Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the manuscript number!)

Multi-Variable Branching: A 0-1 Knapsack Problem Case Study

Yu Yang, Natashia Boland, Martin Savelsbergh
H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332, yangyu@gatech.edu, natashia.boland@gmail.com, martin.savelsbergh@isye.gatech.edu,

We explore the benefits of multi-variable branching schemes for linear programming based branch and bound algorithms for the 0-1 knapsack problem, i.e., the benefits of branching on sets of variables rather than on a single variable (the current default in integer programming solvers). We present examples where multi-variable branching has advantages over single-variable branching, and partially characterize situations in which this happens. Chvátal (1980) shows that for a specific class of 0-1 knapsack instances a linear programming based branch-and-bound algorithm (employing a single-variable branching scheme) must explore exponentially many nodes. We show that for this class of 0-1 knapsack instances a linear programming based branch-and-bound algorithm employing an appropriately chosen multi-variable branching scheme explores either three or seven nodes. Finally, we investigate the performance of various multi-variable branching schemes for 0-1 knapsack instances computationally and demonstrate their potential; the multi-variable branching schemes explored result in smaller search trees (some in search trees that are an order of magnitude smaller) and some also results in shorter solution times.

Key words: 0-1 knapsack problem, branch and bound, multi-variable branching, branching scheme

1. Introduction

Branch and bound, first proposed by Land and Doig (1960), has grown into a standard framework for solving discrete optimization problems. Using bounds on the optimal solution value to avoid searching parts of the solution space that cannot contain an optimal solution, branch and bound can be far more effective than exhaustive search. We focus on linear programming (LP) based branch and bound for solving integer programs, which is the algorithmic framework at the heart of all commercial, and most open source, software for solving integer programs. As dual bounds at the nodes in the search tree are obtained by solving the LP relaxation of the integer program at the nodes, the branching scheme employed has a significant impact on the effectiveness of the algorithm. The typical branching scheme identifies a variable x_i with fractional value f in the solution to the LP relaxation and creates two branches (i.e., two new nodes in the search tree) one in which the constraint $x_i \geq \lceil f \rceil$ is imposed and one in which the constraint $x_i \leq \lfloor f \rfloor$ is imposed, i.e., branching is based on a *single-variable* dichotomy. As a result, most research on branching schemes (in the context of LP based branch and bound) has focused on the identification of the variable to branch on, e.g., pseudo-cost branching (Bénichou et al. 1971), strong branching (Applegate et al. 1995), and reliability branching (Achterberg et al. 2005). Branching based on sets of variables, the focus of our research, has only been considered in very special cases, e.g., *special ordered set* branching (see Beale and Tomlin (1970) and Beale (1979)). We note that in their recent survey, Morrison et al. (2016) list multi-variable branching schemes as one of the under-explored areas of research that deserves further investigation.

A multi-variable branching scheme is a natural generalization of a single-variable branching scheme. Rather than branching on a single variable with a fractional value in the solution to the LP relaxation, branching is performed on a set of variables such that the sum of their values in the solution to the LP relaxation is fractional. More specifically, a set S of variables with $\sum_{i \in S} x_i = f$ with f fractional is identified and two branches are created, one in which the constraint $\sum_{i \in S} x_i \geq \lceil f \rceil$ is imposed and one in which the constraint $\sum_{i \in S} x_i \leq \lfloor f \rfloor$ is imposed. In this case, of course, the question becomes how to choose the set S.

The "desirability" of branching on a particular variable in a single-variable branching scheme has historically been defined by the value $\max\{z^-, z^+\}$, in case of a maximization problem, where z^- and z^+ represent the optimal value of the LP relaxation on the "down" branch and the "up" branch of that variable, respectively, and where a smaller value of $\max\{z^-, z^+\}$ is more desirable. The same definition can be used to define the desirability of branching on a particular set of variables in a multi-variable branching scheme. We note that evaluating the desirability of every fractional variable, in the context of a single-variable branching scheme, has been shown to lead to small search trees, but not necessarily to shorter solution time, as the evaluation of desirability is expensive (it involves solving two LPs for each evaluation). This highlights one of the major challenges for the use of multi-variable branching schemes: evaluating the desirability of every possible set of variables such that the sum of their values in the solution to the LP relaxation is fractional will be computationally prohibitive.

To explore the benefit of multi-variable branching schemes, we focus on the 0-1 knapsack problem, i.e., on the integer program of the following form:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq b, \\ & x_i \in \{0,1\}, \quad \forall\, i \in [n]. \end{aligned}$$

where $[n] = \{1, \ldots, n\}$ and $b \in \mathbb{Z}_{>0}$, $p_i, w_i \in \mathbb{Z}_{>0}$ and $w_i \leq b$ for all $i \in [n]$.

Chvátal (1980), using a class of 0-1 knapsack instances, demonstrates that branch and bound algorithms employing a single-variable branching scheme may have to explore exponentially many nodes. One of our contributions is to show that there is a branch and bound algorithm employing a multi-variable branching scheme that can solve any instance in this class by exploring either three or seven nodes. That some instances in this class can be solved easily by a multi-variable branching scheme was known, as it follows from investigations of basis reduction methods for decomposable knapsack problems (Krishnamoorthy and Pataki 2009), but we are the first to show that *all* instances in this class can be solved easily by a multi-variable branching scheme. Another contribution is that we present and partially characterize situations in which multi-variable branching schemes lead to better dual bounds, i.e., smaller values of $\max\{z^-, z^+\}$, which should lead to smaller search trees. Finally, we demonstrate the potential benefits of multi-variable branching schemes in an extensive computational study in which we assess their relative performance, where our primary measure for the quality of a branching scheme is the number of nodes explored in the search tree, but, at the same time, we seek branching schemes that are computationally efficient.

For completeness and to establish notation, we include a high-level description of the LP based branch and bound algorithm for maximizing a 0-1 integer program used in our research in Algorithm 1, where $\mathcal{L}$ denotes the list of active 0-1 integer programs, x_{best} denotes the incumbent with objective value z_{best}, and z_i^{UP} an upper bound on the value of the 0-1 integer program P_i at node i.

Algorithm 1: LP based based branch and bound algorithm.

```
Initialization : $\mathcal{L} = \{$original 0-1 integer program$\}$ and $z_{best} = -\infty$.
while $\mathcal{L} \neq \emptyset$ do
    Select and remove a 0-1 integer program $P_i$ from $\mathcal{L}$.
    Solve the LP relaxation of $P_i$.
    if the LP relaxation is feasible then
        Retrieve the LP solution $x_i^{LP}$ and its objective value $z_i^{LP}$.
        if $z_i^{LP} > z_{best}$ then
            if $x_i^{LP}$ is integer feasible then
                $x_{best} \leftarrow x_i^{LP}$ and $z_{best} \leftarrow z_i^{LP}$
                Remove 0-1 integer programs $P_j$ from $\mathcal{L}$ with $z_j^{UB} \leq z_{best}$.
            else
                Apply a branching strategy to create two new 0-1 integer programs $P_{i,1}$ and
                  $P_{i,2}$, set $z_{i,1}^{UP} = z_{i,2}^{UP} = z_i^{LP}$, and add $P_{i,1}$ and $P_{i,2}$ to $\mathcal{L}$.
```

The remainder of the paper is organized as follows. In Section 2, we review relevant literature on branching paradigms. In Section 3, we consider a class of difficult 0-1 knapsack problems used in (Chvátal 1980). In Section 4, we give examples showing that branching on sets of variables can be better than branching on a single variable, and derive conditions for branching on sets of two variables to be better than branching on a single variable. In Section 5, we report the results of a number of numerical experiments which further reinforce that branching on sets of variables holds promise. Finally, in Section 6, we provide some concluding remarks.

2. Relevant Literature

2.1. Single variable selection

Over the years, much effort has been devoted to finding schemes for selecting a (single) variable to branch on that result in small search trees. The simplest scheme is to select the variable x_i with fractional part $f - \lfloor f \rfloor$ closest to 0.5. Let z be the value of the LP solution. The intuition (or hope) is that enforcing $x_i \leq \lfloor f \rfloor$ and enforcing $x_i \geq \lceil f \rceil$ result in noticeable changes to the value of the linear program solution, and, therefore, we will have $\max\{z^-, z^+\}$ noticeably less than z, which improves the dual bound. The computational requirements of this naive method are minimal, but, unfortunately, studies have shown that it is not much better than randomly selecting a variable (Achterberg et al. 2008).

Rather than hoping that $\max\{z^-, z^+\}$ is noticeably less than z, pseudo-cost branching (PB) (Bénichou et al. 1971) observes and records actual changes to the values of the LP relaxations and maintains for each variable x_i an average change and uses these values to choose a variable to branch on. The major weakness of PB is that initially little or no information is available and initial branching decisions, at the top of the search, are critically important. However, PB is computationally cheap, and computational studies have shown that it performs reasonably well.

Instead of hoping or observing and recording, strong branching (SB) (Applegate et al. 1995) goes ahead and computes the change in dual bound for each variable x_i with a fractional value, i.e., it computes the value z_i^- at the "down" child where $x_i \leq \lfloor f_i \rfloor$ is imposed and the value z_i^+ at the "up" child where $x_i \geq \lceil f_i \rceil$ is imposed. SB then chooses the variable x_i for which $\max\{z_i^-, z_i^+\}$ is the smallest. SB results in small search trees, but, unfortunately, is computationally intensive and the benefits of smaller search trees are often undone by the effort required to select the variable to branch on.

Hybrid strong / pseudo-cost branching tries to overcome the weaknesses of the two methods by applying SB to nodes high in the search tree (i.e., depth up to d) and switching to PB for nodes low in the search tree (i.e., depth more than d). Since d is typically chosen to be relatively small, fractional variables without a pseudo-cost may still be encountered at depth more than d, in which case SB is used to initialize the pseudo-cost (Gauthier and Ribière 1977, Linderoth and Savelsbergh 1999). These ideas have been taken one step further in reliability branching (RB) (Achterberg et al. 2005) which uses SB to initialize variables without a pseudo-cost or to re-initialize variables with an unreliable pseudo-cost. Extensive computational experiments have shown that RB performs the best among all aforementioned methods in terms of solution time. Furthermore, several generalizations and alternatives to the max scoring function for computing SB scores have been considered. For example, the linear scoring rule (Linderoth and Savelsbergh 1999) generalizes SB: it chooses the variable x_i for which $(1-\mu)\max\{z-z_i^-, z-z_i^+\} + \mu\min\{z-z_i^-, z-z_i^+\}$ is the largest, where $\mu \in [0,1]$ is a parameter ($\mu = 1$ gives SB). An alternative, which performs well in practice, is the product scoring rule (Achterberg 2007), in which the variable x_i chosen is the one for which the product $(\max\{z-z_i^-, \epsilon\}) \cdot (\max\{z-z_i^+, \epsilon\})$ is the largest, where ϵ is a small positive tolerance. Other rules are discussed in Balcan et al. (2018), which also provides theoretical and computational insights into the challenges of designing branching rules.

Instead of deciding candidate branching variables purely from one optimal solution of the current LP relaxation, Berthold and Salvagnin (2013) take advantage of the multiple alternative optimal solutions and potentially avoid solving unpromising LPs in strong branching. Achterberg and Berthold (2009) combine reliability branching and criteria from conflict analysis widely used in satisfiability problems (SAT) into hybrid branching which solves standard integer programs faster than reliability branching.

The recent success of machine learning has inspired new ways to guide the selection of the branching variable. Di Liberto et al. (2016) introduce DASH (Dynamic Approach for Switching Heuristics) which chooses a branching scheme from a collection of branching schemes to be applied at a node in the search tree based on the features of the integer program at that node. The learning is offline, i.e., it is based on the performance of the branching schemes on a set of training instances. Marcos Alvarez et al. (2017) seek to use machine learning to derive a branching scheme that mimics strong branching in the hope of achieving small search trees without the need to spend huge amounts of computing to select a variable to branch on. Again, the learning is offline. An online learning variant enhanced by a reliability mechanism is proposed in Marcos Alvarez et al. (2016). Khalil et al. (2016) also employ online learning to mimic SB. The novelty of their approach lies in that the learning involves solving a ranking problem rather than using regression or classification. The learning based branching methods do not yet outperform state-of-the-art implementations of RB, but are able to achieve better results than PB or hybrid SB / PB. For a recent survey and more in-depth discussion of learning and branching for integer programming, see Lodi and Zarpellon (2017). Finally, Balcan et al. (2018) show how to learn near-optimal parameters in a convex combination of scoring functions from a sample of instances drawn from a distribution over an instance class and provide complexity guarantees on the number of samples needed.

2.2. Multi-variable branching

Two types of special ordered sets (SOS) were introduced in Beale and Tomlin (1970) for modeling nonconvex problems. Special order sets of type 1 (SOS1), in which at most one element from an ordered set can have a non-zero value, can be used to model multiple choice problems, while special ordered sets of type 2 (SOS2), in which up to two consecutive elements from an order set can have non-zero values, can be used when a nonlinear function is approximated by a piecewise linear function. Multi-variable branching can be readily applied to variables appearing in special ordered sets. For example, let x_i for $i \in \{1, \dots, n\}$ be the variables in a special ordered set of type 1, and let $x_0 = x_1 = \dots x_{k-1} = 0$, $x_k = x_{k+1} = 0.5$, and $x_{k+2} = \dots, x_n = 0$, then we can branch by imposing $x_i = 0$ for $i = 1, \dots, k$ on one branch and imposing $x_i = 0$ for $i = k+1, \dots, n$ on the other branch. Similar branching ideas can be developed for special ordered sets of type 2. Beyond the well-known SOS1 and SOS2, Beale (1979) also introduced Linked Ordered Sets (LOS), which handles sums and products of functions of nonlinear variables in either the coefficients or the right hand sides of an IP. We note that these multi-variable branching rules target special structures occurring in certain types of optimization problems.

Basis reduction methods, following the groundbreaking work of Lenstra, Jr. (1983), also give rise to multi-variable branching, often referred to as *hyperplane branching* in that context. (For an introduction to basis reduction methods, see Pataki and Tural (2011), and for an overview of "non-standard" methods for integer programming, see Aardal et al. (2002).) Krishnamoorthy and Pataki (2009) establish the potential of hyperplane branching for decomposable knapsack problems (DKPs). They show that for certain classes of 0-1 knapsack instances a feasibility version can be generated and demonstrate how the infeasibility of this feasibility version can be proven by branching on hyperplanes. Obtaining these hyperplanes, however, involves basis reduction computations which are time-consuming (in spite of their polynomial complexity for fixed dimension). One of the classes of 0-1 knapsack instances considered by Krishnamoorthy and Pataki (2009) are the Chvátal instances we study in Section 3. They show that a feasibility version can be generated for instances with an odd number of variables and that therefore infeasibility of these instances can be proven instantly via branching on hyperplanes. Unfortunately, a feasibility version cannot be generated for the instances with an even number of nodes, which are the more interesting instances, and, therefore, their difficulty cannot be assessed using the Krishnamoorthy and Pataki (2009) framework. Using a traditional, but multi-variable branching perspective, we show how to choose a set S of variables to branch on that will solve all instances (regardless of whether the number of variables is odd or even) evaluating only a few nodes.

3. Chvátal Instances and Multi-Variable Branching

First, we demonstrate the potential of multi-variable branching on a class of 0-1 knapsack problem instances analyzed in Chvátal (1980). The class of instances is characterized by the following formulation:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} a_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_i x_i \leq \left\lfloor \sum_{i=1}^{n} a_i/2 \right\rfloor, \\ & x_i \in \{0,1\}, \quad i \in \{1,\dots,n\}, \end{aligned}$$

where

- $\sum_{i\in I} a_i \leq \sum_{i=1}^n a_i/2$ for all sets $I \subseteq \{1,\dots,n\}$ with $|I| \leq n/10$,
- every integer greater than one divides fewer than $9n/10$ of the integers a_i,
- $\sum_{i\in I} a_i \neq \sum_{j\in J} a_j$, when $I \neq J$, and
- there is no set I such that $\sum_{i\in I} a_i = \lfloor \sum_{i=1}^n a_i/2 \rfloor$.

Chvátal (1980) shows that for this class of 0-1 knapsack instances, a branch and bound algorithm employing a singe-variable branching scheme must explore at least $2^{n/10}$ nodes.

Chvátal (1980) also considers a "specific example" with $a_j = 2^{k+n+1} + 2^{k+j} + 1$ for $j = 1,\dots,n$ and with $k = \lfloor \log(2n) \rfloor$. For this class of 0-1 knapsack instances, it can be shown that a branch and bound algorithm employing a singe-variable branching scheme must explore at least $2^{(n-1)/2}$ nodes. We will show, for this class of instances, that a branch and bound algorithm employing an appropriately chosen multi-variable branching scheme solves instances by evaluating at most 7 nodes. We start by observing some properties of the instance data and showing that each instance has a unique optimal solution.

Lemma 1. *When n is odd, the sum of the largest $\frac{n-1}{2}$ coefficients in the "specific example" is less than b, while the sum of the smallest $\frac{n+1}{2}$ coefficients exceeds b:*

$$\sum_{j=\frac{n+1}{2}+1}^{n} a_j < b \qquad \textit{and} \qquad \sum_{j=1}^{\frac{n+1}{2}} a_j > b. \tag{3.1}$$

Proof. For n odd,

$$b := \left\lfloor \sum_{i=1}^{n} a_i/2 \right\rfloor = \left\lfloor \sum_{j=1}^{n} \left(2^{k+n} + 2^{k+j-1} + 1/2\right) \right\rfloor = (n+1)2^{k+n} - 2^k + \frac{n-1}{2}.$$

Summing the largest $\frac{n-1}{2}$ elements, we have

$$\sum_{j=\frac{n+1}{2}+1}^{n} a_j = \sum_{j=\frac{n+1}{2}+1}^{n} \left(2^{k+n+1} + 2^{k+j} + 1\right) = (n+1)2^{k+n} - 2^{k+\frac{n+1}{2}+1} + \frac{n-1}{2} < b. \tag{3.2}$$

Summing the smallest $\frac{n+1}{2}$ elements, we have

$$\sum_{j=1}^{\frac{n+1}{2}} a_j = \sum_{j=1}^{\frac{n+1}{2}} \left(2^{k+n+1} + 2^{k+j} + 1\right) = (n+1)2^{k+n} + 2^{k+\frac{n+1}{2}+1} - 2^{k+1} + \frac{n+1}{2} > b. \tag{3.3}$$

□

Corollary 1 *When n is odd, the inequality*

$$\sum_{j=1}^{n} x_j < \frac{n+1}{2}$$

is valid for the LP relaxation of the "specific example".

Proof. This is obvious from the second part of Lemma 1. □

Corollary 2 *When n is odd, the optimal solution to an instance in the "specific example" class is:*

$$x^* = (\overbrace{0,\ldots,0}^{\frac{n+1}{2}},\overbrace{1,\ldots,1}^{\frac{n-1}{2}}).$$

Proof. By Corollary 1 and integrality, an optimal solution can have at most $\frac{n+1}{2} - 1 = \frac{n-1}{2}$ non-zero variables. By Lemma 1, packing the last $\frac{n-1}{2}$ items into the knapsack is feasible, and since these have the largest coefficients, will give the best possible value. □

Proposition 1 *If n is even, the optimal solution to an instance in the "specific example" class is:*

$$x^* = (\underbrace{0,\ldots,0}_{\frac{n}{2}-1},\underbrace{1,\ldots,1}_{\frac{n}{2}},0).$$

Proof. When n is even,

$$b = \left\lfloor \sum_{i=1}^{n} a_i/2 \right\rfloor = \left\lfloor \sum_{j=1}^{n} \left(2^{k+n} + 2^{k+j-1} + 1/2\right) \right\rfloor = (n+1)2^{k+n} - 2^k + \frac{n}{2}.$$

Summing the smallest $\frac{n}{2}$ elements, we have

$$\sum_{j=1}^{\frac{n}{2}} a_j = \sum_{j=1}^{\frac{n}{2}} \left(2^{k+n+1} + 2^{k+j} + 1\right) = n2^{k+n} + 2^{k+\frac{n}{2}+1} - 2^{k+1} + \frac{n}{2} < b. \tag{3.4}$$

Summing the first $\frac{n}{2}+1$ elements, we have

$$\sum_{j=1}^{\frac{n}{2}+1} a_j = \sum_{j=1}^{\frac{n}{2}+1} \left(2^{k+n+1} + 2^{k+j} + 1\right) = (n+2)2^{k+n} + 2^{k+\frac{n}{2}+2} - 2^{k+1} + \frac{n}{2} + 1 > b. \tag{3.5}$$

Thus, we can pack at most $\frac{n}{2}$ items. The most valuable $\frac{n}{2}$ items have total weight

$$\sum_{j=\frac{n}{2}+1}^{n} a_j = \sum_{j=\frac{n}{2}+1}^{n} \left(2^{k+n+1} + 2^{k+j} + 1\right) = (n+1)2^{k+n} + 2^{k+n} - 2^{k+\frac{n}{2}+1} + \frac{n}{2} > b,$$

and, thus, we cannot simply pack the last $\frac{n}{2}$ items. However, note that

$$\left(\sum_{j=1}^{\frac{n}{2}-1} a_j\right) + a_n = \left(\sum_{j=1}^{\frac{n}{2}-1} \left(2^{k+n+1} + 2^{k+j} + 1\right)\right) + 2^{k+n+1} + 2^{k+n} + 1 = (n+1)2^{k+n} + 2^{k+\frac{n}{2}} - 2^{k+1} + \frac{n}{2} > b. \tag{3.6}$$

This shows that if the n-th item is included, we can pack at most $\frac{n}{2}-1$ items with total value no more than

$$\alpha := \sum_{j=\frac{n}{2}+2}^{n} a_j = \sum_{j=\frac{n}{2}+2}^{n} \left(2^{k+n+1}+2^{k+j}+1\right) = n2^{k+n} - 2^{k+\frac{n}{2}+1} + \frac{n}{2} - 1. \tag{3.7}$$

Excluding the n-th item and packing the most valuable $\frac{n}{2}$ items among the remaining $n-1$ items has value

$$\alpha < \sum_{j=\frac{n}{2}}^{n-1} a_j = \sum_{j=\frac{n}{2}}^{n-1} \left(2^{k+n+1}+2^{k+j}+1\right) = (n+1)2^{k+n} - 2^{k+\frac{n}{2}} + \frac{n}{2} < b. \tag{3.8}$$

Thus, we obtain that the optimal solution is $x^* = (\underbrace{0,\dots,0}_{\frac{n}{2}-1},\underbrace{1,\dots,1}_{\frac{n}{2}},0)$. $\square$

Next, we define the multi-variable branching scheme to be used for this class of instances. Let $\hat{x}$ be the solution to LP relaxation at a node, let $X_I = \{i:\ \hat{x}_i \in \{0,1\}, i\in[n]\}$ be the set of variables with an integral value, i.e. 0 or 1, in that solution, $X_f = \{i:\ 0<\hat{x}_i<1, i\in[n]\}$ be the set of variables with a fractional value in that solution, and $j=\min\{i:\ i\in X_f\}$ be the index of the first variable with a fractional value. Then we branch on the set $S = X_I \cup \{j\}$.

Theorem 1. *A branch and bound algorithm employing a multi-variable branching based on the set S as defined above solves an instance in at most 7 nodes when n is even and at most 3 nodes when n is odd.*

Proof. We introduce the following notation. The root node of the search tree is identified by $(0,\cdot)$ and other nodes are identified either by a pair $(k,+)$ or by a pair $(k,-)$ where k represents the depth of the node, $-$ indicates the node was reached from its parent by branching down, and $+$ indicates the node was reached from its parent by branching up. In general, this does not uniquely identify a node in the search tree, but, as we will see shortly, in this case it does. The solution $x^{(0,\cdot)}$ at the root node has at most one fractional variable, which implies that $S=[n]$.

When n is odd, by Lemma 1, we have $\frac{n-1}{2} < \sum_{i=1}^n x_i^{(0,\cdot)} < \frac{n+1}{2}$. Branching on $S=[n]$ generates two branches defined by $\sum_{i=1}^n x_i \le \frac{n-1}{2}$ and $\sum_{i=1}^n x_i \ge \frac{n+1}{2}$. By Lemma 1, LP$^{(1,+)}$ is infeasible, while LP$^{(1,-)}$ has solution $x^{(1,-)} = (\underbrace{0,\dots,0}_{\frac{n+1}{2}},\underbrace{1,\dots,1}_{\frac{n-1}{2}})$, which is the optimal solution to the instance. Thus, in this case, at most 3 nodes are evaluated.

When n is even, by (3.5), $\sum_{i=1}^n x_i^{(0,\cdot)} < \frac{n}{2}+1$. By reasoning in the proof of Proposition 1, $\frac{n}{2}-1 < \sum_{i=1}^n x_i^{(0,\cdot)}$. To see this, suppose otherwise, i.e., suppose $\frac{n}{2}-1 \ge \sum_{i=1}^n x_i^{(0,\cdot)}$. In this case, the objective value of $x^{(0,\cdot)}$ cannot be better than that obtained by packing the $\frac{n}{2}-1$ most valuable items, given by α in (3.7). But $\alpha < z^*$, the value of the integer program, which cannot exceed the value of the LP relaxation (which must be b).

We consider several cases for $\sum_{i=1}^n x_i^{(0,\cdot)}$ in the open interval $(\frac{n}{2}-1, \frac{n}{2}+1)$.

Case 1: $\frac{n}{2}-1<\sum_{i=1}^n x_i^{(0,\cdot)}<\frac{n}{2}$. Branching on $S=[n]$ generates two branches defined by $\sum_{i=1}^n x_i \le \frac{n}{2}-1$ and $\sum_{i=1}^n x_i \ge \frac{n}{2}$. For LP$^{(1,-)}$, it is easy to see that the solution is $x^{(1,-)}=(\underbrace{0,\dots,0}_{\frac{n}{2}+1},\underbrace{1,\dots,1}_{\frac{n}{2}-1})$, which is integer but not optimal. Next, we analyze two subcases for LP$^{(1,+)}$. (The search trees are shown in Figure 1.)

Case 1(a): If solution $x^{(1,+)}$ has exactly one fractional component, then $S=[n]$ and $\frac{n}{2}<\sum_{i=1}^n x_i^{(1,+)}<\frac{n}{2}+1$. Branching on $S=[n]$ generates two branches defined by $\sum_{i=1}^n x_i\le\frac{n}{2}$ and $\sum_{i=1}^n x_i\ge\frac{n}{2}+1$. Clearly, LP$^{(2,+)}$ is infeasible. Observe that LP$^{(2,-)}$ has both $\sum_{i=1}^n x_i\le\frac{n}{2}$ and $\sum_{i=1}^n x_i\ge\frac{n}{2}$, which implies that $\sum_{i=1}^n x_i=\frac{n}{2}$, which forces $x^{(2,-)}$ to have exactly two fractional variables. One of them has to be x_n, because if $x_n^{(2,-)}=0$, it will not give the largest possible value by (3.7) and (3.8). If $x_n^{(2,-)}=1$, even if all $\frac{n}{2}-1$ items with smallest weight are included, i.e., a_1 through $a_{\frac{n}{2}-1}$, the weight constraint is still violated in view of (3.6). Therefore, at this node, $S=[n-1]$, which generates two branches defined by $\sum_{i=1}^{n-1} x_i\le\frac{n}{2}-1$ and $\sum_{i=1}^{n-1} x_i\ge\frac{n}{2}$. LP$^{(3,-)}$ has both $\sum_{i=1}^n x_i=\frac{n}{2}$ and $\sum_{i=1}^{n-1} x_i\le\frac{n}{2}-1$, which implies $x_n=1$ and $\sum_{i=1}^{n-1}x_i=\frac{n}{2}-1$, which is infeasible. LP$^{(3,+)}$ has solution $x^{(3,+)}=(\underbrace{0,\dots,0}_{\frac{n}{2}-1},\underbrace{1,\dots,1}_{\frac{n}{2}},0)$ which is optimal to the original integer program. Thus, in this case, at most 7 nodes will be evaluated.

Case 1(b): If solution $x^{(1,+)}$ has two fractional components, then $x_n^{(1,+)}$ must be fractional using a similar argument to the one given for LP$^{(2,-)}$ in Case 1(a). Therefore, at this node, $S=[n-1]$, which generates two branches defined by $\sum_{i=1}^{n-1}x_i\le\frac{n}{2}-1$ and $\sum_{i=1}^{n-1}x_i\ge\frac{n}{2}$. LP$^{(2,-)}$ has both $\sum_{i=1}^n x_i\ge\frac{n}{2}$ and $\sum_{i=1}^{n-1}x_i\le\frac{n}{2}-1$ which implies $x_n=1$ and $\sum_{i=1}^{n-1}x_i=\frac{n}{2}-1$, which is infeasible by (3.6). LP$^{(2,+)}$ has both $\sum_{i=1}^n x_i\ge\frac{n}{2}$ and $\sum_{i=1}^{n-1}x_i\ge\frac{n}{2}$, one of which is redundant. Therefore, its solution $x^{(2,+)}$ has exactly one fractional variable, and, consequently, $S=[n]$ and $\frac{n}{2}<\sum_{i=1}^n x_i^{(2,+)}<\frac{n}{2}+1$. Branching on $S=[n]$ generates two branches defined by $\sum_{i=1}^n x_i\le\frac{n}{2}$ and $\sum_{i=1}^n x_i\ge\frac{n}{2}+1$. Clearly, LP$^{(3,+)}$ is infeasible. LP$^{(3,-)}$ has $\sum_{i=1}^n x_i\ge\frac{n}{2}$, $\sum_{i=1}^{n-1}x_i\ge\frac{n}{2}$ and $\sum_{i=1}^n x_i\le\frac{n}{2}$, which implies $\sum_{i=1}^n x_i=\frac{n}{2}$ and $x_n=0$, which gives solution $x^{(3,-)}=(\underbrace{0,\dots,0}_{\frac{n}{2}-1},\underbrace{1,\dots,1}_{\frac{n}{2}},0)$, which is the optimal to the instance. Thus, in this case too, at most 7 nodes will be evaluated.

Case 2: $\frac{n}{2}<\sum_{i=1}^n x_i^{(0,\cdot)}<\frac{n}{2}+1$. Branching on $S=[n]$ generates two branches defined by $\sum_{i=1}^n x_i\le\frac{n}{2}$ and $\sum_{i=1}^n x_i\ge\frac{n}{2}+1$. LP$^{(1,+)}$ is infeasible. Next, we analyze two subcases for LP$^{(1,-)}$. (The search trees are shown in Figure 2.)

Case 2(a): If $x^{(1,-)}$ has only one fractioning variable, then $S=[n]$ and $\frac{n}{2}-1<\sum_{i=1}^n x_i^{(1,-)}<\frac{n}{2}$ and two branches are generated defined by $\sum_{i=1}^n x_i\le\frac{n}{2}-1$ and $\sum_{i=1}^n x_i\ge\frac{n}{2}$, respectively. LP$^{(2,-)}$ and LP$^{(2,+)}$ are the same as LP$^{(1,-)}$ in case Case 1 and LP$^{(2,-)}$ in case Case 1(a), which completes the argument for this case.

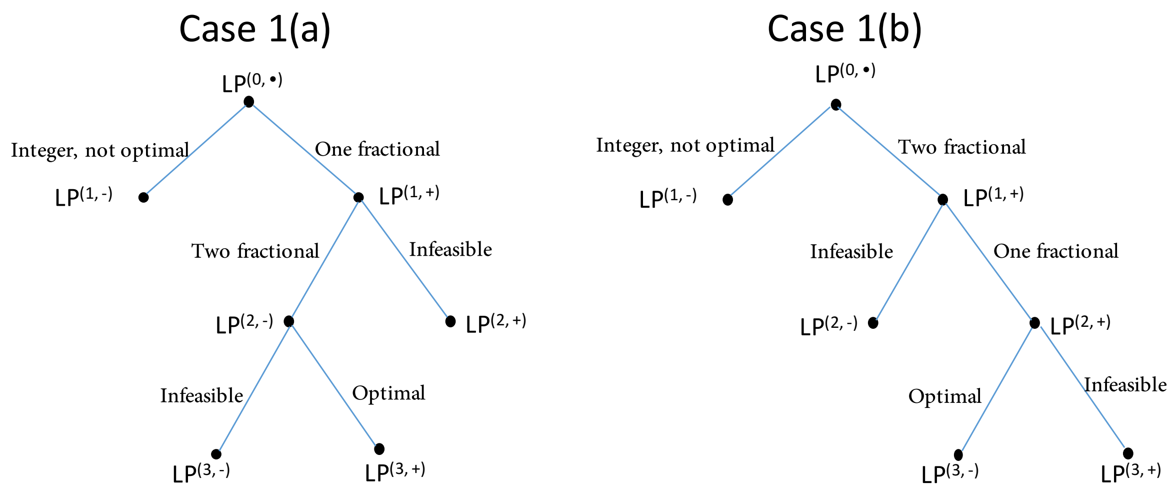

Figure 1 **Search trees when n is even for Case 1.**

Case 2(b): If $x^{(1,-)}$ has two fractional variables, by similar argument for LP$^{(2,-)}$ in Case 1(a), we know $x_n^{(1,-)}$ is fractional. Then $S=[n-1]$ and $\frac{n}{2}-1<\sum_{i=1}^{n-1}x_i^{(1,-)}<\frac{n}{2}$ and two branches are generated defined by $\sum_{i=1}^{n-1}x_i\le\frac{n}{2}-1$ and $\sum_{i=1}^{n-1}x_i\ge\frac{n}{2}$, respectively. LP$^{(2,+)}$ has both $\sum_{i=1}^{n}x_i\le\frac{n}{2}$ and $\sum_{i=1}^{n-1}x_i\ge\frac{n}{2}$, which yields the optimal solution $x^{(2,+)}=(\underbrace{0,\dots,0}_{\frac{n}{2}-1},\underbrace{1,\dots,1}_{\frac{n}{2}},0)$. LP$^{(2,-)}$ has both $\sum_{i=1}^{n}x_i\le\frac{n}{2}$ and $\sum_{i=1}^{n-1}x_i\le\frac{n}{2}-1$, one of which is redundant. $x^{(2,-)}$ can only have one fractional variable, and $\sum_{i=1}^{n-1}x_i<\frac{n}{2}-1$ and $x_n^{(2,-)}$ is fractional. Thus $S=[n]$ and $\frac{n}{2}-1<\sum_{i=1}^{n}x_i^{(2,-)}<\frac{n}{2}$ and two branches are generated defined by $\sum_{i=1}^{n}x_i\le\frac{n}{2}-1$ and $\sum_{i=1}^{n}x_i\ge\frac{n}{2}$, respectively. LP$^{(3,-)}$ are the same as LP$^{(1,-)}$ in Case 1, which has non-optimal integer solution, and LP$^{(3,+)}$ is infeasible. □

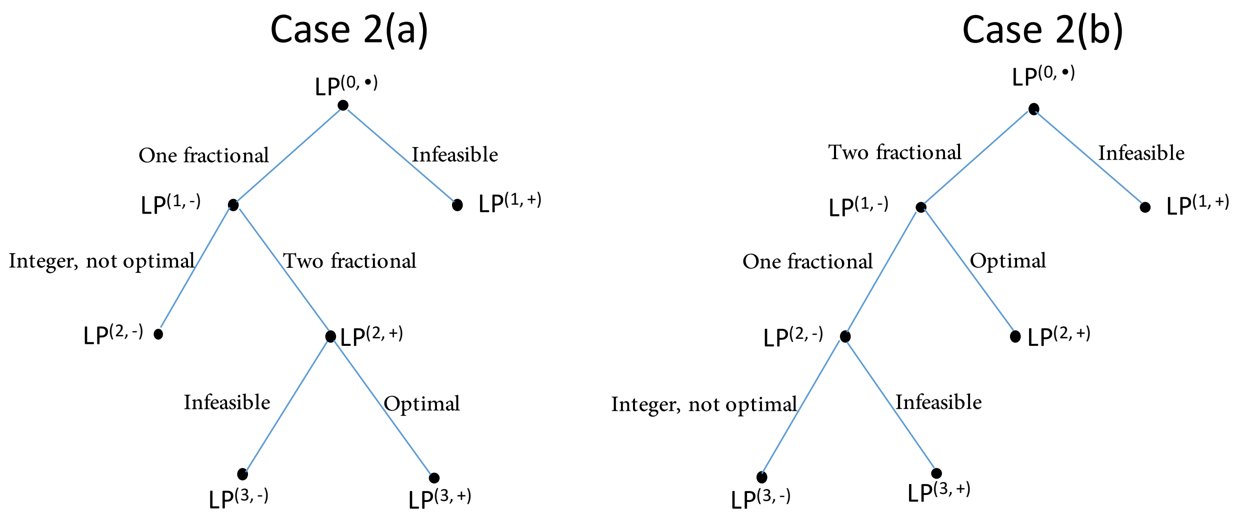

Figure 2 **Search trees when n is even for Case 2.**

4. Examples and Analysis

The multi-variable branching scheme used to solve Chvátal's specific example class of instances shows the dramatic improvements that are possible, but it is a specific branching scheme for a specific class of instances. Next, we present and analyze examples in which branching on a set of variables of size two improves over branching on a set of variables of size one, and branching on a set of variables of size three improves over branching on a set of variables of size two. The insights obtained from the analysis suggest branching schemes that are more general and that may be computationally viable.

The following example illustrates that branching on sets of variables of size two can be better than branching on sets of variables of size one (standard single-variable branching). Consider the 0-1 knapsack problem

$$\begin{aligned} z^* = \ \max \quad & 9x_1 + 4.2x_2 + x_3 \\ \text{subject to} \quad & 3x_1 + 2x_2 + x_3 \leq 5.7, \\ & x_i \in \{0,1\}, \quad i \in [3]. \end{aligned}$$

The solution to the LP relaxation is $x^{LP} = (1,1,0.7)$ with value $z^{LP} = 13.9$. Using standard single-variable branching, the solution to LP relaxation at the node on the down branch is $x^- = (1,1,0)$ with value $z^- = 13.2$, and the solution to LP relaxation at the node on the up branch is $x^+ = (1,0.85,1)$ with value $z^+ = 13.57$. If, instead, we branch on the set of variables $S = \{2,3\}$, then the solution to LP relaxation at the node on the down branch is $x^- = (1,1,0)$ with value $z^- = 13.2$, and the solution to LP relaxation at the node on the up branch is $x^+ = (0.9,1,1)$ with value $z^+ = 13.3$. Thus, we obtain a better bound as $\max\{13, 13.3\} = 13.3 < \max\{13, 13.57\} = 13.57$. See Figure 3 for an illustration.

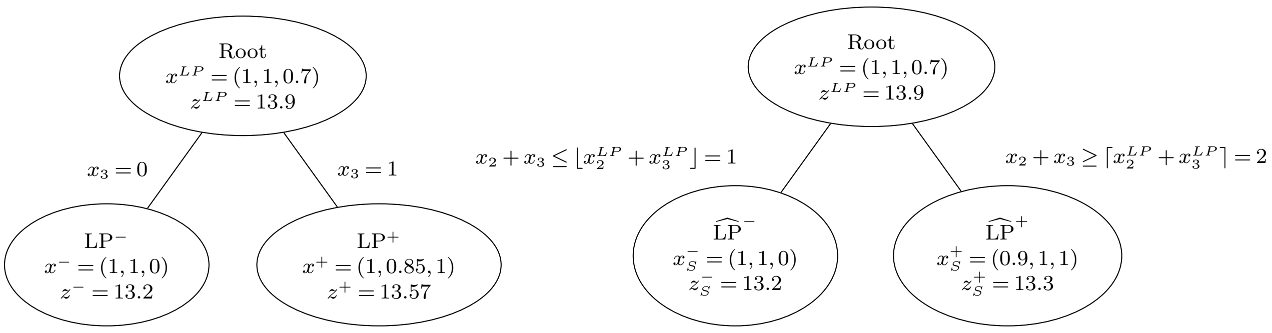

Figure 3 **Example in which multi-variable branching (with set of branching variables $|S| = 2$) improves over single-variable branching.**

Next, consider the 0-1 knapsack problem

$$\begin{aligned} z^* = \ \max \quad & 16x_1 + 9x_2 + 4x_3 + x_4 \\ \text{subject to} \quad & 4x_1 + 3x_2 + 2x_3 + x_4 \leq 5.2, \\ & x_i \in \{0,1\}, \quad i \in [4]. \end{aligned}$$

The solution to the LP relaxation is $x^{LP} = (1, 0.4, 0, 0)$ with value $z^{LP} = 19.6$. Using standard single-variable branching, the solution to the LP relaxation at the node on the down branch is $x^- = (1, 0, 0.6, 0)$ with value $z^- = 18.4$, and the solution to the LP relaxation at the node on the up branch is $x^+ = (0.55, 1, 0, 0)$ with value $z^+ = 17.8$. If, instead, we branch on the set of variables $S = \{2, 3\}$, then the solution to the LP relaxation at the node on the down branch is $x^- = (1, 0, 0, 1)$ with value $z^- = 17$, and the solution to LP relaxation at the node on the up branch is $x^+ = (0.55, 1, 0, 0)$ with value $z^+ = 17.8$. Thus, we obtain a better bound: $\max\{17, 17.8\} = 17.8 < \max\{18.4, 17.8\} = 18.4$. (Branching on $S = \{1, 2\}$ gives the same bound as single-variable branching: the up branch is infeasible and the down branch gives bound 18.4.) However, if, instead, we branch on the set of variables $S = \{1, 2, 3\}$, then the solution to the LP relaxation at the node on the down branch is $x^- = (1, 0, 0, 1)$ with value $z^- = 17$, but the solution to the LP relaxation at the node on the up branch is $x^+ = (0.2, 0.8, 1, 0)$ with value $z^+ = 14.4$. Thus, we obtain an even better bound as $\max\{17, 14.4\} = 17 < \max\{17, 17.8\} = 17.8$. See Figure 4 for an illustration.

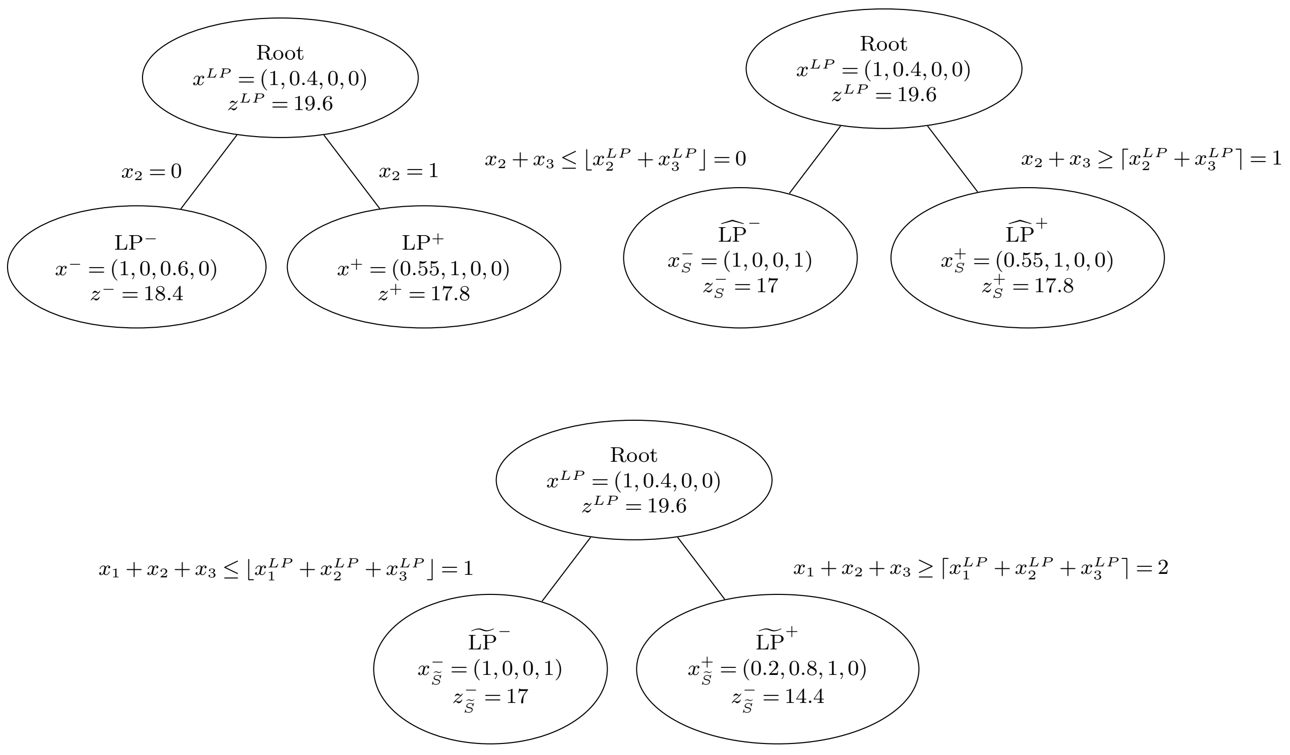
Root
$x^{LP} = (1, 0.4, 0, 0)$
$z^{LP} = 19.6$
$x_2 = 0$
$x_2 = 1$
LP$^-$
$x^- = (1, 0, 0.6, 0)$
$z^- = 18.4$
LP$^+$
$x^+ = (0.55, 1, 0, 0)$
$z^+ = 17.8$
Root
$x^{LP} = (1, 0.4, 0, 0)$
$z^{LP} = 19.6$
$x_2 + x_3 \leq \lfloor x_2^{LP} + x_3^{LP} \rfloor = 0$
$x_2 + x_3 \geq \lceil x_2^{LP} + x_3^{LP} \rceil = 1$
$\widehat{\text{LP}}^-$
$x_S^- = (1, 0, 0, 1)$
$z_S^- = 17$
$\widehat{\text{LP}}^+$
$x_S^+ = (0.55, 1, 0, 0)$
$z_S^+ = 17.8$
Root
$x^{LP} = (1, 0.4, 0, 0)$
$z^{LP} = 19.6$
$x_1 + x_2 + x_3 \leq \lfloor x_1^{LP} + x_2^{LP} + x_3^{LP} \rfloor = 1$
$x_1 + x_2 + x_3 \geq \lceil x_1^{LP} + x_2^{LP} + x_3^{LP} \rceil = 2$
$\widetilde{\text{LP}}^-$
$x_{\tilde{S}}^- = (1, 0, 0, 1)$
$z_{\tilde{S}}^- = 17$
$\widetilde{\text{LP}}^+$
$x_{\tilde{S}}^+ = (0.2, 0.8, 1, 0)$
$z_{\tilde{S}}^- = 14.4$

Figure 4 **Example in which multi-variable branching using a set of branching variables of size three improves over multi-variable branching using a set of branching variables of size two.**

Next, we present a necessary, but not sufficient, condition as well as a sufficient, but not necessary, condition for multi-variable branching using a set of variables of size two to outperform single-variable branching. In all that follows, when we say "outperforms" or "is better than" we mean that the dual bound after branching is lower. Without loss of generality, suppose

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}$$

and that the solution to the LP relaxation at the root node is fractional, i.e., the solution has the form

$$x^{LP} = (\underbrace{1, \ldots, 1}_{i-1}, f, \underbrace{0, \ldots, 0}_{n-i}),$$

with $1 < i \leq n$. (Note that we have assumed that $w_i \leq b$ for $i = 1, \ldots, n$.) We consider two cases, $i = n$ and $1 < i < n$. In the former case, we give a necessary condition for branching on a set of two variables to outperform single-variable branching. In the latter case, we provide a sufficient condition.

Proposition 2 *Suppose $i = n$ and x^+ is the solution to the LP relaxation obtained on the up branch when branching on $\{n\}$. Furthermore, suppose that branching on $S' = \{k, n\}$ is better than branching on $\{n\}$, then x^+ is fractional and k is the index of the fractional variable in x^+.*

Proof. Since $i = n$, it must be that $x^{LP} = (1, 1, \ldots, 1, f)$ with $0 < f < 1$. It is easy to see that branching on S' gives

$$x^- = (\underbrace{1, \ldots, 1}_{n-1}, 0), \quad x^+ = (\underbrace{1, \ldots, 1}_{k-1}, f^+, \underbrace{0, \ldots, 0}_{n-k-1}, 1), \quad 0 \leq f^+ \leq 1,\ 1 \leq k \leq n-1.$$

If f^+ is integer, then we have integer solutions on both branches, and $z = \max\{z^-,\ z^+\}$ is the optimal value for the original integer program. Therefore, branching on any set S cannot be better than branching on S'.

So assume $0 < f^+ < 1$. Consider $S = \{\ell, n\}$ with $\ell < k$ $(k \geq 2)$. On the down branch we add $x_\ell + x_n \leq \lfloor x_\ell^{LP} + x_n^{LP} \rfloor = 1$, and on the up branch we add $x_\ell + x_n \geq \lceil x_\ell^{LP} + x_n^{LP} \rceil = 2$. Note that x^- is feasible for the down branch, and, thus, $z_S^- \geq z^-$. Similarly, x^+ is feasible for the up branch, and, thus, $z_S^+ \geq z^+$. This implies $z_S = \max\{z_S^-, z_S^+\} \geq \max\{z^-, z^+\} = z$, which means that branching on S cannot be better than branching on S'. Next, consider $S = \{\ell, n\}$ with $k < \ell < n$ $(k \leq n-2)$. On the down branch, we add $x_\ell + x_n \leq \lfloor x_\ell^{LP} + x_n^{LP} \rfloor = 1$, which is satisfied by both x^- and x^+. Again, branching on S cannot be better than branching on S'. $\square$

Proposition 3 *Suppose $1 < i < n$, $\frac{p_{i+1}}{w_{i+1}} > \frac{p_{i+2}}{w_{i+2}}$, and $w_{i+1} \geq w_i$. Furthermore, suppose that the value of the solution to the LP relaxation on the down branch is greater than the value of the solution to the LP relaxation on the up branch, i.e., $z^- > z^+$, when branching on $\{i\}$. Then branching on $S = \{i, i+1\}$ is better than branching on $\{i\}$ and achieves the best bound that can be achieved by branching on any set of variables of size two.*

Proof. Since $w_{i+1} \geq w_i$, branching on $\{i\}$ results, on the down branch, in

$$x^- = (\underbrace{1, \ldots, 1}_{i-1}, 0, f^-, \underbrace{0, \ldots, 0}_{n-i-2}), \quad 0 < f^- < 1.$$

When we branch on S, we add $x_i + x_{i+1} \leq \lfloor x_i^{LP} + x_{i+1}^{LP} \rfloor = 0$ on the down branch. Because $\frac{p_{i+1}}{w_{i+1}} > \frac{p_{i+2}}{w_{i+2}}$, we have that $z_S^- < z^-$.

Let $\bar{b} = b - \sum_{k=1}^{i-1} w_k$. Since $x_i^{LP} = f$ is fractional, $w_i > \bar{b}$, thus, $w_{i+1} > \bar{b}$. We will show that $x_S^+ = x^+$, and as a result $z_S^+ = z^+$. When we branch on S, we add $x_i + x_{i+1} \geq \lceil x_i^{LP} + x_{i+1}^{LP} \rceil = 1$, which causes some "waste" of the resource as items i and $(i+1)$ are less desirable than items 1 through $i-1$. By setting $x_i = 1$ and $x_{i+1} = 0$, the branching constraint is satisfied and the waste is minimized. As a consequence, we obtain solution x^+. Thus, $z_S = \max\{z_S^-, z_S^+\} = \max\{z_S^-, z^+\} < z^- = z$.

Next, consider branching on $S' = \{k, i\} \neq \{i, i+1\}$. When $k < i$, we add $x_k + x_i \leq \lfloor x_k^{LP} + x_i^{LP} \rfloor = 1$ on the down branch. We have $z_{S'}^- \geq z^-$ because x^- satisfies $x_k^- + x_i^- \leq 1$. This implies $z_{S''}^- \geq z^- > z^+$, i.e., branching on S' cannot be better than branching on S (in fact, it cannot even better than branching on $\{i\}$). When $k > i+1$, we add $x_i + x_k \leq \lfloor x_i^{LP} + x_k^{LP} \rfloor = 0$ on the down branch, and have $x_i = x_k = 0$. Since $w_{i+1} \geq w_i > \bar{b}$, all of the resource can be consumed by items in $\{1, \ldots, i-1, i+1\}$. When branching on S, on the down branch only items in $\{1, \ldots, i-1, i+2, \ldots, n\}$ can be used, which implies $z_{S'}^- \geq z_S^-$. We have already argued that when we branch on S, on the up branch we will have $x_i^+ = 1$ and $x_{i+1}^+ = 0$. This solution is feasible when branching on S', as $x_i + x_k \geq \lfloor x_i^{LP} + x_k^{LP} \rfloor = 1$ is added on the up branch. Therefore, we also have $z_{S'}^+ \geq z_S^+$, which implies branching on S' cannot be better than branching on S. □

5. Computational Study

In this section, we evaluate the performance of four multi-variable branching schemes motivated by the observations in the previous section. For all instances, we order the ratios $\frac{p_i}{w_i}$ in non-increasing order for $1 \leq i \leq n$.

5.1. Branching on sets of size two

The first two multi-variable branching schemes involve sets of size two in order to control the computation time required to identify the set of variables to branch on. In the first scheme, we use strong branching to compute SB scores for all variables with fractional values and all sets of two variables with a fractional sum and then select either the single variable or the set of two variables with the best SB score. This scheme, which we refer to as "B1", is an analogue to standard single-variable strong branching; it becomes computationally prohibitive when the number of variables in the instance becomes (too) large. In the second scheme, which is inspired by the analysis in Section 4, we choose $S = \{i, i+1\}$ where i is the largest index such that x_i^* is fractional and $i < n$, and choose $S = \{i, j\}$, when $i = n$ and where j the largest index such that $x_j^* \in \{0, 1\}$. Observe that this multi-variable branching scheme is attractive computationally as determining S does not require the solution of any linear programs. We refer to this scheme as "B2".

5.2. Branching on dynamically determined sets

The last two multi-variable branching schemes may involve sets of size larger than two as this may improve the performance (as we have seen in Section 4). To control the computation time somewhat, we generate the set S of variables to branch on dynamically. More specifically, we start from $S = \{i\}$ with i the index of the fractional variable that gives the best bound among all fractional variables when branching on these variables. Note that i is the index of the variable that would be chosen if strong branching was used. Next, we evaluate sets $S = \{i, j\}$ for j such that $x_i^{LP} + x_j^{LP}$ is fractional, and, again, choose the set that gives the best bound, say $S = \{i, \hat{j}\}$. If the best bound associated with $\{i, \hat{j}\}$ is better than the best bound associated with $\{i\}$, then the process continues, i.e., we explores all sets of size three, extending $\{i, \hat{j}\}$, to see if there is one among them that results in a better best bound. We continue extending the set S with one more variable as long as it results in an improved bound. We refer to this scheme as "B3". Clearly, the B3 branching scheme is computationally intensive, and, therefore, we consider a variant in which the cardinality of the set is limited to at most K_1 and it is only applied at the top of the search tree, i.e., at nodes of depth less than or equal to K_2. At nodes of depth more than K_2, standard single-variable branching is used, where we branch on the fractional variable with the smallest index. We refer to this scheme as "B4".

5.3. Computational experiments

In our computational experiments, we compare the five branching schemes:

B0: Standard single-variable branching;

B1: Standard two-variable branching;

B2: Restricted two-variable branching;

B3: Dynamic multi-variable branching; and

B4: Restricted multi-variable branching with $K_1 = 3$ and $K_2 = 10$.

We implemented these five variable selection schemes in the branch-and-bound framework of CPLEX using their callback functions. All experiments to compare the performance of the five branching schemes use CPLEX 12.8 in single-thread mode with a time limit of one hour for each instance. Since our focus is on the impact of the branching strategy, we turn off cuts, heuristics, and presolve.

For our computational experiments, we use three classes of instances introduced in Pisinger (2005) (the code to generate the instances is available at GitHub (`https://github.com/Yu1423/Multi-Variable-Branching-Data`): uncorrelated instances, weakly correlated instances, and strongly correlated instances with coefficient ranges $R = 10^3$ and 10^4. For uncorrelated and weakly correlated instances, we use sizes $n =$100, 200, 500, and, 1000, and for strongly correlated instances, because they are significantly more difficult to solve, we use sizes $n =$ 20, 30, 40, and 50. Thus, for each class of instances, we have eight different combinations of coefficient range and size. For each combination, we generate 100 instances. The performance metrics of interest are: the number of nodes evaluated, N, and the solution time, t.

Many commercial (and open-source) solvers have switched to using the product scoring rule (Achterberg 2007), rather than $\max\{z^-, z^+\}$, to assess the desirability of branching on a particular variable. However, because preliminary computational experiments with 0-1 knapsack instances revealed no clear performance improvement when using the product form (in fact, in many cases showed performance deterioration) and because our theoretical analysis uses max form, we present results for both. Note that a score function is needed only for branching schemes B1, B3, and B4. With branching scheme B0, every LP solution will have at most one fractional variable, and branching scheme B2 employs an unambiguous rule to identify a pair of variables to branch on. Therefore, in Tables 1 and 2, we show the results for branching scheme B2 only in the $\text{Max}\{z^-, z^+\}$ score function section.

The results of our computational experiments can be found in Tables 1 and 2. For branching scheme B0, we report the number of nodes evaluated (Table 1) and the solution time (Table 2) averaged over all instances of a given size. For each of the other branching schemes, we report the number of nodes evaluated and the solution time *relative* to branching scheme B0, i.e., we report ratios of the number of nodes evaluated and the solution time. Since the solution time for nearly all weakly corrected instances with $n = 1000$ exceed the 1 hour time limit for branching scheme B1 (for both max and product functions) and for branching scheme B3 (for the product function), we do not report results for these settings. Furthermore, in Figure 5, we show performance profiles (Dolan and More 2002) for the five branching schemes using the number of nodes evaluated N (on the left) and the solution time t (on the right) for the uncorrelated, weakly correlated, and strongly correlated instances, respectively.

As the performance of the different branching schemes is similar for the uncorrelated and weakly correlated instances, but somewhat different for the strongly correlated instances, we discuss the results for these classes of instances separately.

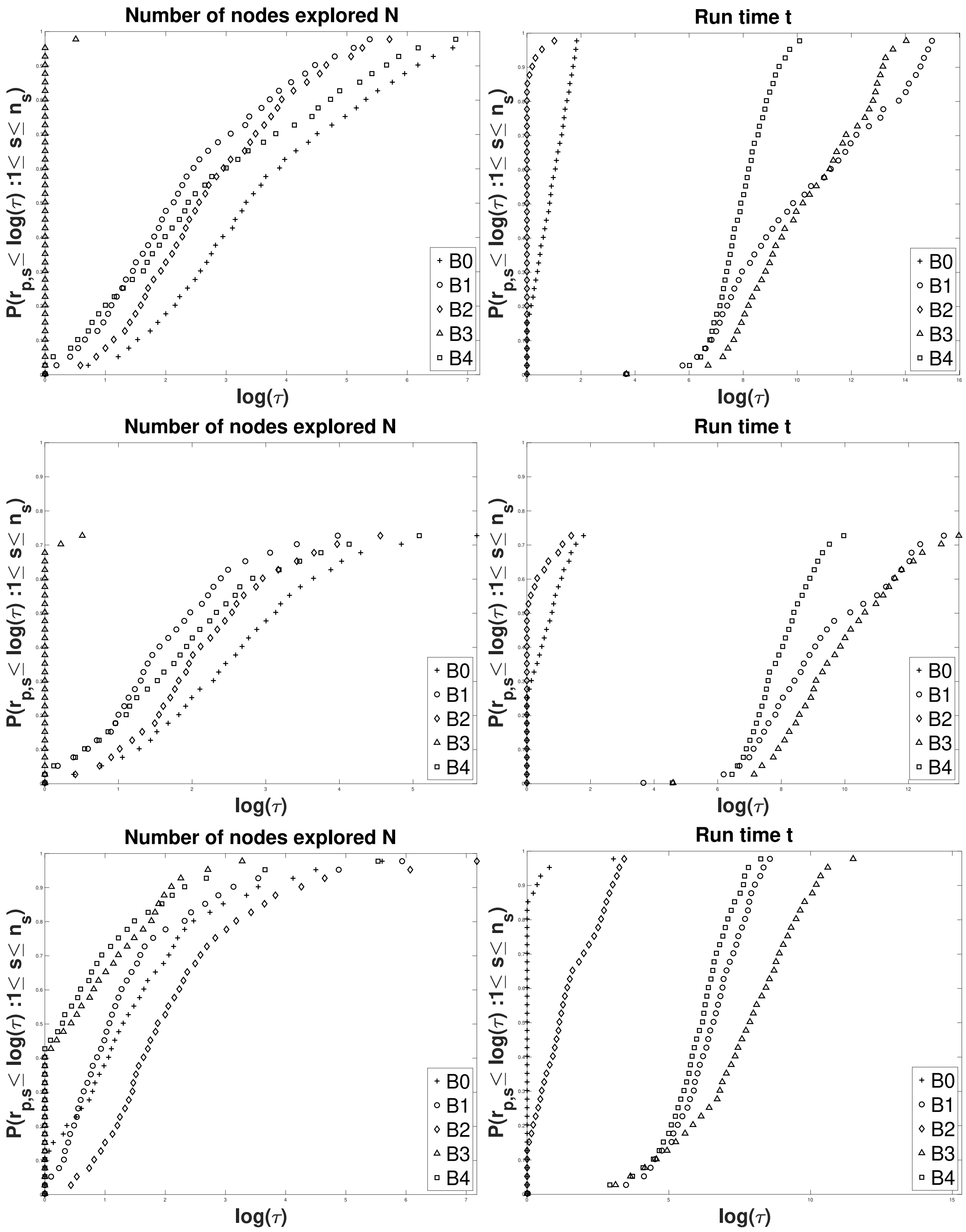

Figure 5 **Performance profiles for uncorrelated (the first row), weakly correlated (the second row), and strongly correlated instances (the bottom row). Left: # of nodes explored N; Right: run time t. The x-axis represents a factor $\tau > 0$ and the y-axis represents the probability that the performance ratio of the corresponding method is less than or equal to τ (see Dolan and More (2002)).**

Table 1 N **of B0 and ratios of** N **relative to B0.**

R=10^3	Score / n		Max$\{z^-, z^+\}$				Product		
		B0	B1	B2	B3	B4	B1	B3	B4
Uncorrelated	100	103	0.55	0.80	0.23	0.56	0.64	0.44	0.87
	200	202	0.46	0.69	0.16	0.58	0.56	0.38	0.85
	500	450	0.36	0.52	0.08	0.67	0.46	0.28	0.96
	1,000	1,291	0.34	0.45	0.04	1.04	0.42	0.22	4.67
Weakly correlated	100	264	0.75	1.31	0.42	0.69	0.92	1.02	1.34
	200	485	0.62	1.08	0.26	0.73	0.74	0.93	1.30
	500	653	0.45	0.75	0.17	1.26	0.59	0.81	1.70
	1,000	843	—	0.63	0.19	4.13	—	—	3.93
Strongly correlated	20	162	1.02	1.97	0.87	0.65	0.88	1.98	0.78
	30	1,208	0.67	2.02	0.23	0.61	0.59	0.75	0.43
	40	2,860	0.78	2.56	0.21	0.74	0.64	0.47	0.63
	50	22,615	0.63	2.15	0.04	0.93	0.35	0.12	0.42

R=10^4	Score / n		Max$\{z^-, z^+\}$				Product		
		B0	B1	B2	B3	B4	B1	B3	B4
Uncorrelated	100	103	0.52	0.74	0.20	0.52	0.63	0.43	0.81
	200	209	0.47	0.71	0.16	0.53	0.55	0.40	0.86
	500	459	0.36	0.50	0.08	0.51	0.45	0.30	0.79
	1,000	775	0.28	0.32	0.02	0.53	0.33	0.15	0.92
Weakly correlated	100	290	0.76	1.28	0.39	0.66	0.84	0.97	1.25
	200	534	0.62	1.09	0.29	0.63	0.71	0.83	1.25
	500	707	0.42	0.63	0.12	0.63	0.52	0.64	1.20
	1,000	987	—	0.36	0.04	0.93	—	—	2.19
Strongly correlated	20	485	1.22	1.55	0.63	0.40	0.54	1.79	0.56
	30	617	1.09	2.66	0.77	0.62	0.82	1.93	0.84
	40	4,571	0.80	2.47	0.13	1.07	0.49	0.51	0.41
	50	12,875	0.61	2.20	0.06	0.90	0.36	0.25	0.20

In Figure 5 (and also in Table 1), we see that all four multi-variable branching schemes achieve better node-efficiency than standard single-variable branching for almost all combinations of R and n. It is also clear that branching scheme B3 results in the largest reduction in number of nodes evaluated for all instances. However, as Figure 5 (and also Table 2) reveal, this reduction in number of nodes evaluated comes at a high price in terms of solution time. As expected, branching scheme B4 is faster than branching scheme B3 even though it explores many more nodes. Tuning the two restriction parameters of B4 may achieve an even better balance between node-efficiency and time-efficiency. Importantly, we observe that one of the multi-variable branching schemes, B2, does better than the standard single-variable scheme, B0, in terms of solution time (for all uncorrelated instances and for large weakly correlated instances).

Table 2 t **of B0 in seconds and ratios of** t **relative to B0.**

R=10^3	Score / n		Max$\{z^-, z^+\}$				Product		
		B0	B1	B2	B3	B4	B1	B3	B4
Uncorrelated	100	0.01	121.80	**0.84**	212.88	96.77	186.21	664.66	161.35
	200	0.01	391.10	**0.72**	469.27	153.50	630.03	1,635.47	252.10
	500	0.04	2,004.81	**0.50**	1,332.98	222.53	3,250.63	7,639.95	365.70
	1,000	0.24	8,355.19	**0.45**	3,378.24	221.88	12,653.67	35,272.41	314.00
Weakly correlated	100	0.01	229.33	1.63	542.09	214.71	359.89	2,506.82	329.76
	200	0.02	507.03	1.27	970.21	326.34	801.70	6,930.39	436.65
	500	0.06	2,389.17	**0.69**	4,011.84	303.46	4,259.84	32,596.95	368.25
	1,000	0.14	—	**0.60**	17,242.40	332.22	—	—	305.71
Strongly correlated	20	0.00	67.95	3.03	133.82	55.10	67.66	413.55	57.45
	30	0.03	64.59	2.99	73.05	28.68	73.34	322.66	26.85
	40	0.07	112.17	4.28	127.32	34.61	121.78	427.90	29.57
	50	0.74	95.06	3.09	26.73	8.19	74.58	125.22	6.63

R=10^4	Score / n		Max$\{z^-, z^+\}$				Product		
		B0	B1	B2	B3	B4	B1	B3	B4
Uncorrelated	100	0.01	124.83	0.82	224.26	97.21	197.89	678.29	156.23
	200	0.01	379.42	0.73	449.02	139.80	598.36	1,655.14	251.13
	500	0.04	1,946.13	0.48	1,271.18	182.79	3,115.66	5,386.72	310.01
	1,000	0.14	7,114.82	0.32	3,093.03	124.79	10,592.71	13,749.53	199.88
Weakly correlated	100	0.01	242.40	1.63	549.79	219.07	347.88	2,423.86	327.81
	200	0.02	493.22	1.29	1,138.47	328.47	768.23	5,595.65	455.22
	500	0.07	1,818.82	0.53	2,738.93	249.18	2,846.89	16,816.90	339.70
	1,000	0.17	—	0.35	6,594.44	150.62	—	—	222.64
Strongly correlated	20	0.11	153.18	2.86	216.70	55.43	76.96	947.32	58.99
	30	0.14	141.32	5.06	273.04	80.92	141.25	1,235.41	86.09
	40	0.13	108.59	3.86	75.58	20.77	84.94	404.95	16.56
	50	0.36	105.89	3.59	50.84	10.23	85.96	337.08	8.26

Comparing the results for the different score functions, for B1, B3, and B4, we see that for these 0-1 knapsack instances, it is not the case that the product form should be preferred, even though the computational study by Achterberg (2007) shows that the product form outperforms the max form by 14% on average. This is interesting and suggests that the score function also plays a role in the effectiveness of multi-variable branching.

To further explore the benefits of multi-variable scheme B2, for uncorrelated and weakly correlated instances, we generated and solved additional instances of significantly larger size: $n = 5000$ and 10000. As before, for each combination of R and n, we generate 100 instances. The results can be found in Table 3. These results provide further evidence of the potential and promise of multi-variable branching: for coefficient range $R = 10^4$ both the number of nodes and the solve time are

Table 3 N and t of B0 (absolute values) and B2 (ratios with respect to B0) for instances of larger size with $R = 10^4$.

	R		B0		B2	
		n	N	t	N	t
Uncorrelated	10^3	5000	1,161	0.73	1.46	1.52
		10000	1,360	1.59	2.15	2.55
	10^4	5000	3,116	1.89	0.32	**0.43**
		10000	4,146	4.85	0.54	**0.63**
Weakly correlated	10^3	5000	1,494	0.89	1.37	1.56
		10000	1,594	1.80	2.17	2.67
	10^4	5000	3,469	2.06	0.37	**0.47**
		10000	5,679	6.31	0.60	**0.69**

reduced. The reason why branching scheme B2 does not perform so well when the coefficient range is $R = 10^3$ is that the number of variables, n, is larger than the coefficient range, which implies that it is more likely that there are items with similar $^{p}/_{w}$ ratios, which makes these instances similar to the more difficult strongly correlated instances.

As mentioned above, the results are somewhat different for the strongly correlated instances. For example, branching scheme B2 no longer has any benefits compared to B0. This is not too surprising, however, as Proposition 3 indicates that the performance of B2 depends on the ratios $^{p_i}/_{w_i}$ for $1 \leq i \leq n$.

Branching scheme B3 continues to significantly reduce the number of nodes evaluated, especially when using the max scoring rule, but, importantly, so does branching scheme B4, especially for the product scoring rule.

Because branching scheme B4 exhibited promising trends for both the number of nodes evaluated and the solution time for strongly correlated instances, we generated and solved additional instances of larger size: $n = 60$, 80, and 100. The results can be found in Table 4.

Table 4 N and t of B0 (absolute values) and B4 (ratios with respect to B0) for strongly correlated instances of larger size.

		B0		B4 Max		B4 Product	
	n	N	t	N	t	N	t
$R = 10^3$	60	130,707	5.06	1.05	3.20	0.39	2.23
	80	2,646,631	108.07	0.81	1.34	0.83	1.14
	100	2,690,262	108.41	0.84	1.40	0.74	1.08
$R = 10^4$	60	59,881	2.23	0.75	4.90	0.51	3.93
	80	2,108,874	148.97	0.79	1.12	0.74	**0.81**
	100	2,559,953	132.32	0.71	1.16	0.76	**0.83**

We see that branching scheme B4 not only outperforms the default branching scheme B0 in terms of number of nodes evaluated, but also in terms of solution time when the product scoring rule is used and the coefficient range of the instances is 10^4 and the instances size is 80 and 100.

6. Final Remarks

In this paper, we explored the potential benefits of branching on sets of variables rather than standard single-variable branching. Preliminary computational experiments on randomly generated instances of the 0-1 knapsack problem show promise and reveal the trade off between node-efficiency and time-efficiency. Our current research is focused on reproducing the promise of multi-variable branching on other classes of mixed integer programs and using machine learning to develop more efficient implementations of multi-variable branching schemes.

Acknowledgment

We want to thank Andrea Lodi for reading an earlier version of the paper and providing valuable comments and suggestions, and Gabor Pataki for informative discussions on basis reduction methods and their connection with multi-variable branching.

References

Aardal K, Weismantel R, Wolsey LA (2002) Non-standard approaches to integer programming. *Discrete Applied Mathematics* 123(1):5 – 74.

Achterberg T (2007) *Constraint integer programming*. Ph.D. thesis, TU Berlin.

Achterberg T, Berthold T (2009) Hybrid branching. *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 309–311 (Springer).

Achterberg T, Berthold T, Koch T, Wolter K (2008) Constraint integer programming: A new approach to integrate CP and MIP. *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, 6–20 (Springer).

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

Applegate D, Bixby R, Chvatal V, Cook B (1995) Finding cuts in the tsp (a preliminary report). Technical report, Center for Discrete Mathematics & Theoretical Computer Science.

Balcan MF, Dick T, Sandholm T, Vitercik E (2018) Learning to branch. *arXiv preprint arXiv:1803.10150* .

Beale E (1979) Branch and bound methods for mathematical programming systems. Hammer P, Johnson E, Korte B, eds., *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, 201 – 219 (Elsevier).

Beale EML, Tomlin JA (1970) Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. Lawrence J, ed., *Proceedings 5th IFORS Conference, Tavistock*, 447–454 (Wiley).

Bénichou M, Gauthier JM, Girodet P, Hentges G, Ribière G, Vincent O (1971) Experiments in mixed-integer linear programming. *Mathematical Programming* 1(1):76–94.

Berthold T, Salvagnin D (2013) Cloud branching. *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 28–43 (Springer).

Chvátal V (1980) Hard knapsack problems. *Operations Research* 28(6):1402–1411.

Di Liberto G, Kadioglu S, Leo K, Malitsky Y (2016) Dash: Dynamic approach for switching heuristics. *European Journal of Operational Research* 248(3):943–953.

Dolan ED, More JJ (2002) Benchmarking Optimization Software with Performance Profiles. *Mathematical Programming* 91:201–213.

Gauthier JM, Ribière G (1977) Experiments in mixed-integer linear programming using pseudo-costs. *Mathematical Programming* 12(1):26–47.

Khalil EB, Le Bodic P, Song L, Nemhauser GL, Dilkina BN (2016) Learning to branch in mixed integer programming. *AAAI*, 724–731.

Krishnamoorthy B, Pataki G (2009) Column basis reduction and decomposable knapsack problems. *Discrete Optimization* 6(3):242 – 270.

Land AH, Doig AG (1960) An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society* 497–520.

Lenstra, Jr H (1983) Integer programming with a fixed number of variables. *Mathematics of Operations Research* 8:538—548.

Linderoth JT, Savelsbergh MW (1999) A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing* 11(2):173–187.

Lodi A, Zarpellon G (2017) On learning and branching: a survey. *Top* 25(2):207–236.

Marcos Alvarez A, Louveaux Q, Wehenkel L (2017) A machine learning-based approximation of strong branching. *INFORMS Journal on Computing* 29(1):185–195.

Marcos Alvarez A, Wehenkel L, Louveaux Q (2016) Online learning for strong branching approximation in branch-and-bound. Technical report, University of Liege.

Morrison DR, Jacobson SH, Sauppe JJ, Sewell EC (2016) Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization* 19:79 – 102, ISSN 1572-5286.

Pataki G, Tural M (2011) Basis reduction methods. Cochran JJ, Cox Jr LA, Keskinocak P, Kharoufeh JP, Smith JC, eds., *Wiley Encyclopedia of Operations Research and Management Science.*

Pisinger D (2005) Where are the hard knapsack problems? *Computers & Operations Research* 32(9):2271–2284.