# Interdiction Branching

ANDREA LODI

Department of Mathematical and Industrial Engineering, Polytechnique Montréal

TED K. RALPHS

Department of Industrial Engineering, Lehigh University

FABRIZIO ROSSI AND STEFANO SMRIGLIO

Dipartimento di Ingegneria e scienze dell'informazione e matematica
Università di L'Aquila

COR@L Technical Report 11T-012-R2

# Interdiction Branching

ANDREA LODI*[1], TED K. RALPHS†[2], FABRIZIO ROSSI‡[3], AND STEFANO SMRIGLIO§[3]

[1]Department of Mathematical and Industrial Engineering, Polytechnique Montréal
[2]Department of Industrial Engineering, Lehigh University
[3]Dipartimento di Ingegneria e scienze dell'informazione e matematica , Università di L'Aquila

Original Publication: November 21, 2009
Last Revised: September 29, 2011

**Abstract**

This paper introduces *interdiction branching*, a new branching method for binary integer programs that is designed to overcome the difficulties encountered in solving problems for which branching on variables is inherently weak. Unlike traditional methods, selection of the disjunction in interdiction branching takes into account the best feasible solution found so far. In particular, the method is based on computing an *improving solution cover*, which is a set of variables of which at least one must be nonzero in any improving solution. From an improving solution cover, we can obtain a branching disjunction with desirable properties. Any minimal such cover yields a disjunction in which multiple variables are fixed in each child node and for which each child node is guaranteed to contain at least one improving solution. Computing a minimal improving solution cover amounts to solving a discrete bilevel program, which is difficult in general. In practice, a solution cover, although not necessarily minimal nor improving, can be found using a heuristic that achieves a profitable trade-off between the size of the enumeration tree and the computational burden of computing the cover. An empirical study on a test suite of difficult binary knapsack and stable set problems shows that an implementation of the method dramatically reduces the size of the enumeration tree compared to branching on variables, yielding significant savings in running times.

*andrea.lodi@polymtl.ca
†ted@lehigh.edu
‡fabrizio.rossi@univaq.it
§stefano.smriglio@univaq.it

## 1 Introduction

We consider mixed integer linear programs (MIPs) of the form

$$\max\{c^\top x \mid Ax \leq b,\ x \geq 0,\ x_j \in \{0,1\}\ \forall j \in I^p\}, \tag{1}$$

where $A \in \mathbb{Q}^{m\times n}$, $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$, and $I^p = \{1, \ldots, p\}$. We denote the set of feasible solutions to (1) by $\mathcal{F}$. A key ingredient in branch-and-cut algorithms for solving such binary MIPs is a method of partitioning the search space, referred to as the *branching scheme*. The most popular approach to this is to choose a binary variable $j \in I^p$ such that $0 < x^*_j < 1$ in the current optimal solution $x^*$ to the linear programming (LP) relaxation of (1) and to partition the feasible region using the associated *variable disjunction*, which arises from the fact that $x_j$ must take on either value one or value zero. The partitioning procedure then consists of creating one new subproblem in which $x_j$ is fixed to value one (the *left* child of the original problem) and another in which it is fixed to value zero (the *right child* of the original problem). This strategy is referred to in the rest of the paper as *variable branching* or *binary branching*. One of its primary advantages is fast reoptimization, since the child subproblems differ from the parent only in the bound of a single variable. This means the size of the basis does not increase after imposing the disjunction.

It is well-known that the effectiveness of a variable disjunction for branching—typically evaluated by considering the change in the optimal value of the LP relaxations of the two children with respect to that of the parent—depends strongly on the choice of branching variable [1, 10]. One major drawback of the variable branching strategy is that it can produce unbalanced enumeration trees. Indeed, if the branching variable is chosen poorly, then one of the resulting subproblems may be nearly as difficult to solve as the original problem. In some cases, this situation is endemic and cannot easily be avoided because the problem's structure is such that setting any single variable to zero has little effect due to the presence of other variables that are near substitutes.

A possible remedy is to consider branching with *general disjunctions*. This consists in adding a constraint $\pi^\top x \leq \pi_0$ in one child subproblem and a constraint $\pi^\top x \geq \pi_0 + 1$ in the other, where $\pi_j \in \mathbb{Z}$ for $j \in I^p$, $\pi_0 \in \mathbb{Z}$, and $\pi_0 < \pi^\top x^* < \pi_0 + 1$. Such a branching scheme can result in a dramatic reduction in tree size, as demonstrated in [9, 12, 13]. The major drawbacks of this general approach are the difficulty of choosing an effective branching disjunction and the increased computational burden caused by the additional constraints. A third alternative, which we consider here, is to branch using a multi-term (non-binary) disjunction arising from a judiciously chosen *set* of variables.

*Contribution of the paper*. We present a generalized branching method based on a different rationale and motivated by our attempts to overcome the difficulties discussed above. The method consists in choosing a set $S$ of variables and fixing each of these variables to one in turn to produce $|S|$ subproblems. The set $S$ is chosen in such a way that this is a valid branching disjunction, as we describe below. As with many of the procedures that constitute a branch-and-cut algorithm, the branching method presented here has both exact and heuristic variants that allow a tradeoff between strength and computational efficiency. The scheme specifically takes into account both the current incumbent value and the lower bound provided by the LP relaxation and can thus be seen as targeting improvements in both the upper and lower bounds. In its strongest form, given the best feasible solution found so far, say $\bar{x} \in \mathcal{F}$, the idea is to branch in such a way as to either prove that no improving solution exists in the feasible region of the parent node or produce $k \geq 2$ child subproblems, each of which is guaranteed to contain an *improving solution*, i.e., a solution with objective function value strictly better than $\bar{x}$.

Of course, there is no "free lunch," and the notable difficulty with the exact version of this method is that selecting the branching disjunction requires the solution of a *discrete bilevel programming problem*, which is a very difficult problem in theory as well as in practice. Thus, more practical versions of the method are obtained by solving those bilevel programs heuristically or in a relaxed form. In such a case, despite the weakening of the theoretical properties, the method can still effectively ensure that the bound improvement in the child subproblems is stronger and better balanced than with branching on variables. The bilevel program that constitutes the disjunction selection procedure is one of a special class of bilevel program called an *interdiction problem* and we thus call the method *interdiction branching* (IB).

Interdiction branching has some distinct advantages in practice. First, as in the traditional branching on variables, child subproblems are generated by imposing variable bounds, without introducing additional constraints. Second, in all but one of the $k$ children, at least two variables are fixed, often yielding a significant improvement in the bound provided by the LP relaxation. In other words, the method is robust with respect to problems with weak LP relaxations or for which variable branching results in one child exhibiting much better bound improvement than the other.

We have developed a branch-and-bound algorithm based on interdiction branching and have performed extensive experiments on a test suite of difficult binary knapsack problem (BKP) and stable set problem (SSP) instances. The results show that the method consistently outperforms a standard branch-and-bound both in terms of size of the enumeration tree and computing time.

The paper is organized as follows. In Section 2, we introduce the necessary definitions, present the method, and analyze its theoretical characteristics. In Section 3, we discuss the bilevel nature of the decision associated with branching and we formulate the interdiction branching problem. Section 6 reports computational results on difficult BKP instances by comparing with standard branch-and-bound and IBM-CPLEX as well. Some concluding remarks are reported in Section 7.

## 2 Interdiction Branching

Although most of the ideas presented in this paper are not restricted to binary programs, the overall approach is much simpler to describe when all variables are integer and restricted to be binary. We therefore address the canonical instance

$$\max_{x \in \{0,1\}^n} \{c^\top x \mid Ax \leq b\}. \tag{2}$$

in the remainder of the paper using the notation introduced earlier. An underlying motivation for our method is that for many classical combinatorial problems, such as the stable set problem, the set partitioning problem, and the binary knapsack problem, it is the case that standard variable branching results in unbalanced search trees due to the relative ineffectiveness of fixing a variable to zero. This is mainly because avoiding the use of a specific variable is usually not very costly in the LP relaxation, particularly at higher levels in the tree and/or in the presence of symmetry. The effect is generally to produce one (strong) child node with a substantially improved bound and another (weak) child node in which the bound remains nearly the same[1].

Let us denote the set of variable indices by $I^n = \{1, \ldots, n\}$ and characterize a node $a$ of the branch-and-bound tree by the indices $F_1^a$ of variables fixed to one and $F_0^a$ of variables fixed to zero at node $a$. The resulting set of free variables is denoted by $N^a = I^n \setminus (F_0^a \cup F_1^a)$, while the set of feasible solutions to subproblem $a$ is denoted by $\mathcal{F}^a$. Finally, let $\bar{x} \in \mathcal{F}$ be the *incumbent* solution, that is, the best feasible solution found so far, and let $\bar{z} = c^\top \bar{x}$ be its value. In what follows, any feasible solution $\tilde{x} \in \mathcal{F}$ such that $c^\top \tilde{x} > \bar{z}$ is referred to as an *improving solution*. We also denote by $\mathcal{F}^a(\bar{z})$ the set of improving solutions at subproblem $a$ with respect to the incumbent value $\bar{z}$.

Interdiction branching is an $n$-ary branching strategy based on the following principle. Given an index set $S \subseteq N^a$, the feasible region $\mathcal{F}^a$ of a given subproblem can be split into $|S| + 1$ subproblems, the first $|S|$ of these obtained by fixing to one each variable $x_i$, $i \in S$ in turn and the last obtained by fixing all variables in $S$ to zero. Formally, the disjunction

$$x_{i_1} = 1 \vee x_{i_2} = 1 \vee \ldots \vee x_{i_{|S|}} = 1 \vee \sum_{i \in S} x_i = 0 \tag{3}$$

is satisfied by all feasible solutions. This disjunction can be strengthened so as to induce a partition of $\mathcal{F}^a$. Indeed, for any sequence $(i_1, \ldots, i_{|S|})$, the disjunction

$$x_{i_1} = 1 \vee (x_{i_2} = 1 \wedge x_{i_1} = 0) \vee \ldots \vee (x_{i_{|S|}} = 1 \wedge x_{i_1} = 0 \wedge \ldots \wedge x_{i_{|S|-1}} = 0) \vee \sum_{i \in S} x_i = 0 \tag{4}$$

is still valid and represents a partition of $\mathcal{F}^a$. The key to effectiveness of the method is the way in which the set $S$ is chosen. The method exhibits very strong theoretical properties when the set $S$ is chosen to be an *improving solution cover*, as described in the next section.

## 2.1 Theoretical Properties

We say that a variable index $i \in I^n$ *covers* a feasible solution $\hat{x} \in \mathcal{F}$ if $\hat{x}_i = 1$, and that an index set $S$ covers a set of solutions $X$ if every solution in $X$ is covered by at least one index in $S$. Furthermore, $S(\hat{x}) \subseteq S$ denotes the set of indices in $S$ covering the solution $\hat{x} \in X$. Let us consider a subproblem $a$ and assume that $\mathcal{F}^a(\bar{z}) \neq \emptyset$.

**Definition 1.** *An* improving solution cover *for a with respect to $\bar{z}$ is an index set $S^a(\bar{z}) = \{i_1, \ldots, i_{|S^a(\bar{z})|}\} \subseteq N^a$ covering $\mathcal{F}^a(\bar{z})$.*

[1]In cases for which fixing variables to zero is strong and fixing to one is not, one can always complement the variables.

Note that an improving solution cover may not exist. However, this occurs if and only if either $\mathcal{F}^a(\bar{z}) = \emptyset$ or there exists a member of $\mathcal{F}^a(\bar{z})$ for which $x_k = 0$ for all $k \in N^a$. The second condition can be easily checked and the incumbent updated. Therefore, in the remainder of the paper we make the following assumption.

**Assumption 1.** *For any subproblem $a$, either $\mathcal{F}^a(\bar{z}) = \emptyset$ or at least one nonempty improving solution cover exists.*

Interdiction branching is based on the computation of an improving solution cover $S^a(\bar{z})$, which can then be exploited by taking $S = S^a(\bar{z})$ in disjunction (4). By construction, the subproblem in which all variables in $S^a(\bar{z})$ are set to zero does not contain improving solutions and need not be explored. As a consequence, the disjunction (4) can be strengthened to

$$x_{i_1} = 1 \vee (x_{i_2} = 1 \wedge x_{i_1} = 0) \vee \ldots \vee (x_{i_{|S^a(\bar{z})|}} = 1 \wedge x_{i_1} = 0 \wedge \ldots \wedge x_{i_{|S^a(\bar{z})|-1}} = 0) \qquad (5)$$

The resulting branching method is summarized in Algorithm 1.

**Algorithm 1** Interdiction Branching: General Scheme

```
Input:    Subproblem $a = (F_1^a, F_0^a)$, incumbent value $\bar{z}$.
Output:   A set of child subproblems.

Step 1.   Attempt to compute an improving solution cover $S^a(\bar{z})$.
Step 2.   If $\mathcal{F}^a(\bar{z}) = \emptyset$, then prune $a$. STOP.
Step 3.   Otherwise, choose a sequence $(i_1, \ldots, i_{|S^a(\bar{z})|})$.
Step 4.   Return subproblems $(F_1^a \cup \{i_1\}, F_0^a), (F_1^a \cup \{i_2\}, F_0^a \cup \{i_1\}), \ldots, (F_1^a \cup \{i_{|S^a(\bar{z})|}\}, F_0^a \cup \{1_1, \ldots, i_{|S^a(\bar{z})|-1}\})$.
```

Obviously, the main challenge is in computing the solution cover in **Step 1**. The remainder of the paper is devoted to discussing the details of how this is done. Note that **Step 1** implicitly includes testing whether an improving solution is obtained by setting $x_k = 0,\ \forall k \in N^a$. If this is the case, the incumbent solution is updated and the search is repeated. Thus, the test in **Step 2** is the same as asking whether there exists an improving solution cover.

The strength of interdiction branching is captured by the following property.

**Theorem 1.** *If $S^a(\bar{z})$ is a minimal improving solution cover*[2]*, then each term of the disjunction (5) is satisfied by at least one solution in $\mathcal{F}^a(\bar{z})$.*

**Proof.** Given the sequence $(i_1, \ldots, i_{|S^a(\bar{z})|})$ and a solution $\hat{x} \in \mathcal{F}^a(\bar{z})$, we denote by $r(\hat{x})$ the index of the term of the disjunction for which $r(\hat{x}) = \arg\min_{k=1,\ldots,S^a(\bar{z})}\{i_k \in S(\hat{x})\}$. If the cover is minimal, then for each index $r \in S^a(\bar{z})$, we must have $r = r(\hat{x})$ for at least one solution

[2]An improving solution cover $S^a(\bar{z})$ is said to be *minimal* if $S^a(\bar{z}) \setminus \{i\}$ is not an improving solution cover for any $i \in S^a(\bar{z})$.

$\hat{x} \in \mathcal{F}^a(\bar{z})$ (otherwise $r$ could be removed while $\mathcal{F}^a(\bar{z})$ is still covered). Therefore, each term of the disjunction (5) is satisfied by at least one solution in $\mathcal{F}^a(\bar{z})$. □

As a consequence, interdiction branching generates a set of child subproblems inducing a partition of the set $\mathcal{F}^a(\bar{z})$ of the improving solutions into non-empty subsets. This has an important consequence on the size of the enumeration tree. Indeed, an upper bound on the total number of generated subproblems can be provided as a function of the number of improving solutions, though we need to be precise about exactly how the subproblems are counted to prove this result.

Specifically, when $|S^a(\bar{z})| = 1$, disjunction (5) has a single term. Therefore, a (single) variable is fixed to one and no branching occurs. In this case, we do not count this as a new subproblem, but consider it to be the fixing of a variable in the parent node. We refer to this situation as *interdiction fixing*. A special case arises when $|\mathcal{F}^a(\bar{z})| = 1$. If we denote by $\widetilde{x}$ the unique improving solution contained in $\mathcal{F}^a(\bar{z})$, the procedure first tests whether $\widetilde{x}_i = 0$ for all $i \in N^a$. If this is the case, $\widetilde{x}$ is found immediately and no improving solutions remain. Otherwise, all minimal improving solution covers will have cardinality one and through successive application of interdiction fixing, $\widetilde{x}$ will again be found. Hence, if branching occurs, there must exist at least two improving solutions.

Let us denote the root node by 0 and $\mathcal{F}^0(\bar{z})$ the set of improving solutions at the root node with respect to a given incumbent value $\bar{z}$. We can now prove that:

**Corollary 1.** *The number of subproblems generated by interdiction branching is at most* $2|\mathcal{F}^0(\bar{z})| - 1$.

**Proof.** The largest enumeration tree is produced when interdiction branching always results in a binary branching where (i) $|\mathcal{F}^a(\bar{z})| - 1$ improving solutions belong to one subproblem and just one solution belongs to the other subproblem and (ii) the solution that is the unique optimum in its subproblem is no better than any of the $|\mathcal{F}^a(\bar{z})| - 1$ solutions in the other node. Subproblems including a unique improving solution will be pruned after a sequence of (interdiction) fixings. In this worst case, the branching process stops at depth $|\mathcal{F}^0(\bar{z})| - 1$ after $2|\mathcal{F}^0(\bar{z})| - 2$ nodes have been generated. When we include the root node, the result follows. □

## 3 Improving Solution Covers and Bilevel Programming

Algorithm 1 requires a method to either prove that $\mathcal{F}^a(\bar{z}) = \emptyset$ or compute an improving solution cover $S^a(\bar{z})$. This task can be carried out without computing the set of improving solutions explicitly, by solving a bilevel programming problem. Consider a generic subproblem $a$ and let $\bar{z}$ be the current incumbent solution. We have the following characterization for an improving solution cover $S^a(\bar{z})$.

**Theorem 2.** *A nonempty index set $S^a(\bar{z}) \subseteq N^a$ is an improving solution cover for $a$ with respect to $\bar{z}$ if and only if*

$$\max_{x \in \{0,1\}^n} \{c^\top x \mid x \in \mathcal{F}^a, x_i = 0 \textit{ for all } i \in S^a(\bar{z})\} \leq \bar{z}. \tag{6}$$

**Proof.** Recalling Assumption 1, if condition (6) holds, then no solution $\hat{x} \in \mathcal{F}^a(\bar{z})$ can exist such that $\hat{x}_i = 0, \forall i \in S^a(\bar{z})$. Thus, $S^a(\bar{z})$ is a solution cover for $a$ with respect to $\bar{z}$. If $S^a(\bar{z})$ is an improving solution cover for $a$ with respect to $\bar{z}$, then at least one variable $x_i, i \in S^a(\bar{z})$ must assume value one to obtain a solution in $\mathcal{F}^a(\bar{z})$. Thus, condition (6) holds. □

Thanks to the above result, the problem of computing the smallest improving solution cover admits a straightforward bilevel programming formulation. Let $x_i$ ($i \in I^n$) be an original variable and $y_i$ a binary artificial variable taking value one if index $i$ is in the improving solution cover $S^a(\bar{z})$ and zero otherwise. Let us consider the following *interdiction branching problem* (IBP):

$$\min \sum_{i \in I^n} y_i \tag{7}$$

$$\sum_{i \in I^n} c_i x_i \leq \bar{z} \tag{8}$$

$$y \in \{0,1\}^n \tag{9}$$

$$x \in \arg\max_{x} c^\top x \tag{10}$$

$$x_i + y_i \leq 1, \; i \in I^n \tag{11}$$

$$x \in \mathcal{F}^a. \tag{12}$$

The lower-level problem includes the original constraints (12) plus the interdiction constraints (11) that enforce that the variables in the cover are fixed to zero in the original problem. The upper-level problem has only one constraint, namely, constraint (8), which imposes that the set defined by variables $y$ is a cover, according to Theorem 2. The objective function enforces that a minimum cardinality improving solution cover is identified.

Note that if the optimal value to IBP is zero, then no improving solution exists with respect to the incumbent value $\bar{z}$, that is, $\mathcal{F}^a(\bar{z}) = \emptyset$ and subproblem $a$ can be pruned (**Step 1** in Algorithm 1). Moreover, IBP may be infeasible in the case that an improving solution can be obtained by setting $x_k = 0, \; \forall k \in N^a$. Indeed, there is no way in this case of forcing the optimal solution value of the lower level integer problem to be smaller than the incumbent value by fixing variables to zero, so constraint (8) cannot be satisfied. However, as briefly mentioned in the previous section, it is better in practice to simply check for such a condition by solving the corresponding linear program and possibly updating the incumbent solution before considering the bilevel program.

Three main goals are achieved by IBP with respect to the bound of the LP relaxation: (i) in case of unbalanced branching decisions (recall the discussion at the beginning of Section 2), the child nodes associated with all but the right-most term of disjunction (4) are "strong" in the sense that at least one variable is fixed to one; (ii) by constraint (8), we can force the right-most branching node, the "weak" one, to be immediately pruned by bound; and (iii) in addition to having one variable fixed to one, every branching node but the left-most one has at least one variable fixed to zero.

In the remainder of the paper, we use the binary knapsack problem to illustrate a number of concepts. The binary knapsack problem is one of the most well-known combinatorial optimization problems and we introduce it here for completeness. Given a set $N = \{1, \ldots, n\}$ of items with a profit $p_j$ and weight $w_j$ associated with each item $j \in N$, one is required to select a subset of items such that the overall profit of the selected items is maximized and the sum of the weights of the selected items does not exceed a given capacity $C$.

We report in the following example a small BKP instance together with the branch-and-bound tree obtained by an implementation of Algorithm 1 in which we compute minimal covers in **Step 1**.

**Example 1.** *Given the BKP instance*

$$\begin{aligned} \max \quad & 19620x_1 + 13407x_2 + 10464x_3 + 7521x_4 + 16848x_5 + 15876x_6 + 4860x_7 + 2592x_8 \\ & 1680x_1 + 1148x_2 + 896x_3 + 644x_4 + 1456x_5 + 1372x_6 + 420x_7 + 224x_8 \le 2613 \\ & x \in \{0,1\}^8, \end{aligned}$$

*and using the (initial) incumbent solution $\bar{x}_1 = \bar{x}_4 = 1$ of value $\bar{z} = 27141$ one has:*

**Root node:** $F_0^0 = \emptyset, F_0^1 = \emptyset$
*List of improving solutions:*

| *Variables* | *Objective* | *Variables* | *Objective* |
|---|---|---|---|
| $\{1,3\}$ | *30084* | $\{3,5\}$ | *27312* |
| $\{1,4,8\}$ | *29733* | $\{3,5,8\}$ | *29904* |
| $\{2,5\}$ | *30255* | $\{3,6,8\}$ | *28932* |
| $\{2,6\}$ | *29283* | $\{4,5,7\}$ | *29229* |
| $\{2,3,7\}$ | *28731* | $\{4,6,7\}$ | *28257* |
| $\{2,4,7,8\}$ | *28380* | | |

*Minimum cover:* $S^0(27141) = \{2,3,4\}$
*Branching decision:*
$F_C^0 = \emptyset$, $F_C^1 = \{2\}$*;*
$F_B^0 = \{2\}$, $F_B^1 = \{3\}$*;*
$F_A^0 = \{2,3\}$, $F_A^1 = \{4\}$*.*

**Node A:** $F_A^0 = \{2,3\}$, $F_A^1 = \{4\}$
*List of improving solutions:*

| *Variables* | *Objective* | *Variables* | *Objective* |
|---|---|---|---|
| $\{1,4,8\}$ | *29733* | $\{4,6,7\}$ | *28257* |
| $\{4,5,7\}$ | *29229* | | |

*Minimum cover:* $S^A(27141) = \{1,7\}$
*Branching decision:*
$F_E^0 = \{2,3\}$, $F_E^1 = \{1,4\}$*;*
$F_D^0 = \{1,2,3\}$, $F_D^1 = \{4,7\}$*.*

**Node D:** $F_D^0 = \{1,2,3\}$, $F_D^1 = \{4,7\}$
*List of improving solutions:*

| *Variables* | *Objective* | *Variables* | *Objective* |
|---|---|---|---|
| $\{4,5,7\}$ | *29229* | $\{4,6,7\}$ | *28257* |

*Minimum cover:* $S^D(27141) = \{5,6\}$
*Branching decision:*
$F_G^0 = \{1,2,3\}$, $F_G^1 = \{4,5,7\}$*;*
$F_F^0 = \{1,2,3,5\}$, $F_F^1 = \{4,6,7\}$*.*

**Node F:** $F_F^0 = \{1,2,3,5\}$, $F_F^1 = \{4,6,7\}$ *Unique improving solution:* $\{4,6,7\}$*, objective value: 28257. Subproblem closed by optimality and incumbent updated to 28257.*

**Node G:** $F_G^0 = \{1,2,3\}$, $F_G^1 = \{4,5,7\}$ *Unique improving solution:* $\{4,5,7\}$*, objective value: 29229. Subproblem closed by optimality and incumbent updated to 29229.*

**Node E:** $F_E^0 = \{2,3\}$, $F_E^1 = \{1,4\}$ *Unique improving solution:* $\{1,4,8\}$*, objective value 29733.*

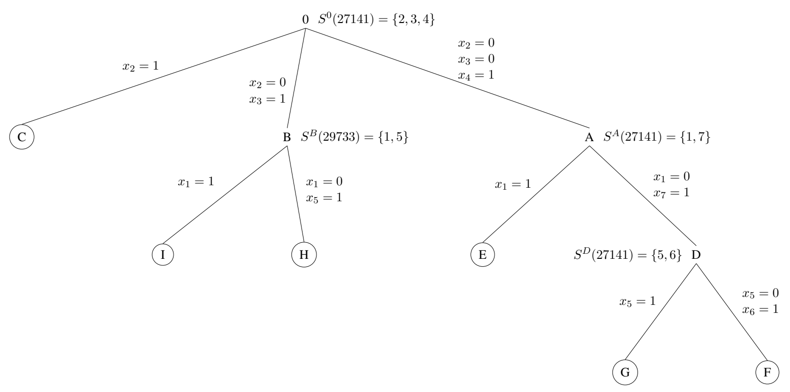


Figure 1: Enumeration tree from exact interdiction branching on BKP example.

*Subproblem closed by optimality and incumbent updated to 29733.*

**Node B:** $F_B^0 = \{2\}$, $F_B^1 = \{3\}$
*List of improving solution:*

| *Variables* | *Objective* | *Variables* | *Objective* |
|---|---|---|---|
| $\{1,3\}$ | *30084* | $\{3,5,8\}$ | *29904* |

*Minimum cover:* $S^B(29733) = \{1,5\}$
*Branching decision:*
$F_I^0 = \{2\}$, $F_I^1 = \{1,3\}$;
$F_H^0 = \{1,2\}$, $F_H^1 = \{3,5\}$

**Node H:** $F_H^0 = \{2\}$, $F_H^1 = \{3,5\}$ *Unique improving solution:* $\{3,5,8\}$, *objective value: 29904. Subproblem closed by optimality and incumbent updated to 29904.*

**Node I:** $F_I^0 = \{2\}$, $F_I^1 = \{1,3\}$ *Unique improving solution:* $\{1,3\}$, *objective value: 30084. Subproblem closed by optimality and incumbent updated to 30084.*

**Node C:** $F_C^0 = \emptyset$, $F_C^1 = \{2\}$ *Unique improving solution:* $\{2,5\}$, *objective value: 30255. Subproblem closed by optimality and incumbent updated to 30255.*

*The resulting enumeration tree is depicted in Figure 1.*
*A textbook branch-and-bound implementation with standard variable branching (initialized with the same incumbent of value* $\bar{z} = 27141$*) yields the enumeration tree drawn in Figure 2 when applied to the same BKP instance (the MIP solver CPLEX, used for comparison in Section 6, explores 60 nodes.) In the figure, square nodes represent subproblems pruned by infeasibility, while circle nodes those pruned by optimality. Note that no nodes are pruned by bound. One specific feature of this BKP instance is that this still holds if an optimal solution is provided at the root node of the standard branch-and-bound. Therefore, the enumeration tree does not change in this case.*

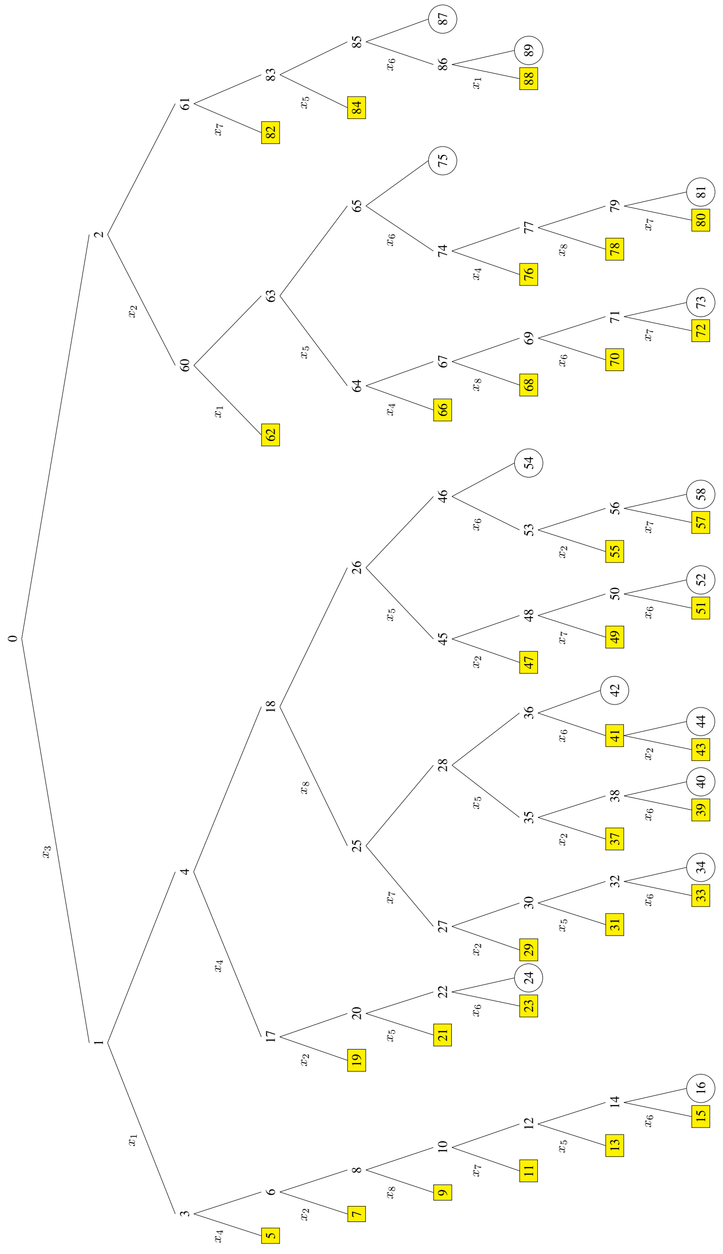


Figure 2: Enumeration tree from branching on variables

Bilevel programs are in the complexity class $\mathcal{N}P$-hard in general, even when all variables are continuous [3, 7, 2, 6], though in this special case, it is well-known that the problem can be reformulated as a mathematical program with equilibrium constraints (MPEC) [11]. In the more general case of the IBP, which has discrete variables in the lower level, the problem is difficult not only in complexity terms, but also in practice [15, 4]. Thus, the advantages of interdiction branching do not come for free, and we quickly encounter a very strong tradeoff between the strength of the disjunctions one obtains by solving the IBP to optimality and the effort required to do so. It is clear that some sacrifices have to be made in order to improve the tractability of the branching subproblem.

## 4 Relaxations and Practical Alternatives

The results in the previous sections show that IB is strong at the price of solving subproblems that are difficult both in theory and in practice. In the following, we show how to substantially reduce such computational complexity while keeping the spirit of the method unchanged.

### 4.1 Relaxing the Integrality Requirement on the Lower Level

Because solving the IBP (7)–(12) to optimality is not practical in general, one way to limit the computational effort is to relax the integrality requirement $x \in \mathcal{F}^a$ in (12) with the weaker requirement that $x$ satisfy just the linear constraints of the original problem and not integrality. We call the resulting problem the *interdiction branching linear program* (IBLP) and denote by $\mathcal{F}^a_{LP}$ the set of feasible solutions of the linear programming relaxation of node $a$. It is easy to show a counterpart of Theorem 2 for IBLP.

**Theorem 3.** *A nonempty index set $S^a(\bar{z}) \subseteq N^a$ is an improving solution cover for $a$ with respect to $\bar{z}$ if*

$$\max\{c^\top x \mid x \in \mathcal{F}^a_{LP}, x_i = 0 \textit{ for all } i \in S^a(\bar{z})\} \leq \bar{z}. \tag{13}$$

**Proof.** The optimal value of the left-hand-side of (13) is an upper bound on the optimal value of the left-hand-side of (6). Thus, if inequality (13) is satisfied, then inequality (6) is satisfied as well and the corresponding set is a cover. □

Condition (13) is sufficient but not necessary in this case and the cover computed by solving IBLP might not therefore be minimal. Thus, the property that, after branching, the feasible region of each of the child nodes contains at least one improving solution is no longer guaranteed (recall Theorem 1). This is to be expected, given the strength of that property.

The resulting IBLP reads as follows.

$$\begin{aligned}
&\min \sum_{i \in I^n} y_i \\
&\quad \sum_{i \in I^n} c_i x_i \le \bar{z} \\
&\quad y \in \{0,1\}^n \\
&\quad x \in \arg\max_x c^\top x \\
&\qquad\qquad x_i + y_i \le 1,\ i \in I^n \\
&\qquad\qquad x \in \mathcal{F}^a_{LP}, \qquad (14)
\end{aligned}$$

where (14) relaxes the integrality requirement in (12).

## 4.2 Heuristically Solving IBP and IBLP

Although the alternative to the IBP described above does indeed improve the tractability of finding a branching disjunction substantially, it is not even strictly necessary to solve either IBP or IBLP to optimality in order to obtain a cover. In practice, the cover can be found by any means, as long the conditions of either Theorems 2 or 3 can be verified.

In the following, we concentrate on IBLP (instead of IBP) and propose a simple and general heuristic algorithm for generating a cover. The algorithm is based on the trivial observation that feasible solutions to IBLP correspond to sets $S$ of variable indices such that the optimal value of the linear program $\{\max c^\top x : x \in \mathcal{F}_{LP},\ x_j = 0\ \forall j \in S\}$ is less than or equal to $\bar{z}$. This observation is exploited in the straightforward heuristic illustrated by Algorithm 2.

**Algorithm 2** Heuristic for IBLP

```
Input:       A linear program {max c^T x : x ∈ F_LP}, a value z̄.
Output:      A feasible solution S ⊆ N to IBLP.

Initialize   $S := \emptyset$
While
             $\widetilde{z} := \{\max c^\top x : x \in \mathcal{F}_{LP}, x_j = 0\ \forall j \in S\}$
             if $\widetilde{z} > \bar{z}$
                if the corresponding solution $\widetilde{x}$ is integer (an improving solution is found) then STOP
                else choose an index $j$ such that $0 < \widetilde{x}_j < 1$ and set $S := S \cup \{j\}$
Until        ($\widetilde{z} > \bar{z}$)
Return       $S$
```

It is easy to see that if Algorithm 2 returns $S = \emptyset$, then the subproblem can be pruned. Notice that the first iteration of Algorithm 2 amounts to solving the current LP relaxation. Another important remark is that in two consecutive iterations the corresponding LPs differ from a single variable bound: this yield a relevant benefit in terms of fast reoptimization. In practice, Algorithm 2 requires the specification of procedures to determine the order in which to add variables to the set $S$, as well as to determine the order in which the variables are considered in constructing the disjunction 4. We address these question later in Section 5.

The reprise of Example 1 shows that, even with such a heuristic, IB outperforms standard variable branching.

**Example 2.** *Figure 3 shows the enumeration tree obtained by IB when* **Step 1** *of Algorithm 1 is carried out by Algorithm 2. Dashed lines labeled by one or more variables denote interdiction fixings. Notice that, in the BKP case, only one variable is fractional at each iteration (corresponding to the critical item computed by Dantzig's algorithm) and no ambiguity arises in its selection. As for* **Step 3** *of Algorithm 1, variables are ordered by non-increasing values of the weight $w_j$. The covers $S^k$ in Figure 3 are then ordered sets: the leftmost child of subproblem $k$ has the first variable in $S^k$ fixed to $1$ (while the others are free) and the rigthmost child has all variables in $S^k$ fixed to $0$ but the last one that is fixed to $1$, as in* **Step 4** *of Algorithm 1.*

*Similarly to what happens with the search tree from standard variable branching (Figure 2), the tree of Figure 3 does not change if the algorithm is initialized with the initial incumbent value 27141 of Example 1, or with the optimal incumbent value 30255. In the first case, we can compare the tree of Figure 3 with the one in of Figure 1 and observe that the heuristic IB requires 33 nodes to solve the problem to optimality versus 10 nodes of the exact version.*

*When the initial incumbent value is the optimal one, we have an additional insight. In this case the exact version of IB would return an empty cover at the root node, providing a certificate of optimality. On the contrary, the covers computed by Algorithm 2 are not minimal and all the subproblems in the enumeration tree drawn in Figure 3 do not contain improving solutions. Thus, the size of the tree gives us a more direct measure of the impairment of the method due to heuristically computing the covers.*

*However, the heuristic IB still largely outperforms standard variable branching (Figure 2). Notice also that all subproblems are pruned by infeasibility. A possible explanation is that IB is trying to fix variables in order to find improving solution which do not exists any more, so falling into infeasibilities. This may bring some some advantage since proving infeasibility of a subproblem can be easier than solving its LP relaxation. More insights about this comparison are presented in Section 6.*

The characteristics and the performance of Algorithm 2 applied to BKPs are discussed in more detail in Section 5.1. In the general case, the choice of which fractional variables to include in the set $S$ is a crucial issue affecting the quality of the resulting cover. It should also be emphasized that Algorithm 2 is only the simplest option among a large family of alternatives that comes into play by exploiting the result presented in the next section, where we discuss the possibility of reformulating IBLP as a MIP.

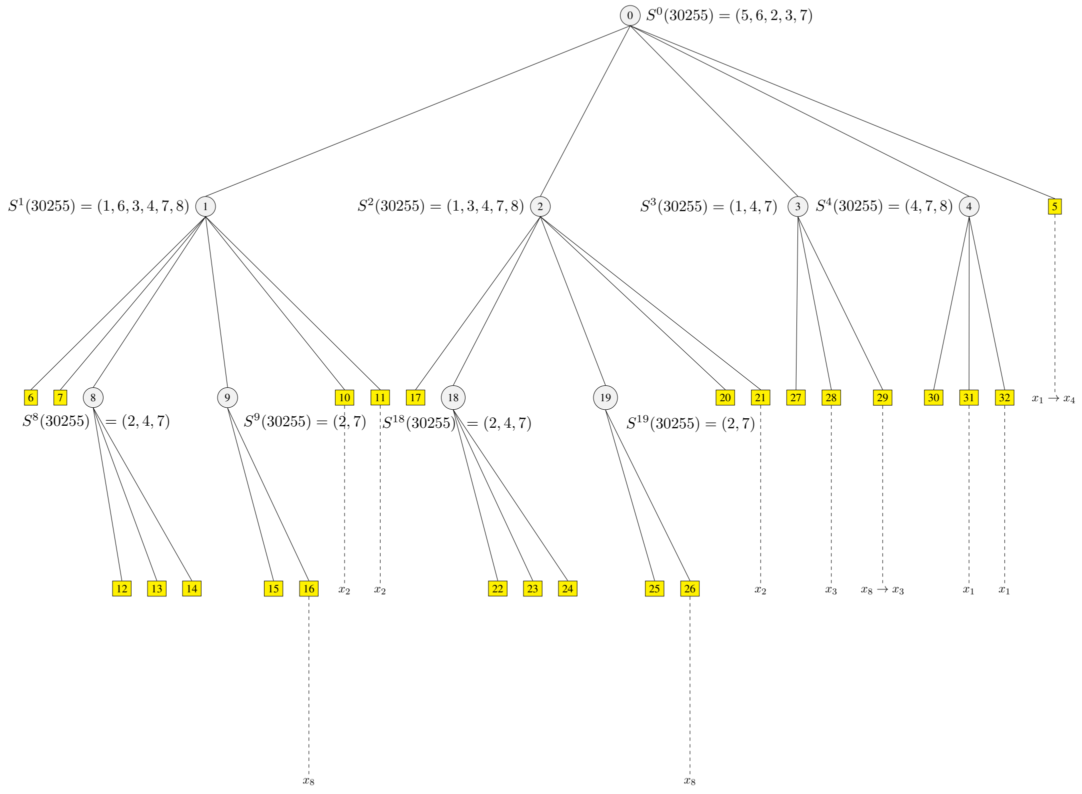


Figure 3: Heuristic interdiction branching on a small BKP.

## 4.3 Reformulating IBLP into a MIP

Because of the continuous nature of the lower level problem, we can reformulate IBLP as a standard MIP by replacing the linear relaxation of (2) with its dual, namely

$$\begin{aligned} \min\ & b^\top u + \mathbf{1}^\top w \\ & u^\top A_i + w_i \geq c_i \quad i \in I^n \\ & u, w \geq 0, \end{aligned} \tag{15}$$

where $w_i$ is a dual variable associated with variable upper bound constraint $x_i \leq 1$ for all $i \in I^n$. Then, a straightforward MIP reformulation of IBLP (7)–(11) and (14) reads as

$$\text{(MIP.1)} \quad \min \sum_{i \in I^n} y_i$$

$$b^\top u + \mathbf{1}^\top w \leq \bar{z} \tag{16}$$

$$b^\top u + \mathbf{1}^\top w - c^\top x = 0 \tag{17}$$

$$u^\top A_i + w_i \geq c_i \qquad i \in I^n \tag{18}$$

$$a_i^\top x \leq b \qquad i = 1, \ldots, m \tag{19}$$

$$x_i + y_i \leq 1 \qquad i \in I^n \tag{20}$$

$$u, w \geq 0, \quad x \in [0,1]^n, \quad y \in \{0,1\}^n,$$

where constraints (19) guarantee primal feasibility, constraints (18) guarantee dual feasibility, and constraint (17) establishes optimality by requiring the values of the primal and the dual solutions to coincide. In other words, constraints (17)–(19) guarantee that the lower level continuous problem has been solved to optimality and that its optimal value is indeed not greater than $\bar{z}$ (constraint (16)).

The above MIP can be substantially simplified as

$$\text{(MIP.2)} \quad \min \sum_{i \in I^n} y_i$$

$$b^\top u + \mathbf{1}^\top w \leq \bar{z} \tag{21}$$

$$u^\top A_i + w_i \geq c_i - M y_i \qquad i \in I^n \tag{22}$$

$$u, w \geq 0, \quad y \in \{0,1\}^n,$$

The correctness of the above simplification is proven by the following proposition.

**Proposition 4.** *(MIP.1) and (MIP.2) are equivalent.*

**Proof.** First, recall that, by construction, constraints (20) ensure that a variable $x_i$ is set to zero (removed from the problem) if the associated binary variable $y_i$ is set to one. This corresponds to deactivating the $i^{\text{th}}$ constraint in the dual and is accomplished similarly in (MIP.2) by adding to the right hand side of each inequality (15) a term $My_i$, for a sufficiently large value $M$. This guarantees dual feasibility of the continuous solution of the former lower level. It remains to show that such solution is also optimal. However, because the dual is in minimization form, if there exists a feasible solution with value smaller than or equal to $\bar{z}$ (as required by constraint (21)), then the optimal value will certainly not be greater. This is enough to show that the lower level of the corresponding IBLP does not contain any improving solution. □

The difficulty of solving the MIP formulation of IBLP to optimality is strongly problem dependent. However, it is worth recalling that any feasible solution of (MIP.2) identifies a valid improving solution cover. In the Section 6, we show that state-of-the-art MIP heuristics allow the method to perform effectively.

## 5 Implementation

Before getting to the details of the computation, we discuss a simple way of implementing our $n$-ary branching method within a binary tree. This is necessary because binary trees are standard for commercial and non-commercial MIP solvers at present, with limited exceptions. Thus, we use the binary implementation of IB to provide a more convincing and accurate comparison with state-of-the-art variable branching methods.

### 5.1 Implementing Interdiction Branching within a Binary Tree

A binary version of IB can be obtained by successive application of binary branchings. Let us consider a generic subproblem $a = (F_a^1, F_a^0)$ and denote by $d(a,i) = (F_a^1, F_a^0 \cup \{i\})$ the right branch obtained by applying the standard variable branching to variable $x_i$, $i \in N^a$, and by $u(a,i) = (F_a^1 \cup \{i\}, F_a^0)$ the associated left branch.

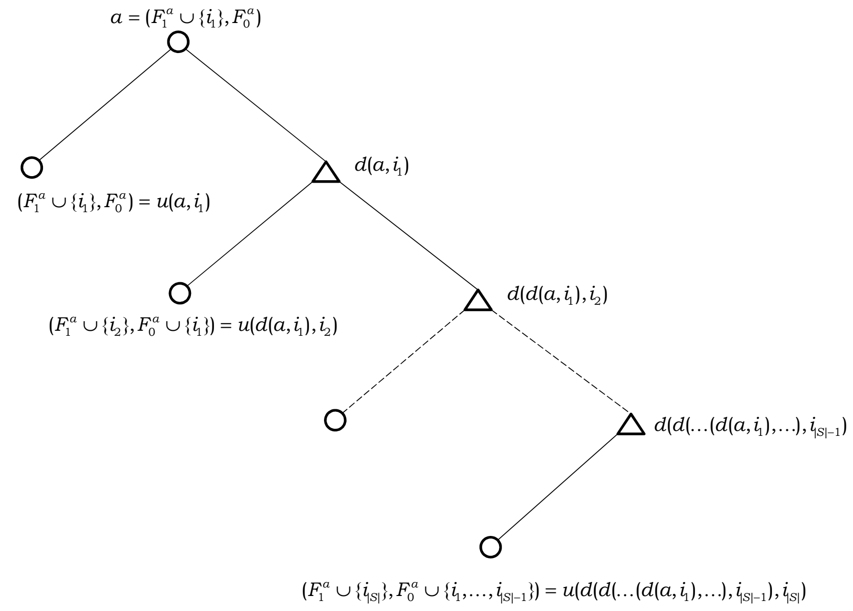
Suppose we are given a branching set $S = S^a(\bar{z})$ along with an ordering $(i_1, \ldots, i_{|S|})$ of its variable indices. The IB scheme generates the subproblems $(F_1^a \cup \{i_1\}, F_0^a), (F_1^a \cup \{i_2\}, F_0^a \cup \{i_1\}), \ldots, (F_1^a \cup \{i_{|S|}\}, F_0^a \cup \{i_1, \ldots, i_{|S|-1}\})$. These can also be obtained by the successive binary branchings

$$
\begin{aligned}
&(F_1^a \cup \{i_1\}, F_0^a) = u(a, i_1),\\
&(F_1^a \cup \{i_2\}, F_0^a \cup \{i_1\}) = u(d(a, i_1), i_2),\\
&\ldots\\
&(F_1^a \cup \{i_{|S|}\}, F_0^a \cup \{i_1, \ldots, i_{|S|-1}\}) = u(d(d(\ldots(d(a, i_1), \ldots), i_{|S|-1}), i_{|S|}).
\end{aligned}
$$

The corresponding binary tree is drawn in Figure 4.

Notice that binary branchings also generate right branches (represented by triangles in the figure) that are not generated by IB. These need not be explored (i.e., one could avoid solving the associate LP relaxation) and we refer to them as *pseudo-subproblems*.

Again, let's continue Example 1 to have more insights.



$a = (F_1^a \cup \{i_1\}, F_0^a)$
$d(a, i_1)$
$(F_1^a \cup \{i_1\}, F_0^a) = u(a, i_1)$
$d(d(a, i_1), i_2)$
$(F_1^a \cup \{i_2\}, F_0^a \cup \{i_1\}) = u(d(a, i_1), i_2)$
$d(d(\ldots(d(a, i_1), \ldots), i_{|S|-1})$
$(F_1^a \cup \{i_{|S|}\}, F_0^a \cup \{i_1, \ldots, i_{|S|-1}\}) = u(d(d(\ldots(d(a, i_1), \ldots), i_{|S|-1}), i_{|S|})$

Figure 4: Binary tree implementing interdiction branching.

**Example 3.** *In Figure 5 the binary version of the enumeration tree of Figure 3 is drawn. Triangular nodes represent pseudo-subproblems. For the sake of clarity, the branching variable is indicated only in the left branch (where it is fixed to* $1$*). Also, dashed lines labeled by one or more variables denote a sequence of interdiction fixings.*

*The structure of the BKP and the binary representation of the IB tree allow a somewhat more straightforward comparison of IB with standard variable branching. Indeed, we can now exactly compare the number of LPs solved by the two methods. Let us look at the root node. The application of Algorithm 2 returns the sequence* $(3, 2, 7, 5, 6)$ *of (fractional) variables, corresponding to a sequence of critical items in Dantzig's algorithm. The LPs solved correspond exactly to those of subproblems* $2, 61, 83, 85$ *in the tree of Figure 2 (this implies that IB always cuts off the current fractional solution.) In this respect, the application of IB seems of no value because the portion of the (binary) tree that is not explored by IB is indeed explored implicitly when solving IBLP by Algorithm 2. This issue will be reconsidered in the next subsection.*

The example suggests a distinction between "genuine" subproblems and pseudo-subproblems: the number of the latter ones provides a fairly accurate measure of the size of the binary subtrees corresponding to each original branching disjunction produced by IB, so the overall number of IB (binary) nodes can be roughly compared with that of the traditional variable branching scheme. On the other hand, the computing time for the pseudo-nodes should in principle be removed because it is otherwise counted twice. However, in the experiments presented below, we have not been able to completely remove those times within our IB implementation using CPLEX callback functions.

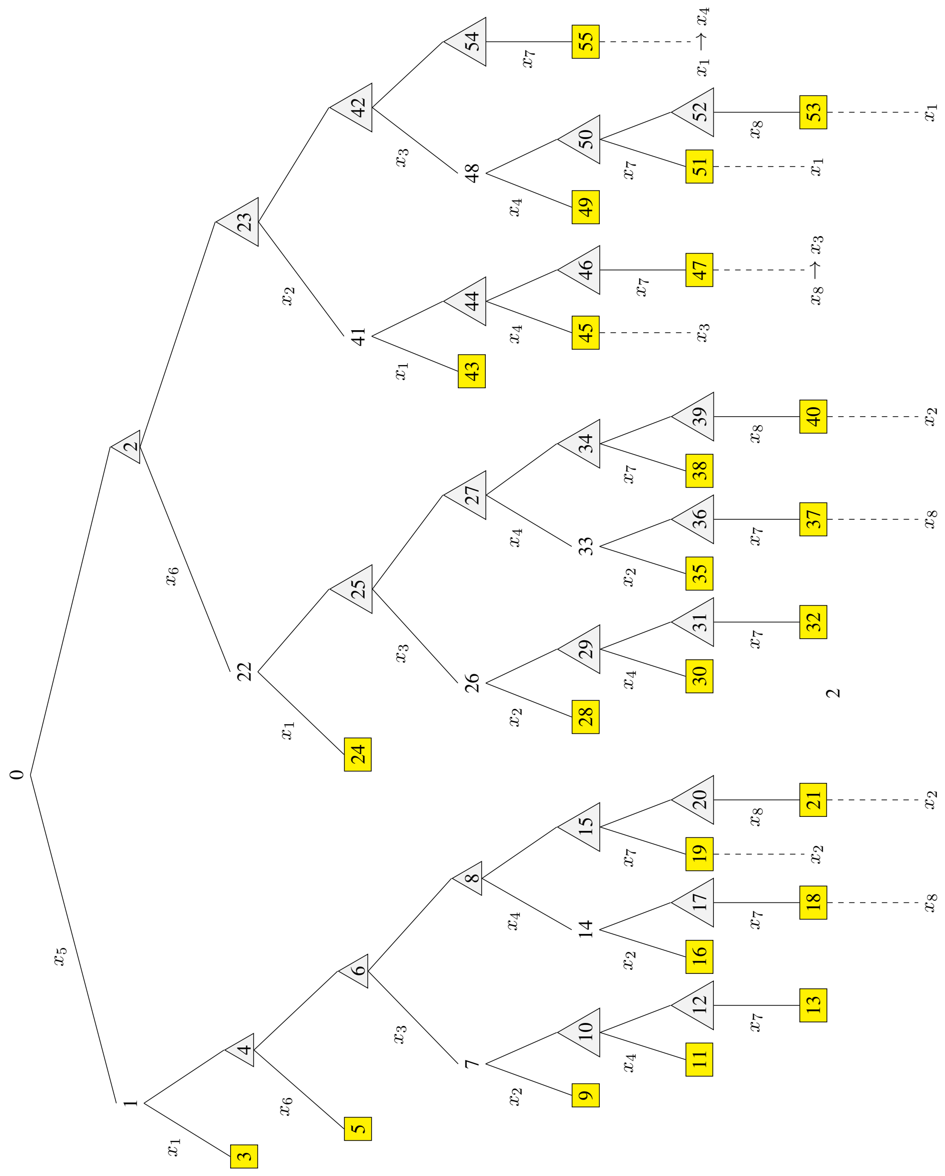


Figure 5: Enumeration tree by binary implementation of IB on the BKP example

Before closing this section, we mention that for general binary programs, in order to guarantee that IB always cuts off the current fractional solution (in each of the genuine subproblems), the search for the solution cover must restricted to variables whose value is strictly less than one in the current LP relaxation.

## 5.2 Ranking variables in the cover

Let us now discuss the role of Step 3 of Algorithm 1, where an arbitrary sequence of the variables in the cover is chosen. Let's first complete the example:

**Example 4.** *Recall that the application of Algorithm 2 returns the sequence* $(3, 2, 7, 5, 6)$ *of (fractional) variables, corresponding to a sequence of critical items in Dantzig's algorithm. Here comes into play the ordering option: ranking such variables by non-increasing weights yields the sequence* $(5, 6, 2, 3, 7)$*, which produces subproblems* $1, 2, 3, 4, 5$ *of Figure 3 as well as the corresponding subproblems* $1, 22, 41, 48, 55$ *in the binary tree of Figure 5. Exploiting such a ranking IB now outperforms standard variable branching. Notice that this occurs even if the optimal solution has been provided as initial incumbent, which nullifies a primary advantage of IB.*

As clearly shown by the example, a critical issue is finding an ordering such that the left-most (least constrained) branches are as strong as possible. This is apparently the real strength of IB—it allows one to focus on choosing a few "strategic" branching variables from a small set to form the left-most nodes, while trusting in the structural "reliability" (and strength) of the right-most branches. Moreover, one can use problem-specific structure to aid in this choice. In Section 6.1 we show that ordering the items by non-increasing weight is effective for some difficult BKP instances.

An important observation here is that the reordering step can produce sequences that may lead to branching on integer variables (in the binary implementation). This is not allowed in traditional variable branching schemes, but our results show that this may actually be profitable. In other words, it is not just that the binary version of IB selects variables to branch in a different order than traditional variable branching. It is in fact the case that *no variable selection rule* employed in a variables branching scheme could produce the binary trees obtained by interdiction branching.

# 6 Computational Experiments

In this section, we analyze of the performance of interdiction branching on two binary combinatorial optimization problems for which, indeed, fixing variables to one is a much stronger condition than fixing them to zero. Namely, we consider the BKP in Section 6.1 and the SSP in Section 6.2.

## 6.1 Binary Knapsack Experiments

It is well-known (see, e.g., [17]) that difficult BKP instances can be randomly generated according to the following rules. A set of $v$ items is generated with weights in $[1, R]$ and profits $p_j = w_j + R/10$. Each item is then normalized by dividing both weight and profit by a value $m + 1$. This set of elements is called *spanner set*: $\mathrm{span}(v, m)$. The $n$ items are then generated by repeatedly choosing an item $k$ from the spanner set and a random value $d$ in $[1, m]$, yielding $(d \cdot p_k, d \cdot w_k)$. Instances of this type are generally refereed to as *strongly correlated spanner instances*, and their difficulty arises from multiple items having the same profit/weight ratio, which might confound the branching process[3].

We set $R = 1000$, $v = \{2, 3\}$ and $m = \{100, 120\}$ and problem size $n = \{50, 60, 70, 80, 90, 100\}$. Capacity $C$ is set as $C := \sum_{j=1}^{N} w_j/2$. For each size, we randomly generated 20 instances. The computations were performed on a 2.8 GHz Intel Quad Core with 24 GB of RAM running Linux. A limit of 2 hours of CPU time was imposed for all experiments.

We compared interdiction branching (implemented in the binary tree version described in the previous section) with CPLEX 12.2. Specifically, CPLEX parameters have been chosen to force the enumeration algorithm to behave as a standard branch-and-bound scheme. The nondefault CPLEX settings are:

`CPX_PARAM_MIPSEARCH=CPX_MIPSEARCH_TRADITIONAL`: traditional branch-and-cut search;

`CPX_PARAM_PREIND=CPX_OFF`: no presolve;

`CPX_PARAM_PRESLVND=-1`: no node presolve. This setting is used to prevent CPLEX from changing the solution of the pseudo-subproblems, so as to avoid the generation of inconsistent covers;

`CPX_PARAM_THREADS=1`: sequential mode.

Moreover, in the first experiment we are disabling cut generation (i.e., `CPX_PARAM_CUTPASS=-1`) to allow a straightforward comparison with respect to 0–1 branching in terms of nodes, as discussed at the end of the previous section. Indeed, in such a case, the fractional variable is unique, thus removing any ambiguity in the implementation of Algorithm 2: the (unique) fractional variable is iteratively fixed to zero until the optimal value of the continuous relaxation is not (anymore) larger than the incumbent value. (CPLEX cuts are then reconsidered before the end of the section.) The variables in the resulting branching set are ranked by non-increasing weight. As anticipated in the previous section, a key issue for the practical effectiveness of IB is to *explicitly* take into account the current (fractional) solution when branching. This corresponds to avoiding inclusion in the branching set of any variable whose value is 1 in the solution to the LP relaxation. Finally, as initial incumbent solution, we use the one computed by CPLEX at the root node.

The aggregated results comparing the standard variable branching by CPLEX (0–1 in the table) and interdiction branching (IB) are reported in Table 1. For each triple $(v, n, m)$ and for each branching scheme (0–1 or IB) we report (*i*) the number of solved instances (in parentheses) and on the same line the average time over instances solved by both methods, and (*ii*) in the line below, the

[3]In our generator we forbid items replication to avoid CPLEX variable aggregations. Instances are available upon request from the authors.

average number of branch-and-bound nodes (again only over the instances solved within the time limit by both schemes). The same information (except for the number of solved instances) is then reported in terms of geometric mean.

| | | $m = 100$ | | | | $m = 120$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | arithmetic mean | | geometric mean | | arithmetic mean | | geometric mean | |
| $v$ | $n$ | 0–1 | IB | 0–1 | IB | 0–1 | IB | 0–1 | IB |
| 2 | 50 | (20) 62.98 | (20) 53.33 | 0.98 | 0.84 | (20) 461.72 | (20) 400.03 | 3.10 | 2.18 |
| | | 1,761,896.95 | 602,378.60 | 607.97 | 240.50 | 18,136,555.70 | 6,064,691.55 | 6,213.88 | 1,935.62 |
| | 60 | (19) 421.00 | (19) 388.93 | 2.65 | 1.96 | (19) 379.79 | (19) 169.37 | 6.17 | 4.76 |
| | | 8,629,773.32 | 4,155,480.05 | 1,587.98 | 711.14 | 14,390,370.89 | 2,408,517.37 | 5,890.59 | 2,903.68 |
| | 70 | (15) 76.11 | (17) 24.11 | 1.24 | 0.98 | (17) 688.23 | (17) 300.94 | 1.80 | 1.52 |
| | | 1,974,794.93 | 229,331.47 | 58.87 | 35.92 | 25,163,713.53 | 3,969,008.53 | 108.51 | 66.11 |
| | 80 | (13) 422.47 | (14) 312.62 | 8.72 | 7.44 | (17) 137.04 | (18) 77.41 | 1.78 | 1.34 |
| | | 11,081,012.15 | 3,142,002.54 | 1,829.19 | 997.03 | 4,972,139.94 | 968,274.35 | 198.06 | 96.14 |
| | 90 | (9) 5.38 | (9) 1.60 | 0.52 | 0.45 | (12) 0.36 | (13) 0.53 | 0.33 | 0.36 |
| | | 129,098.56 | 12,118.44 | 443.78 | 6.87 | 5,770.17 | 3,737.83 | 9.95 | 6.55 |
| | 100 | (11) 113.00 | (11) 20.00 | 0.64 | 0.55 | (13) 2.79 | (15) 2.64 | 0.56 | 0.52 |
| | | 2,668,660.45 | 168,958.91 | 6.11 | 464.39 | 88,137.38 | 28,888.23 | 7.68 | 6.09 |
| 3 | 50 | (19) 0.76 | (19) 0.55 | 0.36 | 0.35 | (20) 2.49 | (20) 1.79 | 0.48 | 0.47 |
| | | 22,715.47 | 5,846.42 | 149.85 | 84.72 | 89,537.25 | 25,462.15 | 177.78 | 69.49 |
| | 60 | (17) 0.64 | (17) 0.40 | 0.36 | 0.34 | (20) 69.30 | (20) 49.45 | 1.35 | 1.28 |
| | | 19,217.47 | 3,961.82 | 245.56 | 118.86 | 2,580,582.25 | 718,594.15 | 2,172.03 | 815.89 |
| | 70 | (16) 60.27 | (16) 34.29 | 1.65 | 1.35 | (20) 864.54 | (20) 482.04 | 39.58 | 31.58 |
| | | 2,163,095.25 | 480,265.19 | 1,753.17 | 731.01 | 28,228,921.40 | 6,746,964.65 | 267,434.32 | 90,363.41 |
| | 80 | (15) 197.87 | (18) 199.67 | 3.60 | 2.72 | (18) 676.07 | (18) 307.61 | 6.05 | 5.57 |
| | | 7,111,671.06 | 2,842,043.17 | 9,815.79 | 3,510.46 | 24,299,781.50 | 4,156,789.22 | 13,440.31 | 5,175.79 |
| | 90 | (15) 925.29 | (16) 555.09 | 5.71 | 4.79 | (17) 844.93 | (17) 704.64 | 30.26 | 25.66 |
| | | 31,585,227.67 | 7,378,931.40 | 37,753.68 | 9,461.52 | 27,628,006.06 | 9,224,236.94 | 71,091.72 | 30,279.05 |
| | 100 | (18) 329.09 | (18) 242.63 | 4.98 | 3.43 | (12) 443.63 | (13) 203.04 | 11.26 | 8.40 |
| | | 11,123,070.73 | 2,911,413.00 | 3,741.34 | 1,223.74 | 15,135,092.75 | 2,483,053.67 | 17,447.84 | 6,116.46 |

Table 1: Comparing variable branching with IB for the BKP.

The results in Table 1 clearly show a very satisfactory behavior for IB as compared to standard variable branching. Specifically, IB is able to solve 12 more instances within the time limit of 2 hours of CPU time and shows very competitive performance both in terms of average CPU time and number of nodes, both in arithmetic and geometric mean.

The performance profiles comparing the number of nodes and the CPU times are reported in Figures 6 and 7, respectively. Specifically, in this first experiment we are comparing the red and green curves as IB and binary branching.

It is interesting to note that since for BKP there is only one fractional variable to branch on, it is clear that if IB is different than binary branching, then it is because IB does not always branch on the fractional variable. Otherwise, the tree would come out exactly the same in both cases.

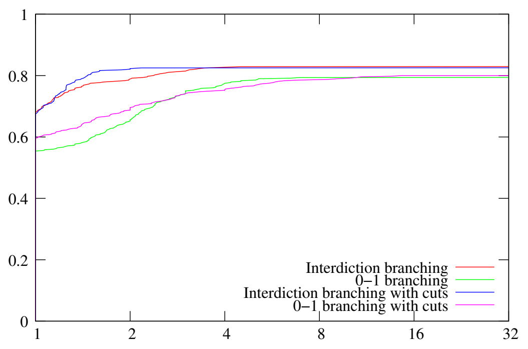


Figure 6: Variable branching vs IB: number of subproblems.

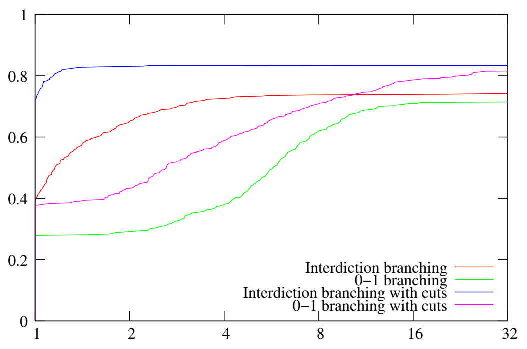


Figure 7: Variable branching vs IB: CPU Times.

Indeed, this is due to the reordering of the variables once the branching set $S$ is computed. Because of such a reordering, it might be (and it is actually the case) IB branches on variables that are *integral* in the current LP relaxation (see, e.g., $x_5$ in Figure 5). Note that for BKP branching on variables at integral value is not strange as in the classical branch-and-bound scheme that considers the variables in a prefixed order (see, e.g., Martello and Toth [14]). However, the difference here is that IB branches on variables at 0 in the current LP relaxation instead of branching on variables at 1 as in the classical scheme. In other words, the power of IB consists in its ability to define a strong (non-binary) disjunction involving variables different from 1 in the current LP relaxation. The experiment with the binary implementation then shows that one cannot reproduce the same disjunction with traditional branching rules (that typically work only on fractional variables).

Finally, for BKP we also evaluated both IB and binary branching when employing cut generation in CPLEX. Because of the addition of cuts, it might be the case here that the LP relaxations have more than one fractional variable. Thus, the implementation of Algorithm 2 has to be further specified to determine which of the fractional variable to be selected to enter at each iteration of the loop that builds the set $S$ (see Section 4.2). In principle, such a decision could have been taken in a very informed way knowing that the problem at hand is a BKP (and this is the case for the entire computation of $S$), but we avoided any tailored choice and simply resorted to selecting the variable, among the fractional ones ($0 < \tilde{x}_j < 1$), with highest pseudocost. We note, however, that even in presence of cuts, the LP relaxation of BKP generally produces very few fractional variables, very often only one.

The aggregated results are reported in Table 2, while performance profiles of both nodes and CPU times are the blue and pink curves in Figures 6 and 7. Both Table 2 and the performance profiles essentially show the same effect described before for the case in which no cutting planes were used. Specifically, both IB and binary branching clearly benefit from the use of CPLEX cutting planes, thus resulting in a larger number of problems solved within the time limit and shorter computing times. However, IB continues to show clear advantages with respect to classical binary branching and the trends of the performance profiles, with the red-green and blue-pink pairs

| | | $m = 100$ | | | | $m = 120$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | arithmetic mean | | geometric mean | | arithmetic mean | | geometric mean | |
| $v$ | $n$ | 0–1 | IB | 0–1 | IB | 0–1 | IB | 0–1 | IB |
| 2 | 50 | (20) 39.41 | (20) 28.83 | 0.94 | 0.80 | (19) 118.65 | (20) 77.49 | 2.24 | 1.61 |
| | | 1,257,841.45 | 360,106.85 | 396.96 | 145.51 | 3,705,276.47 | 971,255.95 | 2,481.64 | 763.41 |
| | 60 | (19) 252.77 | (19) 197.75 | 2.18 | 1.58 | (19) 403.16 | (19) 159.47 | 7.50 | 4.82 |
| | | 6,654,547.21 | 2,377,041.68 | 926.16 | 390.41 | 12,056,479.58 | 1,897,376.26 | 3,987.08 | 1,473.12 |
| | 70 | (16) 531.98 | (17) 367.60 | 2.31 | 1.66 | (17) 678.88 | (17) 288.35 | 1.83 | 1.50 |
| | | 16,104,171.69 | 4,321,404.50 | 119.99 | 64.85 | 20,126,021.82 | 3,247,175.41 | 94.44 | 55.03 |
| | 80 | (13) 154.85 | (14) 125.08 | 5.26 | 4.72 | (17) 308.04 | (18) 89.41 | 2.16 | 1.38 |
| | | 4,552,826.54 | 1,395,661.54 | 1,174.92 | 672.48 | 9,309,853.12 | 966,178.47 | 184.31 | 86.39 |
| | 90 | (9) 5.38 | (9) 1.60 | 0.52 | 0.45 | (12) 0.30 | (12) 0.30 | 0.30 | 0.30 |
| | | 129,098.56 | 12,118.44 | 443.78 | 6.87 | 1,131.08 | 676.33 | 3.18 | 2.82 |
| | 100 | (11) 113.00 | (11) 20.00 | 0.64 | 0.55 | (13) 3.95 | (13) 1.64 | 0.55 | 0.50 |
| | | 2,668,660.45 | 168,958.91 | 6.11 | 464.39 | 100,859.31 | 14,329.77 | 7.36 | 5.66 |
| 3 | 50 | (19) 0.39 | (19) 0.39 | 0.33 | 0.33 | (20) 2.37 | (20) 1.99 | 0.44 | 0.47 |
| | | 4,496.63 | 1,757.11 | 61.30 | 35.53 | 34,418.71 | 23,336.35 | 50.67 | 28.48 |
| | 60 | (17) 0.52 | (17) 0.46 | 0.35 | 0.34 | (20) 68.45 | (20) 35.20 | 1.28 | 1.27 |
| | | 11,270.29 | 2,845.82 | 100.72 | 50.90 | 2,075,146.30 | 413,624.00 | 891.04 | 437.45 |
| | 70 | (16) 66.46 | (16) 35.27 | 1.80 | 1.45 | (19) 682.41 | (20) 427.78 | 31.17 | 25.00 |
| | | 1,964,402.38 | 400,697.94 | 767.96 | 334.25 | 17,056,587.16 | 4,952,817.16 | 116,284.75 | 36,971.57 |
| | 80 | (18) 97.32 | (18) 99.32 | 3.44 | 2.74 | (17) 377.66 | (18) 252.76 | 3.83 | 3.87 |
| | | 2,799,942.61 | 1,074,442.56 | 3,801.27 | 1,700.36 | 11,321,562.00 | 2,756,197.59 | 3,880.31 | 1,679.92 |
| | 90 | (15) 647.10 | (16) 488.76 | 7.19 | 4.41 | (17) 799.93 | (17) 583.11 | 30.73 | 23.71 |
| | | 18,198,739.47 | 5,170,702.40 | 39,832.30 | 7,517.93 | 21,841,217.59 | 6,113,590.65 | 42,675.89 | 16,982.53 |
| | 100 | (18) 1,223.36 | (18) 750.19 | 17.92 | 11.47 | (13) 528.19 | (13) 368.81 | 16.11 | 12.47 |
| | | 34,983,410.33 | 7,476,192.89 | 13,284.01 | 4,588.13 | 14,847,604.69 | 3,674,453.31 | 29,246.47 | 9,797.45 |

Table 2: Comparing variable branching with IB for the BKP **with CPLEX cuts**.

of curves being highly similar.

## 6.2 Stable Set Experiments

Let $G = (V, E)$ be an undirected graph with $|V|$ vertices. A vertex subset $W \subseteq V$ is called *stable* if no two elements of $W$ are adjacent. The *stability number* $\alpha(G)$ of $G$ is the size of a maximum cardinality stable set of $G$. The SSP consists in computing a stable set of maximum cardinality. It is well known that a set of *clique inequalities* [16] associated with any set of cliques covering all edges of $G$ provide a valid formulation for the SSP. In our experience, we consider a clique-cover of $G$ generated by a greedy heuristic, as in [18]. The testbed consists of the 18 graphs with $|V| < 400$ from the DIMACS Second Challenge [8], available at the web site [5]. Details on the instances are given in Table 3. The computations were run on a machine equipped by 2 Intel Xeon 5150 processors clocked at 2.6 GHz and having 8MB RAM with CPLEX 12.2. CPLEX settings are identical to those described in the previous section.

| Graph | $\lvert V \rvert$ | $\lvert E \rvert$ | $\alpha(G)$ | LP relaxation | # of rows (clique ineq.) |
|---|---|---|---|---|---|
| brock200_1 | 200 | 5,066 | 21 | 39.58 | 1,637 |
| brock200_2 | 200 | 10,024 | 12 | 23.78 | 1,422 |
| brock200_3 | 200 | 7,852 | 15 | 29.74 | 1,626 |
| brock200_4 | 200 | 6,811 | 17 | 32.81 | 1,668 |
| C125.9 | 250 | 3,141 | 34 | 43.06 | 481 |
| c-fat200-5 | 200 | 11,427 | 58 | 66.67 | 7,561 |
| DSJC125.1 | 125 | 736 | 34 | 43.23 | 456 |
| DSJC125.5 | 125 | 3,891 | 10 | 17.05 | 643 |
| mann_a9 | 378 | 702 | 16 | 18.00 | 48 |
| mann_a27 | 1,035 | 1,980 | 126 | 135.00 | 468 |
| keller4 | 171 | 5,100 | 11 | 15.00 | 511 |
| p_hat300-1 | 300 | 33,917 | 8 | 17.71 | 1,124 |
| p_hat300-2 | 300 | 22,922 | 25 | 36.51 | 2,017 |
| p_hat300-3 | 300 | 11,460 | 36 | 57.72 | 3,147 |
| san200_0.7_2 | 200 | 5,970 | 18 | 22.19 | 990 |
| san200_0.9_3 | 200 | 1,990 | 44 | 47.17 | 1,136 |
| sanr200_0.7 | 200 | 6,032 | 18 | 35.75 | 1,671 |
| sanr200_0.9 | 200 | 2,037 | 42 | 60.20 | 1,130 |

Table 3: DIMACS graphs: instances description and formulations data.

In the SSP experiments instead of using Algorithm 2 to (heuristically) solve the IBLP problems, we used the results of Section 4.3, to reformulate IBLP as an MIP and then used CPLEX itself to provide a heuristic solution. Specifically, the objective function of the IBLP evaluated at the root node is $\min \sum y_i$ (see, Section 4.1), while in the other subproblems, the objective coefficients (instead of all 1s) are set to the negation of the pseudocosts evaluated by CPLEX (a variable with high pseudocost is preferred to be in the branching set). When building the IBLP one can exploit the fact that a variable (vertex) $k$ fixed to one implies that all variables in its neighborhood (in $G$) are fixed to zero. This results in smaller IBLP problems and produces a considerable speed-up.

The reason for not using Algorithm 2 in the SSP context is that, as opposed to BKP, the solution of each LP relaxation is time consuming and, in general, not very informative due to high amount of symmetry of the classical formulation. Thus, formulating IBLP as a MIP allows better exploitation of preprocessing techniques, fast enumeration and heuristics that result in branching sets of smaller size in competitive computing times with respect to those obtained by a straightforward implementation of Algorithm 2. Note that the power of ingredients like preprocessing, fast enumeration and heuristics also leads to (slightly) smaller branching sets in the BKP case but at a much higher price in terms of computing time because of the negligible amount of time needed to solve the LP relaxation (even by using the simplex algorithm instead of Dantzig's procedure).

Again for SSP, we used as initial incumbent solution the one provided by CPLEX at the root node. Concerning cutting plane generation, we performed experiments with and without CPLEX cuts without noticing appreciable differences in the trends. This was somehow expected because clique constraints, which would be the powerful ones in the CPLEX arsenal for SSP, do not play a fundamental role while adopting the so-called "intermediate" formulation with a clique-cover of $G$ greedily generated. Thus, for the sake of conciseness and uniformity with respect to the initial configuration used for BKP, the experiments reported below are performed *without* CPLEX cuts activated.

CPLEX parameters for the ILBP solution are chosen so as to balance the tradeoff between solution quality (size of the branching set) and CPU time. The best (nondefault) settings found were:

`CPX_PARAM_THREADS=4`: parallel mode with 4 cores;

`CPX_PARAM_MIPEMPHASIS=CPX_MIPEMPHASIS_FEASIBILITY`;

`CPX_PARAM_NODELIM=1000`: node limit at the root node;

`CPX_PARAM_NODELIM=10`: node limit at subproblems different from root.

Last, but not least, the branching set is sorted in nonincreasing order of the vertex degree in the graph $G$.

The computational results are presented in Table 4, where the variable and IB branching schemes are compared in terms of computing time (Time in the table, expressed in CPU seconds), number of nodes (# Subproblems) and time for node (Time per sub, expressed in milliseconds of CPU). In addition, for IB, Table 4 reports the amount of time spent to solve IBLPs (IBLP, expressed in CPU seconds), the number of "genuine" decision nodes (# Genuine) and the number of fixings (# IB Fixings).

The results presented in Table 4 are quite satisfactory. Interdiction branching is faster than variable branching in 10 of the 18 of the instances, sometimes much faster, especially for harder instances. Because of the use of the MIP reformulation of IBLP and of CPLEX for solving it, it is hard(er) to compare the two approaches in terms of tree size. On the one hand, some enumeration is clearly done solving the IBLPs and that is not anymore tightly approximated by the overall number of binary nodes as was the case for BKP (see the discussion at the end of Section 5.1). On the other hand, the number of "genuine" subproblems gives a tight evaluation of the decision points that the enumeration scheme based on IB needs to face. That number is always much smaller than the number of branching nodes of a traditional variable branching approach and the general reduction in terms of computing time shows that spending some more effort to take more sophisticated decisions tends to pay off.

| Graph | 0-1 branching | | | Interdiction Branching | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Time | Subproblems | Time per sub | Time | IBLP | Subproblems | Genuine | IB fixings | Time per sub |
| name | (sec) | (#) | (msec) | (sec) | (sec) | (#) | (#) | (#) | (msec) |
| brock200_1 | 1,551.96 | 370,372 | 4.19 | **1,028.37** | 547.91 | **104,795** | 57,603 | 5,464 | 9.81 |
| brock200_2 | 81.50 | 11,769 | 6.92 | **63.20** | 21.41 | **6,523** | 3,441 | 181 | 9.69 |
| brock200_3 | 213.70 | 33,142 | 6.45 | **130.35** | 55.03 | **10,663** | 5,973 | 271 | 12.22 |
| brock200_4 | 475.79 | 104,572 | 4.55 | **308.49** | 136.27 | **35,079** | 19,150 | 827 | 8.79 |
| C125.9 | **5.19** | 6,093 | 0.85 | 33.61 | 28.25 | **3,070** | 1,754 | 496 | 10.95 |
| c-fat200-5 | **6.30** | **47** | 134.04 | 8.02 | 0.45 | 51 | 27 | 0 | 157.25 |
| DSJC125.1 | **3.44** | 4,259 | 0.81 | 40.13 | 36.35 | **2,792** | 1,505 | 747 | 14.37 |
| DSJC125.5 | **4.34** | 1,307 | 3.32 | 4.88 | 2.32 | **760** | 417 | 13 | 6.42 |
| mann_a9 | **0.01** | 5 | 2.00 | 0.02 | 0.01 | **3** | 3 | 0 | 6.67 |
| mann_a27 | **3.32** | 10,755 | 0.31 | 44.82 | 42.67 | **1,975** | 1,176 | 416 | 22.69 |
| keller4 | **16.91** | 6,429 | 2.63 | 18.44 | 13.10 | **1,701** | 955 | 168 | 10.84 |
| p_hat300-1 | 56.05 | 5,198 | 10.78 | **43.25** | 14.00 | **5,074** | 2,674 | 27 | 8.52 |
| p_hat300-2 | 167.59 | 8,576 | 19.54 | **117.24** | 51.03 | **3,924** | 2,064 | 228 | 29.88 |
| p_hat300-3 | 22,449.20 | 1,104,172 | 20.33 | **6,307.62** | 2937.63 | **211,749** | 116,737 | 18,125 | 29.79 |
| san200_0.7_2 | 4.42 | 822 | 5.38 | **4.00** | 1.85 | **321** | 176 | 7 | 12.46 |
| san200_0.9_3 | **2.88** | **595** | 4.84 | 13.49 | 9.73 | 1,247 | 744 | 55 | 10.82 |
| sanr200_0.7 | 1,067.42 | 203,077 | 5.26 | **626.92** | 307.15 | **66,509** | 36,429 | 2,921 | 9.43 |
| sanr200_0.9 | 6,223.24 | 2,054,850 | 3.03 | **5,318.50** | 4,295.58 | **351,634** | 203,835 | 41,772 | 15.13 |

Table 4: Comparing variable branching with IB for the SSP.

## 7 Conclusions

We proposed an $n$-ary branching method for binary programs designed to overcome the difficulties often encountered with standard branching on variables. This is based on looking at the branching decision as a bilevel problem in which the current incumbent solution is taken into account in the attempt of only targeting improving tree directions.

We have shown that it is not necessary to solve the bilevel programming problem neither to optimality nor even directly, in the sense that one can solve heuristically relaxed versions of it. Finally, we have shown that the $n$-ary branching scheme can be efficiently implemented within a binary tree, thus making the approach compatible for most of the commercial and noncommercial MIP solvers.

We gave evidence through a computational study on difficult BKP and SSP instances from the literature that such a method helps reducing significantly the size of the search tree with respect to variable branching, and, in general, the running time.

Interdiction branching can be incorporated within any branch-and-cut framework, even in a hybrid manner, i.e., interleaved with standard variable branching. We believe that the framework could still be improved by good primal heuristics or cutting planes, and the extension of the method to general MIPs is the subject of continuing research.

## References

[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.

[2] O. Ben-Ayed and C. Blair. Computational difficulties of bilevel linear programming. *Operations Research*, 38:556–560, 1990.

[3] P. Calamai and L. Vicente. Generating quadratic bilevel programming problems. *ACM Transactions on Mathematical Software*, 20:103–119, 1994.

[4] S.T. DeNegre and T.K. Ralphs. A Branch-and-Cut Algorithm for Bilevel Integer Programming. In *Proceedings of the Eleventh INFORMS Computing Society Meeting*, pages 65–78, 2009.

[5] DIMACS. `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique`.

[6] P. Hansen, B. Jaumard, and G. Savard. New branch-and-bound rules for linear bilevel programming. *SIAM Journal on Scientific and Statistical Computing*, 13:1194–1217, 1992.

[7] R. Jeroslow. The polynomial hierarchy and a simple model for competitive analysis. *Mathematical Programming*, 32:146–164, 1985.

[8] D.S. Johnson and M.A. Trick (Eds.). *Cliques, Coloring and Satisfiability: the 2nd DIMACS Implementation Challenge*. American Mathematical Society, Providence, RI, 1996.

[9] M. Karamanov and G. Cornuéjols. Branching on general disjunctions. Working Paper, 2007.

[10] J.T. Linderoth and M.W.P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11:173–187, 1999.

[11] Z-Q. Luo, J-S. Pang, and D. Ralph. *Mathematical Programs with Equilibrium Constraints*. Cambridge University Press, 1996.

[12] A. Mahajan and T.K. Ralphs. Experiments with branching using general disjunctions. In *The Proceedings of the Eleventh INFORMS Computing Society Meeting*, pages 101–118, 2009.

[13] A. Mahajan and T.K. Ralphs. On the complexity of selecting disjunctions in integer programming. *SIAM Journal on Optimization*, 20:2181–2198, 2010.

[14] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, Chichester, 1990.

[15] J.T. Moore and J.F. Bard. The mixed integer linear bilevel programming problem. *Operations Research*, 38(5):911–921, 1990.

[16] M.W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[17] D. Pisinger. Where are the hard knapsack problems? *Comput. Oper. Res.*, 32:2271–2284, 2005.

[18] F. Rossi and S. Smriglio. A branch-and-cut algorithm for the maximum cardinality stable set problem. *Operations Research Letters*, 28:63–74, 2001.