A Theoretical and Computational Analysis of Full Strong-Branching

Santanu S. Dey*[1], Yatharth Dubey†[1], Marco Molinaro‡[2], and Prachi Shah§[1]

[1]School of Industrial and Systems Engineering, Georgia Institute of Technology
[2]Computer Science Department, Pontifical Catholic University of Rio de Janeiro

November 9, 2021

Abstract

Full strong-branching (henceforth referred to as strong-branching) is a well-known variable selection rule that is known experimentally to produce significantly smaller branch-and-bound trees in comparison to all other known variable selection rules. In this paper, we attempt an analysis of the performance of the strong-branching rule both from a theoretical and a computational perspective. On the positive side for strong-branching we identify vertex cover as a class of instances where this rule provably works well. In particular, for vertex cover we present an upper bound on the size of the branch-and-bound tree using strong-branching as a function of the additive integrality gap, show how the Nemhauser-Trotter property of persistency which can be used as a pre-solve technique for vertex cover is being recursively and consistently used through-out the strong-branching based branch-and-bound tree, and finally provide an example of a vertex cover instance where not using strong-branching leads to a tree that has at least exponentially more nodes than the branch-and-bound tree based on strong-branching. On the negative side for strong-branching, we identify another class of instances where strong-branching based branch-and-bound tree has exponentially larger tree in comparison to another branch-and-bound tree for solving these instances. On the computational side, we conduct experiments on various types of instances like the lot-sizing problem and its variants, packing integer programs (IP), covering IPs, chance constrained IPs, vertex cover, etc., to understand how much larger is the size of the strong-branching based branch-and-bound tree in comparison to the optimal branch-and-bound tree. The main take-away from these experiments is that for all these instances, the size of the strong-branching based branch-and-bound tree is within a factor of two of the size of the optimal branch-and-bound tree.

1 Introduction

The branch-and-bound scheme, invented by Land and Doig [30], is the method of choice for solving mixed integer linear programs (MILP) by all modern state-of-the-art MILP solvers.

*santanu.dey@isye.gatech.edu
†yatharthdubey7@gatech.edu
‡molinaro@inf.puc-rio.br
§prachi.shah@gatech.edu

We present a quick outline of this well-known method for binary MILPs below; see [43, 13] for more discussion on the branch-and-bound method. The optimal objective function value of the linear programming (LP) relaxation of a given MILP provides a dual bound (upper/lower bound for maximization-type/minimization-type objective respectively) on the optimal objective function value of the MILP. This LP relaxation corresponds to the root node of the branch-and-bound tree. In order to improve this bound and to find feasible solutions, after solving the LP corresponding to a node, the feasible region of the LP is partitioned into two sub-problems which correspond to the child nodes of the given node. Such a partitioning of the feasible region for binary MILPs is usually accomplished in practice by selecting a variable x_j that is currently fractional and adding the constraint $x_j = 0$ to one of the child nodes and the constraint $x_j = 1$ to the other child node. The process of partitioning the feasible regions of the LP at a node continues recursively for the child nodes (thus forming a tree) and is stopped (sometimes referred to as pruning a node), if one of the following conditions hold: (i) the LP at the node is infeasible, (ii) the LP's optimal solution is integer feasible, or (iii) the LP's optimal objective function value is worse than an already known integer feasible solution. The procedure terminates when all nodes have been pruned.

Given an underlying LP solver, formally speaking, the *branch-and-bound algorithm* is well-defined by fixing two rules:

- Rule for selecting an open node to be branched on next and,
- Rule for deciding the variable to branch on.

It is natural to measure the efficiency of a branch-and-bound algorithm by the number of nodes (corresponding to number of LPs solved) in the tree, i.e., lesser the number of nodes, faster the algorithm.

Selecting an open node to branch on next (node selection rule). It is well established [43] that the *worst-bound rule* (i.e., select node with the maximum LP optimal objective function value for a maximization-type MILP or select node with minimum LP optimal objective function value for a minimization-type MILP) for selecting the next node to branch on, leads to small branch-and-bound trees. The intuition behind this is the following: one cannot ignore the node with worst-bound if one wants to solve the MILP. Thus, it is best to select to branch on it first. In the rest of the paper, we always assume to use the worst-bound rule.

Deciding which variable to branch on (variable selection rule). Given that the rule for selecting the node to be branched on is well-understood, much of the research in the area of branch-and-bound algorithms has focused on the topic of deciding the variable to branch on – see for example [25, 16, 19, 7, 34, 21, 11, 23, 29, 20, 4, 32, 2]. Most of the above work develops various intricate greedy rules for determining the branching variable. A popular concept is that of *pseudocost branching*: the value of pseudocost (variable with largest pseudocost gets branched on) keeps a history of the success (in terms of improving dual bound) of the variables on which branching has already been done. Many of the papers cited above differ in how pseudocost is initialized and updated during the course of the branch-and-bound tree. Other successful methods like *hybrid branching* and *reliability branching* [2] are combinations of pseudocost branching and *full strong-branching*, that we discuss next.

The focus of this work is *full strong-branching* [4], henceforth referred to as strong-branching for simplicity. This rule works as follows: branching on all the current fractional variables is computed (i.e., the child nodes are solved for every choice of fractional variable) and improvement measured in the left and right child node. Branching is now done on the variable with the most 'combined improvement', where the combined improvement is computed as a 'score' function of the left and right improvement. Formally, let z be the optimal objective function value of the LP at a given node, and let z_j^0 and z_j^1 be the optimal objective function values of the LPs corresponding to the child nodes where the variable x_j is set to 0 and 1 respectively; we define $\Delta_j^+ := z - z_j^1$ and $\Delta_j^- := z - z_j^0$ (assuming the MILP's objective function is of maximizing-type). Note that $\Delta_j^+ = +\infty$ is the child node with x_j set to 0 is infeasible. Similarly for Δ_j^-. Two common score functions used are:

$$\text{score}_L(j) = (1-\mu)\cdot \min\{\Delta_j^-, \Delta_j^+\} + \mu \cdot \max\{\Delta_j^-, \Delta_j^+\}$$

for a constant $\mu \in [0,1]$, and

$$\text{score}_P(j) = \max(\Delta_j^+, \epsilon)\cdot \max(\Delta_j^-, \epsilon),$$

for a constant $\epsilon > 0$, where the first score is recommended in [32, 2] (the paper [2] recommends using $\mu = 1/6$) and the second score function is recommended in [1], where $\epsilon > 0$ is chosen close to 0 (for example, $\epsilon = 10^{-6}$) to break ties. We will refer to the first score function as the *linear score function* and the second score function as the *product score function*. Finally, the variable selected to branch on belongs to the set:

$$\arg\max_j \{\text{score}(j)\}.$$

Empirically, strong-branching is well-known to produce significantly smaller branch-and-bound trees [2] compared to all other known techniques, but is extremely expensive to implement as one has to solve $2K$ LPs where K is the number of fractional variables for making just one branching decision. This experimentally observed fact is so well established in the literature that almost all recent methods to improve upon branching decisions are based on using *machine learning techniques to mimic strong-branching* that avoid solving the $2K$ LPs, see for example [28, 33, 3, 5, 22, 24, 35]. Finally, see [31] that describes a more sophisticated way to decide the branching variable based on left and right improvement rather than a static 'score' function.

1.1 Our contributions

As explained in the previous section, empirically it is well understood that strong-branching produces very small trees in comparison to other rules. However, to the best of our knowledge there is *no understanding of how good strong-branching is in absolute terms.* In particular, we would like to answer questions such as:

- How large is the tree produced by strong-branching in comparison to the smallest possible branch-and-bound tree for a given instance? Answering this question may lead us to finding better rules.
- A more refined line of questioning: Intuitively, we do not expect strong-branching to work well for all types of MILP models and instances. On the other hand, it may be possible that for some classes of MILPs strong-branching based branch-and-bound tree may be quite

close to the smallest possible branch-and-bound tree. It would be, therefore, very useful to understand the performance of strong-branching vis-á-vis different classes of instances.

In this paper, we attempt an analysis of the performance of the strong-branching rule – both from a theoretical and a computational perspective, keeping in mind the above questions.

- Strong branching is provably good: We show that for the vertex cover problem, the strong-branching rule has several benefits. First, we present a fixed parameter-type (FPT) result that uses the additive gap between the IP's and the LP's optimal objective function value to bound the size of the branch-and-bound tree using strong-branching.

  Nemhauser and Trotter [37] proved that one may fix variables that are integral in the optimal solution of the LP relaxation of vertex cover and still find an optimal solution to the IP. Note that in the branch-and-bound tree, every node corresponds to a sub-graph of the original vertex-cover instance, and thus, ideally we would like to continue to use the Nemhauser-Trotter property at each node. Instead of designing a specialized implementation of branch-and-bound algorithm (where we fix variables that have an integer value in the LP optimal solution at each node), our second result is to show that strong-branching naturally incorporates "fixing" the integral variables of the LP solution recursively and consistently thoughout the branch-and-bound tree.

  Finally, we construct an instance where strong-branching yields a branch-and-bound tree that is exponential-times smaller than a branch-and-bound tree generated using a very reasonable alternative variable selection rule.

- Strong branching is provably bad: We present a class of instances where the size of the strong-branching based branch-and-bound tree is exponentially larger than a special branch-and-bound tree that solves these instances. In fact, the result we prove is stronger – we show that if we only branch on variables that are fractional, then the size of the branch-and-bound tree is exponentially larger than the given special branch-and-bound tree to solve these instances. This special branch-and-bound tree branches on variables that are integral in the optimal LP solutions at certain nodes.

- Computational evaluation of the size of strong-branching based branch-and-bound tree against the "optimal" branch-and-bound tree[1]: We first present a dynamic programming algorithm for generating the optimal branch-and-bound tree whose running time is $\text{poly}(\text{data}(\mathcal{I})) \cdot 3^{O(n)}$ where n is the number of binary variables. Then we conduct experiments on various types of instances like the lot-sizing problem and its variants, packing IPs, covering IPs, chance constrainted IPs, vertex cover, etc., to understand how much larger is the size of the strong-branching based branch-and-bound tree in comparison to the optimal branch-and-bound tree. The main take-away from these experiments is that for all these instances, the size of the strong-branching based branch-and-bound tree is within a factor of two of the size of the optimal branch-and-bound tree.

To the best of our knowledge, this is the first such study of this kind on strong-branching, that provides a better understanding of why strong-branching often performs so well in practice, and gives insight into when an instance may be challenging for strong-branching.

[1]We write "optimal" branch-and-bound tree with quotes, since the size of the optimal branch-and-bound tree depends not only on the branching decisions taken at each node, but also on the properties of the LP solver. We discuss this issue in detail in Section 2.

The rest of the paper is organized in the following fashion. In Section 2 we formally present all our theoretical and computational results. In Section 3 and Section 4, we present proofs of the theoretical results presented in Section 2. In Section 5 we present the details of the dynamic programming algorithm mentioned above. Finally, in Section 6 we present all the details of our computational experiments.

2 Main results

2.1 Theoretical results

Note that in this section (Section 2.1), whenever we refer to strong-branching, we assume that it has been used in conjunction with the product score function [1] score_P, where $\epsilon = 0$ and we use the convention that $0 \cdot \infty = 0$. Finally, we refer to the *size* of a branch-and-bound tree to denote its number of nodes; we note that in any binary tree, the number of nodes is at most $2\ell + 1$, where ℓ is the number of leaves.

2.1.1 Strong branching works well for vertex cover

There are simple MILPs that require exponential size branch-and-bound trees [26, 12, 18, 27, 15]. A common way to meaningfully analyze an algorithm with exponential worst-case performance is to show its performance with respect to some parameter [42, 14]. Arguably the most well-studied problem in parameterized complexity is *vertex cover*.

Definition 1 (Vertex cover)**.** *The vertex cover problem over a graph $G = (V, E)$ can be expressed as the following integer program (IP)*

$$\begin{aligned} \textit{min} \quad & \sum_{v \in V} x_v \\ \textit{s.t.} \quad & x_u + x_v \geq 1, \quad uv \in E \\ & x_v \in \{0, 1\}, \quad v \in V \end{aligned}$$

Given an instance $\mathcal{I}$ of this IP, we let $L(\mathcal{I})$ denote its LP relaxation (i.e. when the variable constraints instead are $x_v \in [0, 1]$). We denote the optimal objective function value of an instance by $\text{OPT}(\mathcal{I})$ and the optimal objective function value of its LP relaxation by $\text{OPT}(L(\mathcal{I}))$. We denote its additive integrality gap $\Gamma(\mathcal{I}) := \text{OPT}(\mathcal{I}) - \text{OPT}(L(\mathcal{I}))$. For results pertaining to vertex cover, we use n to denote the number of vertices (i.e. $n := |V|$).

Upper bound on the size of branch-and-bound tree using strong-branching. Let $\mathcal{T}_S(\mathcal{I})$ represent a branch-and-bound tree for solving instance $\mathcal{I}$ using strong-branching. Unfortunately, the number of nodes in this tree depends not only on the node selection rule and variable selection rule, but also on the underlying LP solver as well. For example, a solver may report an integral solution at a given node and allow us to prune the node. Another solver might report a different optimal solution to the LP, which is not integral. Therefore, we will be careful to not refer to *the branch-and-bound tree generated by strong-branching*. Instead, we will use $\mathcal{T}_S(\mathcal{I})$ to represent

some branch-and-bound tree generated by strong-branching. As such, we conduct a parameterized analysis of strong-branching as an algorithm for vertex cover.

We show an upper bound on strong-branching for vertex cover parameterized by its additive integrality gap $\Gamma(\mathcal{I})$.

Theorem 1. *Let $\mathcal{I}$ be any instance of vertex cover. Assume we break ties within the worst-bound rule for node selection rule by selecting a node with the largest depth. Let $\mathcal{T}_S(\mathcal{I})$ be some branch-and-bound tree generated by strong-branching with the above version of worst-bound node selection rule that solves $\mathcal{I}$. Then independent of the underlying LP solver used,*

$$|\mathcal{T}_S(\mathcal{I})| \leq 2^{2\Gamma(\mathcal{I})+2} + \mathcal{O}(n).$$

Moreover, it is impossible to find another branch-and-bound rule that has a much better upper bound, so in this sense strong-branching is in the worst-case almost optimal for vertex cover parametrized by integrality gap. This is because of the following bad example.

Remark 1. *There is an instance $\mathcal{I}$ of vertex cover such that any branch-and-bound tree that solves $\mathcal{I}$ has size $2^{2\Gamma(\mathcal{I})+1} - 1$.*

This is the instance of m disjoint triangles presented in [6]. Note that the smallest vertex cover in this instance has value $2m$ while the optimal solution to the LP relaxation has value $\frac{3}{2}m$, therefore $\Gamma(\mathcal{I}) = \frac{1}{2}m$. It follows from the discussion in [6], that all branch-and-bound trees for this instance have 2^m leaves, i.e., at least $2^{m+1} - 1$ nodes.

We note that the result of Theorem 1 matches the guarantee of the classic parameterized-complexity algorithm that uses bounded search trees tailored to this problem; see Theorem 3.8 of [14].

We present the proof of Theorem 1 in Section 3.1.

Strong branching and persistency. Nemhauser and Trotter [37] prove the following property regarding the LP relaxation of vertex cover.

Fact 1 (Persistency; Theorem 2 of [37])**.** *Let $\mathcal{I}$ be an instance of vertex cover, $\hat{x}$ be an optimal solution of $L(\mathcal{I})$ and I be the set of variables on which $\hat{x}$ is integer (i.e. $I = \{j : \hat{x}_j \in \{0, 1\}\}$). Then, there exists an optimal solution $\hat{y}$ to $\mathcal{I}$ such that $\hat{y}$ agrees with $\hat{x}$ on all of its integer components (i.e. $\hat{y}_j = \hat{x}_j$ for all $j \in I$).*

One way to use this property is to use it as a pre-solve routine, i.e., fix variables that are integral in the optimal solution of the LP relaxation, and then work with the sub-graph induced by the vertices with value $\frac{1}{2}$ in the optimal solution of the LP. (The extreme points of the LP relaxation of the vertex cover problem are half integral [36].) However, note that in the branch-and-bound tree, every node corresponds to a sub-graph of the original vertex-cover instance, and thus, ideally we would like to continue to use the Nemhauser-Trotter property at each node. Instead of designing a specialized implementation of branch-and-bound algorithm (where we fix variables that have an integer value in the LP optimal solution at each node), we show that strong-branching naturally incorporates "fixing" the integral variables of the optimal solution of LP at each node recursively throughout the branch-and-bound tree. In fact, it does even better in the following sense: due

to dual degeneracy, there may be alternative linear programming optimal solutions with different corresponding sets of variables being integral. Strong branching "avoids branching" on all the variables that are integral in any of the alternative optimal LP solutions, i.e., strong-branching is not fooled by the LP solver.

In order to present our results, we need to define the notion of *maximal set of integer variables* and present some properties regarding this set of variables.

Fact 2 (Lemma 1 in [39])**.** *Consider instance of vertex cover and its LP relaxation. Let x^1, x^2 be two optimal solutions to this LP relaxation and let $I^1, I^2 \subseteq [n]$ be the indices of the integer valued variables in x^1, x^2 respectively. Then, there exists an optimal solution of the LP relaxation, $\hat{x}$, such that the set of integer valued variables in $\hat{x}$ is $I = I^1 \cup I^2$.*

Based on the above fact we obtain the following observation: Given a vertex cover instance, all optimal solutions of the LP relaxation that have a maximal number of integral coordinates actually have the same set $I \subseteq [n]$ of integral coordinates. This, given an instance of vertex cover $\mathcal{I}$, we call this subset the *maximal set of integer variables* and denote as $I(\mathcal{I})$. Fact 1 then implies that there exists an optimal solution to the vertex-cover instance where the *maximal set of integer variables* are fixed to integer values from the corresponding values of an maximal optimal solution of the LP relaxation.

As discussed before, in a branch-and-bound tree, each node corresponds to a vertex cover instance on a sub-graph. Therefore, we can define the notion of maximal set of integer variables at a given node N, which we denote as $I(\mathcal{I}, N)$. Given an instance of vertex cover $\mathcal{I}$, we refer to $\mathcal{TP}(\mathcal{I})$ as a partial branch-and-bound tree for $\mathcal{I}$ if all the nodes of $\mathcal{TP}(\mathcal{I})$ cannot be pruned. For such a partial branch-and-bound tree, we use $\mathcal{TP}^B(\mathcal{I})$ to refer to the dual bound that one can infer from the partial tree.

The strength of strong-branching with regards to the maximal set of integer variables at a node N is explained by the next result: Let j belong to the maximal set of integer variables at node N. If the LP solver returns an optimal solution with x_j integral, then clearly we do not branch on this variable. However, if the LP solver returns an optimal solution where x_j is fractional and we decide on branching on this variable based on strong-branching, then it must be that we have "nearly solved" the instance, i.e., the dual bound that can be inferred from the partial tree must be equal to the optimal objective function value of the instance. Formally we have the following:

Proposition 1. *Let $\mathcal{I}$ be any instance of vertex cover. Assume we break ties within the worst-bound rule for node selection by selecting a node with the largest depth. Consider a partial tree $\mathcal{TP}_S$ generated by strong-branching with the above version of worst-bound node selection rule. Let N be a node of this tree that is not pruned. If $j \in I(\mathcal{I}, N)$ and we decide to branch on variable x_j at node N (using strong-branching), then*

1. *$\mathcal{TP}_S^B(\mathcal{I}) = \mathrm{OPT}(\mathcal{I})$.*
2. *After branching on x_j,in at most $\mathcal{O}(n)$ further branchings the algorithm returns an integral optimal solution.*

The above result shows that we "almost never" branch on maximal set of integer variables at a node if we use strong-branching. However, after branching on a variable that is not in the maximal

set of integer variables, what happens to the set of maximal set of integer variables at the child nodes? A very favorable property would be if the maximal set of integer variables of the parent node is inherited by the child nodes. Otherwise, while we may not branch on x_j where $j \in I(\mathcal{I}, N)$ at node N, but we may end up branching on j for a child N' of N — in other words, we are then not really "fixing" variable x_j. As it turns out the above bad scenario does not occur. Formally we have the following:

Theorem 2. *Let $\mathcal{I}$ be an instance of vertex cover. Consider any internal node N of $\mathcal{T}_S(\mathcal{I})$ and let N' be a child node of N that results from branching on x_v where $v \notin I(\mathcal{I}, N)$. Then, $I(\mathcal{I}, N) \subset I(\mathcal{I}, N')$.*

Therefore, using Proposition 1 and Theorem 2 together, we can conclude that when using strong-branching, we are essentially repeatedly using Nemhauser-Trotter property recursively and consistenly through-out the branch-and-bound tree: If j is in the maximal set of integer variables at node N, it continues to remain in the maximal set of integer variables for all the child nodes of N; and we do not branch on such a variable x_j at node N or any of its children unless the instance is essentially solved (i.e., dual bound inferred from the tree equals the values of the IP).

Here we present an example to illustrate that there are variable selection rules for which the property described in Theorem 2 does not hold. See the instance $\mathcal{I}^*$ shown in Figure 1. Consider the following partial branch-and-bound tree, letting N denote the root node. Observe that there is an optimal LP solution that sets $x_a = x_c = 1$ and $x_b = x_d = 0$; therefore, $I(\mathcal{I}^*, N) = \{a, b, c, d\}$. However, suppose that an adversarial LP solver returns the optimal basic feasible solution $x_a = x_b = x_c = x_d = \frac{1}{2}$. Suppose we branch on x_b and consider the sub-problem resulting from $x_b = 1$, which we denote N'. The unique optimal solution to N' sets $x_b = 1$ and $x_a = x_c = x_d = \frac{1}{2}$; therefore, $I(\mathcal{I}^*, N') = \{b\} \subset I(\mathcal{I}^*, N)$. In fact, this example can be easily extended to that of Theorem 3 (see below) to show that this variable selection rule similarly results in an exponentially larger branch-and-bound tree as compared to one that is obtained by strong-branching.

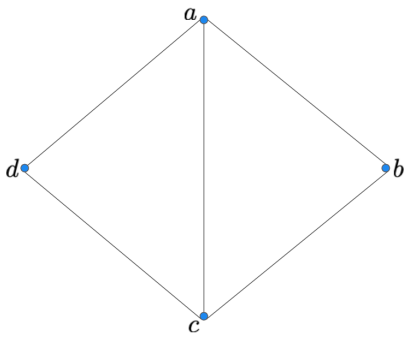

Figure 1: Instance $\mathcal{I}^*$: example illustrating that the property of Theorem 2 is not true for every variable selection rule.

We present proof of Proposition 1 and Theorem 2 in Section 3.2.

Superiority of strong-branching. There are very few papers that give upper bounds on sizes of branch-and-bound tree (when we use 0-1 branching) [17, 10]. These papers show that certain class of IPs with random data can be solved using polynomial-size branch-and-tree with high probability. However, these results do not depend on the variable selection rule used. Theorem 1 above is the

first result of its kind that we are aware of, which uses a specific variable selection rule to prove upper bounds on size of branch-and-bound tree. To further highlight the importance of strong-branching in obtaining this upper bound result, we next show that if we do not use the strong-branching rule and we have an "adversarial" LP solver, then we may need exponentially larger trees to solve the instance.

In particular, we next demonstrate the superiority of strong-branching by comparing it with another variable selection rule for the vertex cover problem (which we call the *greedy-rule*) when the LP solver is adversarial, i.e. the LP solver always gives the most fractional extreme point solution. The greedy-rule for variable selection is intuitive and natural for vertex cover: branch on the vertex with most fractional neighbors. Since setting such a vertex to 0 would enforce all of its fractional neighbors to be 1, one might think this rule provides the most "local improvement".

Theorem 3. *Consider an LP solver with the following property: Among all optimal extreme point solutions, it reports an optimal extreme point with maximal number of fractional components. Given an instance $\mathcal{I}$ and the above LP solver, let $\mathcal{T}_S(\mathcal{I})$ and $\mathcal{T}_G(\mathcal{I})$ be some branch-and-bound trees that solves instance $\mathcal{I}$ obtained using the strong-branching rule and the greedy rule respectively.*

There is an instance $\mathcal{I}^*$ of vertex cover such that

$$|\mathcal{T}_G(\mathcal{I}^*)| \geq 2^{\Omega(n)} \cdot |\mathcal{T}_S(\mathcal{I}^*)|$$

and furthermore

$$|\mathcal{T}_G(\mathcal{I}^*)| \geq 2^{cst \cdot \Gamma(\mathcal{I}^*)}$$

where cst is a constant strictly greater than 2.

We present the proof of Theorem 3 in Section 3.3.

2.1.2 Strong branching does not work well for some instances

Next we present a negative result regarding strong-branching, showing that strong-branching based branch-and-bound tree can have an exponential times as many nodes as compared to number of nodes in an alternative tree. In fact, the example shows something even stronger: any tree that branches only on variables fractional in the current nodes optimal solution will have exponential size, while an alternative tree has linear size.

We begin by showing a seemingly surprising result about the existence of an extended formulation for any binary IP, that leads to a linear size branch-and-bound tree.

Proposition 2. *For any integer program $\mathcal{I}$ with n binary variables, there is an equivalent integer program that uses an extended formulation of the feasible region of $\mathcal{I}$ with $2n$ binary variables, which we refer to as $BDG(\mathcal{I})$, that has the following property: there exists a branch and bound tree $\mathcal{T}^*(BDG(\mathcal{I}))$ that solves the instance $BDG(\mathcal{I})$ and $|\mathcal{T}^*(BDG(\mathcal{I}))| \leq 4n + 1$.*

The extended formulation corresponding to $BDG(\mathcal{I})$ used in Proposition 2 was first introduced in [9] to show that every binary integer program has an extended formulation with split rank of 1. We also remark here that Proposition 2 does not imply that the decision version of binary IPs is in

co-NP. This is because the $BDG(\mathcal{I})$ formulation may be of exponential-size in comparison to the original formulation.

In Corollary 1 below, we take the cross-polytope [18] and apply the extended formulation of Proposition 2 to obtain an example where strong-branching based branch-and-bound tree can have an exponential times as many nodes as compared to number of nodes in an alternative tree.

Corollary 1. *There exists an instance $\mathcal{I}^*$ with $2n$ binary variables, such that the following holds: Let $\mathcal{T}(\mathcal{I}^*)$ be any tree that solves $\mathcal{I}^*$ satisfying the following property: if x is the optimal solution to an internal node N of $\mathcal{T}(\mathcal{I}^*)$, then the variable j branched on at N must be such that $x_j \in (0,1)$. Then, $|\mathcal{T}(\mathcal{I}^*)| \geq 2^{n+1} - 1$. In particular, if $\mathcal{T}_S(\mathcal{I}^*)$ is a branch-and-bound tree generated using strong-branching that solve $\mathcal{I}^*$, then $|\mathcal{T}_S(\mathcal{I}^*)| \geq 2^{n+1} - 1$. On the other hand, there exists a tree $\mathcal{T}^*$ that solves $\mathcal{I}^*$ such that $|\mathcal{T}^*(\mathcal{I}^*)| \leq 4n + 1$.*

Typically, when implementing a branch-and-bound algorithm, one might be inclined to restrict the algorithm to branch on variables that are fractional in the current optimal solution. Therefore the above example is counter-intuitive in that it shows there can be a significant separation between branch-and-bound trees that are restricted to branch on variables that are fractional in the current optimal solution and branch-and-bound trees that are allowed to branch on integer valued variables. To our knowledge, this is the first such explicit example in the literature.

We present a proof of Proposition 2 and Corollary 1 in Section 4.

2.2 Computational results

In the previous section, we have shown that strong-branching works well for vertex cover and on the other hand, strong-branching can sometimes produce exponentially larger trees than alternative trees to solve an instance. However, in general it seems very difficult to analyze strong-branching for general MILPs on a case-to-case basis. Moreover, we would really like to answer the question: how good is the strong-branching based branch-and-bound tree in comparison to the optimal tree? In this section we try to shed light on this question using computational experiments.

Optimal branch-and-bound tree. We begin with a discussion of an "optimal branch-and-bound tree" for a given instance.

First note that it is clear that the optimal branch-and bound tree uses the worst bound rule for node selection [43]. Moreover, we will consider branching on all variables at a given node, whether it is integral or not in the optimal solution of the LP relaxation. Since we are using the worst bound rule for node selection, we only branch on nodes whose objective function value is at least as good as that of the MILP optimal objective function value.

Now consider a node whose LP optimal objective function value is equal to that of the MILP optimal objective function value. There are two possible scenarios:

- The optimal face of this LP is integral, i.e., all the vertices of the optimal face are integer feasible and therefore the LP solver (like simplex) is guaranteed to find an integral solution and prune the node.

- At least one vertex of the optimal face is fractional. In this case, depending on whether the LP solver returns an integral vertex or not, we are able to prune the node or not. Moreover, if we arrive at a fractional vertex, then depending on properties of other nodes and how we break ties for node selection among nodes with same objective function value, we may or may not end up branching on this node.

In other words, in the second case, the size of tree may depend on the type of optimal vertex reported by the LP solver. In order to simplify our analysis and to remove all ambiguity regarding the definition of the optimal branch-and-bound tree, we make the following assumption for results presented in this section.

Assumption 1. *We assume that if there exists an optimal solution to the LP relaxation at a given node that is integral, then the LP solver finds it.*

Observation 1. *When using an LP solver which satisfies Assumption 1, a branch-and-bound tree using the worst bound rule never branches on a node whose optimal objective function value is equal to that of the IP solver.*

Proof of Observation 1 is presented in Section 5.1. The above observation clearly makes the notion of "optimal branch-and-bound tree" for a given instance well-defined.

Dynamic programming algorithm for finding the optimal branch-and-bound tree. In Section 5.2 we present a dynamic programming (DP) algorithm that computes an optimal branch-and-bound tree for a given instance under Assumption 1 for the LP solver.

Theorem 4. *Under Assumption 1, there exists an algorithm with running time* $\text{poly}(data) \cdot 3^{O(n)}$ *time to compute an optimal branch-and-bound tree for any binary MILP instance* $\mathcal{I}$ *defined on* n *binary variables.*

Section 5.2 also presents some computational enhancements (like exploiting parallel computing) to improve the wall clock run time of the DP algorithm.

Computationally comparison of various variable selection rules against the optimal branch-and-bound tree. We conduct the first study comparing branch-and-bound trees employing different variable selection rules to the optimal branch-and-bound tree. Detailed results are presented in Section 6.

We consider the following variable selection rules: strong-branching with linear score function, strong-branching with product score function, most infeasible, and random. We evaluate the performance of these rules on a wide range of problems: general packing and covering IPs (packing-type, covering-type, and mixed packing and cover instances), lot-sizing and variants, vertex cover, chance constraint programming (CCP) models for multi-period power planning and portfolio optimization, and stable set on a bipartite graph with knapsack sides constraint (100 instances for each model). Note that all of these instances have up to 20 binary variables since we are unable to run the DP algorithm of Theorem 4 for larger instances. See Section 6.1 for a detailed description of all these instances.

We present a discussion of all our results (together with tables and explanatory figures) in Section 6.2. Here are some of our notable findings on sizes of branch-and-bound tree trees:

- Random consistently performs the worst.
- Strong branching *always* performs the best.
- While the performance of two variants of strong-branching is comparable on all problems considered in this study, strong branching with product score function (SB-P) dominates over strong branching with linear score function (SB-L) on 8 out of 10 problems, although by a small margin.
- The geometric mean of branch-and-bound tree size using strong-branching remains less than twice the size of optimal branch-and-bound tree for all problems considered in this study. This is not that case for all branching rules: the Random rule (and sometimes even the most-infeasible rule) is typically many more times larger than the optimal branch-and-bound tree.

Finally, see Figure 2 for a summary of the computational results. It is clear that while strong-branching does quite well (within a factor of 2), there is clearly scope for coming up with better rules for deciding branching variables, for example for problems such as CCP portfolio optimization. It would be interesting to see if one can use machine learning techniques to learn from the optimal branch-and-bound trees.

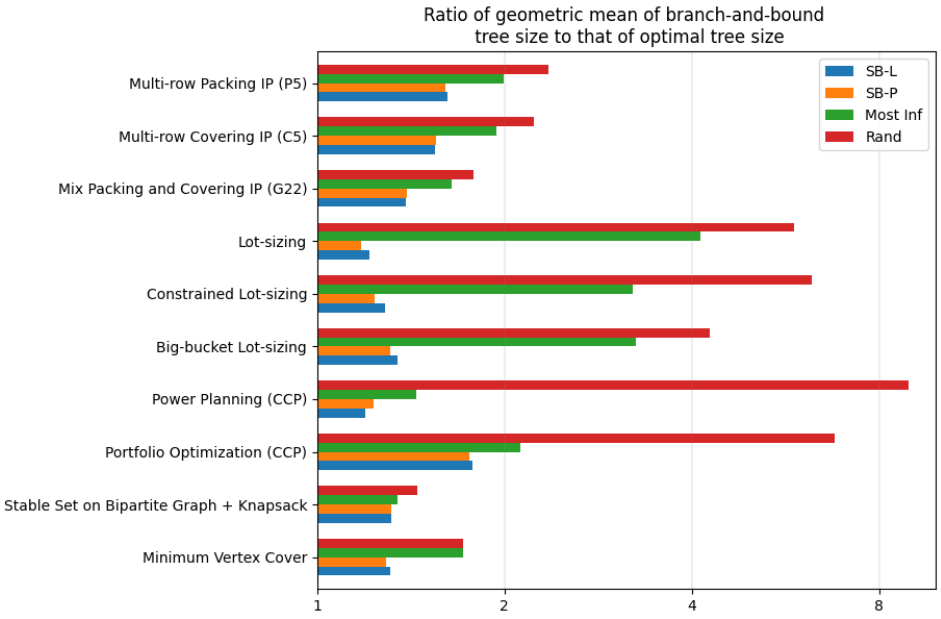

Figure 2: Ratio of geometric mean of branch-and-bound tree sizes to geometric mean of optimal tree sizes over all instances of a problem for various branching strategies. "Rand" stands for random, "Most Inf" stands for most infeasible, "SB-P" stands for strong-branching with product score function, and "SB-L" stands for strong-branching with linear score function.

In previous section (see Corollary 1), we have seen an example where strong-branching performs badly while alternative branch-and-bound tree that has exponentially lesser nodes, branches on

integer variables. So a natural question we would like to understand is the percentage of times the optimal branch-and-bound tree branches on integer variables for the various instances. These results are presented in Table 1. We note that there are multiple optimal trees. So it may be possible that these exists other optimal trees with a slightly different number of branchings on integer variables.

Here are some of our notable findings:

- Strong branching does relatively poorly on general packing and covering IPs which has a high fraction of integer branchings in the optimal tree and it does very well on lot-sizing where the optimal tree rarely branches on integers. So it would appear consistent with our hypothesis that more branchings on integer variables in the optimal tree implies strong-branching performs poorly.
- On the other hand, for stable set on bipartite graph with knapsack side constraint, strong-branching does very well in spite of a lot of integer branchings in the optimal tree.

Table 1: Summary of average tree sizes (geometric) and percentage of branching on integral variable in optimal tree for all problems across 100 instances

Problem	Opt Tree	SB-L	SB-P	Most inf	Rand	% Int Branch
Multi-row Packing IP (P5)	25.4	41.2	40.9	50.8	59.9	48.2%
Multi-row Covering IP (C5)	25.2	39.0	39.1	48.9	56.4	49.0%
Mix Packing and Covering IP (G22)	10.2	14.2	14.2	16.8	18.3	48.0%
Lot-sizing	111.5	135.0	131.1	461.0	651.6	0.2%
Constrained Lot-sizing	101.9	131.1	125.9	328.0	635.0	0.2%
Big-bucket Lot-sizing	81.8	110.3	107.1	266.4	349.6	1.1%
Power Planning (CCP)	37.9	45.3	46.8	54.8	337.9	1.1%
Portfolio Optimization (CCP)	97.4	172.8	171.1	206.5	659.1	4.4%
Stable Set on Bipartite Graph + Knapsack	137.6	180.8	180.8	185.5	199.6	47.7%
Minimum Vertex Cover	7.1	9.3	9.1	12.1	12.2	0.0%

In other words, there does not seem to be a direct relationship between the performance of strong-branching and the number of branching on integer variable of the optimal branch-and-bound tree.

Finally we end this section with a word of caution regarding over-interpreting the computational results above: As mentioned above, due of the exponential nature of the DP algorithm to compute the optimal branch-and-bound tree, computational experiments could be performed only on relatively smaller problem sizes with up to 20 binary variables. Some of the observations derived here may not extrapolate to larger instances.

3 Analysis of Strong Branching for Vertex Cover

3.1 Proof of Theorem 1.

Throughout this section, we use $N_{j,0}$ to denote the child node of N that results from the branch $x_j = 0$; we use $N_{j,1}$ similarly.

Throughout this section we make the following assumptions. Let $\mathcal{I}$ be any instance of vertex cover. Assume we break ties within the worst-bound rule for node selection rule by selecting a node with the largest depth. Let $\mathcal{T}_S(\mathcal{I})$ be *some branch-and-bound tree generated by strong-branching* with the above version of worst-bound node selection rule that solver $\mathcal{I}$. Moreover, all results are independent of the underlying LP solver used.

We will require two preliminary results for the proof of Theorem 1.

Lemma 1. *Let N be a node of $\mathcal{T}_S(\mathcal{I})$ with optimal objective value less than $\mathrm{OPT}(\mathcal{I})$. Then strong-branching will branch on $v \notin I(\mathcal{I}, N)$ at node N and $N_{v,0}, N_{v,1}$ have optimal value at least $\frac{1}{2}$ more than that of N.*

Proof. Since N has optimal objective value less than $\mathrm{OPT}(\mathcal{I})$, the LP relaxation at N must not have an optimal solution that is integer. Note that if at N we branch on x_u where $u \in I(\mathcal{I}, N)$, it must hold that, at least one of $N_{u,0}$ or $N_{u,1}$ have optimal value the same as N. Therefore, $\mathrm{score}_P(u) = 0$. We now show that there exists $v \in [n]$ such that $\mathrm{score}_P(v) > 0$. If there is no optimal solution to N that is integer, then $I(\mathcal{I}, N) \neq [n]$ by Fact 2. Therefore, there exists a $v \notin I(\mathcal{I}, N)$ such that $x_v = \frac{1}{2}$ in every optimal solution to N. It follows that fixing $x_v \in \{0, 1\}$ must result in a feasible solution to N that has strictly greater value, and so branching on x_v at N leads to child nodes $N_{v,0}$ and $N_{v,1}$ where both have optimal value more than that of N. Further, since $N_{v,0}$ and $N_{v,1}$ have objective value strictly more than N and all basic feasible solutions are half-integral, it holds that $N_{v,0}$ and $N_{v,1}$ have objective value at least $\frac{1}{2}$ more than N. □

Lemma 2. *Let N be a node of $\mathcal{T}_S(\mathcal{I})$ with an optimal LP solution that is integer. Then the sub-tree rooted at N will have size $\mathcal{O}(n)$, where it finds an integer optimal solution.*

Proof. Let $y \in \{0, 1\}^n$ be an optimal solution to the LP of node N. If the LP solver returns an integer solution, we are done. Suppose not, and suppose we branch on some variable v, and without loss of generality let $y_v = 0$. Then, of course $N_{v,0}$ will have y as an optimal solution and therefore have value $\mathrm{OPT}(\mathcal{I})$. If $N_{v,1}$ has value greater than $\mathrm{OPT}(\mathcal{I})$, this node gets pruned by bound and we continue by branching on $N_{v,0}$ since it is now the open node with largest depth. Suppose instead $N_{v,1}$ has value $\mathrm{OPT}(\mathcal{I})$. Then, we argue in the next paragraph that there is an optimal solution to the LP corresponding to N that is integer feasible, denote z, with $z_v = 1$; in this case we can break the tie between $N_{v,0}, N_{v,1}$ arbitrarily. Then, for every node N' in the sub-tree rooted at N, either one child of N' has value greater than $\mathrm{OPT}(\mathcal{I})$ and is pruned by bound, or both children of N' have optimal solutions that are integer. In the second case, only one of these two children will ever be branched on, since we break ties by branching on the node with largest depth, and therefore will find an integer solution before revisiting a shallower node. The result of the lemma follows.

Here we argue that if $N_{v,1}$ has a LP optimal objective function value $\mathrm{OPT}(\mathcal{I})$, then there is an optimal solution to the LP corresponding to N that is integral, denote z, with $z_v = 1$. Let x' denote any optimal solution of $N_{v,1}$ and observe that x', y are both optimal solutions to N. It follows from the proof of Lemma 1 in [39] that: given two half-integral optimal solutions x', y, there exists an optimal solution z constructed as follows:

$$z_j = \begin{cases} 1 & \text{if } x'_j = 1 \text{ or } x'_j = \frac{1}{2}, y_j = 1 \\ 0 & \text{if } x'_j = 0 \text{ or } x'_j = \frac{1}{2}, y_j = 0 \\ \frac{1}{2} & \text{if } x'_j = y_j = \frac{1}{2} \end{cases}$$

In particular, [39] shows that that z constructed as above is feasible and satisfies, $\sum_j z_j = \sum_j x'_j = \sum_j y_j$. Clearly $z_v = x'_v = 1$ and since $y \in \{0,1\}^n$, it follows that $z \in \{0,1\}^n$ and is an optimal solution of the LP corresponding to N. □

Theorem 1. *Let $\mathcal{I}$ be any instance of vertex cover. Assume we break ties within the worst-bound rule for node selection rule by selecting a node with the largest depth. Let $\mathcal{T}_S(\mathcal{I})$ be* some branch-and-bound tree generated by strong-branching *with the above version of worst-bound node selection rule that solver $\mathcal{I}$. Then independent of the underlying LP solver used,*

$$|\mathcal{T}_S(\mathcal{I})| \leq 2^{2\Gamma(\mathcal{I})+2} + \mathcal{O}(n).$$

Proof. Fix any branch-and-bound tree (using strong-branching) for $\mathcal{I}$. Let N be a node at depth $2\Gamma(\mathcal{I})+1$. Suppose that no ancestor of N had an optimal LP solution that was integer; it follows from Lemma 1 that N has optimal objective value at least $\mathrm{OPT}(\mathcal{I}) + \frac{1}{2}$. Denote the set of such nodes $\mathcal{N}_{\text{no ancestor}}$ and note that these nodes are pruned by bound. Observe all other nodes (if any) at depth $2\Gamma(\mathcal{I})+1$ do have such an ancestor. Let $\mathcal{N}_{\text{integer}}$ denote the set of nodes at depth $\leq 2\Gamma(\mathcal{I})$ that have an optimal solution that is integer, but none of its ancestors have an optimal solution that is integer. Finally, observe that $|\mathcal{N}_{\text{no ancestor}}| + |\mathcal{N}_{\text{integer}}| \leq 2^{2\Gamma(\mathcal{I})+1}$. We conclude the proof by observing that Lemma 2 gives: if N is the node in $\mathcal{N}_{\text{integer}}$ with largest depth, then the sub-tree rooted at N has size $\mathcal{O}(n)$ where it finds an integer solution with value $\mathrm{OPT}(\mathcal{I})$. Therefore, no other nodes in $\mathcal{N}_{\text{integer}}$ will be branched on. Since the number of leaves in this tree is at most $2^{2\Gamma(\mathcal{I})+1} + \mathcal{O}(n)$, the result of the theorem follows.

□

3.2 Proof of Proposition 1 and Theorem 2.

Proposition 1. *Let $\mathcal{I}$ be any instance of vertex cover. Assume we break ties within the worst-bound rule for node selection by selecting a node with the largest depth. Consider a partial tree $\mathcal{TP}_S$ generated by strong-branching with the above version of worst-bound node selection rule. Let N be a node of this tree that is not pruned. If $j \in I(\mathcal{I}, N)$ and we decide to branch on variable x_j at node N (using strong-branching), then*

1. *$\mathcal{TP}_S^B(\mathcal{I}) = \mathrm{OPT}(\mathcal{I})$.*
2. *After branching on x_j, in at most $\mathcal{O}(n)$ further branchings the algorithm returns an integral optimal solution.*

Proof. We begin by proving property 1. Suppose for sake of contradiction, there exists an open node N' in $\mathcal{TP}_S^B(\mathcal{I})$ with optimal objective value less than $\mathrm{OPT}(\mathcal{I})$. It follows from the worst-bound rule and Lemma 1 that strong-branching will choose to branch on j' at N' with $j' \notin I(\mathcal{I}, N')$. Thus j' and N' are not the same as j and N, giving the desired contradiction.

Property 2 follows directly from Lemma 2 and the fact that $I(\mathcal{I}, N) = [n]$, since otherwise strong-branching would branch on some $j \notin I(\mathcal{I}, N)$. □

Theorem 2. *Let $\mathcal{I}$ be an instance of vertex cover. Consider any internal node N of $\mathcal{T}_S(\mathcal{I})$ and let N' be a child node of N that results from branching on x_v where $v \notin I(\mathcal{I}, N)$. Then, $I(\mathcal{I}, N) \subset I(\mathcal{I}, N')$.*

Proof. Throughout this proof, we use $N_{j,0}$ to denote the child node of N that results from the branch $x_j = 0$; we use $N_{j,1}$ similarly. We use $\delta(S)$ to denote the neighbors of any subset of vertices $S \subseteq V$. Also, throughout the proof, let x^* be an optimal solution to N with maximal set of integer components (i.e. $x^*_j \in \{0, 1\}$ for all $j \in I(\mathcal{I}, N)$).

We will first consider the child $N_{v,1}$. Consider the solution y to $N_{v,1}$ constructed from x^* as follows: y takes the same values as x^*, but $y_v = 1$ instead of $\frac{1}{2}$. Note that this is feasible for $N_{v,1}$. Since $v \notin I(\mathcal{I}, N)$ we have that $x_v = \frac{1}{2}$ in every optimal solution to N. In other words, optimal objective function value of $N_{v,1}$ is at least $\frac{1}{2}$ more than that of N. Also, $\langle 1, y\rangle = \langle 1, x^*\rangle + \frac{1}{2}$, and so, y is an optimal solution to $N_{v,1}$. So $I(\mathcal{I}, N) \subset I(\mathcal{I}, N')$ follows in the case of $N' = N_{v,1}$.

We will now consider the child $N_{v,0}$. Let $V^*_1 = \{u \in V : x^*_u = 1\}$ and V^*_0 be defined similarly. We will need the following technical result later in the proof.

Lemma 3. *$|\delta(S) \cap V^*_0| \geq |S|$ for all $S \subseteq V^*_1$.*

Proof. Assume, for the sake of contradiction, there is a subset $S \subseteq V^*_1$ such that $|\delta(S) \cap V^*_0| < |S|$, then it must hold that x^* is not an optimal solution. This is because, we can set all of S and $\delta(S) \cap V^*_0$ to be $\frac{1}{2}$. This remains feasible, since we are only decreasing the value of the vertices in S, but all of the vertices with value 0 adjacent to these vertices are also being raised to $\frac{1}{2}$. It also decreases the cost, since $\frac{1}{2}(|\delta(S) \cap V^*_0| + |S|) < |S|$. □

Let x' be any optimal solution to LP corresponding to $N_{v,0}$. Let $\Phi = \{u \in V^*_1 : x'_u \neq 1\}$. We will construct a feasible solution z such that $z_u = 1$ for all $u \in V^*_1$ with cost no more than that of x'. Note that since $\delta(V^*_0) \subseteq V^*_1$, this implies the existence of an optimal solution to $N_{v,0}$ with the same integer variables as x^* (since any optimal solution with all variables in V^*_1 set to 1 will have all variables in V^*_0 set to 0).

We begin by constructing an intermediate solution y. Notice Φ can be partitioned into $\Phi^0 = \{u \in \Phi : x'_u = 0\}$ and $\Phi^{\frac{1}{2}} = \{u \in \Phi : x'_u = \frac{1}{2}\}$. Let y be defined by setting the variables in Φ^0 to $\frac{1}{2}$ and setting the variables in $\delta(\Phi^0) \cap V^*_0$ to $\frac{1}{2}$ (note that $x'_u = 1$ for all $u \in \delta(\Phi^0) \cap V^*_0$). This maintains feasibility, since the variables with decreasing value are a subset of V^*_0, which are only adjacent to vertices in V^*_1 which now all have value at least $\frac{1}{2}$ in y. This also maintains optimality, since the cost increases by $\frac{1}{2}|\Phi^0|$ and decreases by $\frac{1}{2}|\delta(\Phi^0) \cap V^*_0|$, and by Lemma 3, $|\delta(\Phi^0) \cap V^*_0| \geq |\Phi^0|$. We now construct z similarly, from this intermediate solution y. Notice now, that all variables in Φ have value $\frac{1}{2}$ in y. We construct z by setting all of these variables to 1 and setting all variables in $\delta(\Phi) \cap V^*_0$ to 0 (note that $y_u \geq \frac{1}{2}$ for all $u \in \delta(\Phi) \cap V^*_0$). This maintains feasibility, since the variables with decreasing value are a subset of V^*_0, which are adjacent only to vertices in V^*_1, which all have value at least 1 in z. This also maintains optimality, since the cost increases by $\frac{1}{2}|\Phi|$ and decreases by at least $\frac{1}{2}|\delta(\Phi) \cap V^*_0|$, and by Lemma 3, $|\delta(\Phi) \cap V^*_0| \geq |\Phi|$. Finally, we note that z maintains feasibility for $N_{v,0}$, since $x'_v = 0$ and therefore $x'_u = 1$ for all $u \in \delta(v)$; since $\delta(v) \cap \Phi = \emptyset$, all vertices v and $\delta(v)$ keep their value in z. This concludes the proof.

□

3.3 Proof of Theorem 3.

In this section, we will demonstrate an instance of vertex cover where the worst tree can have $2^{\Omega(n)}$ times as many nodes as the tree generated by strong-branching. Here we assume that the LP solver finds a basic feasible solution at every node, but the one with the *most* fractional components.

The instance. Here we refer to a *triangle* as a K_3 and to a *diamond* as a K_4 minus an edge; see Figure 3. Let G be a union of m triangles and $\frac{1}{2}m$ diamonds. We will consider the instance of finding the maximum unweighted vertex cover on G. Observe the following properties of this instance:

- The number of vertices is $5m$.
- An optimal IP solution picks both vertices of degree 3 in each diamond and any two vertices in each triangle, so the optimal IP value is $3m$. This also implies that, at the root node, the maximal set of integer variables I is the set of vertices that make up the diamonds.
- An optimal LP solution sets all vertices to have value $\frac{1}{2}$, so the optimal LP value is $\frac{5}{2}m$.
- The additive integrality gap of the instance is then $\frac{1}{2}m$.

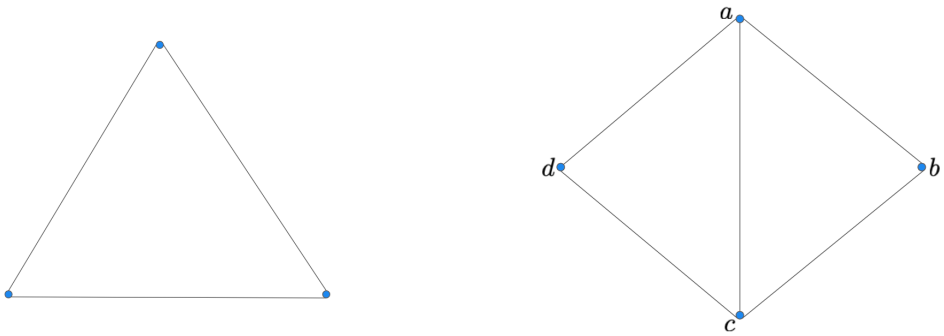

Figure 3: We refer to the component on the **left** as a *triangle* and the component on the **right** as a *diamond*. We label the vertices of the diamond for ease of following the arguments of this section.

Branching on a diamond. Consider any diamond, and suppose all of its vertices are set to $\frac{1}{2}$ in the LP optimal solution. Let a be a vertex of degree 3 and b be a vertex of degree 2; see Figure 3. First, we consider the case of branching on a. Setting $x_a = 0$, the only feasible solution is to set the three other vertices in the diamond to 1. This increases the objective value by 1. Setting $x_a = 1$, the only optimal solution sets $x_c = 1$ and $x_b = x_d = 0$. This results in no objective value change. Therefore, $\text{score}_P(a) = 0$. Now, we consider the case of branching on b. Setting $x_b = 0$, any feasible solution sets $x_a = x_c = 1$ and the only optimal solution then sets $x_d = 0$. This results in no change to the objective value. Setting $x_v = 1$, the only optimal solution sets $x_a = x_c = x_d = \frac{1}{2}$. This results in an objective value increase of $\frac{1}{2}$. Therefore, $\text{score}_P(b) = 0$.

Branching on a triangle. Consider any triangle, and suppose all of its vertices are set to $\frac{1}{2}$ in the LP optimal solution. Let u be any vertex of the triangle. Setting $x_u = 0$, the only optimal solution is to set the other two vertices to 1. This results in an objective value increase of $\frac{1}{2}$. Setting $x_u = 1$, any optimal basic feasible solution sets one of the other two vertices to 1 and the other to 0. This results in an objective value increase of $\frac{1}{2}$. Therefore, $\text{score}_P(u) = \frac{1}{4}$.

Strong branching. Observe that, given an instance with at least one diamond with all of its vertices set to $\frac{1}{2}$ and at least one triangle with all of its vertices set to $\frac{1}{2}$, strong-branching will choose to branch on a vertex in such a triangle. Then, observe that each node on the m-th level of the strong-branching tree "looks the same": its LP optimal solution has an integer value for each vertex in every triangle, in particular, sets two vertices to 1 and one vertex to 0; and all other vertices (i.e. all vertices in the diamonds) are set to $\frac{1}{2}$. Observe that each of these nodes has LP value the *same* as the optimal IP value. Since each node on the m-th level has an optimal solution that is integer, it follows from Lemma 2 that strong-branching results in a tree of size at most $2^{m+1} + \mathcal{O}(m)$.

A greedy rule. Consider instead the *greedy* variable selection rule: branch on the vertex with most fractional neighbors. Observe that, given an instance with at least one diamond with all of its vertices set to $\frac{1}{2}$ and at least one triangle with all of its vertices set to $\frac{1}{2}$, greedy will choose to branch on a vertex with degree 3 in such a diamond (without loss of generality, we assume its vertex a). Now, consider the nodes on the $\frac{1}{2}m$-th level of the tree generated by greedy. Each node's optimal LP solution has an integer value for each vertex in every diamond, and all other vertices (i.e. all vertices in the triangles) are set to $\frac{1}{2}$. In particular, consider any specific node on this level and suppose that, on the path from the root to this node in the tree, k vertices are set to 1 and $\frac{1}{2}m-k$ vertices are set to 0. Then, this node has objective value $\text{OPT}(L(\mathcal{I}))+(\frac{1}{2}m-k)$. Notice then that the sub-tree at this node will have to branch on $2k+1$ triangles and the resulting 2^{2k+1} leaves will have objective value $\text{OPT}(L(\mathcal{I})) + (\frac{1}{2}m - k) + \frac{1}{2}(2k+1) = \text{OPT}(L(\mathcal{I})) + \frac{1}{2}m + \frac{1}{2} > \text{OPT}(\mathcal{I})$ (branching on any fewer triangles will not allow the sub-tree to be pruned). On the $\frac{1}{2}m$-th level, there are $\binom{m/2}{k}$ nodes with k vertices set to 1 on its path from the root. Then, the final number of nodes in the tree is more than

$$\sum_{k=1}^{m/2} \binom{m/2}{k} 2^{2k}.$$

We can bound this quantity using the Binomial Theorem. In particular, we use the form

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}. \tag{1}$$

Plugging $n = \frac{1}{2}, x = 4, y = 1$ into (1), we have the bound

$$\sum_{k=0}^{m/2} \binom{m/2}{k} 4^k = 5^{m/2} \geq 2^{1.15m}. \tag{2}$$

4 Proof of Proposition 2 and Corollary 1: A bad example

Proposition 2. *For any integer program $\mathcal{I}$ with n binary variables, there is an equivalent integer program that uses an extended formulation of the feasible region of $\mathcal{I}$ with $2n$ binary variables, which we refer to as $BDG(\mathcal{I})$, that has the following property: there exists a branch and bound tree $\mathcal{T}^*(BDG(\mathcal{I}))$ that solves the instance $BDG(\mathcal{I})$ and $|\mathcal{T}^*(BDG(\mathcal{I}))| \leq 4n+1$.*

Proof. Consider the instance $\mathcal{I}$:

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & x \in P,\ x \in \{0,1\}^n. \end{aligned} \tag{$\mathcal{I}$}$$

where $P \subseteq [0,1]^n$ is a polytope. In [9], Bodur, Dash and Gunluk construct the extended formulation $Q \subseteq [0,1]^{2n}$ of P as follows. For every vertex x of P, construct a vertex (x,y) of Q where

$$y_i = \begin{cases} 1 & \text{if } x_i \in \{0,1\} \\ 0 & \text{if } x_i \in (0,1) \end{cases}.$$

Define Q to be the convex hull of these vertices, we call this the *BDG extended formulation for P*. We construct the equivalent IP, $BDG(\mathcal{I})$, as follows:

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & (x,y) \in Q,\ x \in \{0,1\}^n,\ y \in \{0,1\}^\cdot \end{aligned} \tag{BDG($\mathcal{I}$)}$$

For any IP $\mathcal{I}$, there exists a branch-and-bound tree with at most $4n+1$ nodes that solves $BDG(\mathcal{I})$. However, this tree does not remove the current LP optimal fractional point when branching. See Figure 4.

This follows since, by the definition of Q, all of its vertices that have $y_j = 0$ must have $x_j \in (0,1)$. Therefore, the branch that has $y_j = 0$ and $x_j = 0$ (and similarly $x_j = 1$) must be empty. Also, note that the branch that has $y_1 = 1, ..., y_n = 1$ must be integral. □

Corollary 1. *There exists an instance $\mathcal{I}^*$ with $2n$ binary variables, such that the following holds: Let $\mathcal{T}(\mathcal{I}^*)$ be any tree that solves $\mathcal{I}^*$ satisfying the following property: if x is the optimal solution to an internal node N of $\mathcal{T}(\mathcal{I}^*)$, then the variable j branched on at N must be such that $x_j \in (0,1)$. Then, $|\mathcal{T}(\mathcal{I}^*)| \geq 2^{n+1} - 1$. In particular, if $\mathcal{T}_S(\mathcal{I}^*)$ is a branch-and-bound tree generated using strong-branching that solve $\mathcal{I}^*$, then $|\mathcal{T}_S(\mathcal{I}^*)| \geq 2^{n+1} - 1$. On the other hand, there exists a tree $\mathcal{T}^*$ that solves $\mathcal{I}^*$ such that $|\mathcal{T}^*(\mathcal{I}^*)| \leq 4n+1$.*

Proof. Note that the y variables are not fractional in any vertex of Q. So when the tree of Figure 4 branches on a y variable, it does not remove the current LP optimal fractional point (because it does not remove any vertex of Q).

Now suppose we restrict ourselves to branching on a variable that does remove the current optimal point. Such a tree would only branch on x variables (i.e. the original variables). Then, if P is the n-dimensional cross polytope [18], and Q is its BDG extended formulation, we know that branching on only the x variables will require a tree of size at least $2^{n+1} - 1$, as shown in [18].

The final statement follows from Proposition 2. □

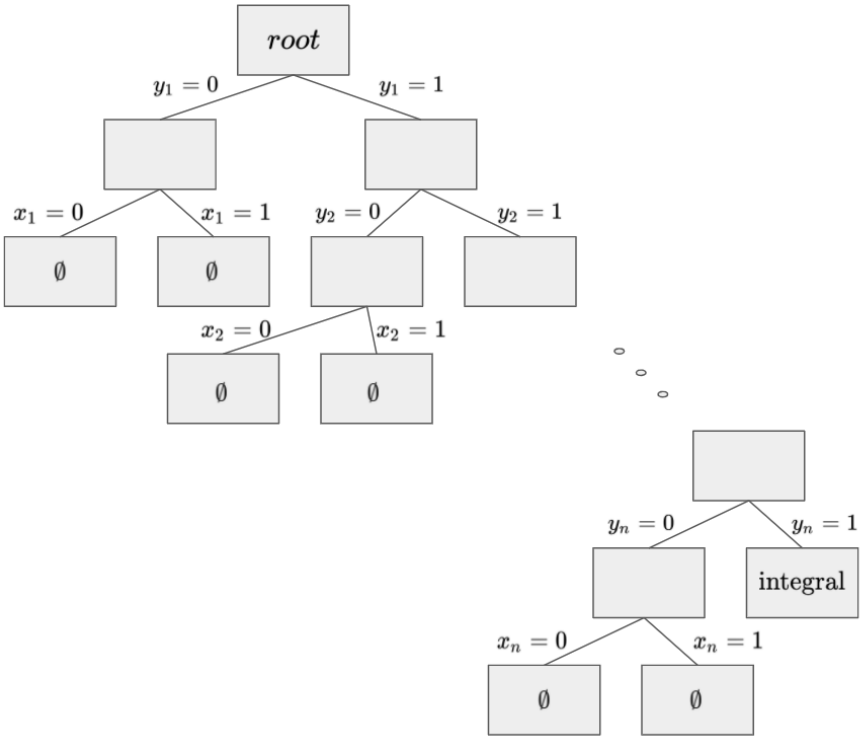

Figure 4: Branch-and-bound on BDG extended formulation

5 Computing an optimal branch-and-bound tree

5.1 Proof of Observation 1.

Observation 1. *When using an LP solver which satisfies Assumption 1, a branch-and-bound tree using the worst bound rule never branches on a node whose optimal objective function value is equal to that of the IP solver.*

Proof. Consider a linear program at a node whose optimal function value is equal to that of the MILP optimal objective function value. There are two cases to consider. Case 1: if there exists an integral optimal solution to the LP relaxation at a given node, then the LP solver finds it and the node is pruned. Case 2: If not, then this node does not contain an integral solution with objective function value equal to that of the MILP optimal objective function value. This implies that there must exist another node whose feasible region contains an integer feasible solution with the same objective function value as that of the MILP optimal objective function value. In particular, this implies that there must exist a node whose optimal linear programming solution is such an integer feasible solution. Since the LP solver will discover this solution before branching on the current node, we will never branch on the current node. □

5.2 The dynamic programming algorithm to compute optimal branch-and-bound tree.

We will now present the algorithm to compute the size of an optimal branch-and-bound tree for a fixed IP $\max_{x \in P \cap \{0,1\}^n} \langle c, x \rangle$ under Assumption 1. Let $\mathcal{F}$ denote the set of faces of $[0,1]^n$ and note that each face can be defined as a string in $\{\star, 0, 1\}^n$. For example, $(0, \star, 1)$ denotes the face $\{x \in [0,1]^3 : x_1 = 0, x_3 = 1\}$. Thus, $|\mathcal{F}| = 3^n$. Also, $\mathcal{F}$, is in one-to-one correspondence with all the possible nodes in the branch-and-bound tree. Let $\overline{\text{OPT}}(F)$ denote the size of the optimal branch-and-bound tree for the sub-problem restricted to F, i.e., $\max_{x \in F \cap P \cap \{0,1\}^n} \langle c, x \rangle$.

Based on Assumption 1 and Observation 1, with the WDB rule for node selection, a node in the branch-and-bound tree is pruned if and only if it is either infeasible, or the optimal objective function value of its LP relaxation is less than or equal to the optimal MILP optimal objective value. In Phase-1 of our algorithm (Algorithm 1), this fact is used to identify nodes that are pruned by infeasibility or by bound, thus, $\overline{\text{OPT}}(F) = 0$ for corresponding faces.

Now, given a face F that is not pruned in the branch-and-bound tree, and variable x_j that is free in F, define $F_{j,0}, F_{j,1}$ to be the faces of F that result from fixing x_j to 0 and 1 respectively, i.e. $F_{j,0} = \{x \in F : x_j = 0\}$. The fact that the optimal sub-tree at the node corresponding to F branches on the variable that produces two child nodes having the smallest optimal sub-trees, leads to the following recurrence relation,

$$\overline{\text{OPT}}(F) = 1 + \min_{j \in J_F} \left\{ \overline{\text{OPT}}(F_{j,0}) + \overline{\text{OPT}}(F_{j,1}) \right\},$$

where J_F denotes the set of variables that are free in F. We use this recurrence relation in the bottom-up computation of $\overline{\text{OPT}}(F)$ for the remaining faces (i.e. faces where $\overline{\text{OPT}}(F) \neq 0$) in $\mathcal{F}$ as Phase-2 of the algorithm. Thus, it can be inductively seen that the algorithm is correct. Additionally, the actual branch-and-bound tree can be found by storing $\arg\min_j \left\{ \overline{\text{OPT}}(F_{j,0}) + \overline{\text{OPT}}(F_{j,1}) \right\}$ at every iteration.

Notice it takes $2^{O(n)}$ time to execute line 1 and poly($data$) time to execute line 3 of Algorithm 1 for a particular face, where poly($data$) is the running time for solving an LP. Therefore, Phase I takes at most $2^{O(n)} + \text{poly}(data) \cdot 3^n$ time. Also notice Phase-2 takes $n \cdot 3^n$ time; this is because line 11 of Algorithm 1 takes at most n comparisons. So, in total Phase-1 and Phase-2 take $2^{O(n)} + (\text{poly}(data) + n) \cdot 3^n = \text{poly}(data) \cdot 3^{O(n)}$ time.

5.3 Implementation enhancements

Cascading 0-nodes Observe that for $F_1, F_2 \in \mathcal{F}$ such that $F_2 \subset F_1$, if $\overline{\text{OPT}}(F_1) = 0$, then $\overline{\text{OPT}}(F_2) = 0$ as well. Thus, at the start of Phase-1, faces in $\mathcal{F}$ are arranged in decreasing order of their dimension and upon finding a face, F_1, that is infeasible or pruned by bound, all other faces that are contained in F_1 can be removed from $\mathcal{S}$.

Parallelization Phase-1 of the proposed algorithm can be directly parallelized, as each face can be independently solved. However, to benefit from Cascading 0-nodes, we group faces that have the same values from $\{0, 1, \star\}$ for the first n_1 dimensions and run Phase-1 for the resulting 3^{n_1} groups

Algorithm 1 Computing Optimal Branch-and-bound Tree

Phase-1: Pruning by Infeasibility or Bound

Solve $\max_{x \in P \cap \{0,1\}^n} \langle c, x \rangle$; let x^* be the solution
Initialise: $\mathcal{S} \leftarrow \mathcal{F}$
for F in $\mathcal{S}$ **do**
    Solve $\max_{x \in F \cap P} \langle c, x \rangle$; let x^*_F be the optimal solution ($x^*_F = \emptyset$ if LP is infeasible)
    if $x^*_F = \emptyset$ **or** $\langle c, x^*_F \rangle \leq \langle c, x^* \rangle$ **then**
        $\overline{\text{OPT}}(F) \leftarrow 0$
        $\mathcal{S} \leftarrow \mathcal{S} \setminus \{F\}$
    end if
end for

Phase-2: Recursive bottom-up computation

Sort $\mathcal{S}$ in order of increasing dimension
for F in $\mathcal{S}$ **do**
    $\overline{\text{OPT}}(F) \leftarrow 1 + \min_j(\overline{\text{OPT}}(F_{j,0}) + \overline{\text{OPT}}(F_{j,1}))$
end for
return $\overline{\text{OPT}}([0,1]^n)$

independently. Note that this also reduces peak memory usage as all 3^n faces in $\mathcal{F}$ are not required to be generated at the start of the algorithm and the remaining $n - n_1$ dimensions are generated on the fly for each group. In our experiments, we set $n_1 = \lfloor \frac{n}{2} \rfloor$. In addition, this manner of grouping also enables the LP solver to benefit from smaller incremental changes between consecutive faces.

6 Computational Experiments

We evaluate the following branching strategies on problems mentioned in Section 6.1 by comparing the number of branching to the optimal number of branching computed using the dynamic programming based algorithm presented in Section 5.2. Recall the definition of Δ_j^+, Δ_j^- from Section 1.

- **SB-L**: Strong branching where branching candidates are compared using a convex combination of the maximum and minimum. The scoring function used is
$$\text{score}_L(j) = \frac{5}{6} \min(\Delta_j^+, \Delta_j^-) + \frac{1}{6} \max(\Delta_j^+, \Delta_j^-).$$
- **SB-P**: Strong branching where the product of improvements is used to score branching candidates. Thus, the scoring function is,
$$\text{score}_P(j) = \max(\Delta_j^+, \epsilon) \cdot \max(\Delta_j^-, \epsilon),$$
where $\epsilon > 0$ is small. For the computations presented in this section, ϵ is chosen to be 10^{-4}.
- **Most-Inf**: Most infeasible branching where the variable with fractional value closest to 0.5 is selected for branching
- **Rand**: Branching variable is randomly chosen from variables with fractional values.

6.1 Instance Generation

In this section, we discuss the problems considered for computational experiments and the details of generating randomized instances.

6.1.1 General Packing and Covering IPs

We consider packing problems, covering problems and general problems with multiple covering and packing inequalities. Let I_p and I_c represent the set of indices corresponding to packing and covering constraints respectively. The general formulation is as follows,

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & a_i^\top x \leq b_i \ \forall \ i \in I_p \\ & a_i^\top x \geq b_i \ \forall \ i \in I_c \\ & x \in \{0,1\}^n. \end{aligned}$$

For the experiments in this paper, $n = 20$ is considered. The constraint matrix is generated randomly while incorporating sparsity. Each element a_{ij} is 0 with probability $p = 0.25$. Otherwise, a random integer selected uniformly from the set $\{1, 2, \ldots, 200\}$ otherwise. The capacity parameter b_i is 50% of the sum of weights of the constraint, rounded down to integer value. The objective function is dependent on the number of packing and covering constraints as follows,

- $P5$ is a purely packing type problem with 5 constraints ($I_p = 5, I_c = 0$). We thus choose a non negative vector for the objective function, and each component c_i, is independently selected from the set $\{1, 2, \ldots, 200\}$ with uniform probability.
- $C5$ is a purely covering type problem with 5 constraints ($I_p = 0, I_c = 5$). Each component of the objective, c_i, is independently selected from the set $\{-200, \ldots, -2, -1\}$ with uniform probability.
- $G22$ is a general MILP with 2 packing-type and 2 covering-type constraints ($I_p = I_c = 2$). Each component of the objective, c_i, is independently selected from the set $\{-100, \ldots, 100\}$ with uniform probability.

6.1.2 Lot-sizing Problem and Variants

We consider the classical lot-sizing problem of determining production volumes while minimizing production cost, fixed cost of production and inventory holding cost across the planning horizon.

The MILP model for the lot-sizing problem with n time periods is as follows,

$$\min \quad \sum_{i=1}^{n} (c_i\, x_i + f_i\, y_i) + \sum_{i=1}^{n-1} h_i\, s_i \tag{3}$$

$$\text{s.t.} \quad x_1 = s_1 + d_1 \tag{4}$$

$$s_{i-1} + x_i = d_i + s_i \;\forall\; i \in \{2, \ldots, n\} \tag{5}$$

$$s_{n-1} + x_n = d_n \tag{6}$$

$$x_i \leq \left(\sum_{j=i}^{n} d_j \right) y_i \;\forall\; i \in \{1, \ldots, n\} \tag{7}$$

$$x \in \mathbb{R}^n_+,\; s \in \mathbb{R}^{n-1}_+,\; y \in \{0,1\}^n. \tag{8}$$

where variable x_i is the quantity produced in period i and y_i is a binary variable with value 1 if production occurs in period i and 0 otherwise. Lastly variable s_i is the inventory at the end of period i. Unit cost of production for each period, c_i, is independently and uniformly sampled from $\{1, \ldots, 10\}$, fixed cost of production, f_i, from $\{200, \ldots, 400\}$ and unit inventory holding cost h_i from $\{1, \ldots, 10\}$. Similarly, the demand for each time period, d_i is independent and uniformly distributed in $\{0, \ldots, 100\}$. In our computational experiments, we consider problems with $n = 17$.

Capacitated Lot-sizing In the capacitated lot-sizing problem, the maximum quantity produced in every time period is constrained. Let parameter u_i denote the upper-bound on the quantity that can be produced in period i. In our experiments, u_i are sampled uniformly and independently from $\{150, \ldots, 250\}$. All other parameters are generated in the same was as the uncapacitated problem. Equation (7) is thus replaced with the constraint,

$$x_i \leq u_i\, y_i, \qquad \forall i \in \{1, \ldots, n\}$$

Big-bucket Lot-sizing The last variant of lot-sizing problem that we consider is the big-bucket lot-sizing problem where resources are shared amongst multiple products [41]. We do not consider unit cost of production here. On the other hand, set up time and processing time are considered to be constrained. The following are the parameters of the corresponding MILP model,

P	Number of products, $\mathcal{P} = \{1, \ldots, P\}$
T	Number of time periods, $\mathcal{T} = \{1, \ldots, T\}$
f_i^p	Fixed cost of producing product p in period i
h_i^p	Inventory holding cost of product p in period i
$t_i^{s,p}$	Set up time of product p in period i
$t_i^{u,p}$	Processing time per unit of product p in period i
C_i	Time available in period i
z^p	Initial inventory of product p at the beginning of planning horizon.

The variables used to model the problem are following,

x_i^p	Quantity product p produced in period i
s_i^p	Quantity of product p stored as inventory at the end of period i
y_i^p	Binary variable indicating if product p was produced in period i ($y_i^p = 1$ if $x_i^p > 0$)

The MILP model used for the big-bucket lot-sizing problem is described below.

$$
\begin{aligned}
\min \quad & \sum_{i\in\mathcal{T}}\sum_{p\in\mathcal{P}} (f_i^p\, y_i^p + h_i^p\, s_i^p) \\
\text{s.t.} \quad & s_0^p = z^p,\ p \in \mathcal{P} \\
& s_T^p = 0,\ p \in \mathcal{P} \\
& s_{i-1}^p + x_i^p = d_i^p + s_i^p, i \in \mathcal{T}, p \in \mathcal{P} \\
& x_i^p \le \left(\sum_{j=i}^{n} d_j^p\right) y_i^p, i \in \mathcal{T}, p \in \mathcal{P} \\
& \sum_{p\in\mathcal{P}} (t_i^{s,p}\, y_i^p + t_i^{u,p}\, x_i^p) \le C_i, i \in \mathcal{T} \\
& x_i^p \in \mathbb{R}_+, s_i^p \in \mathbb{R}_+, y_i^p \in \{0,1\}, \quad i \in \mathcal{T}, p \in \mathcal{P}.
\end{aligned}
$$

In our experiments, we consider problems with 9 time periods ($T = 9$) and 2 products ($P = 2$). Parameters corresponding to demand, fixed cost of production and inventory holding cost are generated as described in the context of previous variants. Set up time for a product in a each period, $t_i^{s,p}$ is independently sampled from $\{200, \ldots, 500\}$ with equal probability, unit processing time $t_i^{u,p}$ from $\{1, \ldots, 10\}$ and time limitation C_i from $\{1000P, \ldots, 2000P\}$. Initial inventory for each product z^p is similarly sampled from $\{0, \ldots, 200\}$.

6.1.3 Minimum Vertex Cover

The vertex cover problem on graphs $G = (V, E)$ concerns with identifying the smallest possible subset V' of V, such that for every edge in E, at least one of its endpoints is included in V'. It is formulated as follows,

$$
\begin{aligned}
\min \quad & \sum_{i\in V} x_i \\
\text{s.t.} \quad & x_i + x_j \ge 1, \qquad \forall (i,j) \in E \\
& x \in \{0,1\}^{|V|}
\end{aligned}
$$

We generated random graphs for the vertex cover problem using the *Erdős-Rényi model*, i.e., the graph $G = (V, E)$ is constructed using two parameters; N indicating the number of nodes and p representing the probability with which each edge of the complete graph on V is independently included in the set E. For our computational experiments, we consider graphs with $N = 20$ and $p = 0.75$.

6.1.4 Chance Constraint Programming - Multiperiod Power Planning

We consider the problem of expanding the electric power capacity [8] of a state by constructing new coal and nuclear power plant to meet with the electricity demand of the state for a time horizon of T periods. Once constructed, coal plants are operational for T_c time periods and nuclear plants for T_n time periods. Legal restrictions mandate that fraction of nuclear power should be at

most f of the total capacity. Capital cost incurred for the construction of coal and nuclear power plants operational from the beginning of time period t are c_t and n_t respectively per megawatt of power capacity. The objective is to minimize the total capital cost of construction. Further, the demand is stochastic and defined on probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Approximated by the sample approximation approach, $\Omega = \{\omega_1, \ldots, \omega_N\}$ is assumed to be a finite sample space. It is required that the probability of the event where the demand is not satisfied be at most ϵ. The deterministic formulation of the problem as an MILP is as follows,

Parameters

T	Number of time periods, $\mathcal{T} = \{1, \ldots, T\}$
N	Size of sample space Ω, $\mathcal{N} = \{1, \ldots, N\}$
c_t	Capital cost per MW for coal plant operational from period t
n_t	Capital cost per MW for nuclear plant operational from period t
T_c	Lifespan of a coal power plant
T_n	Lifespan of a nuclear power plant
f	Upper bound on nuclear capacity as a fraction of total capacity
e_t	Electric capacity from existing resources in period t
d_t^i	Electricity demand (in MW) in period t corresponding to outcome ω_i
p_i	Probability of outcome ω_i
ϵ	Upper bound on the probability that the demand is satisfied

Variables

x_t	Power capacity (in MW) of coal plants operational starting at period t
y_t	Power capacity (in MW) of coal plants operational starting at period t
u_t	Total coal power capacity (in MW) in period t
v_t	Total nuclear power capacity (in MW) in period t
z_i	Binary variable indicating if demand is not satisfied for outcome ω_i

Model - CCP Power Planning

$$\min \quad \sum_{t=1}^{T} (c_t\, x_t + n_t\, y_t) \tag{9}$$

$$\text{s.t.} \quad u_t = \sum_{\max\{1,\, t-T_c+1\}}^{t} x_s, \quad t \in \mathcal{T} \tag{10}$$

$$v_t = \sum_{\max\{1,\, t-T_n+1\}}^{t} y_s, \quad t \in \mathcal{T} \tag{11}$$

$$(1-f)\, u_t - f\, v_t \le f\, e_t, \quad t \in \mathcal{T} \tag{12}$$

$$u_t + v_t \ge (d_t^i - e_t)(1 - z_i), \quad t, i \in \mathcal{T} \times \mathcal{N} \tag{13}$$

$$\sum_{i=1}^{n} z_i\, p_i \le \epsilon \tag{14}$$

$$x_t, y_t, u_t, v_t \in \mathbb{R}_+, \quad t \in \mathcal{T} \tag{15}$$

$$z_i \in \{0, 1\}, \quad i \in \mathcal{N}. \tag{16}$$

The objective function (9) minimizes total capital expenditure of constructing power plants. Equations (10) and (11) compute total coal and nuclear power capacity for a given time period from active power plants based on their lifespan. Equation (12) enforces the regulatory limit on nuclear capacity is satisfied. Equations (13) and (14) ensure that the outcomes for which the demand is not satisfied has probability at most ϵ.

For the experiments in Section 6, we generate instances with $T = 30$ and $N = 20$ which corresponds to 30 time periods and 20 outcomes in sample space. Parameters d_t^i are independent random integers uniformly distributed in $\{300, \ldots, 700\}$. Similarly, c_t are uniformly distributed in $\{100, \ldots, 300\}$ and n_t in $\{100, \ldots, 200\}$. Electric capacity from existing resources for the first period, e_1 is a random integer in $\{100, \ldots, 500\}$. Capacity from existing resources is then modelled to decline by a factor of r in every subsequent period where r is uniformly distributed in $[0.7, 1)$, ie. $e_i = e_1\, r^{i-1}$. Lifespan of coal and nuclear power plants are 15 and 10 periods respectively. Nuclear capacity is constrained to be at most 20% of total capacity. All outcomes in Ω are equally probable with $p_i = 0.05$ and demand satisfiability can be violated with a probability of at most 0.2.

6.1.5 Chance Constraint Programming - Portfolio Optimization

We consider the probabilistically-constrained portfolio optimization problem for n asset types, approximated by the sample approximation approach [38], where the constraint on overall return may be violated for at most k out of the m samples. The MILP formulation of this problem is as

follows:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} x_i \\ \text{s.t.} \quad & a_i^T x + r z_i \geq r, \quad \forall i = 1, \ldots, m \\ & \sum_{i=1}^{m} z_i \leq k \\ & x \in \mathbb{R}^n_+,\ z \in \{0,1\}^m. \end{aligned}$$

We sample scenarios from the distribution presented in [40], which is shown to be computationally difficult to solve. Each component of the constraint matrix, a_{ij} is independently sampled from a uniform distribution in $[0.8, 1.5]$ and r is equal to 1.1. For our experiments, we set $n = 30$, $m = 20$ and $k = 4$.

6.1.6 Stable Set Polytope on Bipartite Graph With Knapsack Constraint

Stable set polytope corresponding to a bipartite graph is known to have a totally unimodular matrix and thus integral vertices. We consider the problem of solving a maximization problem on the stable set polytope of a bipartite graph where the optimal extreme point is cut off with a knapsack constraint with the same coefficients as the objective function. The details of model are explained below. A bipartite graph $G = (N, E)$ is generated for a n nodes and m edges as follows. The partition of $N = N_1 \cup N_2$ is generated, by setting N_1 as a randomly selected subset of $\lfloor fn \rfloor$ nodes and N_2 as its complement, where f is sampled from a uniform distribution over $[0.3, 0.5]$. From the $N_1 \times N_2$ possible edges, m are then randomly selected to form set E. Lastly, each component c_i of the objective function is a randomly selected integer from 1 to 50. In our experiments, we consider instances with 20 nodes and 30 edges. Let δ^* be the objective function value of the corresponding maximum weight stable set problem,

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} c_i\, x_i && (17) \\ \text{s.t.} \quad & x_i + x_j \leq 1, \quad \forall i, j \in E && (18) \\ & x \in \{0,1\}^m. && (19) \end{aligned}$$

The following constraint is then added to (17)- (19),

$$\sum_{i=1}^{n} c_i\, x_i \leq r\delta^* \tag{20}$$

where r is uniformly distributed in $[0.75, 0.9]$.

6.2 Results

We present some preliminary results of various variable selection rules in comparison to optimal tree. SB-L stands for strong-branching with linear score, SB-P stands for strong-branching with

product score, Most infeasible is selecting the variable with fractionality closest to 0.5, and Random just selects a variable randomly from a list of fractional variables.

Figure 2 shows a comparison of branching strategies based on the ratio of geometric mean of BB tree sizes to geometric mean of optimal tree sizes over all instances of the problem.

Figure 5 shows performance profiles. The way to read the plots is the following: Consider the green curve for multi row packing problem. The point $(80\%, 0.3)$ means that in the case of using the most infeasible rule for multi row packing problem, $0.3 \times 100 = 30$ instances out of 100, have branch-and-bound trees that have at most 80% more nodes than the optimal branch-and-bound tree. The x-axis represents percentage differences in size of the BB tree in comparison to the optimal BB tree. The y-axis is the cumulative frequency of instances.

Tables 1 through 12 present the number of nodes for all the instances we tested and all the variable selection rules.

6.2.1 Discussion On Findings

It is clear from the Fig. 2-5 and Tables 1 through 12 that random always performs the worst and as expectated strong-branching is always the best. Furthermore, as seen in Fig. 2, the geometric mean of tree size for strong-branching remains less than twice of the optimal tree for all problems considered in this study. The overall geometric mean of optimal tree size of instances across all problems is 42.65, whereas the same for SB-L, SB-P, most infeasible branching and random branching is 61.47, 61.09, 90.19 and 145.37 respectively. While the performance of two variants of strong-branching is comparable on all problems considered in this study, SB-P dominates over SB-L on 8 out of 10 problems, although by a small margin.

Strong branching is near optimal for the lot-sizing instances and performs exceptionally well on all other lot sizing variants as well. This is an interesting result considering that the tree sizes of the other two strategies can be significantly larger for almost all instances as seen in Fig 6d. Thus, this indicates that very large branch-and-bound trees exist for the problem, but strong-branching is indeed effective at finding an effective branching strategy. This is in contrast to the problem of Stable Set on Bipartite Graphs with additional knapsack constraint, where all strategies have comparable performance as seen in Fig 6i. In comparison with the optimal tree, its performance on general packing and covering IPs is also relatively poor and is worst on Chance Constraint Programming Portfolio Optimization problem. The performance of random branching is also worst for CCP problems, thus making these an interesting class of problems for which better variable selection rules could be discovered.

References

[1] Tobias Achterberg. Constraint integer programming. *PhD Thesis*, 2007.

[2] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[3] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[4] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Finding cuts in the tsp (a preliminary report). Technical report, Citeseer, 1995.

[5] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International conference on machine learning*, pages 344–353. PMLR, 2018.

[6] Amitabh Basu, Michele Conforti, Marco Di Summa, and Hongyi Jiang. Complexity of cutting planes and branch-and-bound in mixed-integer optimization. *arXiv preprint arXiv:2003.05023*, 2020.

[7] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

[8] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Belmont, MA: Athena Scientific, 1997.

[9] Merve Bodur, Sanjeeb Dash, and Oktay Günlük. Cutting planes from extended lp formulations. *Mathematical Programming*, 161(1-2):159–192, 2017.

[10] Sander Borst, Daniel Dadush, Sophie Huiberts, and Samarth Tiwari. On the integrality gap of binary integer programs with gaussian data. In *IPCO*, pages 427–442, 2021.

[11] Raymond Breu and Claude-Alain Burdet. Branch and bound experiments in zero-one programming. In *Approaches to Integer Programming*, pages 1–50. Springer, 1974.

[12] Vasek Chvátal. Hard knapsack problems. *Operations Research*, 28(6):1402–1411, 1980.

[13] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer programming*, volume 271. Springer, 2014.

[14] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

[15] Daniel Dadush and Samarth Tiwari. On the complexity of branching proofs. *arXiv preprint arXiv:2006.04124*, 2020.

[16] Robert J Dakin. A tree-search algorithm for mixed integer programming problems. *The computer journal*, 8(3):250–255, 1965.

[17] Santanu S Dey, Yatharth Dubey, and Marco Molinaro. Branch-and-bound solves random binary ips in polytime. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 579–591. SIAM, 2021.

[18] Santanu S Dey, Yatharth Dubey, and Marco Molinaro. Lower bounds on the size of general branch-and-bound trees. *arXiv preprint arXiv:2103.09807*, 2021.

[19] Norman J Driebeek. An algorithm for the solution of mixed integer programming problems. *Management Science*, 12(7):576–587, 1966.

[20] Jonathan Eckstein. Parallel branch-and-bound algorithms for general mixed integer programming on the cm-5. *SIAM journal on optimization*, 4(4):794–814, 1994.

[21] JJH Forrest, JPH Hirst, and JOHN A Tomlin. Practical solution of large mixed integer programming problems with umpire. *Management Science*, 20(5):736–773, 1974.

[22] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019.

[23] J-M Gauthier and Gerard Ribiere. Experiments in mixed-integer linear programming using pseudo-costs. *Mathematical Programming*, 12(1):26–47, 1977.

[24] Prateek Gupta, Maxime Gasse, Elias B Khalil, M Pawan Kumar, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *arXiv preprint arXiv:2006.15212*, 2020.

[25] WC Healy Jr. Multiple choice programming (a procedure for linear programming with zero-one variables). *Operations Research*, 12(1):122–138, 1964.

[26] Robert G Jeroslow. Trivial integer programs unsolvable by branch-and-bound. *Mathematical Programming*, 6(1):105–109, 1974.

[27] Hongyi Jiang. Complexity of branch-and-bound and cutting planes in mixed-integer optimization-ii. In *Integer Programming and Combinatorial Optimization: 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19–21, 2021, Proceedings*, page 383. Springer Nature.

[28] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[29] A Land and S Powell. Computer codes for problems of integer programming. In *Annals of Discrete Mathematics*, volume 5, pages 221–269. Elsevier, 1979.

[30] Alisa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[31] Pierre Le Bodic and George Nemhauser. An abstract model for branching and its application to mixed integer programming. *Mathematical Programming*, 166(1):369–405, 2017.

[32] Jeff T Linderoth and Martin WP Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[33] Andrea Lodi and Giulia Zarpellon. On learning and branching: a survey. *Top*, 25(2):207–236, 2017.

[34] Gautam Mitra. Investigation of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4(1):155–170, 1973.

[35] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[36] George L Nemhauser and Leslie Earl Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical programming*, 6(1):48–61, 1974.

[37] George L Nemhauser and Leslie Earl Trotter. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975.

[38] Bernardo K Pagnoncelli, Shabbir Ahmed, and Alexander Shapiro. Computational study of a chance constrained portfolio selection problem. *Journal of Optimization Theory and Applications*, 142(2):399–416, 2009.

[39] Jean-Claude Picard and Maurice Queyranne. On the integer-valued variables in the linear vertex packing problem. *Mathematical Programming*, 12(1):97–101, 1977.

[40] Feng Qiu, Shabbir Ahmed, Santanu S Dey, and Laurence A Wolsey. Covering linear programming with violations. *INFORMS Journal on Computing*, 26(3):531–546, 2014.

[41] Daniel Quadt and Heinrich Kuhn. Capacitated lot-sizing with extensions: a review. *4OR*, 6(1):61–83, 2008.

[42] Tim Roughgarden. Beyond worst-case analysis. *Communications of the ACM*, 62(3):88–96, 2019.

[43] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

A Graphs and tables

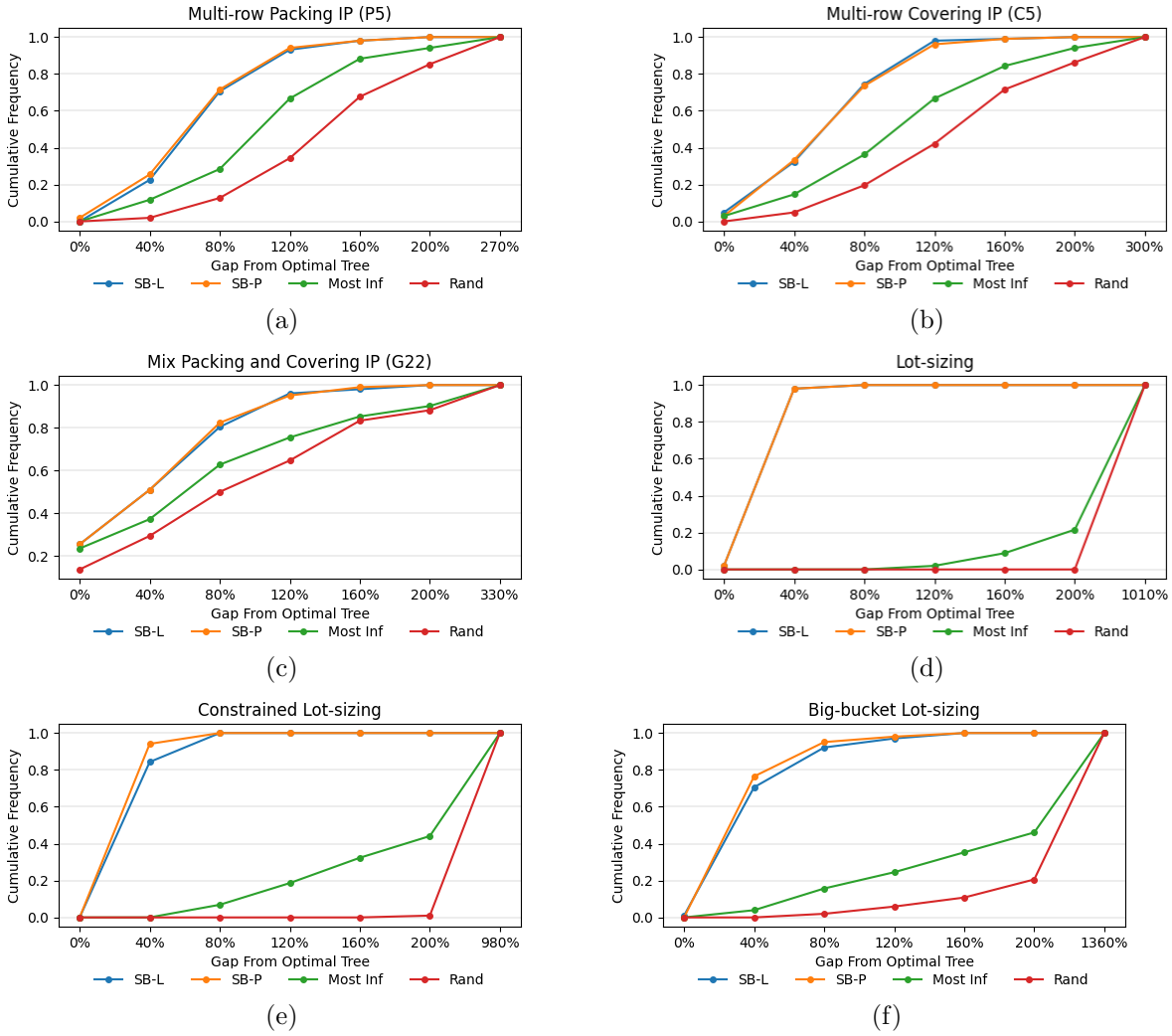

Figure 5: Cumulative frequency of instances in terms of optimality gap for different branching strategies

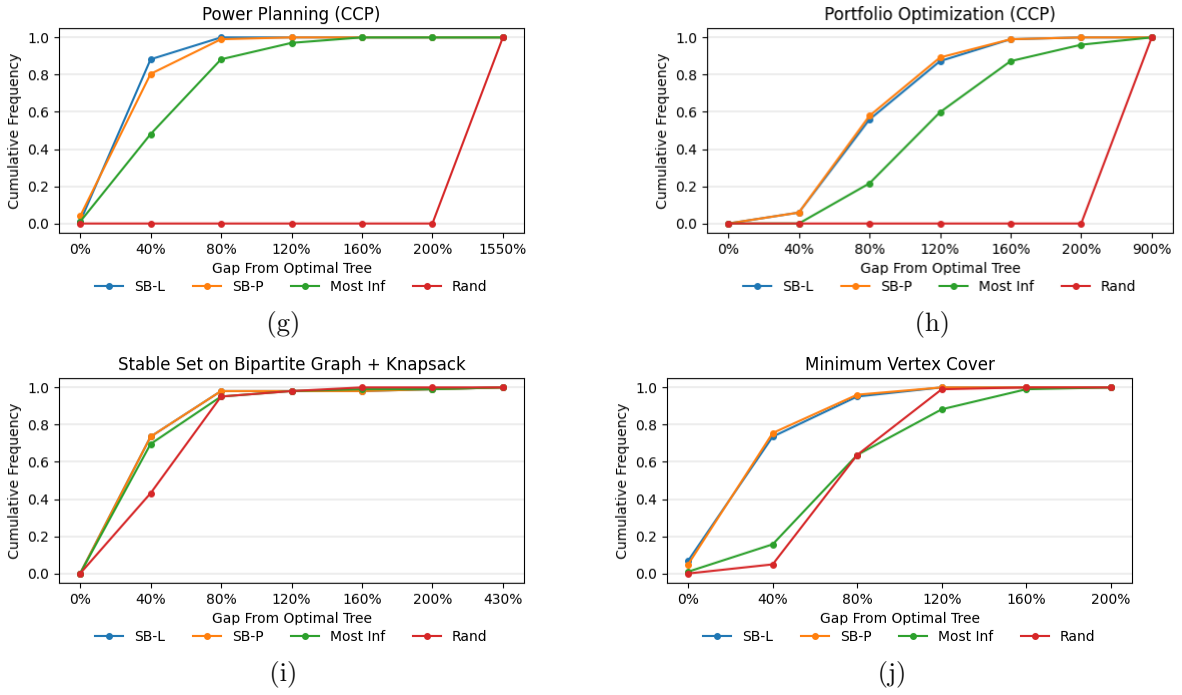

Figure 5: Cumulative frequency of instances in terms of optimality gap for different branching strategies, cont'd.

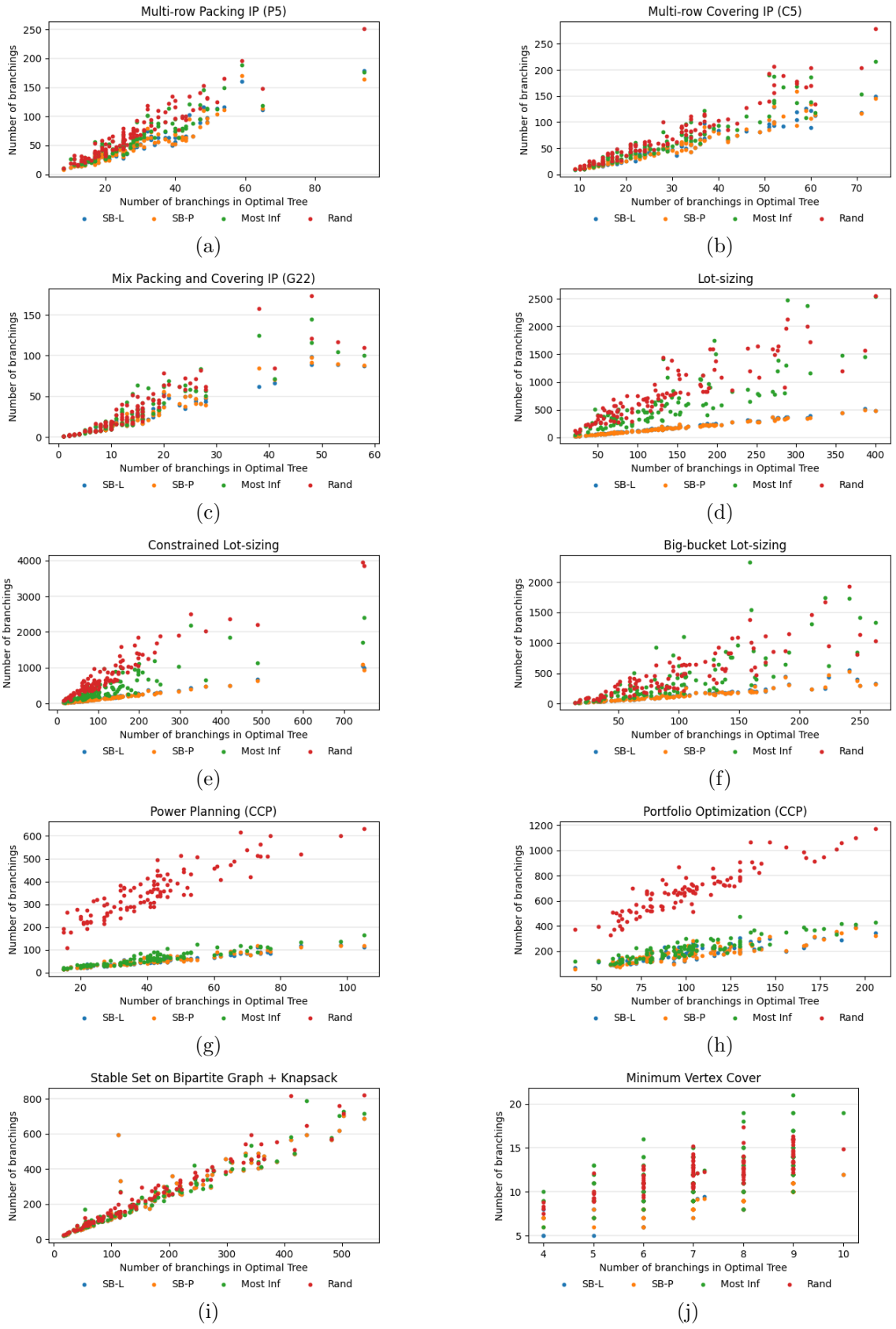

Figure 6: Number of branching operations for all instances in comparison with the optimal tree

Table 2: Computational Results for Packing IP (P5) instances, $n = 20,\ |I_p| = 5,\ |I_c| = 0,\ p = 0.25$

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	37	73	61	90	91
2	17	27	27	27	29
3	28	44	56	60	80
4	14	19	18	24	30
5	49	98	89	113	132
6	42	62	60	79	98
7	49	90	97	114	131
8	22	35	36	36	43
9	20	26	26	34	39
10	16	23	20	21	24
11	41	64	63	63	77
12	14	21	21	29	26
13	25	42	41	48	58
14	33	60	63	74	85
15	8	9	8	10	11
16	26	35	37	36	51
17	65	112	114	119	149
18	35	56	56	88	110
19	13	14	14	22	25
20	25	50	49	60	70
21	19	50	50	54	59
22	29	59	59	51	67
23	11	17	15	27	33
24	21	27	30	53	62
25	59	160	171	189	197
26	22	40	43	57	66
27	38	63	57	114	123
28	28	54	54	64	76
29	19	30	30	49	47
30	20	28	28	38	44
31	34	51	55	64	78
32	25	37	35	47	54
33	40	56	53	75	127
34	10	12	12	26	19
35	18	28	27	31	38
36	20	25	24	29	33
37	21	26	24	30	48
38	14	17	15	17	21
39	48	116	110	146	154
40	25	38	38	49	55
41	25	28	30	56	46
42	43	66	63	83	94
43	37	64	61	74	107
44	17	26	33	37	37
45	29	47	50	61	76
46	33	66	73	89	95
47	18	20	21	36	31
48	17	32	32	56	54
49	29	65	63	90	93
50	32	62	62	101	119
51	94	179	164	176	251
52	27	47	46	52	63
53	23	38	38	40	44
54	13	19	19	18	22
55	42	65	65	76	101
56	47	95	96	120	141
57	17	26	24	23	40
58	45	66	66	88	112
59	23	39	34	45	48
60	54	116	111	150	165
61	28	44	41	59	72
62	27	46	45	46	70
63	39	50	52	88	135
64	32	75	79	101	113
65	30	52	51	62	69
66	25	49	48	68	80
67	11	14	14	17	26
68	31	45	47	70	90
69	15	22	23	23	28
70	25	56	56	56	66
71	12	16	16	16	16
72	11	16	15	17	20
73	19	32	35	41	43
74	22	36	33	44	57
75	47	89	82	96	114
76	23	32	41	43	44
77	24	43	42	49	50
78	40	63	70	95	116
79	31	60	58	73	76
80	12	16	14	15	17
81	20	30	30	34	36
82	17	18	17	29	35
83	43	62	58	79	101
84	17	21	20	23	38
85	31	61	65	65	77
86	44	103	95	121	134
87	25	36	35	59	74
88	25	34	33	34	44
89	52	114	104	113	125
90	35	64	70	83	99
91	28	45	49	56	71
92	18	28	26	33	39
93	25	46	51	51	63
94	13	21	19	18	32
95	20	34	34	43	52
96	29	59	55	64	77
97	20	26	26	27	31
98	28	44	49	61	64
99	20	36	36	37	40
100	40	74	68	79	97
Average	**28.2**	**48.7**	**48.4**	**60.2**	**71.2**
Geo mean	**25.4**	**41.2**	**40.9**	**50.8**	**59.9**

Table 3: Computational Results for Covering IP (C5) instances, $n = 20,\ |I_p| = 0,\ |I_c| = 5,\ p = 0.25$

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	21	37	31	32	43
2	49	82	82	101	138
3	11	11	13	11	18
4	25	42	49	56	64
5	36	68	67	77	98
6	22	26	32	41	47
7	28	49	50	82	101
8	12	14	13	18	17
9	32	54	46	78	90
10	18	29	29	43	48
11	46	83	87	112	128
12	38	72	72	91	93
13	24	35	33	39	44
14	19	23	24	38	44
15	37	66	65	113	116
16	12	16	16	24	25
17	13	14	15	19	23
18	28	51	51	49	57
19	24	35	35	43	58
20	32	61	63	81	90
21	15	17	17	21	22
22	71	118	117	154	204
23	29	49	51	45	58
24	13	13	13	15	25
25	18	24	23	23	34
26	17	24	25	27	32
27	17	25	22	39	40
28	34	62	58	73	106
29	35	51	52	65	85
30	15	17	23	34	33
31	19	24	25	29	27
32	61	113	114	118	135
33	23	33	29	37	47
34	32	57	58	89	92
35	31	37	42	60	62
36	33	50	48	71	84
37	30	48	46	54	68
38	13	17	16	21	25
39	59	127	122	109	167
40	57	120	159	169	178
41	23	39	35	35	47
42	14	18	16	17	20
43	17	23	20	23	26
44	21	44	40	40	54
45	24	38	43	52	48
46	27	41	40	49	60
47	52	129	131	188	207
48	26	42	36	40	47
49	24	40	39	56	63
50	36	58	58	60	78
51	12	15	13	17	19
52	21	31	38	39	50
53	39	84	83	88	95
54	16	20	25	30	38
55	33	71	63	67	76
56	37	69	65	78	83
57	29	48	45	55	59
58	42	67	65	71	85
59	15	22	22	24	24
60	15	20	20	19	28
61	11	12	12	12	17
62	10	11	11	12	16
63	57	104	94	138	174
64	60	123	134	187	204
65	34	57	57	65	71
66	32	54	58	76	82
67	52	100	100	142	171
68	9	9	10	9	11
69	17	23	23	32	42
70	44	75	75	86	98
71	10	10	10	13	11
72	74	149	146	216	279
73	37	77	78	81	101
74	15	16	18	19	20
75	60	90	107	139	171
76	22	39	33	46	57
77	22	45	45	61	54
78	34	43	43	82	83
79	18	20	24	30	27
80	21	42	43	43	50
81	15	26	25	29	27
82	33	72	59	97	95
83	42	68	63	93	107
84	13	17	15	17	20
85	51	96	85	112	140
86	20	26	26	35	37
87	13	18	15	20	21
88	22	36	38	57	57
89	21	37	37	57	58
90	11	11	11	11	12
91	16	19	20	34	41
92	37	98	91	122	113
93	52	94	98	137	156
94	34	57	58	101	112
95	40	84	79	94	103
96	17	24	28	34	39
97	13	16	16	21	23
98	15	16	17	20	24
99	51	91	104	191	193
100	54	92	111	167	189
Average	**28.8**	**49.1**	**49.5**	**63.9**	**73.7**
Geo mean	**25.2**	**39.0**	**39.1**	**48.9**	**56.4**

Table 4: Computational Results for Mix Packing and covering IP (G22) instances, $n = 20,\ |I_p| = 2,\ |I_c| = 2,\ p = 0.25$

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	10	17	15	18	20
2	38	62	85	125	158
3	48	99	98	145	173
4	2	2	2	2	2
5	18	27	30	29	45
6	20	54	51	62	63
7	8	15	15	14	16
8	10	11	14	17	19
9	48	89	92	116	121
10	2	2	2	2	3
11	53	89	90	105	117
12	19	35	34	42	46
13	16	31	31	34	36
14	28	47	49	57	62
15	9	9	9	12	14
16	16	17	17	22	19
17	12	16	16	15	16
18	1	1	1	1	1
19	16	35	33	42	42
20	7	7	7	7	7
21	14	15	15	15	17
22	2	2	2	2	2
23	15	28	21	34	34
24	19	28	27	34	42
25	13	19	19	20	20
26	24	35	38	64	59
27	11	17	17	16	16
28	12	16	18	31	24
29	3	4	4	3	4
30	2	2	2	2	2
31	2	2	2	2	2
32	8	8	8	8	9
33	17	22	21	60	48
34	15	30	21	33	39
35	23	39	41	62	62
36	2	2	2	2	2
37	13	17	17	15	22
38	7	8	8	7	7
39	19	33	31	33	37
40	26	44	41	57	61
41	4	4	4	4	4
42	4	4	4	4	5
43	10	12	12	12	15
44	6	8	8	8	10
45	8	12	12	12	14
46	5	6	6	9	8
47	2	2	2	2	3
48	15	28	25	34	38
49	24	50	50	57	72
50	9	9	9	11	16
51	11	19	23	27	35
52	12	16	17	18	22
53	9	11	11	14	15
54	20	38	37	42	51
55	2	2	2	2	2
56	15	26	25	26	30
57	9	12	12	13	15
58	15	35	33	64	50
59	13	14	14	22	24
60	10	10	12	13	15
61	7	12	15	13	17
62	18	35	32	41	46
63	12	17	18	34	40
64	28	44	39	51	59
65	2	2	2	3	3
66	14	17	17	53	52
67	58	87	88	100	110
68	10	10	10	10	10
69	14	17	20	21	19
70	21	48	52	69	65
71	9	9	9	19	14
72	16	32	28	27	32
73	5	8	6	5	7
74	10	15	16	16	17
75	8	8	8	8	8
76	15	28	25	26	26
77	27	41	41	84	82
78	41	66	71	72	85
79	18	25	27	46	53
80	16	23	24	22	26
81	6	7	7	8	11
82	10	11	11	11	11
83	3	3	3	3	3
84	7	7	7	7	7
85	26	46	47	57	71
86	20	53	56	64	79
87	7	8	7	8	8
88	6	6	6	6	7
89	1	1	1	1	1
90	12	17	22	32	28
91	3	3	3	3	3
92	9	14	11	13	18
93	14	19	18	20	25
94	6	7	7	7	8
95	25	51	51	59	66
96	16	21	23	28	25
97	7	7	7	7	7
98	9	14	14	19	19
99	11	21	18	23	27
100	13	21	29	43	39
Average	**13.9**	**22.4**	**22.6**	**29.0**	**31.6**
Geo mean	**10.2**	**14.2**	**14.2**	**16.8**	**18.3**

Table 5: Computational Results for Lot-sizing instances, $n = 17$

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	79	94	94	339	416
2	21	21	21	51	124
3	76	102	87	504	278
4	78	94	86	240	628
5	358	448	442	1481	1204
6	72	85	85	312	573
7	122	156	151	601	849
8	64	78	78	446	465
9	26	28	27	70	97
10	75	92	87	475	493
11	81	93	90	280	555
12	69	83	72	339	402
13	53	65	69	185	259
14	206	233	226	591	1085
15	57	66	66	380	345
16	191	224	221	905	1601
17	161	176	173	588	1137
18	69	83	80	225	692
19	126	166	163	459	696
20	182	222	213	1125	1117
21	70	96	96	275	294
22	277	356	339	1385	1648
23	53	63	62	179	397
24	107	131	120	489	690
25	132	155	152	1418	1440
26	188	251	221	406	823
27	198	257	226	1503	1377
28	46	51	51	511	252
29	270	368	372	823	1599
30	197	229	211	1749	1223
31	153	231	234	419	1056
32	93	121	121	210	515
33	23	23	23	82	87
34	109	132	128	456	692
35	116	151	142	364	484
36	219	276	274	826	848
37	179	221	204	1057	952
38	314	351	344	2378	2004
39	180	210	206	1048	927
40	287	362	348	1297	1959
41	62	69	69	402	459
42	56	71	74	144	536
43	387	519	495	1458	1575
44	110	122	119	318	705
45	145	166	167	837	771
46	134	165	159	567	723
47	253	296	275	767	1078
48	195	236	231	535	1592
49	241	306	308	644	1195
50	74	82	86	265	474
51	70	80	74	378	570
52	37	44	44	102	239
53	186	226	225	608	824
54	137	191	174	1087	1249
55	121	150	145	640	953
56	251	291	279	784	1645
57	74	94	89	364	384
58	112	127	125	348	707
59	129	169	160	667	502
60	182	224	221	581	831
61	130	197	195	313	579
62	189	214	218	966	897
63	53	58	58	260	281
64	126	156	141	672	770
65	141	173	171	476	763
66	289	368	356	2477	2128
67	59	66	63	180	249
68	318	387	360	1155	1726
69	112	137	126	287	572
70	47	58	54	261	281
71	101	122	118	464	616
72	50	59	59	147	388
73	93	114	109	255	461
74	400	486	477	2545	2550
75	239	317	287	869	1612
76	40	49	49	158	204
77	131	143	143	304	774
78	118	141	140	366	653
79	34	40	40	199	222
80	58	69	67	362	499
81	27	29	30	129	157
82	93	109	110	240	290
83	143	164	162	722	1384
84	152	185	180	710	1138
85	102	114	115	333	594
86	273	321	303	778	1491
87	154	195	185	451	817
88	86	99	98	498	757
89	66	79	80	212	301
90	85	105	105	184	351
91	276	351	346	1196	1567
92	285	347	326	808	908
93	42	47	48	265	245
94	147	178	180	422	807
95	50	62	59	408	332
96	118	158	147	446	557
97	100	126	115	473	747
98	67	84	84	349	451
99	151	181	173	636	1208
100	163	205	200	609	794
Average	**136.4**	**167.7**	**162.0**	**605.5**	**804.1**
Geo mean	**111.5**	**135.0**	**131.1**	**461.0**	**651.6**

Table 6: Computational Results for Capacitated Lot-sizing instances, $n = 17$

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	60	85	71	178	572
2	243	292	298	630	1694
3	19	20	22	27	93
4	198	233	225	901	1842
5	114	158	148	983	969
6	85	110	105	172	352
7	90	144	109	207	404
8	68	97	91	130	364
9	197	287	258	933	1274
10	56	68	69	276	368
11	71	85	95	171	487
12	95	119	116	228	533
13	72	94	94	218	518
14	237	271	267	1187	1431
15	75	89	89	457	695
16	48	59	58	128	354
17	197	271	255	811	1052
18	121	160	160	232	715
19	160	202	203	881	1209
20	92	141	117	342	566
21	102	134	123	180	464
22	191	235	228	301	1099
23	137	167	162	392	615
24	75	85	86	182	563
25	79	105	125	240	356
26	40	45	45	113	308
27	62	71	73	106	380
28	490	684	636	1126	2203
29	62	79	73	185	480
30	126	152	155	215	644
31	66	86	74	118	613
32	149	203	170	397	878
33	172	242	251	430	879
34	60	76	75	251	494
35	181	230	217	384	1011
36	143	182	176	888	1085
37	749	991	942	2401	3854
38	72	86	82	394	507
39	83	108	106	294	526
40	130	174	145	570	654
41	65	84	82	470	634
42	195	243	239	657	1412
43	160	200	197	506	990
44	85	104	103	389	487
45	135	172	169	435	878
46	89	111	111	345	581
47	26	33	31	98	163
48	143	169	172	636	660
49	82	110	102	392	771
50	209	263	252	878	1371
51	124	154	151	285	879
52	33	37	38	106	246
53	96	120	124	333	563
54	326	441	397	2189	2498
55	118	177	153	456	666
56	21	28	27	134	143
57	146	202	198	344	660
58	90	120	120	223	663
59	33	37	37	219	203
60	28	34	34	59	181
61	104	135	133	535	891
62	154	219	194	476	1041
63	72	87	88	206	454
64	33	42	38	71	95
65	421	505	496	1855	2357
66	97	117	112	505	578
67	54	68	64	155	269
68	152	179	178	468	1329
69	174	211	201	262	1040
70	73	103	89	309	444
71	149	182	181	640	844
72	95	138	120	540	566
73	58	69	72	156	382
74	90	118	111	274	968
75	362	479	477	659	2030
76	101	134	118	256	555
77	92	137	134	401	441
78	118	140	137	258	581
79	117	149	142	481	519
80	68	75	81	136	339
81	100	122	117	194	395
82	99	128	117	364	617
83	62	77	71	170	462
84	65	96	90	205	368
85	37	48	43	85	218
86	50	62	60	147	337
87	189	256	234	949	1626
88	16	28	28	36	86
89	745	1048	1099	1717	3948
90	251	327	309	538	1881
91	222	387	361	679	1248
92	157	194	188	995	1583
93	82	105	99	229	435
94	41	47	47	164	265
95	140	214	178	273	682
96	68	82	79	267	398
97	155	185	176	609	1182
98	200	255	244	1119	1392
99	51	67	67	164	305
100	297	360	340	1042	1907
Average	**132.8**	**173.3**	**166.4**	**460.0**	**828.1**
Geo mean	**101.9**	**131.1**	**125.9**	**328.0**	**635.0**

Table 7: Computational Results for Big-bucket Lot-sizing instances, $T = 9, P = 2$

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	172	202	210	642	683
2	102	184	179	312	464
3	80	92	93	402	460
4	63	68	70	271	249
5	24	29	27	58	93
6	73	129	123	159	177
7	60	71	69	216	303
8	59	63	63	311	368
9	104	168	172	321	501
10	71	93	89	149	191
11	131	204	218	222	346
12	90	109	102	292	284
13	35	41	39	142	131
14	221	255	271	1747	1673
15	109	142	139	517	643
16	83	102	97	275	281
17	39	54	59	137	208
18	69	87	82	170	257
19	54	62	65	91	228
20	263	330	316	1338	1034
21	100	117	119	628	754
22	102	143	144	480	638
23	104	131	125	1108	646
24	129	170	155	731	632
25	54	66	65	101	169
26	63	79	71	100	147
27	104	135	132	167	214
28	65	120	106	229	381
29	163	301	290	436	599
30	250	300	300	1417	1134
31	23	28	30	28	83
32	33	43	40	47	76
33	172	234	212	756	1112
34	133	194	190	647	925
35	130	161	147	565	840
36	37	44	43	106	127
37	162	199	188	875	921
38	129	164	153	258	403
39	56	82	77	127	224
40	73	131	129	274	458
41	49	62	62	209	315
42	160	213	213	314	552
43	49	66	68	130	238
44	224	441	474	623	956
45	138	180	172	406	544
46	14	16	16	16	20
47	67	77	76	153	385
48	65	91	82	180	274
49	99	155	150	192	249
50	159	224	209	2325	1389
51	91	122	118	265	409
52	160	191	191	1549	1004
53	77	109	96	194	383
54	32	40	40	58	84
55	248	397	379	847	811
56	29	36	33	45	64
57	149	193	184	961	1092
58	106	274	259	447	636
59	63	74	76	104	142
60	125	187	173	225	456
61	95	122	115	332	547
62	143	192	167	761	833
63	17	18	19	20	32
64	38	45	44	84	112
65	105	142	131	435	500
66	92	117	113	164	272
67	35	42	38	55	105
68	136	191	194	359	582
69	81	105	101	923	640
70	188	444	433	652	873
71	141	179	179	820	833
72	59	113	117	255	429
73	44	54	49	118	142
74	86	130	129	275	316
75	77	98	96	132	303
76	114	184	180	387	648
77	51	63	61	322	368
78	64	82	79	152	333
79	144	204	196	762	1082
80	35	46	43	91	127
81	92	120	116	464	683
82	96	108	112	218	268
83	178	262	257	446	852
84	84	101	101	180	219
85	18	19	19	62	64
86	89	153	134	175	237
87	164	207	195	400	468
88	59	98	82	98	128
89	191	326	314	843	1155
90	210	242	240	1318	1470
91	95	122	116	799	611
92	139	166	165	858	572
93	25	25	26	94	97
94	63	82	78	506	415
95	241	549	528	1730	1935
96	44	53	53	163	184
97	36	48	49	94	143
98	47	58	55	196	281
99	34	44	44	44	54
100	122	186	183	334	695
Average	**98.6**	**140.2**	**136.2**	**422.2**	**486.6**
Geo mean	**81.8**	**110.3**	**107.1**	**266.4**	**349.6**

Table 8: Computational Results for Multi-period power planning (CCP), $T = 30, N = 20$

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	29	34	32	54	269
2	53	61	58	89	343
3	73	111	118	90	515
4	70	85	87	96	539
5	42	58	59	66	290
6	16	16	16	20	264
7	41	51	67	78	307
8	36	43	39	44	271
9	43	49	46	66	369
10	51	56	55	84	455
11	36	38	41	63	288
12	32	34	40	40	361
13	35	47	49	58	290
14	42	50	51	89	307
15	42	44	46	61	385
16	47	57	58	64	362
17	41	42	49	54	289
18	23	27	27	27	222
19	51	59	51	67	443
20	74	84	86	115	564
21	32	35	40	35	238
22	65	78	94	88	474
23	44	52	53	54	346
24	28	29	30	37	251
25	27	31	37	30	215
26	34	40	44	50	278
27	22	23	31	29	224
28	86	118	112	133	520
29	38	43	46	60	388
30	21	24	23	28	216
31	34	35	35	39	288
32	27	40	40	39	328
33	60	66	69	77	458
34	24	31	33	33	315
35	23	30	28	32	225
36	16	20	20	19	110
37	43	53	45	51	494
38	40	44	69	73	352
39	61	84	89	111	467
40	42	50	46	55	402
41	53	57	57	88	434
42	22	23	25	34	219
43	105	113	118	164	632
44	41	61	62	63	358
45	23	25	24	37	254
46	35	36	38	38	361
47	30	38	38	32	290
48	43	46	48	53	428
49	49	71	71	63	391
50	27	36	38	34	297
51	40	49	47	63	415
52	28	31	33	31	263
53	55	64	57	126	508
54	46	48	50	72	386
55	74	91	88	111	511
56	62	71	71	82	407
57	22	38	26	32	192
58	46	48	46	106	434
59	27	40	51	32	230
60	28	31	32	36	301
61	45	64	57	59	357
62	45	65	54	76	411
63	40	44	41	50	322
64	39	66	65	45	313
65	42	47	57	60	367
66	21	25	30	33	222
67	98	120	119	136	602
68	25	35	33	36	321
69	36	39	42	75	374
70	32	34	40	44	383
71	43	56	50	58	448
72	51	53	55	66	344
73	43	62	65	79	391
74	15	16	15	15	177
75	46	60	71	72	334
76	19	21	21	34	276
77	50	58	59	70	515
78	44	47	54	81	427
79	43	48	55	90	338
80	22	25	32	34	275
81	20	25	23	32	236
82	44	50	52	58	337
83	33	55	54	49	370
84	68	84	90	117	617
85	52	60	60	87	373
86	33	42	36	48	341
87	33	42	38	47	373
88	66	76	84	96	488
89	17	25	25	22	177
90	71	77	79	113	420
91	20	28	28	22	246
92	47	56	74	66	404
93	77	83	93	107	603
94	44	59	60	60	354
95	32	35	35	42	281
96	15	16	16	23	192
97	27	28	32	31	242
98	32	33	33	42	311
99	38	48	52	85	308
100	76	89	98	112	511
Average	**41.4**	**49.9**	**51.6**	**61.7**	**354.1**
Geo mean	**37.9**	**45.3**	**46.8**	**54.8**	**337.9**

Table 9: Computational Results for Portfolio Optimization (CCP), $n = 30$, $m = 20$, $k = 4$

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	130	237	215	284	761
2	122	201	210	228	722
3	63	149	101	132	488
4	96	235	186	157	668
5	63	89	78	106	400
6	81	126	118	138	522
7	97	201	201	232	599
8	72	129	112	228	572
9	80	186	181	228	617
10	130	186	181	239	779
11	93	107	96	231	669
12	64	107	96	136	520
13	81	128	146	143	655
14	72	104	112	115	511
15	118	191	270	293	757
16	74	148	156	142	475
17	166	230	243	389	989
18	187	288	347	418	1063
19	102	244	228	237	688
20	51	116	124	122	398
21	130	304	218	478	841
22	62	122	119	173	439
23	89	201	179	230	736
24	59	93	118	145	508
25	60	91	93	96	376
26	104	268	284	215	668
27	101	196	242	214	696
28	94	170	217	169	591
29	68	121	143	112	525
30	67	125	98	161	435
31	141	286	302	224	827
32	78	178	181	175	670
33	104	185	201	215	679
34	38	68	58	121	374
35	88	233	177	171	665
36	98	179	169	179	696
37	96	181	183	202	868
38	103	179	173	185	737
39	81	191	200	146	663
40	93	145	148	206	551
41	127	225	194	251	719
42	64	98	95	135	382
43	124	223	212	263	721
44	76	129	152	142	557
45	78	160	140	208	655
46	80	182	146	214	615
47	167	249	250	372	945
48	119	228	237	214	750
49	114	186	186	299	653
50	78	145	162	181	677
51	87	149	146	166	557
52	81	120	132	187	552
53	156	199	204	351	1025
54	137	248	243	255	910
55	102	172	178	170	684
56	116	233	203	244	740
57	147	300	316	255	1066
58	136	278	260	350	1067
59	123	193	196	189	766
60	99	122	129	242	715
61	106	243	213	283	713
62	78	118	120	198	527
63	106	137	150	207	693
64	104	168	164	234	516
65	100	184	149	303	784
66	105	175	181	252	659
67	99	140	140	222	666
68	80	145	145	140	523
69	72	116	144	134	458
70	61	83	79	108	504
71	142	228	218	338	899
72	121	179	178	215	728
73	184	354	360	332	1010
74	87	151	154	182	766
75	109	227	164	252	735
76	172	312	316	370	915
77	80	158	142	114	561
78	64	119	98	131	459
79	86	136	123	206	562
80	69	97	112	144	517
81	195	393	384	413	1099
82	90	156	188	239	656
83	130	196	198	254	792
84	95	196	205	271	651
85	94	157	168	190	683
86	177	299	297	378	947
87	125	263	238	305	787
88	58	95	95	98	327
89	64	133	149	115	377
90	115	231	227	230	859
91	116	166	175	229	793
92	130	199	228	271	909
93	115	217	217	287	792
94	99	155	168	166	646
95	103	220	240	202	578
96	87	156	167	147	530
97	206	346	326	429	1173
98	138	218	203	370	866
99	104	227	196	251	724
100	70	109	111	152	700
Average	**102.4**	**183.6**	**182.2**	**220.7**	**682.3**
Geo mean	**97.4**	**172.8**	**171.1**	**206.5**	**659.1**

Table 10: Computational Results for Stable Set Polytope on Bipartite Graph with Knapsack Constraint

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	17	20	20	21	25.5
2	481	567	567	579	569.5
3	62	73	73	91	83.9
4	213	319	319	282	286.9
5	364	472	472	457	459.6
6	85	102	102	101	117.7
7	170	195	195	194	229.9
8	246	294	294	319	357.7
9	131	156	156	155	192.3
10	159	187	187	207	218.9
11	153	236	236	239	297.7
12	114	131	131	127	127.4
13	176	254	254	271	290.5
14	218	279	279	276	304.9
15	50	56	56	67	67.8
16	141	181	181	195	188.2
17	55	67	67	70	78.1
18	166	176	176	230	230.2
19	495	619	619	704	761.7
20	100	146	146	134	130.8
21	332	491	491	477	542.4
22	418	485	485	492	512.2
23	24	27	27	38	35.9
24	135	194	194	199	225.8
25	244	290	290	423	347.7
26	277	387	387	389	393.4
27	34	54	54	53	52.2
28	195	263	263	301	289.4
29	112	153	153	149	198.3
30	69	89	89	100	99.8
31	115	133	133	151	159.5
32	53	75	75	82	75.8
33	175	219	219	222	243.9
34	146	216	216	207	217.2
35	20	26	26	24	28.2
36	77	105	105	96	103.6
37	538	690	690	718	820.9
38	114	125	125	126	135.3
39	60	87	87	93	102
40	47	53	53	59	56.9
41	105	127	127	132	177.1
42	298	457	457	372	385.5
43	108	135	135	151	146.4
44	80	117	117	128	119.4
45	83	103	103	107	107
46	342	439	439	452	455.3
47	55	95	95	103	93.8
48	329	394	394	401	437.3
49	65	87	87	111	98.9
50	123	144	144	139	158
51	387	443	443	444	554
52	192	243	243	273	294.5
53	134	159	159	174	188.6
54	68	78	78	83	92.9
55	93	104	104	109	151.7
56	221	257	257	266	282.1
57	219	304	304	293	322.6
58	258	310	310	287	337.2
59	272	296	296	304	341.5
60	206	362	362	254	260.9
61	238	321	321	275	315.1
62	342	459	459	533	595.1
63	134	167	167	170	181.2
64	114	136	136	140	151.1
65	116	331	331	273	265.4
66	133	178	178	185	191.5
67	98	115	115	117	164.6
68	245	350	350	341	379.7
69	266	366	366	342	393.4
70	307	389	389	443	459
71	142	185	185	172	172.4
72	53	85	85	76	75.8
73	86	131	131	120	130.1
74	82	100	100	100	110.2
75	27	39	39	37	41.5
76	98	156	156	139	142.9
77	191	300	300	217	267.7
78	219	261	261	281	335
79	354	492	492	442	475.5
80	360	410	410	413	542.3
81	76	99	99	101	125.2
82	190	246	246	248	265.8
83	99	114	114	125	131.3
84	199	273	273	275	251.5
85	95	138	138	136	147.6
86	40	54	54	48	51.4
87	59	76	76	71	76.4
88	309	437	437	402	443.7
89	354	404	404	433	435
90	181	264	264	240	306.2
91	40	70	70	76	77.5
92	295	430	430	439.9	488
93	247	321	321	313	358.8
94	411	569	569	582	819.3
95	170	222	222	228	276.8
96	112	148	148	154	154.2
97	438	595	595	790	649
98	502	706	706	728	717.5
99	37	46	46	48	50.4
100	275	368	368	410	412.6
Average	**176.3**	**230.9**	**230.9**	**235.8**	**255.8**
Geo mean	**137.6**	**180.8**	**180.8**	**185.5**	**199.6**

Table 11: Computational Results for Vertex Cover instances with $|V| = 20,\ p = 0.75$

Index	Opt Tree	SB-L	SB-P	Most inf	Rand
1	8	8	8	11	13
2	7	8	8	14	14
3	9	10	11	14	15
4	7	9	9	12	12
5	9	14	10	15	13
6	7	9	10	13	15
7	7	9	8	9	11
8	9	10	11	12	15
9	7	7	9	9	11
10	9	11	10	13	15
11	7	9	8	11	12
12	9	12	11	12	12
13	8	12	10	13	12
14	8	9	9	14	13
15	9	11	10	15	14
16	9	10	11	14	13
17	7	8	8	10	13
18	6	9	8	10	13
19	6	10	8	8	10
20	9	11	12	17	15
21	7	9	9	12	13
22	8	12	11	10	11
23	7	10	10	12	14
24	9	12	10	13	13
25	9	10	10	17	14
26	7	9	9	10	11
27	9	10	10	14	16
28	8	8	9	10	11
29	8	8	8	14	14
30	9	10	12	16	14
31	8	10	9	19	13
32	7	10	10	12	12
33	9	12	12	14	13
34	7	8	8	10	13
35	4	5	7	9	8
36	5	8	8	7	9
37	9	10	10	12	13
38	6	11	11	13	12
39	6	8	8	14	12
40	7	10	9	15	13
41	5	8	10	11	10
42	9	11	10	19	16
43	6	11	8	11	11
44	6	6	6	10	13
45	6	8	9	12	13
46	6	11	11	10	11
47	5	9	6	13	9
48	7	9	8	12	11
49	5	7	9	11	10
50	8	10	10	12	13
51	9	12	11	21	16
52	9	11	10	17	16
53	5	7	9	12	9
54	8	11	9	8	12
55	8	9	9	13	12
56	8	10	9	18	17
57	9	10	11	10	13
58	8	11	10	13	14
59	9	10	10	15	16
60	7	9	9	11	13
61	6	8	7	11	11
62	9	13	13	14	14
63	9	10	10	19	16
64	6	7	7	9	9
65	8	11	9	11	13
66	10	12	12	19	15
67	6	8	8	9	10
68	9	10	10	14	13
69	7	10	10	10	14
70	7	9	9	12	10
71	7	9	9	10	11
72	5	11	11	9	12
73	8	9	9	10	12
74	4	7	7	10	8
75	8	10	11	15	14
76	6	10	10	14	11
77	6	10	10	13	11
78	7	9	9	11	11
79	7	9	8	11	12
80	6	7	7	11	11
81	7	11	11	11	12
82	7	9	8	11	11
83	7	8	8	12	11
84	8	10	10	11	13
85	7	9	7	12	10
86	7	10	8	15	13
87	8	10	10	14	12
88	5	5	7	13	9
89	4	5	6	6	8
90	8	12	10	15	14
91	5	7	7	9	10
92	6	9	10	12	12
93	4	8	7	9	9
94	6	6	6	16	13
95	7	12	11	9	11
96	8	11	12	15	16
97	8	9	9	14	13
98	7	9	8	14	14
99	6	9	9	12	11
100	7	8	11	14	14
Average	**7.2**	**9.4**	**9.2**	**12.4**	**12.3**
Geo mean	**7.1**	**9.3**	**9.1**	**12.1**	**12.2**