00	3	1	0.00	3	1	0.00	3
pp08a	4	0.00	3	4	0.01	3	4	0.00	3	4	0.01	3
pp08aCUTS	4	0.01	3	4	0.01	3	4	0.01	3	4	0.01	3
protfold	20	6.79	0	18	7.47	1	146	31.23	2	25	10.71	3
qiu	12	0.18	2	18	0.33	2	15	0.20	3	25	0.39	3
rd-rplusc-21	20	4.45	0	20	7.96	0	250	54.30	0	250	84.02	0
set1ch	3	0.01	3	1	0.01	3	3	0.01	3	1	0.01	3
seymour	2	1.30	3	1	1.23	3	2	1.29	3	1	1.23	3
sp97ar	10	1.30	2	2	1.06	3	87	3.71	2	2	1.02	3
swath	20	0.56	0	20	0.87	0	200	5.73	1	165	5.91	3
t1717	20	87.62	0	10	43.08	3	44	188.11	3	10	43.02	3
tr12-30	8	0.02	3	6	0.04	3	8	0.02	3	6	0.04	3
vpm2	2	0.00	3	2	0.00	3	2	0.00	3	2	0.00	3
Overall	437	275.27	76	320	253.76	99	3,314	880.42	96	1,841	374.44	113

Table 4.6: Detailed binary results with 100% optimality constraint.

	$IL = 20$						$IL = 250$					
	std			prop			std			prop		
Instance	nitr	time	#f	nitr	time	#f	nitr	time	#f	nitr	time	#f
10teams	20	1.00	0	7	0.39	3	250	12.12	0	7	0.38	3
a1c1s1	20	1.47	0	20	1.43	0	250	3.09	0	250	5.18	0
aflow30a	20	0.04	0	20	0.05	0	250	0.21	0	216	0.29	1
aflow40b	20	0.17	0	20	0.19	0	250	0.94	0	250	1.33	0
air04	7	16.70	3	5	10.53	3	7	16.86	3	5	10.77	3
air05	16	15.10	2	5	2.46	3	17	15.64	3	5	2.47	3
cap6000	3	0.05	3	1	0.06	3	3	0.05	3	1	0.06	3
dano3mip	4	21.12	3	1	15.40	3	4	21.16	3	1	15.27	3
danoint	20	0.17	0	3	0.13	3	228	2.82	1	3	0.13	3
disctom	2	4.24	3	2	4.03	3	2	4.24	3	2	4.05	3
ds	20	95.47	0	20	123.24	0	78	502.88	0	64	507.18	0
fast0507	5	26.47	3	3	29.54	3	5	25.60	3	3	30.26	3
fiber	20	0.02	0	20	0.04	0	250	0.14	0	250	0.32	0
fixnet6	20	0.01	0	20	0.03	0	250	0.11	0	250	0.28	0
glass4	20	0.02	0	20	0.02	0	250	0.10	0	250	0.16	0
harp2	20	0.02	0	18	0.03	1	250	0.18	0	34	0.06	3
liu	20	0.08	0	20	0.14	0	250	0.55	0	250	1.06	0
markshare1	20	0.00	0	20	0.00	0	250	0.02	0	250	0.03	0
markshare2	20	0.00	0	20	0.00	0	250	0.02	0	250	0.04	0
mas74	20	0.01	0	20	0.01	0	250	0.04	0	250	0.11	0
mas76	2	0.00	3	1	0.00	3	2	0.00	3	1	0.00	3
misc07	20	0.03	0	20	0.04	0	250	0.25	0	176	0.26	2
mkc	20	0.11	0	20	0.25	0	250	0.83	0	250	2.54	0
mod011	4	0.29	3	4	0.33	3	4	0.28	3	4	0.33	3
modglob	3	0.00	3	3	0.01	3	3	0.01	3	3	0.01	3
net12	20	2.75	0	20	4.21	0	250	7.28	0	250	16.56	0
nsrand-ipx	20	0.37	0	20	0.93	0	250	4.80	0	250	11.31	0
nw04	15	3.62	1	20	6.30	1	23	4.87	3	43	11.80	3
opt1217	2	0.01	3	1	0.01	3	2	0.01	3	1	0.01	3
p2756	20	0.03	0	20	0.07	0	250	0.23	0	250	0.76	0
pk1	20	0.01	0	20	0.01	0	250	0.11	0	250	0.15	0
pp08a	20	0.01	0	20	0.02	0	250	0.05	0	250	0.21	0
pp08aCUTS	20	0.02	0	20	0.03	0	250	0.12	0	250	0.33	0
protfold	20	8.76	0	20	12.80	0	250	88.78	0	250	147.56	0
qiu	12	0.19	3	20	0.23	0	12	0.20	3	29	0.35	3
rd-rplusc-21	20	4.74	0	20	8.04	0	250	53.75	0	250	83.79	0
set1ch	20	0.02	0	20	0.04	0	250	0.13	0	250	0.32	0
seymour	2	1.30	3	1	1.24	3	2	1.29	3	1	1.23	3
sp97ar	20	2.56	0	20	3.08	0	250	21.56	0	250	31.02	0
swath	20	0.70	0	20	0.82	0	250	8.09	0	250	11.04	0
t1717	20	87.33	0	13	59.09	3	62	267.10	2	13	58.70	3
tr12-30	20	0.07	0	20	0.19	0	250	0.23	0	250	2.17	0
vpm2	17	0.00	1	20	0.01	0	170	0.04	1	250	0.07	0
Overall	674	295.07	37	628	285.49	44	7,125	1,066.76	43	6,361	959.98	54

Table 4.7: Detailed binary results with 10% optimality constraint.

	$IL = 20$						$IL = 250$					
	`std`			`prop`			`std`			`prop`		
Instance	nitr	time	#f	nitr	time	#f	nitr	time	#f	nitr	time	#f
arki001	20	0.45	0	20	0.49	0	250	5.34	0	250	5.79	0
atlanta-ip	20	25.62	0	20	26.58	0	250	36.94	0	250	45.46	0
gesa2	4	0.02	3	3	0.03	3	4	0.02	3	3	0.03	3
gesa2-o	6	0.02	3	5	0.03	3	6	0.02	3	5	0.02	3
manna81	20	0.45	0	2	0.09	3	50	1.32	3	2	0.09	3
msc98-ip	20	15.97	0	10	13.62	3	21	14.52	3	9	13.65	3
mzzv11	20	19.80	0	4	18.76	3	100	32.07	2	4	18.76	3
mzzv42z	20	8.81	0	2	7.50	3	26	8.76	3	2	7.58	3
noswot	2	0.00	3	1	0.00	3	2	0.00	3	1	0.00	3
roll3000	20	0.31	0	20	0.41	0	250	2.71	0	227	3.14	1
rout	20	0.02	1	18	0.03	2	24	0.02	3	18	0.03	3
timtab1	20	0.04	0	12	0.03	3	250	0.42	0	12	0.03	3
timtab2	20	0.08	0	20	0.10	0	250	0.80	0	250	1.25	0
neos7	19	0.12	1	3	0.06	3	7	0.04	3	3	0.06	3
neos8	20	0.10	0	1	0.05	3	235	0.90	1	1	0.05	3
neos10	20	0.41	0	1	0.04	3	108	1.11	2	1	0.04	3
neos16	20	0.16	0	20	0.19	0	250	1.69	0	250	1.82	0
neos20	20	0.08	0	20	0.10	0	250	0.69	0	250	0.85	0
rococoB10-011000	12	0.22	3	20	0.32	0	12	0.22	3	29	0.34	3
rococoB10-011001	12	0.22	3	20	0.33	0	12	0.22	3	27	0.37	3
rococoB11-010000	16	0.61	3	20	0.97	0	16	0.61	3	33	1.18	3
rococoB11-110001	20	1.63	1	20	2.02	0	22	1.57	3	26	2.10	3
rococoB12-111111	20	2.45	0	20	2.84	0	250	7.36	0	250	8.46	0
rococoC10-001000	12	0.08	3	13	0.09	3	12	0.08	3	13	0.10	3
rococoC10-100001	14	0.15	3	20	0.35	0	14	0.16	3	126	1.47	3
rococoC11-010100	5	0.43	3	2	0.43	3	5	0.43	3	2	0.43	3
rococoC11-011100	6	0.30	3	2	0.31	3	6	0.31	3	2	0.30	3
rococoC12-100000	20	1.39	0	20	1.84	0	45	1.66	3	47	2.41	3
rococoC12-111100	11	0.69	3	2	0.60	3	11	0.67	3	2	0.59	3
Overall	459	80.63	36	341	78.21	47	2,738	120.66	59	2,095	116.40	67

Table 4.8: Detailed general integer results with no optimality constraint.

	$IL = 20$						$IL = 250$					
	`std`			`prop`			`std`			`prop`		
Instance	nitr	time	#f	nitr	time	#f	nitr	time	#f	nitr	time	#f
arki001	20	0.38	0	20	0.56	0	250	5.01	0	250	7.23	0
atlanta-ip	20	23.14	0	20	24.09	0	250	36.01	0	225	42.14	1
gesa2	4	0.02	3	3	0.03	3	4	0.02	3	3	0.03	3
gesa2-o	6	0.02	3	5	0.03	3	6	0.02	3	5	0.03	3
manna81	20	0.46	0	2	0.09	3	50	1.29	3	2	0.09	3
msc98-ip	20	16.00	0	10	13.76	3	21	14.41	3	9	13.65	3
mzzv11	20	24.71	0	11	24.43	3	114	37.19	2	7	23.64	3
mzzv42z	20	12.02	0	3	9.20	3	6	9.24	3	3	9.29	3
noswot	2	0.00	3	1	0.00	3	2	0.00	3	1	0.00	3
roll3000	20	0.36	0	20	0.49	0	55	0.25	3	91	1.23	3
rout	8	0.01	3	8	0.03	3	21	0.02	3	17	0.04	3
timtab1	20	0.05	0	7	0.02	3	250	0.58	0	7	0.02	3
timtab2	20	0.10	0	20	0.14	0	250	1.06	0	250	1.81	0
neos7	20	0.12	0	20	0.16	0	97	0.29	2	14	0.09	3
neos8	4	0.03	3	1	0.05	3	4	0.03	3	1	0.04	3
neos10	17	0.10	2	1	0.06	3	19	0.08	3	1	0.06	3
neos16	20	0.16	0	20	0.19	0	250	1.68	0	250	1.81	0
neos20	20	0.08	0	20	0.11	0	250	0.72	0	250	0.85	0
rococoB10-011000	20	0.47	0	20	0.39	0	250	3.80	0	250	2.92	0
rococoB10-011001	20	0.46	0	20	0.39	0	250	3.87	0	250	2.94	0
rococoB11-010000	20	2.14	0	20	4.04	0	250	19.80	0	250	49.40	0
rococoB11-110001	20	6.24	0	20	4.49	0	250	85.88	0	250	34.12	0
rococoB12-111111	20	9.60	0	20	4.89	0	250	63.87	0	250	29.90	0
rococoC10-001000	20	0.15	0	20	0.15	0	250	0.98	0	250	1.79	0
rococoC10-100001	20	0.88	0	20	0.60	0	250	21.21	0	250	8.15	0
rococoC11-010100	20	2.16	0	20	1.09	0	250	20.92	0	35	1.65	3
rococoC11-011100	20	0.46	0	20	0.81	0	250	3.57	0	121	3.55	2
rococoC12-100000	20	4.86	0	20	4.71	0	250	43.78	0	250	40.73	0
rococoC12-111100	20	0.86	0	17	1.09	1	250	6.46	0	23	1.57	3
Overall	501	106.04	17	409	96.09	34	4,649	382.04	34	3,565	278.77	48

Table 4.9: Detailed general integer results with 100% optimality constraint.

	$IL = 20$						$IL = 250$					
	std			prop			std			prop		
Instance	nitr	time	#f	nitr	time	#f	nitr	time	#f	nitr	time	#f
arki001	20	0.37	0	20	0.59	0	250	4.96	0	250	7.17	0
atlanta-ip	20	26.36	0	20	25.96	0	250	102.96	0	250	69.21	0
gesa2	4	0.02	3	3	0.03	3	4	0.02	3	3	0.03	3
gesa2-o	6	0.02	3	12	0.06	3	6	0.02	3	12	0.06	3
manna81	20	0.49	0	2	0.10	3	50	5.76	3	2	0.09	3
msc98-ip	20	21.52	0	20	19.80	0	250	102.36	0	250	49.42	0
mzzv11	20	26.35	0	20	27.19	0	250	118.20	0	250	76.92	0
mzzv42z	20	12.11	0	20	19.57	0	250	53.67	0	250	65.94	0
noswot	2	0.00	3	1	0.00	3	2	0.00	3	1	0.00	3
roll3000	20	0.60	0	20	0.57	0	250	6.23	0	250	6.88	0
rout	20	0.04	0	20	0.05	0	202	0.21	1	141	0.27	3
timtab1	20	0.07	0	20	0.08	0	250	0.80	0	250	0.95	0
timtab2	20	0.17	0	20	0.23	0	250	1.79	0	250	2.81	0
neos7	16	0.08	1	18	0.12	1	98	0.51	3	89	0.37	3
neos8	4	0.03	3	1	0.05	3	4	0.03	3	1	0.05	3
neos10	20	0.44	0	19	0.47	1	250	3.53	0	127	2.14	3
neos16	20	0.16	0	20	0.19	0	250	1.69	0	250	1.83	0
neos20	20	0.10	0	20	0.27	0	250	0.70	0	250	2.38	0
rococoB10-011000	20	0.48	0	20	0.39	0	250	3.88	0	250	2.87	0
rococoB10-011001	20	0.45	0	20	0.38	0	250	3.90	0	250	2.90	0
rococoB11-010000	20	2.15	0	20	4.02	0	250	19.97	0	250	51.53	0
rococoB11-110001	20	6.62	0	20	4.54	0	250	95.55	0	250	32.95	0
rococoB12-111111	20	8.48	0	20	4.77	0	250	68.65	0	250	28.46	0
rococoC10-001000	20	0.15	0	20	0.16	0	250	1.00	0	250	1.70	0
rococoC10-100001	20	0.86	0	20	0.87	0	250	21.60	0	250	10.65	0
rococoC11-010100	20	2.18	0	5	0.45	3	250	20.57	0	5	0.45	3
rococoC11-011100	20	0.46	0	20	1.29	0	250	3.55	0	250	17.07	0
rococoC12-100000	20	4.90	0	20	4.69	0	250	43.48	0	250	40.28	0
rococoC12-111100	20	0.87	0	20	5.17	0	250	6.48	0	250	58.49	0
Overall	512	116.53	13	481	122.06	20	5,866	692.07	19	5,381	533.87	27

Table 4.10: Detailed general integer results with 10% optimality constraint.

Appendix A

Fast Approaches to Improve the Robustness of a Railway Timetable

The Train Timetabling Problem (TTP) consists in finding an effective train schedule on a given railway network. The schedule needs to satisfy some operational constraints given by capacities of the network and security measures. Moreover, it is required to exploit efficiently the resources of the railway infrastructure.

In practice, however, the maximization of some objective function is not enough: the solution is also required to be robust against delays/disturbances along the network. Very often, the robustness of optimal solutions of the original problem turns out to be not enough for their practical applicability, whereas easy-to-compute robust solutions tend to be too conservative and thus unnecessarily inefficient. As a result, practitioners call for a fast yet accurate method to find the most robust timetable whose efficiency is only slightly smaller than the theoretical optimal one. This is typically obtained by adding "buffer times" to the schedule according to certain simple rules (see §2.2 in [64]).

The purpose of the present chapter is to propose and evaluate new methods to improve the robustness of a given TTP solution. In particular, we addresss the aperiodic (non cyclic) TTP version described in [17]. Our approach combines Linear Programming (LP) with Stochastic Programming (SP) and Robust Optimization techniques.

We propose the following three-stage framework as a practical tool for improving and testing robust solutions for the TTP:

stage 1) nominal problem solution: the TTP is modeled without taking into account robustness, and solved (not necessarily to optimality) by a standard MIP solver or using ad-hoc heuristics.

stage 2) robustness training: borrowing an expression typical of Artificial Intelligence field, starting from the previous stage solution the model is "trained to robustness", typically by exploiting a restricted set of samples (scenarios).

stage 3) robustness validation: the robustness of the final solution found in stage

2 is evaluated by using a validation tool, thus allowing for a fair comparison of different training methods.

In this chapter we focus mainly on the second stage, robustness training. We assume nominal solutions are given in input while, as far as the validation stage is concerned, we use a simple LP validation model.

A.1 Literature review

The TTP problem has two main variants: the *periodic* and *aperiodic* versions. The periodic TTP's goal is to design a timetable that is operated cyclically after a (small) period of time; this is a typical requirement for passenger trains in order to come up with an easy-to-remember timetable. The first authors who developed a model for generating periodic timetables were Serafini and Ukovic [108], who introduced a mathematical model called *Periodic Event Scheduling Problem* (PESP). In PESP, a set of repetitive events is scheduled under periodic time-window constraints. Consequently, the events are scheduled for one cycle in such a way that the cycle can be repeated. Most models for periodic TTP are based on PESP. A natural LP formulation of PESP is quite weak, due to kind of big-M constraints (where M is the period). An alternative stronger formulation is treated in Nachtigall [86] and Liebchen and Peeters [66, 93] among others, and is based on cycle bases and separation of valid inequalities.

As to robustness, Kroon et al. [64] describe a stochastic optimization variant of PESP. Their model explicitly takes into account stochastic disturbances of the railway processes, distinguishing between a planned timetable and several realizations of the timetable under pre-determined stochastic disturbances. The model can be used to allocate time supplements and buffer times to the processes in the planned timetable in such a way that the average delay of the realizations of the trains is minimized. In order to keep the computation times within an acceptable bound, they start with an existing timetable and fix the precedences among trains. They show that a substantial increase in robustness can be achieved by taking into account stochastic disturbances in the design of the timetable. For the case of one trip serving 10 stations, Liebchen and Stiller [67] provide a theoretical explanation for the effects observed empirically in Kroon et al. [64]. Finally a new notion of robustness, called *recoverable robustness*, has been proposed in [68], which integrate the notion of robustness and recoverability into a common framework. Applications to integrated timetabling/delay management in railway systems are described and evaluated in [19, 68, 69].

The aperiodic TTP is especially relevant on heavy-traffic, long-distance corridors, where the capacity of the infrastructure is limited due to greater traffic densities, and competitive pressure among the train operators is expected to increase in the near future. In the Caprara et al. [17] setting, a train operator applies for allocating its trains on the infrastructure, and specify a profit for the "ideal timetable" they are asking for. Then the infrastructure manager collects all requests from train operators and computes a feasible timetable maximizing the overall profit, i.e., the difference

between the profits of the scheduled trains and a cost-penalty function, which takes into account the deviations of the final timetables with respect to the ideal ones (possibly leaving some trains unscheduled).

Different ILP models based on a graph representation of the problem were presented in [17,18]. In these papers the problem is modeled by means of a directed acyclic multi-graph, in which nodes correspond to arrival and departure events from the stations and arise at some discretized time instants, and arcs correspond to train stops within a station or to train trips. A Lagrangian relaxation method is used to derive bounds on the optimal solution value as well as to drive a heuristic procedure.

A.2 The Nominal Model

In this section we describe the specific aperiodic TTP problem we consider, and give a basic event-based formulation for the "nominal" version where robustness is not taken into account.

Following [17], the aperiodic TTP can be described as follows: given a railway network, described as a set of stations connected by tracks, and an ideal train timetable, find an actual train schedule satisfying all the operational constraints and having a minimum distance from the ideal timetable.

The entities involved in modelling the problem are the following:

railway network: a graph $N = (\mathcal{S}, \mathcal{L})$, where $\mathcal{S}$ is the set of stations and $\mathcal{L}$ is the set of tracks connecting them.

trains: a train corresponds to a simple path on the railway network N. The set of trains is denoted by T. For each train $h \in T$ we have an ideal profit π_h (the profit of the train if scheduled exactly as in the ideal timetable), a stretch penalty θ_h (the train *stretch* being defined as the difference between the running times in the actual and ideal timetables) and a shift penalty σ_h (the train *shift* being defined as the absolute difference between the departures from the first station in the actual and ideal timetables).

events: arrivals and departures of the trains at the stations. The set of all the events is denoted by E. With a small abuse of notation, we will denote by t_i^h both the i-th event of train h and its associated time. We also define

- A: set of all arrival events
- D: set of all departure events

whereas A_S, D_S and E_S denote the restriction of the above sets to a particular station S. Each train h is associated with an ordered sequence of length $len(h)$ of departure/arrival events t_i^h such that $t_{i+1}^h \geq t_i^h$, the first and last event of train h being denoted by t_1^h and $t_{len(h)}^h$, respectively. In addition, let $\overline{t_i^h}$ denote the ideal time for event t_i^h.

(partial) schedule: a time assignment to all the events associated with a subset of trains.

objective: maximize the overall profit of the scheduled trains, the profit of train h being computed as

$$\rho_h = \pi_h - \sigma_h \, shift_h - \theta_h \, stretch_h$$

where

$$shift_h = |t_1^h - \bar{t}_1^h|$$

and

$$stretch_h = (t_{len(h)}^h - t_1^h) - (\bar{t}_{len(h)}^h - \bar{t}_1^h)$$

denote the shift and stretch associated with train h, respectively. Trains with negative profit are intended to remain unscheduled and do not contribute to the overall profit.

Operational constraints include:

time windows: it is possible to shift an event from its ideal time only within a given time window;

headway times: for safety reasons, a minimum time distance between two consecutive arrival/departure events from the same station is imposed;

track capacities: overtaking between trains is allowed only within stations (assumed of infinite capacity).

As already mentioned, in the present chapter we do not address the solution of the nominal TTP problem explicitly, in that a nominal solution is assumed to be provided in input. Nevertheless, we next outline the structure of a MIP formulation for the nominal TTP problem, since a relaxed version of it is at the basis of the LP models used in Sections A.4 and A.5.

Although in the nominal problem one is allowed to leave some trains unscheduled, to simplify our presentation we consider a non-congested network where one is required to schedule all the trains. A natural event-based model in the spirit of the Periodic Event Scheduling Problem (PESP) formulation used in the periodic (cyclic) case [108] can be sketched as follows:

$$z^* = \max \sum_{h \in T} \rho_h$$

$$t_{i+1}^h - t_i^h \geq d_{i,i+1}^h \quad \forall h \in T, i = 1, \ldots, len(h) - 1 \tag{A.1}$$

$$|t_i^h - t_j^k| \geq \Delta_a \quad \forall t_i^h, t_j^k \in A_S, \forall S \in \mathcal{S} \tag{A.2}$$

$$|t_i^h - t_j^k| \geq \Delta_d \quad \forall t_i^h, t_j^k \in D_S, \forall S \in \mathcal{S} \tag{A.3}$$

$$t_{i+1}^h < t_{j+1}^k \Leftrightarrow t_i^h < t_j^k \quad \forall t_i^h, t_j^k \in D_S, \forall S \in \mathcal{S} \tag{A.4}$$

$$\rho_h = \pi_h - \sigma_h|t_1^h - \bar{t}_1^h| - \theta_h((t_{len(h)}^h - t_1^h) - (\bar{t}_{len(h)}^h - \bar{t}_1^h)) \quad \forall h \in T \tag{A.5}$$

$$l \leq t \leq u \quad \forall t \in E \tag{A.6}$$

Constraints (A.1) impose a minimum time difference $d_{i,i+1}$ between two consecutive events of the same train, thus imposing minimum trip duration (trains are supposed to travel always at the maximum allowed speed for the track) and minimum stop time at the stations.

Constraints (A.2)-(A.3) model the headway times between two consecutive arrival or departure events in the same station (Δ_d and Δ_a being the minimum departure and arrival headway, respectively). Since these constraints are nonlinear and we do not know in advance the order in which events occur at the stations, in our MIP model we introduce a set of binary variables $x_{i,j}^{h,k}$ to be set to 1 iff $t_i^h \leq t_j^k$ along with big-M coefficients M, so that conditions

$$|t_i^h - t_j^k| \geq \Delta$$

can be translated to

$$t_i^h - t_j^k \geq \Delta - M x_{i,j}^{h,k}$$

$$t_j^k - t_i^h \geq \Delta - M x_{j,i}^{k,h}$$

$$x_{i,j}^{h,k} + x_{j,i}^{k,h} = 1$$

Given the linearization of constraints (A.2)-(A.3), it is easy to translate the track capacity constraints (A.4) as

$$x_{i,j}^{h,k} = x_{i+1,j+1}^{h,k}$$

Constraints (A.5) define the profits of the trains, whereas constraints (A.6) model the user-defined time windows of each event.

It is important to notice that, although we are interested in integer values (minutes) for the events to be published in the final timetable, we do not force the integrality on variables t_j. This has the important consequence that, after fixing the event precedence variables x, the model becomes a plain linear model. On the other hand, the possible fractional value of the final time variables t need to be handled somehow in a post-processing phase to be applied before publishing the timetable. For arrival events, one can just round the corresponding fractional times to the nearest integer since there is no problem of an arrival arises a little earlier (or later) than published. An easy procedure for departure times is instead to simply round down all the t-values even if this results into a slightly infeasible published timetable, so as to guarantee that all events arise not earlier than their published time value. In a sense, this policy amounts to using an "infinite" time discretization during the optimization phase, the difference between the actual and the published event times being perceived by the travelers as a small (less than one minute) delay.

As far as the objective function is concerned, the nonlinear term

$$|t_1^h - \bar{t}_1^h|$$

giving the shift s_h of train h can be easily linearized as

$$s_h \geq t_1^h - \bar{t}_1^h$$

$$s_h \geq \bar{t}_1^h - t_1^h$$

A.3 The Stochastic Programming Paradigm

Having stated the problem as a MIP, a well known tool to find robust solutions is the (two-stage) *Stochastic Programming* approach; see [16], [101] for an introduction to the SP methodology. In SP, the set of constraints is decomposed in *structural* constraints, which represent the deterministic part of the model, and *control* constraints which have a stochastic nature and whose coefficients depend on the particular scenario. Roughly speaking, the approach allows one to take decisions in the first stage by ignoring the stochastic part of the model, but enforces some costly *recourse* action when indeterminacy will eventually occur. Thus a natural optimization objective for this two-stage strategy is to minimize the total expected cost:

$$\min_{x \in X} c^T x + \mathbb{E}[Q(x, \xi(\omega))]$$

where x denotes the first-stage decision whose feasibility set is X, $\omega \in \Omega$ denotes a *scenario* that is unknown when the first-stage decision x has to be made, and $Q(x, \xi(\omega))$ represents the optimal value of the second-stage recourse problem corresponding to first-stage decision x and parameters $\xi(\omega)$.

If Ω contains a finite number of scenarios $\{\omega_1, \omega_2, \ldots, \omega_{|\Omega|}\}$ with associated probabilities p_k, $k \in 1, 2, \ldots, |\Omega|$, then the expectation can be evaluated as a finite sum, and the two-stage model (with linear recourse) becomes a standard linear model:

$$w^* = \min_{x \in X} c^T x + \sum_{k=1}^{|\Omega|} p_k q_k^T r_k, \quad r_k \in Y_k, \forall k = 1 \ldots |\Omega| \tag{A.7}$$

where r_k are the recourse variables with linear costs q_k, and Y_k is a polyhedron depending on the first-stage variables x.

As $|\Omega|$ is often very large, various sampling-based approaches have been proposed to estimate the second-stage objective function. *Interior sampling* methods use samples during the algorithm execution to estimate, from time to time, algorithmic parameters such as function values, gradients, optimality cuts. *Exterior sampling* methods, instead, use the *Sample Average Approximation* (SAA) algorithm to estimate the optimal objective. We have chosen exterior sampling, since it has some advantages over interior sampling [109]: ease of numerical implementation, good theoretical convergence prop-

erties [117], well developed statistical inference (validation and error analysis, stopping rules). Furthermore, it is easily amenable to variance reduction techniques, ideal for parallel computations.

A.3.1 The Sample Average Approximation Method

The SAA consists in approximating the mean of the random variable $Q(x, \xi(\omega))$ with the sample mean of $\{Q(x, \xi(\omega_1)), Q(x, \xi(\omega_2)), \ldots, Q(x, \xi(\omega_N))\}$ independent and identically distributed (i.i.d.) samples from the distribution of $Q(x, \xi(\omega))$. If $Q(x, \xi(\omega))$ has finite mean and variance, the sample mean $\bar{Q}_N(x, \xi(\omega_i)) = \frac{1}{N}\sum_{i=1}^{N} Q(x, \xi(\omega_i))$ is an unbiased estimator of the actual mean:

$$\mathbb{E}[\bar{Q}_N(x, \xi(\omega_i))] = \mathbb{E}[Q(x, \xi(\omega))]$$

and it satisfies the following central limit theorem:

$$\sqrt{N}[\bar{Q}_N(x, \xi(\omega_i)) - \mathbb{E}[Q(x, \xi(\omega))]] \Rightarrow \mathcal{N}(0, \sigma^2) \text{ as } N \to \infty$$

where $\Rightarrow$ denotes convergence in distribution, $\mathcal{N}(0, \sigma^2)$ is a normal random variable with zero mean and variance $\sigma^2 = \text{var } Q(x, \xi(\omega))$.

The SAA approximation of (A.7) reads:

$$w_N^* = \min_{x \in X} c^T x + \frac{1}{N}\sum_{k=1}^{N} q_k^T r_k, \quad r_k \in Y_k, \forall k = 1 \ldots N \tag{A.8}$$

Under mild assumptions it was proved that the optimal value of SAA problem (A.8) converges with probability 1 to w^* (see [117]), that is, the optimal value of the stochastic problem, as N tends to infinity.

Also, it is possible to use SAA to estimate the optimality gap by deriving lower and upper bounds on w^*. These bounds will be used to quantify the confidence intervals of the optimal solution of the stochastic model (see Section A.6, Figure A.9). Indeed, an *upper bound* on w^* is

$$c^T \hat{x} + \mathbb{E}[Q(\hat{x}, \xi(\omega))] = c^T \hat{x} + \mathbb{E}[\bar{Q}_N(\hat{x}, \xi(\omega_i))] \tag{A.9}$$

where $\hat{x}$ is a given feasible, yet suboptimal, first-stage decision vector. The expectation in the right hand side of (A.9), by its own, can be estimated as the mean of $N' \ll N$ (say) independent SSA $\bar{Q}_N^j(\hat{x}, \xi(\omega_i^j))$, obtaining:

$$\overline{UB} = \frac{1}{N'}\sum_{j=1}^{N'} \bar{Q}_N^j(\hat{x}, \xi(\omega_i^j)) \tag{A.10}$$

$$\sigma_u^2 = \text{var } \bar{Q}_N(\hat{x}, \xi(\omega_i)) = \frac{1}{(N'-1)N'}\sum_{j=1}^{N'} (\bar{Q}_N^j(\hat{x}, \xi(\omega_i^j)) - \bar{UB}) \tag{A.11}$$

It is easy to show (see [70]) that a lower bound on w^* is given by $\mathbb{E}[w_N^*]$. Again, we can compute this expectation by sampling:

$$\overline{LB} = \frac{1}{N'} \sum_{j=1}^{N'} w_N^{*j} \tag{A.12}$$

$$\sigma_l^2 = \text{var } \bar{w}_N^* = \frac{1}{(N'-1)N'} \sum_{j=1}^{N'} (w_N^{*j} - \overline{LB}) \tag{A.13}$$

A.3.2 Sampling

Sampling of delays has been carried out using the following per-line model. A *line* $\mathcal{L}$ is defined as a sequence of stations operated by trains during the 24 hours. Each line section (the path between two consecutive stations i and j) can have a certain probability $P_{(i,j)}$ to be affected by delay. Also, each time interval $[l, k]$ in the 24-hour time horizon can have a certain probability of delay, say $P_{[l,k]}$. Then each single train h arrives at the last station with a cumulative random delay δ^h. The actual delay incurred by train h operating on section (i, j) in time interval $[l, k]$ is computed using the following formula:

$$\delta^h_{(i,j)}([l,k]) = \delta^h P_{[l,k]} \frac{P_{(i,j)}}{\sum_{(i,j)\in\mathcal{L}} P_{(i,j)}} \tag{A.14}$$

where we normalize sections delay probabilities in order to distribute the cumulative delay $\delta^T P_{[l,k]}$ incurred by train T operating on line $\mathcal{L}$ through each line section (i, j). Note that $P_{(i,j)}$ and $P_{[l,k]}$ can be deterministic numbers between 0 and 1, but typically they are treated as random variables.

When using random sampling, the outcome can be affected by a large variance, making it difficult to interpret. So we decided to use *Latin Hypercube* (LH) variance reduction technique when sampling from each distribution of $P_{(i,j)}$, $P_{[l,k]}$ and δ^h. Other techniques such as, e.g., Importance Sampling [20] can in principle fit our simulation setting as well, but are quite involved. On the contrary, LH sampling is of general applicability and comes with a straightforward implementation. The standard approach to get a sample from a particular probability distribution is to apply the inverse of the desired Cumulative Distribution Function (CDF) to a sample drawn from a uniform distribution. The process is then repeated until the required number of samples N is collected. Using LH sampling, instead, we first subdivide the $[0, 1]$ interval in N equal subintervals, and from each of them we draw a sample from a uniform distribution spanning the subinterval. Then the obtained sample vector is inverted through the CDF and randomly permuted. For more theoretical insights, the interested reader is referred to [72].

While being a very simple-minded approach, still LH sampling proved to be quite effective in our application. Figure A.1 shows the reduction in variance σ when sampling from an exponential distribution with or without LH sampling. In our computational

testing, we observed an even larger reduction (one order of magnitude or more).

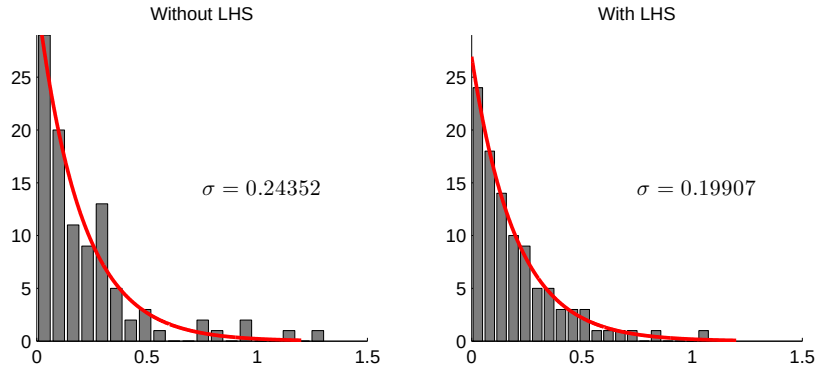


Figure A.1: Reduction of variance σ with LH when approximating through sampling, the exponential probability distribution function (solid line).

A.4 Validation Model

An important component in our framework is robustness validation. Validation is often carried out inside the model itself, as is the case when a SP approach is used. However, we decided to implement an external simulation-based validation module that is independent from the optimization model itself, so that it can be of general applicability and allows one to compare solutions coming from different methods. The module is required to simulate the reaction of the railways system to the occurrence of delays, by introducing small adjustments to the planned timetable received as an input parameter. Validation is a huge topic on its own. Indeed, the set of actions the railway system can make to react to disruptions is quite large—see for example [48]—and the decision making process is often complicated by strict real-time requirements and complex business rules. Validation can be carried out by optimization methods, these as the one proposed in [19,68,69]. However, the complexity of such models grows rapidly as soon as we allow complex decisions to be made. Thus, simulation methods are often used to measure empirically the robustness of a given timetable—see, for example, [10].

For the purpose of the present chapter, we decided to implement a simple LP-based validation tool based on the following simplified assumptions.

- Limited adjustability in response to delays with respect to the given timetable: In this chapter, timetabling robustness is *not* concerned with major disruptions (which have to be handled by the real time control system and require human intervention) but is a way to control delay propagation, i.e., a robust timetable has to favor delay compensation without heavy human action. As a consequence, at validation time no train cancellation is allowed, and event precedences are fixed with respect to the planned timetable.

- Speed of validation: The validation tool should be able to analyze quickly the behavior of the timetable under many different scenarios.

Given these guidelines, we designed a validation model which analyzes a single delay scenario ω at a time. As all precedences are fixed according to the input solution to be evaluated, constraints (A.1-A.3) all simplify to linear inequalities of the form:

$$t_i - t_j \geq d_{i,j}$$

where $d_{i,j}$ can be a minimum trip time, a minimum rest or an headway time. We will denote with $\mathcal{P}$ the set of ordered pairs (i, j) for which a constraint of type (A.1) can be written. The problem of adjusting the given timetable t under a certain delay scenario ω can thus be rephrased as the following simple linear programming model with decision variables t^ω:

$$\min \sum_{j \in E} w_j \left(t_j^\omega - t_j\right) \tag{A.15}$$

$$t_i^\omega - t_j^\omega \geq d_{i,j} + \delta_{i,j}^\omega \quad \forall (i,j) \in \mathcal{P} \tag{A.16}$$

$$t_i^\omega \geq t_i \qquad \forall i \in E \tag{A.17}$$

Constraints (A.16) correspond to the linear inequalities just explained, in which the nominal right-hand-side value $\delta_{i,j}$ is updated by adding the (possibly zero) extra-time $\delta_{i,j}^\omega$ from the current scenario ω. Weights w_j appearing in the objective function are related to the relative importance of the events, and typically depend on the number of passengers affected.

Constraints (A.17) are non-anticipatory constraints stating the obvious condition that one is not allowed to anticipate any event with respect to its published value in the timetable. Since these values are known, these constraints act as simple lower bounds on the decision variables. Instead, we impose no upper bounds since we allow an unlimited stretch of the timetable to recover from delays, i.e. a feasible timetable is always achievable.

The objective function is to minimize the "cumulative delay" on the whole network.

Given a feasible solution, the validation tool keeps testing it against a large set of scenarios, one at a time, gathering statistical information on the value of the objective function and yielding a concise figure (the average cumulative delay) of the robustness of the timetable.

A.5 Finding Robust Solutions

In this section we present three different approaches to cope with robustness. We introduced two simplifying hypotheses: (1) all input trains have to be scheduled; (2) all event precedences are fixed according to a given input "nominal" solution. These strong assumptions were made to obtain tractable (LP) models.

A.5.1 A Fat Stochastic Model

Our first attempt to solve the robust version of the TTP is to use a standard scenario-based SP formulation. The model can be outlined as:

$$\min \frac{1}{|\Omega|} \sum_{j \in E, \omega \in \Omega} \left(t_j^\omega - t_j\right)$$

$$\sum_{h \in T} \rho_h \geq (1-\alpha) z^* \tag{A.18}$$

$$t_i^\omega - t_j^\omega \geq d_{i,j} + \delta_{i,j}^\omega \quad \forall (i,j) \in \mathcal{P}, \forall \omega \in \Omega \tag{A.19}$$

$$t_i^\omega \geq t_i \quad \forall i \in E, \forall \omega \in \Omega \tag{A.20}$$

$$t_i - t_j \geq d_{i,j} \quad \forall (i,j) \in \mathcal{P} \tag{A.21}$$

$$l \leq t \leq u \tag{A.22}$$

The structure of the model is similar to that used in the validation tool, but takes into account several scenarios at the same time. Moreover, the nominal timetable values t_j are now viewed as first-stage decision variables to be optimized–their optimal value will define the final timetable to be published. The model is composed by the original one and a copy of it with a modified right hand side for each scenario; the original variables and the correspondent second-stage copies in each scenario are linked through non-anticipatory constraints.

The objective is to minimize the cumulative delay over all events and scenarios. The original objective function $\sum \rho_h$ is taken into account through constraint (A.18), where $\alpha \geq 0$ is a tradeoff parameter and z^* is the objective value of the reference solution.

For realistic instances and number of scenarios this model becomes very time consuming (if not impossible) to solve–hence we called it "fat". On the other hand, also in view of its similarity with the validation model, it plays the role of a kind of "perfect model" in terms of achieved robustness, hence it is be used for benchmark purposes.

A.5.2 A Slim Stochastic Model

Being the computing time required by the full stochastic model quite large, we present an alternative model which is simpler yet meaningful for our problem. In particular, we propose the following recourse-based formulation:

$$\min \sum_{(i,j) \in \mathcal{P}, \omega \in \Omega} w_{i,j} s_{i,j}^\omega \tag{A.23}$$

$$\sum_{h \in T} \rho_h \geq (1-\alpha) z^* \tag{A.24}$$

$$t_i - t_j + s^{\omega}_{i,j} \geq d_{i,j} + \delta^{\omega}_{i,j} \quad \forall (i,j) \in \mathcal{P}, \forall \omega \in \Omega \tag{A.25}$$

$$s^{\omega}_{i,j} \geq 0 \quad \forall (i,j) \in \mathcal{P}, \forall \omega \in \Omega \tag{A.26}$$

$$t_i - t_j \geq d_{i,j} \qquad \forall (i,j) \in \mathcal{P} \tag{A.27}$$

$$l \leq t \leq u \tag{A.28}$$

In this model we have just one copy of the original variables, plus the recourse variables $s^{\omega}_{i,j}$ which account for the unabsorbed extra times $\delta^{\omega}_{i,j}$ w.r.t. the minimum train trip times. It is worth noting that the above "slim" model is inherently smaller than the fat one. Moreover, one can drop all the constraints of type (A.25) with $\delta^{\omega}_{i,j} = 0$, a situation that occurs very frequently in practice since most extra-times in a given scenario are zero.

As to objective function, it involves a weighted sum of the recourse variables. Finding meaningful values for weights $w_{i,j}$ turns out to be very important. Indeed, we will show in Section A.6 how to define these weights so as to produce solutions whose robustness is comparable with that obtainable by solving the (much more time consuming) fat model.

A.5.3 Light Robustness

A different way to produce robust solutions is to use the *Light Robustness* (LR) approach proposed recently by Fischetti and Monaci [33]. This method is based on the consideration that, roughly speaking, robustness is about putting enough slack on the constraints of the problem. In its simpler version, the LR counterpart of the LP model

$$\min\{c^T x : \quad Ax \leq b, \quad x \geq 0\}$$

reads

$$\min f(\gamma) \tag{A.29}$$

$$Ax - \gamma \leq b - \beta \tag{A.30}$$

$$c^T x \leq (1+\alpha) z^\star \tag{A.31}$$

$$x \geq 0 \tag{A.32}$$

$$0 \leq \gamma \leq \beta \tag{A.33}$$

where β_i is a parameter giving the desired *protection level* (or slack) on constraint i, and $\gamma_i \geq 0$ is a decision variable giving the corresponding *unsatisfied slack*. The objective is to minimize a given function f of the γ variables (typically, a linear or quadratic expression). Moreover, (A.31) gives a bound (controlled by α) on the efficiency loss due to the increased robustness of the solution, where $z^\star$ is the value of the input nominal solution.

Instance	#Stations	#Sched. Trains
BZVR	27	127
BrBO	48	68
MUVR	48	48
PDBO	17	33

Table A.1: Nominal solution characteristics.

In our TTP model, a typical constraint reads

$$t_i - t_j \geq d_{i,j}$$

and its LR counterpart is simply

$$t_i - t_j + \gamma_{i,j} \geq d_{i,j} + \Delta_{i,j} \quad \gamma_{i,j} \geq 0$$

where $\Delta_{i,j}$ is the required protection level parameter.

A.6 Computational Results

We carried out tests on four single-line medium-size TTP instances provided by the Italian railway company, Trenitalia. Data refers to unidirectional traffic on different corridors.

An almost-optimal heuristic solution for each of these instances was computed by using the algorithm described in [17]. The algorithm is a Lagrangian heuristic based on the computation of paths on a time-expanded network, whose computing time was in the order of minutes on a Pentium IV, 2.4 GHz PC. The corresponding solutions were used as the input nominal solutions to freeze the event precedences and to select the trains to schedule. Solution characteristics are given in Table A.1.

We implemented our framework in C++ and carried out our tests on a AMD Athlon64 X2 4200+ computer with 4GB of RAM. ILOG CPLEX 10.1 [56] was used as MIP solver.

According to the sampling model described in Section A.3.2, we generated an extra time $\delta^h(\omega)$ corresponding to each train h and to each scenario ω, drawing them from an exponential distribution with mean $\mu = 5\%$. In lack of more detailed data from the Italian railways operator about the actual distribution of delays in line sections, we assume a proportional distribution of delays along line segments seems to be a fair assumption. Accordingly probabilities $P_{(i,j)}$ in A.14 are proportional to the length of train segments, barring a small additive white Gaussian noise (standard deviation $\sigma = 0.01$, i.e. a random adjustment of 1-2%), and probabilities $P_{[l,k]}$ are deterministically set to 1.

Given this setting, the first test we performed was aimed at comparing four different training methods for each reference solution with different values of the tradeoff

parameter α, namely 1%, 5%, 10% and 20%. We compared the following alternative methods:

- *fat*: fat stochastic model (only 50 scenarios)
- *slim1*: slim stochastic model with uniform objective function–all weights equal (400 scenarios)
- *slim2*: slim stochastic model with enhanced objective function (400 scenarios), where events arising earlier in each train sequence receive a larger weight in the objective function. More specifically, if the i-th event of train h is followed by k events, its weight in A.23 is set to $k+1$. The idea behind this weighing policy is that unabsorbed disturbances $s^{\omega}_{i,j}$ in a train sequence are likely to propagate to the next ones, so the first ones in the line are the most important to minimize.
- *LR*: Light Robustness model with objective function as in *slim2* (using the *slim1* objective function produces significantly worse results). Protection level parameters are set to $\Delta = -\mu \ln \frac{1}{2}$, where μ is the mean of the exponential distribution. This is the protection level required to absorb a delay drawn from such a distribution with probability $\frac{1}{2}$. For example, setting a buffer of 1 minute we can absorb half of the times an exponentially distributed disturbance of mean 1.44 minutes.

As to the validation model, weights w_j appearing in objective function (A.15) are assumed to be equal to 1, i.e., all events are considered equally important. It turn out that the new objective function tipically produces slightly worse results for *LR* while *slim2* takes advantage of it for large values of λ. In any case, the improvement is not substantial (up to 3-4%).

The results are shown in Table A.2 and graphical representations are given in Figures A.2 and A.3.

According to the figures, *slim2* always yields a very tight approximation of *fat*, while *slim1* is often poorer. As to *LR*, it usually produces good results that are only slightly worse than *slim2*, mainly in the most-realistic cases where the tradeoff parameter α is small. As to computing times (reported in Table A.2), the *fat* model is one order of magnitude slower than *slim1* and *slim2*, although it uses only 50 scenarios instead of 400. *LR* is much faster than any other method—more than two orders of magnitude w.r.t the fast stochastic models. Therefore, *LR* qualifies as the method of choice for addressing very large real cases, as it guarantees good levels of robustness within very short computing times.

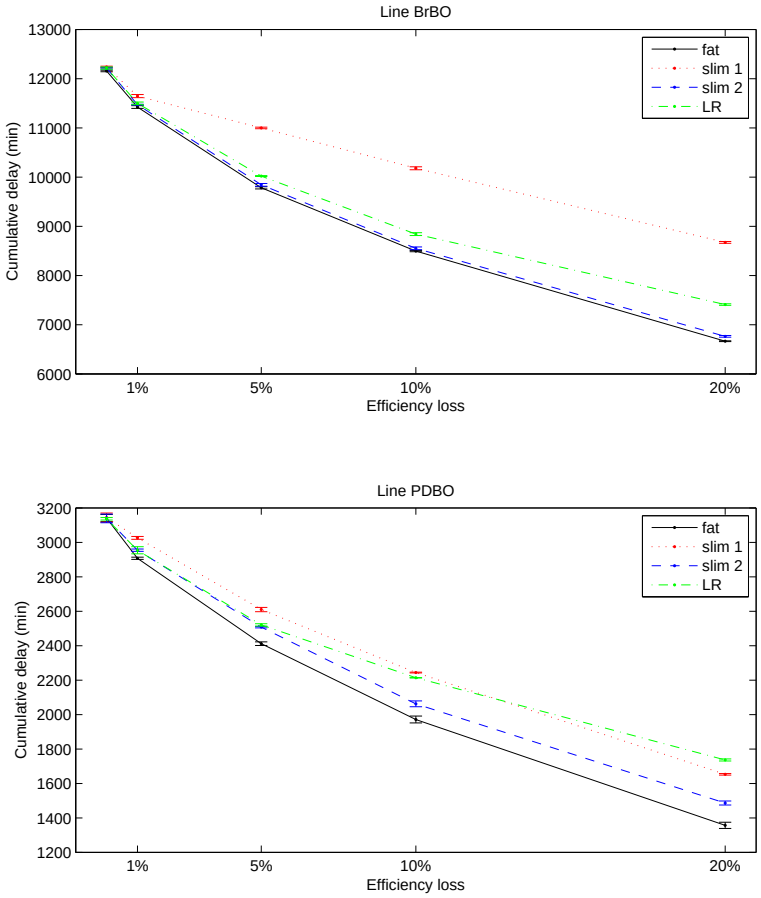


Figure A.2: Comparison of different training models applied to the best reference solution for each instance. On the x-axis there is the efficiency loss (α) while the y-axis reproduces the confidence intervals of the validation figure (run with 500 scenarios).

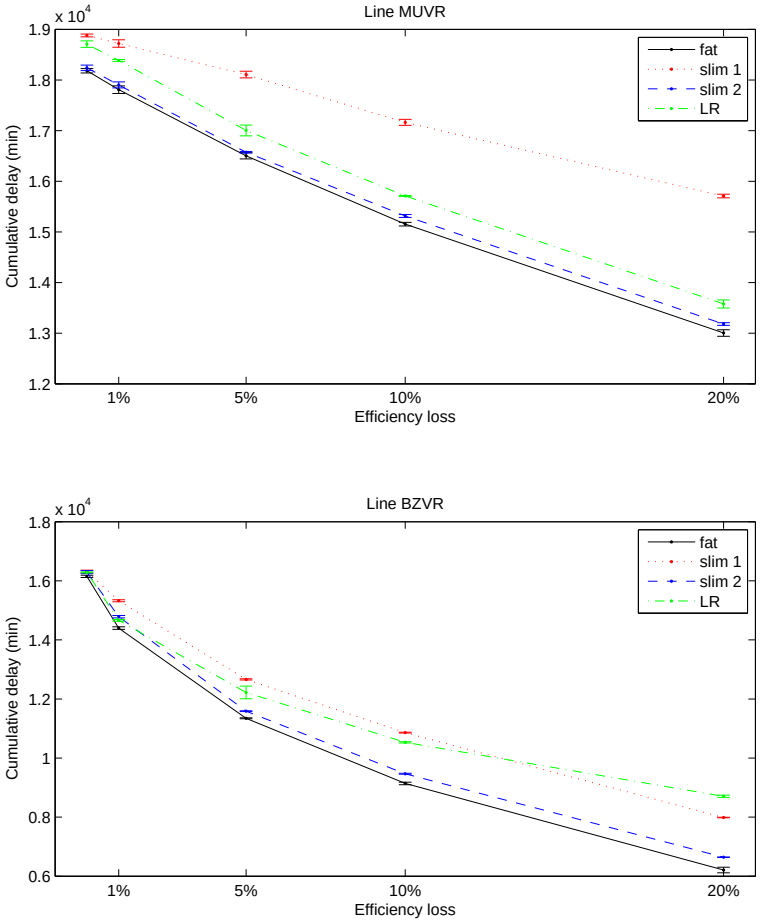


Figure A.3: Comparison of different training models applied to the best reference solution for each instance. On the x-axis there is the efficiency loss (α) while the y-axis reproduces the confidence intervals of the validation figure (run with 500 scenarios).

α		Fat			Slim1			Slim2			LR		
	Line	Delay	WAD(%)	Time(s)	Delay	WAD(%)	Time(s)	Delay	WAD(%)	Time(s)	Delay	WAD(%)	Time(s)
0%	BZVR	16149	–	9667	16316	–	532	16294	–	994	16286	–	2.27
0%	BrBO	12156	–	384	12238	–	128	12214	–	173	12216	–	0.49
0%	MUVR	18182	–	377	18879	–	88	18240	–	117	18707	–	0.43
0%	PDBO	3141	–	257	3144	–	52	3139	–	63	3137	–	0.25
	Tot:	49628	–	10685	50577	–	800	49887	–	1347	50346	–	3.44
1%	BZVR	14399	16.4	10265	15325	45	549	14787	17	1087	14662	18	2.13
1%	BrBO	11423	21.6	351	11646	42	134	11472	21	156	11499	23	0.48
1%	MUVR	17808	12.9	391	18721	37	96	17903	12	120	18386	8	0.48
1%	PDBO	2907	15.6	250	3026	51	57	2954	11	60	2954	13	0.27
	Tot:	46537	66.5	11257	48718	175	836	47116	61	1423	47501	62	3.36
5%	BZVR	11345	15.9	9003	12663	48	601	11588	19	982	12220	22	1.99
5%	BrBO	9782	18.9	357	11000	50	146	9842	22	164	10021	23	0.51
5%	MUVR	16502	14.5	385	18106	41	86	16574	13	107	17003	11	0.45
5%	PDBO	2412	14.7	223	2610	44	49	2508	20	57	2521	19	0.28
	Tot:	40041	64	9968	44379	183	882	40512	74	1310	41765	75	3.23
10%	BZVR	9142	21.4	9650	10862	50	596	9469	24	979	10532	33	2.01
10%	BrBO	8496	19.1	387	10179	51	132	8552	20	157	8842	23	0.51
10%	MUVR	15153	14.7	343	17163	49	84	15315	15	114	15710	13	0.43
10%	PDBO	1971	19.9	229	2244	49	50	2062	27	55	2314	37	0.25
	Tot:	34762	75.1	10609	40448	199	862	35398	86	1305	37398	106	3.2
20%	BZVR	6210	28.5	9072	7986	50	538	6643	31	1019	8707	52	2.04
20%	BrBO	6664	22.1	375	8672	53	127	6763	23	153	7410	30	0.52
20%	MUVR	13004	17.1	384	15708	52	91	13180	18	116	13576	19	0.42
20%	PDBO	1357	28.4	230	1653	49	55	1486	34	60	1736	53	0.28
	Tot:	27235	96.1	10061	34019	204	811	28072	106	1348	31429	154	3.26
40%	BZVR	3389	35.4	10486	4707	50	578	3931	37	998	5241	51	2.31
40%	BrBO	4491	27.7	410	6212	52	130	4544	29	166	6221	52	0.53
40%	MUVR	10289	21.8	376	13613	52	95	10592	25	108	11479	34	0.45
40%	PDBO	676	37.1	262	879	49	55	776	41	57	1010	52	0.28
	Tot:	18845	122	11534	25411	203	858	19843	132	1329	23951	189	3.57

Table A.2: Comparison of different training methods w.r.t. computing time, percentage WAD and validation function (cumulative delay in minutes), for different lines and tradeoff α.

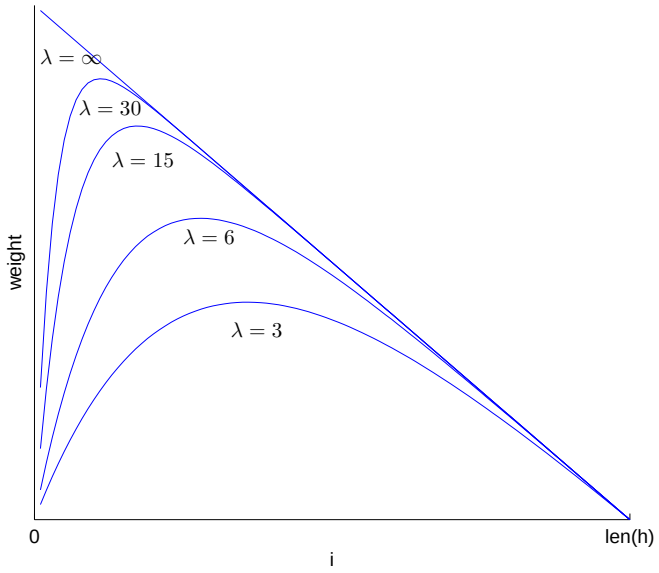


Figure A.4: Alternative weighing functions for slim2 *and* LR*, giving weight w_{ij} as a function of position i in the line.*

We also tried a variation of the *slim2* and *LR* objective function. The variation is motivated by observations in [64] about the optimal distribution of buffers on a single corridor. There, it was observed that buffers that are placed too early risk to be left unused, because the probability to face any delay at this early position is too small. As a consequence, it is worthwhile to lower the weights $w_{i,j}$ arising in the early sections of the line. Figure A.4 plots different parametric variants of *slim2* objective functions. All of the them obey a common formula, namely:

$$(1 - e^{-\lambda i})(len(h) - i)$$

parametrized in λ ($\lambda = \infty$ is the original *slim2* weighing scheme). Table A.3 reports the percentual improvements with respect to case $\lambda = \infty$, *slim2* and *LR*, respectively. It turns out that there is only a small improvement for large λ in *slim2*.

One might also wonder what is the effect of the input nominal solution to the subsequent robustness improving phase. To answer this question we do the following experiment. We take the scheduled trains in the heuristic nominal solutions of [17] formerly used, and we construct a MIP model, as described in Section A.2, where the choice of precedences is left open. Then we collect a series of heuristic nominal solutions of increasing efficiency for that model by running the MIP solver with a 5-minute time limit and storing all the incumbent solutions produced during the run. Moreover we let the solver run with a 1-hour time limit to produce an almost optimal solution z_{ref}. Indeed, all instances terminated with a remaining optimality gap less than 4%.

Then, we compare the robustness achieved by our *fat* model when starting from these solutions and allowing for a relative efficiency loss α with respect to z_{ref}. The

Slim2

λ	BZVR				BrBO				MUVR				PDBO			
$\alpha =$	0.01	0.05	0.10	0.20	0.01	0.05	0.10	0.20	0.01	0.05	0.10	0.20	0.01	0.05	0.10	0.20
3	-1.9	2.6	1.7	3.5	0	-2.5	-2.3	-0.1	-1.2	-0.8	-1.2	-5.3	-2.3	0.2	-0.1	-1.7
6	-0.7	2.6	1.6	3	0.4	-1.3	0.6	2.8	-0.8	-0.9	0.2	-1.5	-1.8	0.2	1	3.6
15	0	3.7	1.7	4.9	1.1	1.2	3.4	3.8	-0.6	-0.3	0	1	-0.3	0.7	1.8	1.4
30	0.4	3.6	0.1	3.5	0.8	1.8	2.2	3.7	0	0.2	0.3	0.7	0.3	-0.2	1.1	1.8

LR

λ	BZVR				BrBO				MUVR				PDBO			
$\alpha =$	0.01	0.05	0.10	0.20	0.01	0.05	0.10	0.20	0.01	0.05	0.10	0.20	0.01	0.05	0.10	0.20
3	-0.3	-0.2	1.1	-2.2	-0.1	0.2	-0.5	-0.5	0.1	-0.4	1.2	0.2	-0.7	-0.7	-2.2	0.2
6	-0.2	-0.8	2	-2	0.1	0.6	0.5	-0.7	0.4	0.2	-0.1	1.3	-1.5	-0.3	0.2	-1.7
15	-0.3	-0.5	1.7	-1.1	0.1	0.3	0.4	-0.5	-0.2	0.7	1.3	0.6	-1.3	0	-0.7	1.3
30	0.1	-0.1	1.2	-0.7	0.3	0.2	0	-0.7	-0.4	0.4	0.9	-0.8	-0.4	-0.8	-0.4	-0.9

Table A.3: Percentage robustness improvement with respect to $\lambda = \infty$ for the different weighing functions plotted in Figure A.4; a negative value corresponds to a worse robustness.

left part of Table A.4 gives the outcome of the experiment, for two instances (BrBO and MUVR). First columns correspond to the 10 best solutions obtained with the 5-minute time limit, sorted by increasing efficiency from left to right. E.g., for instance BrBO, the 10 solutions have a loss of efficiency ranging from 5.5% to 0.4% with respect to z_{ref}. Rows correspond to the different thresholds α used (1%,5%,10% and 20%). The table entries then give the percentage increase in robustness (as measured by the validation tool) with respect to robustness measured when starting from the almost optimal solution z_{ref}. E.g., for BrBO, if we allow a 10% efficiency loss w.r.t. z_{ref} and start from a nominal solution which is already 4.5% worse, we loose 13.9% in terms of robustness achieved through *fat*.

As expected, starting from a worse nominal solution reduces the degrees of freedom in the subsequent training phase, leading to robustness loss (in some cases the robust model become even infeasible, as for missing entries in Table A.4). This negative effect could be counterbalanced by the different precedence structure of the solution. Indeed, each solution has a difference precedence order of trains which could in principle be more suited to deal with disturbances, thus allowing for a more robust timetable. However, in the aforementioned experiment the precedence structure of the solutions seems to play only a secondary role.

To better quantify the role of precedences for robustness in our instances we do a second experiment, namely, we solve a MIP version of the *LR* model where the precedences are left unfixed. Note that this is only viable for *LR*, since the other models are too large to be solved by any state-of-the-art MIP solver. Again we train the model using a loss of efficiency α ranging from 1 to 20% from the same almost optimal solution z_{ref} used in the precedence example. z_{ref} is also used to provide a first incumbent solution to the MIP solver. Since the solver cannot improve robustness by finding a more efficient solution for which the tradeoff constraint would becomes looser,

the robustness improvement, if any, will come entirely from the more robust precedence structure of the solution. Results are reported in the right part of Table A.4.

The remaining integrality gap after 1 hour of computing time was less than 1% for all instances. For PDBO and BZVR the MIP model did not find any more solution than the incumbent, so they are not reported in the table.

The last column of Table A.4 reports the percentage robustness improvement of the MIP *LR* model described above, over the linear *LR* model described in Section A.5.3, both applied to the almost optimal solution z_{ref}. E.g. in Br-BO, for a threshold α of 10% the MIP version of *LR* is able to improve by 4.3% over the simple linear *LR*.

Furthermore, the second-last column of Table A.4 reports, for comparison sake, the percentage difference between the solution robustness obtained by the MIP *LR* and the robustness obtained by using *fat* on the same almost optimal solution z_{ref}.

Results confirm that in our testbed the combinatorial structure of solutions given by precedences is not as important, as their efficiency is, in determining the maximum achievable robustness. However, we expect that for more complex network topologies the combinatorial contribution will become overwhelming.

BrBO

	Fat											LR-MIP	
α												vs Fat	vs LR
eff(%)=	-5.5	-4.5	-3.9	-2.7	-2.2	-1.7	-1.3	-1.2	-0.8	-0.4	0.0	0.0	0.0
1%	–	–	–	–	–	–	–	–	-4.1	-2.7	0.0	-0.4	-0.1
5%	–	-20.3	-18.2	-8.1	-7.4	-4.4	-2.8	-3.1	-1.6	-2.2	0.0	1.7	4.1
10%	-23.9	-13.9	-15.2	-5.2	-5.6	-2.9	-1.9	-2.8	-1.4	-2.6	0.0	-1.0	4.3
20%	-22.2	-11.9	-14.9	-4.6	-4.5	-3.1	-2.1	-2.8	-2.4	-3.0	0.0	-11.8	2.7

MUVR

	Fat											LR-MIP	
α												vs Fat	vs LR
eff(%)=	-27.4	-14.9	-9.9	-9.2	-7.6	-6.8	-2.7	-1.6	-1.6	-1.3	0.0	0.0	0.0
1%	–	–	–	–	–	–	–	–	–	–	0.0	-0.6	0.0
5%	–	–	–	–	–	–	-3.7	-1.7	-1.2	-1.7	0.0	-1.2	-0.1
10%	–	–	-19.2	-16.5	-12.6	-10.1	-1.6	-0.8	-0.3	0.2	0.0	-1.4	1.1
20%	–	-25.5	-13.1	-12.7	-9.1	-8.6	-2.1	-0.9	0.1	-0.8	0.0	-4.3	1.4

Table A.4: Effects of nominal input solution on robustness.

A simple yet often used in practice policy to enforce robustness in a timetable is to allocate a buffer that is just proportional to the train duration. Figure A.5 gives the results of this simple policy on a sample instance, where we first compute the maximum total amount of buffer we can allocate for a given efficiency loss, and then distribute it proportionally along the line. According to the figure, the proportional buffer allocation policy and *slim1* behave quite similarly. This is not surprising, since model *slim1* actually favors a proportional buffer allocation — is confirmed in the other instances as well (not shown in the figure). On the other hand, much better results are obtained by applying more clever optimization methods, showing the practical relevance

of the optimization approaches.

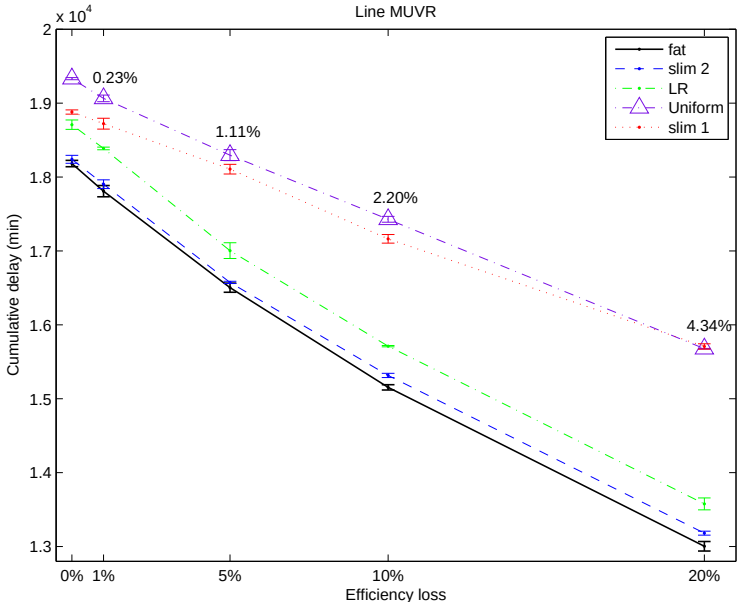


Figure A.5: Comparison of a simple "proportional" buffer allocation strategy against the proposed methods. The percentages shown are the total amount of buffer it was possible to allocate within a given tradeoff.

While the validation output gives a reliable measure of how robust a solution is against delays, other figures exist that summarize somehow the "static" structure of a solution. These figures are useful to get insights into the structure of the solutions obtained with different training methods. In particular, we used the weighted average distance (WAD) of the allocated buffer from the starting point. The WAD of the single train h is calculated as

$$WAD_h = \frac{1}{\sum_{i=1}^{len(h)-1} s_{i,i+1}} \sum_{i=1}^{len(h)-1} \frac{s_{i,i+1}(t^h_{i+1} + t^h_i)/2}{t^h_{len(h)} - t^h_1} \tag{A.34}$$

where $s_{i,i+1}$ is the amount of buffer allocated from t_i to t_{i+1}. The WAD is a number between 0 and 1 which measures how the buffers are distributed along the train trip. For example, a value of 0.5 means that the same amount of buffers were allocated in the first half and in the second half of the trip; values smaller or bigger than 0.5 relate to a shift in buffers distribution towards the begin or the end of the trip, respectively. The WAD of an entire line is calculated as the mean of all the WADs of the trains of the line. The reader is referred to [64] for a more detailed discussion.

A comparison of the various WADs is reported in Table A.2 and illustrated in Figures A.6 and A.7. It can be seen that there is a significative correlation between the degree of approximation of the various WADs with respect to "perfect WAD" (WAD_{fat}) and the robustness of the solution–as computed by the validation tool and reported in Figure A.2 and A.3.

In Figures A.6 and A.7 appears that *slim1* WAD is almost fixed to 50%, meaning a uniform allocation of buffers. On the other hand, the other methods tend to allocate buffers earlier in the line, resulting in a lower value of the WAD. Moreover, as the allowed efficiency loss increases (x axis), the WAD increases as well, meaning that the uniform allocation is more likely to be a good choice. We can also note that *LR* behaves better for small efficiency losses. Indeed, *LR* uses a fixed buffer β to deal with disturbances. When the problem is less constrained in efficiency, these buffers can become too small, and the LP solver will start to distribute, in a somehow unpredictable way, the excess buffers to meet the increased degree of freedom, thus degrading the performance of the method. E.g., this is the case of lines BZVR and PDBO. Moreover BZVR and PDBO are more congested than other two instances, which also explains the better performance of the uniform allocation strategy.

Figure A.8 reports how the buffers are distributed along the line. The figure is obtained by normalizing each line by the length of the corridor, and averaging the buffers allocated in each normalized line section. The averages are then normalized by the total amount of allocated buffer, so that the area of each chart approximately sum up to 1. E.g., *slim1* allocates buffers almost uniformly along the line—the particular structure of the timetable being responsible of local fluctuations. It is clear that *slim2* produces a very tight approximation of *fat*, while *slim1* does not. It is worth noting that *LR* uses a smoother allocation of buffers, while *slim1* yields a better approximation of their oscillations, but misses the global allocation policy. In this respect, *slim2* performs quite well instead. This is due to the fact that *LR* does not exploit directly the scenario information, thus it has to cope with very little information. Again, note that the poorest method (*slim1*) produces an almost uniform distribution of the buffers, whereas the best ones tend to allocate them earlier. This confirms the findings reported in [64].

Finally, given the intrinsic approximation of the stochastic methods due to the evaluation of the expectation, we have computed lower and upper bounds on the optimal solutions of the stochastic models, as described in Section A.3. A typical plot obtained for the slim stochastic model is reported in Figure A.9, showing very narrow estimation gaps. Similar results are obtained with the other models, except *fat* that behaves a little worse due the reduced number of scenarios.

A.7 Conclusions and future work

In this chapter we have described a three-stage framework as a practical tool for building and testing robust solutions for the Train Timetabling Problem. We mainly focused on robustness improvement of a given nominal solution. Robustness was then validated in terms of the total cumulative delay, computed by solving an LP model.

We examined three different robustness improving models. The best performing, in terms of validated cumulative delay, is a huge stochastic reformulation of the nominal TTP problem. However, the solution of this model turned out to be very slow (if not impossible) for practical instances. An approximation of it, called “slim”, performed

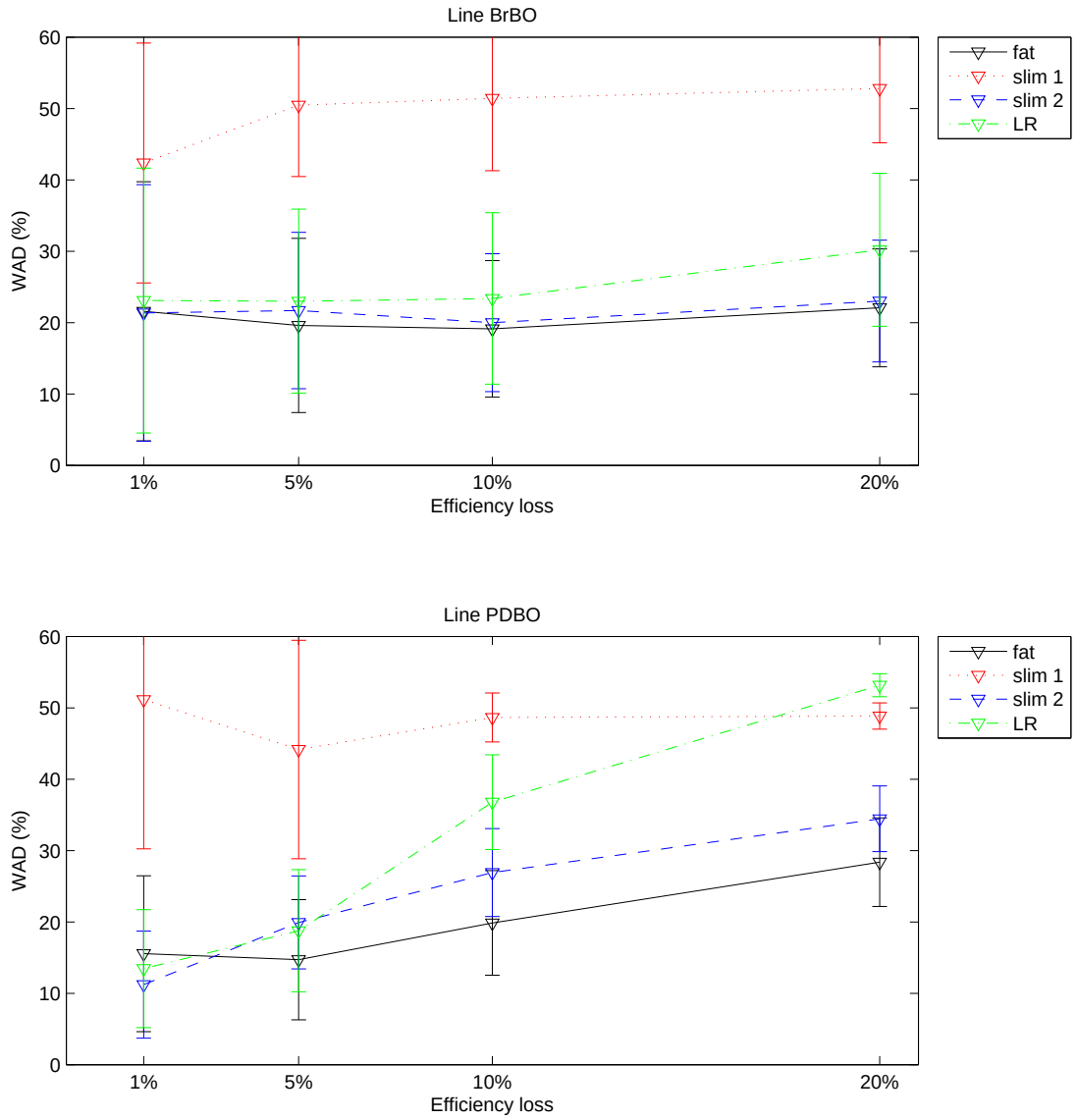


Figure A.6: Comparison of different training models from the WAD point of view (WAD is given within its confidence intervals).

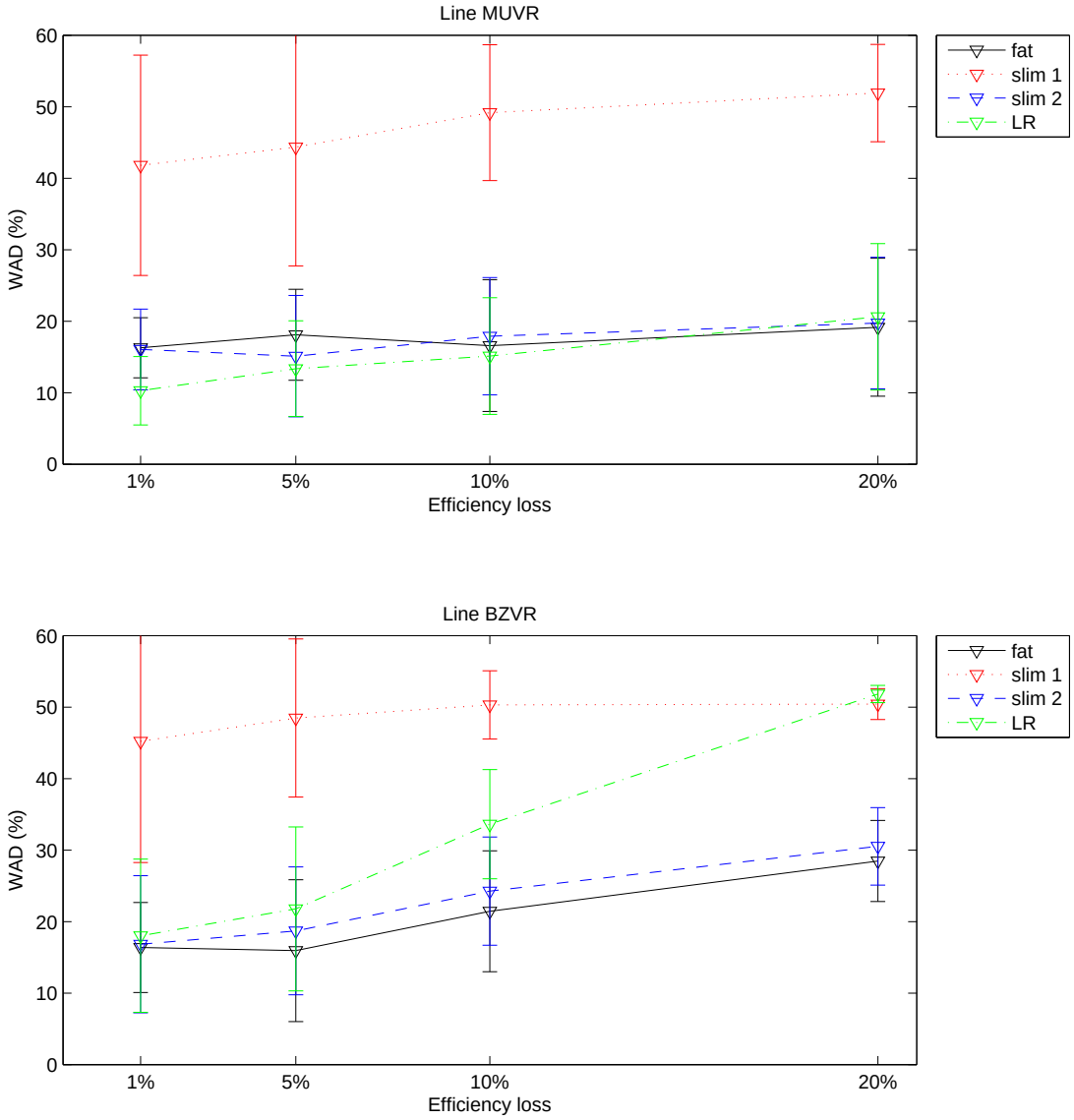


Figure A.7: Comparison of different training models from the WAD point of view (WAD is given within its confidence intervals).

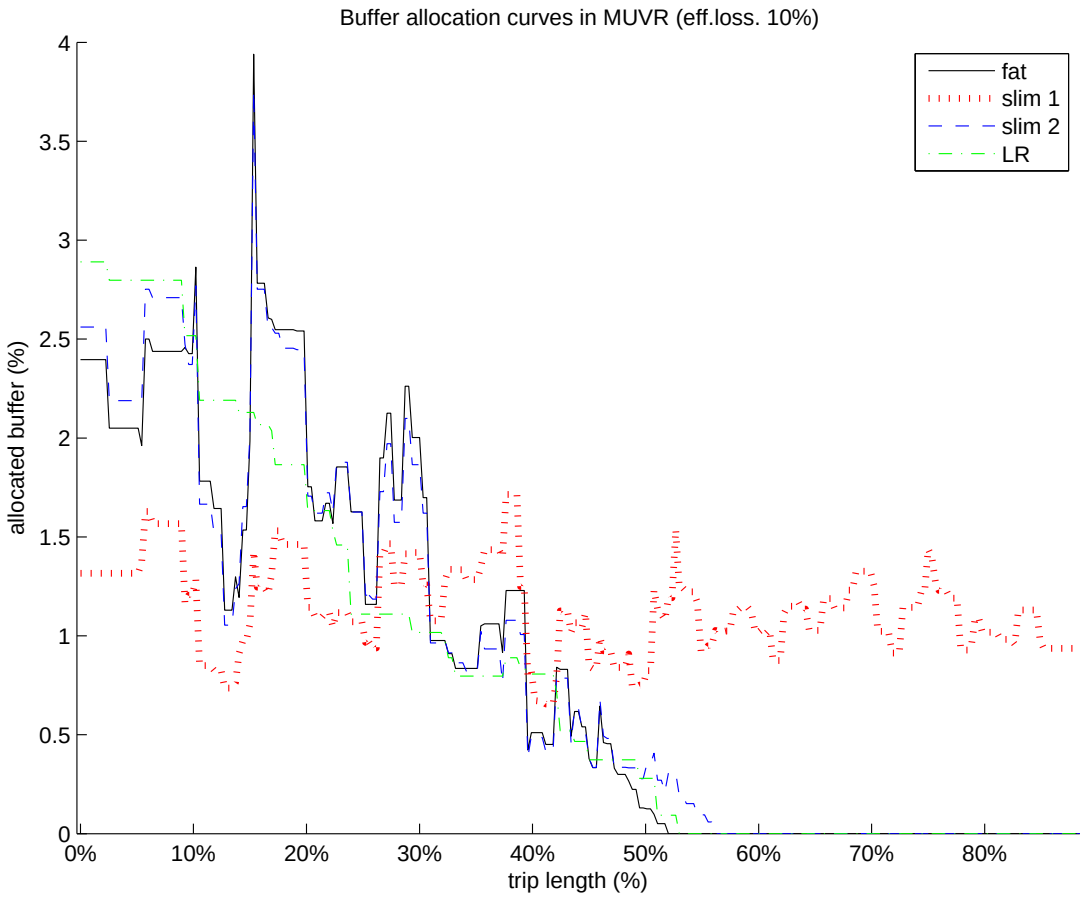


Figure A.8: Comparison of different training models from the allocated-buffer point of view.

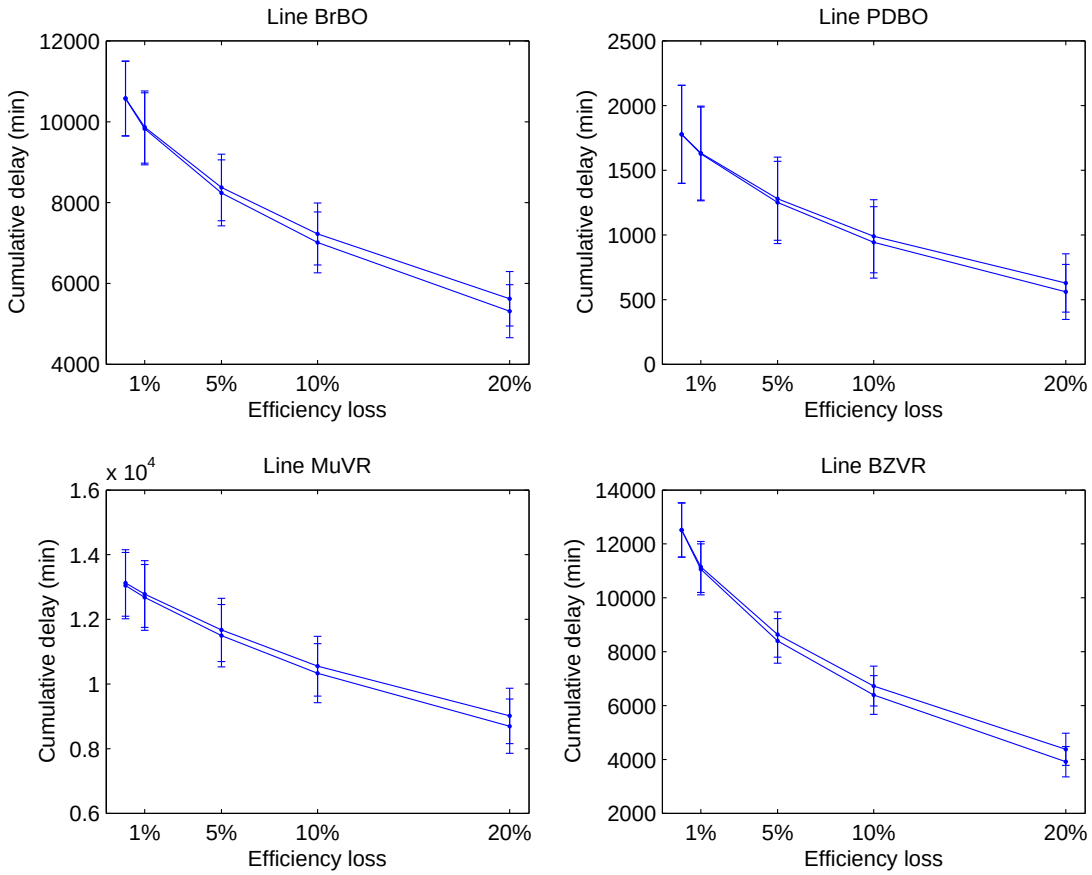


Figure A.9: Confidence intervals of upper and lower bounds of the optimal solution of stochastic model slim2

much better provided that the "right" objective function is given. The fastest method, Light Robustness (LR), proved to be quite accurate when seeking for a reasonable robustness–efficiency tradeoff, allowing for fast solution of large instances. Light Robustness is therefore a suitable tool for addressing large-scale real scenarios, and can be embedded in the nominal solver to efficiently provide alternative, robust-improved timetables.

We performed our computations on real world unidirectional corridors operated by the Italian railways operator. Further work should address more complex network topologies.

In our study we used a LP-based validation tool to estimate the cumulative delay in a set of random scenarios: however, it would be interesting to measure the actual price required to recover a delayed timetable by using the same strategies used in real-world delay management.

Moreover, we have quantified for the *LR* model the gain in terms of robustness resulting from relaxing the requirement that all precedences in the nominal solution must be preserved. It would be interesting to extend this analysis also to the *slim2* model that, at the time of writing, seems intractable even for medium–size instances.

Acknowledgments

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 (PRIN 2006 project "Models and algorithms for robust network optimization"). We thank Paolo Toth, Alberto Caprara and Valentina Cacchiani for providing us with the nominal TTP solutions used in our computational testing. Thanks are also due to three anonymous referees for their constructive comments.

Bibliography

[1] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34:361–372, 2006. Problems available at *http://miplib.zib.de*.

[2] Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 1:4–20, 2007.

[3] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

[4] Tobias Achterberg and Timo Berthold. Improving the feasibility pump. *Discrete Optimization*, 4:77–86, 2007.

[5] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.

[6] E. Amaldi, M. E. Pfetsch, and L.E. Trotter Jr. On the maximum feasible subsystem problem, IISs and IIS-hypergraphs. *Mathematical Programming*, 95(3):533–554, 2003.

[7] D. Applegate, R. Bixby, V. Chvatal, and B. Cook. Finding cuts in the TSP. Technical Report 95-05, DIMACS, 1995.

[8] Alper Atamturk and Deepak Rajan. On splittable and unsplittable flow capacitated network design arc-set polyhedra. *Mathematical Programming*, 92:315–333, 2002.

[9] E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0–1 programming by lift-and-project in a branch-and-cut frameworkbranch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996.

[10] F. Barber, S. Cicerone, D. Delling, G. Di Stefano, M. Fischetti, L. Kroon, D. Salvagnin, P. Tormos, C. Weber, and A. Zanette. New frameworks for the interaction between robust and online timetable planning, and for monitoring the status quo of the system. Technical Report ARRIVAL-D3.4, ARRIVAL Project, 2008.

[11] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, December 1962.

[12] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and D. Vincent. Experiments in mixed integer linear programming. *Mathematical Programming*, 1:76–94, 1971.

[13] Livio Bertacco, Matteo Fischetti, and Andrea Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4:63–76, 2007.

[14] Timo Berthold. Primal Heuristics for Mixed Integer Programs. Master's thesis, Technische Universität Berlin, 2006.

[15] Armin Biere and Wolfgang Kunz. SAT & ATPG: Boolean engines for new generation synthesis and verification moderators. In *ICCAD*, pages 782–790, 2002.

[16] John R. Birge and Francois Louveaux. *Introduction to Stochastic Programming*. Springer, 2000.

[17] A. Caprara, M. Fischetti, and P. Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2002.

[18] A. Caprara, M. Monaci, P. Toth, and P.L. Guida. A Lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete Applied Mathematics*, 154(5):738–753, 2006.

[19] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. On the interaction between robust timetable planning and delay management. Technical Report ARRIVAL-TR-0116, ARRIVAL project, 2007.

[20] Charles E. Clark. Importance sampling in Monte Carlo analyses. *Operations Research*, 9(5):603–620, 1961.

[21] G. Codato and M. Fischetti. Combinatorial Benders' cuts. In *IPCO 2005 Proceedings*, pages 178–195, 2005.

[22] A. Colmerauer. An introduction to Prolog III. draft, 1987.

[23] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158. ACM, 1971.

[24] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *Computer Journal*, 8:250–255, 1965.

[25] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[26] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.

[27] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[28] Mehmet Dincbas, Pascal Van Hentenryck, Helmut Simonis, Abderrahmane Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *FGCS*, pages 693–702, Tokyo, Japan, 1988.

[29] Niklas Eén and Niklas Sörensson. An extensible SAT-solver, October 14 2003.

[30] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1–3):23–47, 2003.

[31] M. Fischetti, A. Lodi, and D. Salvagnin. Just MIP it! Technical report, University of Padova, October 2008.

[32] M. Fischetti, A. Lodi, and A. Tramontani. Experiments with disjunctive cuts. Technical report, November 2007. In preparation.

[33] M. Fischetti and M. Monaci. Light robustness. Technical Report ARRIVAL-TR-0119, ARRIVAL Project, 2008.

[34] Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[35] Matteo Fischetti and Andrea Lodi. Optimizing over the first Chvàtal closure. *Mathematical Programming*, 110(1):3–20, 2007.

[36] Matteo Fischetti and Paolo Toth. A new dominance procedure for combinatorial optimization problems. *Operations Research Letters*, 7:181–187, 1988.

[37] Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In *Principles and Practice of Constraint Programming*, pages 189–203. Springer-Verlag, 1999.

[38] Ricardo Fukasawa and Marcos Goycoolea. On the exact separation of mixed integer knapsack cuts. In *Integer Programming and Combinatorial Optimization*, volume 4513, pages 225–239. Springer, 2007.

[39] Hervé Gallaire. Logic programming: Further developments. In *SLP*, pages 88–96, 1985.

[40] Gecode Team. Gecode: Generic constraint development environment, 2008. Available from `http://www.gecode.org`.

[41] Arthur M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[42] Matthew L. Ginsberg, James M. Crawford, and David W. Etherington. Dynamic backtracking. Technical report, 1996.

[43] J. Gleeson and J. Ryan. Identifying minimally infeasible subsystems of inequalities. *ORSA Journal on Computing*, 2(1):61–63, 1990.

[44] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.

[45] J. Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, 9:73–91, 1997.

[46] Jack E. Graver. On the foundations of linear and integer linear programming i. *Mathematical Programming*, 9(1):207–226, 1975.

[47] William D. Harvey. *Nonsystematic backtracking search*. PhD thesis, Stanford University, 1995.

[48] Mads Hofman, Line Madsen, Julie Jespersen Groth, Jens Clausen, and Jesper Larsen. Robustness and recovery in train scheduling - a case study from DSB S-tog a/s. In *ATMOS 2006*, 2006.

[49] J. N. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley, 2000.

[50] J. N. Hooker. A hybrid method for planning and scheduling. *Constraints*, 10(4):385–401, 2005.

[51] J. N. Hooker. *Integrated Methods for Optimization*. Springer, 2006.

[52] J. N. Hooker and Hong Yan. Logic circuit verification by Benders' decomposition. In *Principles and Practice of Constraint Programming*, pages 267–288, 1995.

[53] John N. Hooker. An integrated method for planning and scheduling to minimize tardiness. *Constraints*, 11:139–157, 2006.

[54] John N. Hooker, Hong Yan, Ignacio E. Grossmann, and R. Raman. Logic cuts for processing networks with fixed charges. *Computers & OR*, 21(3), 1994.

[55] Toshihide Ibaraki. The power of dominance relations in branch-and-bound algorithms. *Journal of the ACM*, 24(2):264–279, 1977.

[56] ILOG Inc. *ILOG CPLEX 10.1 User's Manual*, 2007.

[57] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *POPL*, pages 111–119, 1987.

[58] Roberto J. Bayardo Jr. and Daniel P. Miranker. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *AAAI/IAAI*, pages 298–304, 1996.

[59] George Katsirelos and Fahiem Bacchus. Generalized nogoods in CSPs. In *AAAI*, pages 390–396. MIT Press, 2005.

[60] L. G. Khachian. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

[61] H. J. Kim and J. N. Hooker. Solving fixed-charge network flow problems with a hybrid optimization and constraint programming approach. *Annals of Operations Research*, 115:95–124, 2002.

[62] Anton J. Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

[63] W. H. Kohler and K. Steiglitz. Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. *Journal of the ACM*, 21(1):140–156, 1974.

[64] L.G. Kroon, R. Dekker, and M.J.C.M. Vromans. Cyclic railway timetabling: a stochastic optimization approach. In *Algorithmic Methods for Railway Optimization*, Lecture Notes in Computer Science, pages 41–66. 2007.

[65] Mikael Z. Lagerkvist and Christian Schulte. Advisors for incremental propagation. In *Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 409–422. Springer-Verlag, 2007.

[66] C. Liebchen and L. W.P. Peeters. On cyclic timetabling and cycles in graphs. Technical Report 761-2002, TU Berlin, Mathematical Institute, 2002.

[67] C. Liebchen and S.Stiller. Delay resistant timetabling. Technical Report ARRIVAL-TR-0056, ARRIVAL Project, 2006.

[68] Christian Liebchen, Marco Lübbecke, Rolf H. Möhring, and Sebastian Stiller. Recoverable robustness. Technical Report ARRIVAL-TR-0066, ARRIVAL-Project, 2007.

[69] Christian Liebchen, Michael Schachtebeck, Anita Schöbel, Sebastian Stiller, and André Prigge. Computing delay resistant railway timetables. Technical Report ARRIVAL-TR-0071, ARRIVAL Project, October 2007.

[70] Jeff Linderoth, Alexander Shapiro, and Stephen Wright. The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research*, 142(1):215–241, February 2006.

[71] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11:972–989, 1963.

[72] W. L. Loh. On latin hypercube sampling. *The Annals of Statistics*, 24(5), 1996.

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

[74] T.L. Magnanti and R.T. Wong. Accelerating Benders decomposition: algorithmic enhancement and model selection criteria. *Operations Research*, 29:464–484, 1981.

[75] W. K. Mak, D. P. Morton, and R. K. Wood. Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24(24):10, February 1999.

[76] H. Marchand and L. A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001.

[77] François Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94(1):71–90, 2002.

[78] François Margot. Exploiting orbits in symmetric ILP. *Mathematical Programming*, 98(1):3–21, 2003.

[79] István Maros. *Computational Techniques of the Simplex Method*. Kluwer Academic Publishers, 2002.

[80] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.

[81] M. Milano. *Constraint and Integer Programming: Toward a Unified Methodology*. Kluwer Academic Publishers, 2003.

[82] P. Miliotis. Integer programming approaches to the travelling salesman problem. *Mathematical Programming*, 10:367–378, 1976.

[83] P. Miliotis. Using cutting planes to solve the symmetric travelling salesman problem. *Mathematical Programming*, 15:177–178, 1978.

[84] H. D. Mittelmann. Benchmarks for optimization software: Testcases. Problems available at *http://plato.asu.edu/sub/testcases.html*.

[85] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, pages 530–535, 2001.

[86] K. Nachtigall. Periodic network optimization and fixed interval timetables. Habilitation Thesis, Deutsches Zentrum für Luft-und Raumfahrt, Braunschweig, 1999.

[87] G. Nemhauser and L. A. Wolsey. A recursive procedure to generate all cuts for 0-1 mixed integer programs. *Mathematical Programming*, 46:379–390, 1990.

[88] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6:1–8, 1987.

[89] M. W. Padberg. Covering, packing and knapsack problems. *Annals of Discrete Mathematics*, 4:265–287, 1979.

[90] M. W. Padberg and Rinaldi G. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33, 1991.

[91] M. W. Padberg, T. J. Van Roy, and L. A. Wolsey. Valid Linear Inequalities for Fixed Charge Problems. *Operations Research*, 33(4):842–861, 1985.

[92] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1982.

[93] L. W. P. Peeters. *Cyclic Railway Timetable Optimization*. PhD thesis, Erasmus University Rotterdam, 2003.

[94] David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32:2271–2284, 2005.

[95] Chandra Poojari and John Beasley. Improving benders decomposition using a genetic algorithm. Technical report, Centre for the Analysis of Risk and Optimisation Modelling Applications (CARISMA), School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, 2007.

[96] Jean-Francois Puget. A C++ implementation of CLP. In *Proceedings of the Second Singapore International Conference on Intelligent Systems*, Singapore, 1994.

[97] W. V. O. Quine. The problem of simplifying truth functions. *American Mathematical Monthly*, 59:521–531, 1952.

[98] Rasmus V. Rasmussen and Michael A. Trick. A Benders approach for the constrained minimum break problem. *European Journal of Operational Research*, 177(1):198–213, 2007.

[99] W. Rei, J. F. Cordeau, M. Gendreau, and P. Soriano. Accelerating Benders decomposition by local branching. Technical report, 2006.

[100] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *The Handbook of Constraint Programming*. Elsevier, 2006.

[101] A. Ruszczynski and A. Shapiro, editors. *Stochastic Programming (Hanbooks in Operations Research and Management Series)*. Elsevier, 2003.

[102] ILOG S.A. *CPLEX: ILOG CPLEX 11.0 User's Manual and Reference Manual*, 2007. `http://www.ilog.com`.

[103] T. Sandholm and R. Shields. Nogood learning for mixed-integer programming. Technical Report CMU-CS-06-155, Carnegie Mellon University, 2006.

[104] M. W. P. Savelsbergh. Preprocessing and probing for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[105] Herbert E. Scarf. Neighborhood systems for production sets with indivisibilities. *Econometrica*, 54(3):507–532, 1986.

[106] Christian Schulte. *Programming Constraint Services*. PhD thesis, Universität des Saarlandes, Naturwissenschaftlich-Technische Fakultät I, Fachrichtung Informatik, Saarbrücken, Germany, 2000.

[107] Christian Schulte and Peter J. Stuckey. Speeding up constraint propagation. In *Principles and Practice of Constraint Programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 619–633. Springer, 2004.

[108] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2, 1989.

[109] A. Shapiro. Monte Carlo sampling approach to stochastic programming. In *ESAIM: Proceedings*, volume 13, pages 65–73, December 2003.

[110] João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.

[111] Ivan E. Sutherland. Sketchpad: a man-machine graphical communication system. In *DAC '64: Proceedings of the SHARE design automation workshop*, New York, NY, USA, 1964. ACM.

[112] Rekha Thomas and Robert Weismantel. Test sets and inequalities for integer programs. *Integer Programming and Combinatorial Optimization*, pages 16–30, 1996.

[113] Erlendur S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. *Lecture Notes in Computer Science*, 2239:16–30, 2001.

[114] K. Truemper. *Design of Logic-Based Intelligent Systems*. John Wiley & Sons, 2004.

[115] G. S. Tseitin. On the complexity of derivations in the propositional calculus. In A. O. Slisenko, editor, *Structures in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

[116] Stan P.M. van Hoesel, Arie M.C.A. Koster, Robert L.M.J. van de Leensel, and Martin W.P. Savelsbergh. Polyhedral results for the edge capacity polytope. *Mathematical Programming*, 92:335–358, 2002.

[117] B. Verweij, S. Ahmed, A. J. Kleywegt, G. Nemhauser, and A. Shapiro. The sample average approximation method applied to stochastic routing problems: A computational study. *Computational and Applied Optimization*, 24, 2003.

[118] David Waltz. Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision*, page pages. McGraw-Hill, 1975.