Estimating the Size of Search Trees by Sampling with Domain Knowledge

Gleb Belov[1,*], **Samuel Esler**[1], **Dylan Fernando**[1], **Pierre Le Bodic**[1,†], **George L. Nemhauser**[2]
[1] Faculty of Information Technology, Monash University, Clayton, Australia
[2] School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, USA

Abstract

We show how recently-defined abstract models of the Branch & Bound algorithm can be used to obtain information on how the nodes are distributed in B&B search trees. This can be directly exploited in the form of probabilities in a sampling algorithm given by Knuth that estimates the size of a search tree. This method reduces the offline estimation error by a factor of two on search trees from Mixed-Integer Programming instances.

1 Introduction

Tree-search procedures are typically employed to solve problems that require some degree of enumeration. The efficiency of the tree search lies in whether it can be performed implicitly rather than explicitly, i.e. if parts of the search space can be discarded without being fully explored. One flaw of implicit enumeration procedures is that it is a priori not clear how implicit the search will be, and therefore the running time of these procedures is hard to estimate. Since there are real-life problems for which current state-of-the-art tree-search techniques and hardware cannot provide a provably optimal solution within years, a runtime (or tree-size) estimate would be very beneficial.

Besides informing the user when the program is likely to terminate, a running time estimate could be used to select the best search strategy in a short period of time from a few that have been tested [Lobjois and Lemaître, 1998]. It could also inform a restart procedure that needs to know how long the search is going to take to decide how beneficial a restart would be. Finally, as more tree-search algorithms become parallelized, having a reliable estimate of (sub)tree-sizes can be invaluable for better load balancing and thus faster parallel computations [Marcos Alvarez *et al.*, 2014].

Although the end goal of this research is to allow online predictions of the size of search trees, it seems unreasonable to plan on solving the problem of online prediction without having first solved the problem of offline prediction. Both online and offline methods estimate the tree-size given *partial* knowledge (i.e. a subtree S) of a tree T. The main difference is that in an online prediction, the available subtree S depends on the search procedure, whereas in an offline prediction, S can be chosen in a way that increases the precision of the estimation method, since there is no real search to be performed. Note that with this definition, if one supposed that online tree-size prediction methods were allowed information that is not provided by the search procedure (e.g. by visiting nodes not yet traversed by the search), then it would essentially be equivalent to an offline prediction method. The purpose of this paper is thus to propose and evaluate an offline tree-size estimation method, which, with further research, could be extended to work in an online setting.

1.1 Contributions

Our work builds on the models of the Branch & Bound (B&B) algorithm given in [Le Bodic and Nemhauser, 2017] and on the sampling procedure described by [Knuth, 1975]. Essentially, Knuth gives a sampling method that can use knowledge of the distribution of nodes in a tree to reduce the variance of tree-size estimates, and Le Bodic and Nemhauser give some models from which domain knowledge on B&B trees can be derived with some degree of precision. This paper is technically simple, but establishes how these two methods can be combined to obtain significant increase in estimation precision. In our experiments, the offline estimation error is more than halved. The paper thus emphasizes the complementarity of the two methods. The direct applicability of our method is limited to those applications for which domain knowledge on the distribution of nodes in search trees is available, but we anticipate that this study will lead to results similar to those given in [Le Bodic and Nemhauser, 2017] for other procedures that rely on search trees.

1.2 Related Work

We specifically address previous work on tree-size estimation related to Knuth's method [Knuth, 1975] (presented in Section 2) or to Mixed-Integer Programming (MIP). For a general overview, we refer the reader to [Thayer *et al.*, 2012] and references therein. This paper is not the first attempt at predicting tree-sizes for MIPs, but the fact that there is currently no built-in tree-size prediction in commercial MIP solvers implies that no existing technique satisfies industry standards, and that further research is needed.

[*] Authors are ordered alphabetically.
[†] Corresponding author: pierre.lebodic@monash.edu

Knuth's sampling method [Knuth, 1975] has been extended and applied, notably in [Purdom, 1978], which allows more than one branch to be investigated, and [Chen, 1992], who uses stratified sampling. Some results in [Kilby *et al.*, 2006] further explore Knuth's method in an online fashion. In [Lelis *et al.*, 2013], the authors propose an extension of [Knuth, 1975] and [Chen, 1992] for Branch-and-Bound with a Depth-First Search (DFS) node selection strategy. However, in MIP solvers, DFS requires 30% more time than the default node selection strategy [Achterberg, 2007, p. 79], therefore such an estimation method would not be practical for MIP (see discussion in [Cornuéjols *et al.*, 2006, p. 2]).

For MIPs, in [Cornuéjols *et al.*, 2006], a parameter γ_i is defined for each level i of the B&B tree, which encodes the growth rate of nodes between levels. The sequence of γ_i is estimated using a model (determined offline) linear in three auxiliary tree parameters. A more granular progress measure than the traditional MIP gap, the *sum of subtree gaps* (SSG), is proposed in [Özaltın *et al.*, 2011]. The authors use SSG in a statistical model that can estimate the running time of MIP solvers.

2 Tree Sampling à la Knuth

In this section we revisit the ideas of [Knuth, 1975]. We suppose that a rooted tree $T = (V, A)$ is given, and that $I \subset V$ denotes the set of inner nodes including the root. Knuth proposes to estimate tree-sizes by randomly sampling T. A *sample* is defined as a path from the root r of T to one of its leaves. To obtain a sample, one simply starts at the root r, and at each node v randomly selects an arc $vw \in A$ (if any) with some predefined probability $p(vw)$. At each inner vertex $v \in I$, the probabilities satisfy $\sum_{vw \in A} p(vw) = 1$. The cost of processing a node $v \in V$ may in general vary, and is encoded by $c(v)$, and we suppose that for two nodes v and w at the same depth, the cost function satisfies $c(v) = c(w)$. If the costs are *unitary*, i.e. $c(v) = 1$ for every node $v \in V$, then the cost of the tree corresponds to its size.

Algorithm 1 formally describes the sampling process. It outputs the estimated cost to traverse a tree T given a single random sample. At step 2 of the algorithm, variable v stores the current node, variable D corresponds to the estimated number of nodes at the level of node v, and variable C corresponds to the estimated cost of exploring T minus the estimated cost of exploring T_v, the subtree rooted at v, plus $c(v)$.

Algorithm 1 Knuth's sampling algorithm

Input: A tree $T = (V, A)$ with root r, $p(vw)\ \forall vw \in A$.
Output: The estimated cost to traverse the tree.
1: $v = r, D = 1, C = c(v)$.
2: **while** Node v is not a leaf **do**
3: Select $vw \in A$ with probability $p(vw)$.
4: $D = D/p(vw)$.
5: $C = C + c(v)D$.
6: $v = w$.
7: **end while**
8: Return C.

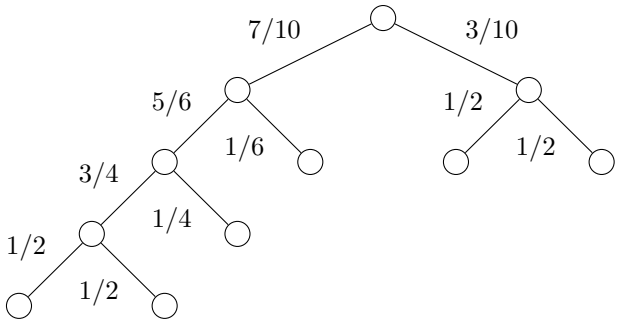


Figure 1: Example of a binary tree $\mathcal{T}$. The arc labels are sampling probabilities that "perfectly" reflect the node distribution of $\mathcal{T}$.

# samples		1	2	3	4	5	6
Var.	w/	64	18.67	11.06	8.95	8.09	7.64
	w/o	64	14.89	7.64	4.49	1.57	0

Table 1: Variance of the estimated size of $\mathcal{T}$, depending on the number of samples, with (w/) or without (w/o) replacement, for uniform probabilities (data is rounded).

2.1 Sampling with Uniform Probabilities

We use the binary tree $\mathcal{T}$ depicted in Figure 1 as a running example, and we suppose costs are unitary, therefore Algorithm 1 estimates the tree-size. In this section, we suppose that the arc probabilities are $1/2$ uniformly (and thus for now we will disregard the arc labels in Figure 1). Using uniform probabilities, Algorithm 1 will return $2^{d+1} - 1$ upon reaching a leaf at depth d. The probability of a sample reaching a leaf at depth 2 (resp. 3, 4) is $3/4$ (resp. $1/8$, $2/16$), therefore the average tree-size of these samples amounts to

$$\frac{3}{4} * 7 + \frac{1}{8} * 15 + \frac{1}{8} * 31 = 11,$$

the exact size of $\mathcal{T}$. In fact, Knuth (Theorem 1) proves that the expected tree-size (resp. cost, in the general case) computed with Algorithm 1 is equal to the actual tree-size (resp. cost), i.e. it is an *unbiased* estimator. However, Knuth indicates that, if one uses uniform probabilities, this sampling method suffers from a high variance (see also [Kilby *et al.*, 2006]).

Table 1 gives the exact variance of the estimated size of $\mathcal{T}$ depending on the number of samples, *with* or *without* replacement. A sample of size 6 drawn from $\mathcal{T}$ without replacement necessarily contains each sample once, and since the tree-size estimation is unbiased, the exact tree-size is retrieved. Indeed, drawing each sample once can be viewed as an exhaustive traversal of $\mathcal{T}$. Table 1 clearly illustrates the benefits that sampling without replacement brings in variance reduction. Furthermore, since our end goal is to have an online estimation method for the B&B algorithm, which traverses each leaf once, we only consider sets of samples obtained without replacement throughout the paper. Even when sampling without replacement, the variance remains high for a number of samples smaller than 6. Knuth provides an improved method, called *importance sampling*, in the case where some knowledge of T is available in the form of non-uniform sampling probabilities.

2.2 Importance Sampling

Importance sampling defines the probabilities $p(vw), vw \in A$ to reflect the node distribution within T supposing one possesses knowledge of it, and in turn reduces the variance of the tree-size estimates.

We consider in this section the example tree $\mathcal{T}$ with non-uniform probabilities, as shown on Figure 1. We refer to these probabilities as "perfect", as the exact sizes of all subtrees are used to define them. At the root node, there are 7 nodes to the left, and 3 nodes to the right, therefore the probability to go left is defined as 7/10, and 3/10 to go right. By doing so at every inner node, the probability to sample left (resp. right) is always proportional to the size of the left (resp. right) subtree. Running Algorithm 1 on $\mathcal{T}$ with these probabilities, we can verify that any sample of $\mathcal{T}$ perfectly estimates its size. For example, if Algorithm 1 terminates at the left-most leaf of $\mathcal{T}$, it returns

$$C = 1 + \frac{10}{7}\left(1 + \frac{6}{5}(1 + \frac{4}{3}(1 + 2))\right) = 11,$$

which is the exact size of $\mathcal{T}$. Knuth (Theorem 2) proves that this property always holds if the probabilities are defined "perfectly", and that the size of any tree can thus be estimated with zero variance using a single sample. This result may seem "too good to be true", and in a sense it is, as defining those "perfect" probabilities requires knowing the exact relative size of every subtree.

Note that when sampling with uniform probabilities as in Section 2.1, we implicitly suppose that all leaves of $\mathcal{T}$ have the same depth d as the sampled leaf. Using probabilities that do not reflect the distribution of nodes in the tree does not change the nature of the sampling procedure, but it may greatly affect the precision of the estimates, as illustrated in Sections 2.1 and 2.2. We show in Section 3 and throughout how the node distribution of B&B trees for MIP can be approximated and used as probabilities in importance sampling.

3 Abstract Models for B&B Trees

3.1 Previous Work

In the B&B algorithm for a minimization MIP, a Linear Program (LP) in which some of the variables are required to be integral, an LP *relaxation* is solved at each node n and provides a *lower bound* b_n on the optimal value in the (sub)space of feasible solutions at n [Nemhauser and Wolsey, 1988]. If the lower bound b_n is at least the value V of the best solution found so far, n can be pruned. Otherwise, a variable i, required to be integer but fractional in the LP relaxation, is branched on at n, creating two children v and w, with lower bounds $b_v \geq b_n$ and $b_w \geq b_n$, respectively. The state-of-the-art branching heuristics try to select i for branching that maximizes the ("*l*eft" and "*r*ight") lower bound improvements $l_i = b_v - b_n$ and $r_i = b_w - b_n$, in order to accelerate pruning in that subtree [Achterberg, 2007]. It is not clear however which i should be branched on if there are multiple non-dominated pairs (l_i, r_i), as this choice affects the balance of the tree as well as its size.

Le Bodic and Nemhauser address this question by designing abstract models to identify the variable that minimizes the B&B tree-size. They define a branching as a couple of values $(l, r) \in \mathbb{Z}_+^2$, with $l \leq r$, such that given an inner node of value $x \in \mathbb{Z}_+$, a branching at that node creates two children of value $x + l$ and $x + r$. In the *Single Variable Branching* problem (SVB), we are given a branching $(l, r) \in \mathbb{Z}_+^2$, as well as a target value $V \in \mathbb{Z}_+$, and we want to determine the minimum size of a binary tree that has value 0 at the root, and for which each node with value less than V is branched on. SVB models the B&B algorithm in a simplified setting, where the lower bound improvements l and r are constant at each node, and the optimal value V is known at the root node. Le Bodic and Nemhauser give Figure 2 as an example, for branching $(2, 5)$ and value $V = 6$. For a branching (l, r) and a target value V, the size $t(V)$ of the SVB tree is given by the recursive formula

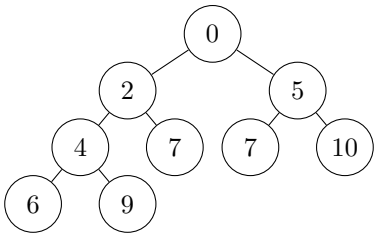


Figure 2: SVB tree of size 9 with branching $(2, 5)$ and $V = 6$.

$$t(V) = \begin{cases} 0 & \text{if } V \leq 0, \\ 1 + t(V - l) + t(V - r) & \text{otherwise.} \end{cases} \quad (1)$$

Le Bodic and Nemhauser show that the characteristic trinomial

$$x^r - x^{r-l} - 1 \quad (2)$$

associated with the recursive formula (1) has a root $\varphi > 1$ that characterizes the growth of the SVB tree as the target value V becomes larger. Formally, they prove

$$\lim_{V \to \infty} \frac{t(V + r)}{t(V)} = \varphi^r.$$

Furthermore, given multiple branchings (l_i, r_i), each a separate input to the SVB problem, the branching that has the minimum φ_i yields the smallest tree for large enough target values V. Perhaps surprisingly, the authors show that the value φ encapsulates enough of the complexity of the B&B algorithm that a rule that selects the branching with minimum φ is better than the state-of-the-art branching heuristics in MIP benchmarks, both in number of nodes in B&B trees and overall computing time.

3.2 Extracting Information on Node Distribution from B&B Models

These results are relevant in the context of sampling, as for a branching (l, r), we have

$$\lim_{V \to \infty} \frac{t(V - l)}{t(V)} = \varphi^{-l}, \quad \lim_{V \to \infty} \frac{t(V - r)}{t(V)} = \varphi^{-r}. \quad (3)$$

In other words, in a SVB tree, the portion to the left (resp. right) of the tree is φ^{-l} (resp. φ^{-r}), with increasing precision

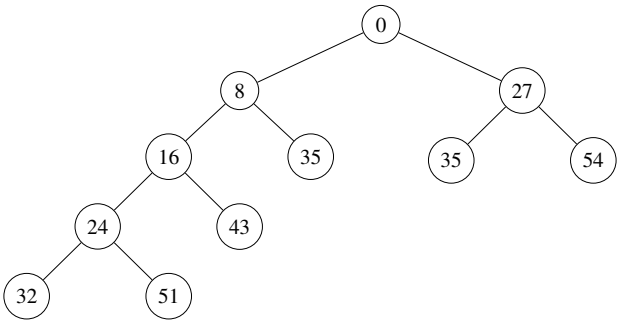


Figure 3: Example of tree $\mathcal{T}$ that closes a target value $V = 32$ by branching on $(8, 27)$ at every node.

Sampl. prob.	0.24	0.10	0.15	0.21	0.21	0.09
Estimated t-s	11.6	17.1	11.3	7.20	9.10	15.4

Table 2: Probability of being sampled and estimated tree-size for each sample (i.e. leaf) of $\mathcal{T}$, from the left to the right (data is rounded).

as V grows large. Note that, although it is not stated in the original article, it follows both from (2) and (3) that

$$\varphi^{-l} + \varphi^{-r} = 1. \tag{4}$$

In a B&B tree T, for each inner node $n \in I$ with branching (l, r) resulting in children v and w, we define

$$p_k(nv) = \varphi^{-l}, \qquad p_k(nw) = \varphi^{-r},$$

as the probabilities established from domain *k*nowledge, to be used in Algorithm 1. By contrast, we will denote p_u the *u*niform probabilities, and we will occasionally use p_k and p_u to refer directly to the sampling method that uses one of these probability schemes. The purpose of the upcoming sections is to ascertain with what degree of precision p_k represents the distribution of nodes in T, and if and how it can improve tree-size estimates if used in Algorithm 1.

4 Importance Sampling on SVB Trees

We show with an example how p_k can be used to increase the precision of tree-size estimates in the SVB problem, by reducing the variance of a sample set. Consider the SVB tree given in Figure 3, built using branching $(l = 8, r = 27)$, and closing a target value $V = 32$. Branching $(8, 27)$ (with $\varphi = 1.04560$) has been chosen because it has the property

$$\varphi^{-l} = 0.69998 \approx \frac{7}{10} \qquad \varphi^{-r} = 0.30002 \approx \frac{3}{10}$$

with a five decimal precision, so that these values correspond closely to the probabilities $7/10$ and $3/10$ at the root node in Figure 1. Using φ^{-l} and φ^{-r} as the left and right arc probabilities at every inner node of $\mathcal{T}$, the probability of drawing a given sample and the tree-size of this sample as estimated by Algorithm 1 is given by Table 2. The effect of introducing arc probabilities that reflect the distribution of the tree can be seen in row 1 of Table 3: the variance decreases drastically compared to Table 1. Even though we are dealing with SVB, a simplification of the B&B model, we find surprising that such an increase in precision can be achieved.

# samples	1	2	3	4	5	6
(0.70, 0.30)	9.50	3.17	1.55	0.80	0.31	0
(0.61, 0.39)	20.74	7.22	3.59	1.83	0.70	0
(0.73, 0.27)	9.93	2.72	1.34	0.75	0.30	0

Table 3: Variance of the estimated size of $\mathcal{T}$, depending on the number of samples, without replacement, for three choices of probabilities (data is rounded).

There are other choices of branching (l, r) and target value V that could fit our example tree $\mathcal{T}$ in the SVB setting, i.e. such that the value at every leaf is $\geq V$, and the value at every inner node is $< V$ (e.g. $(l = 3, r = 10), V = 12$). One could thus ask how "bad" a branching could get (for prediction purposes) while still fitting the tree $\mathcal{T}$. Observe that, since it is an SVB tree, two of the leaves of $\mathcal{T}$ imply

$$4l \geq V, \qquad l + r \geq V,$$

and their parents imply, respectively,

$$3l < V, \qquad r < V,$$

therefore

$$l + r \geq V > 3l \Rightarrow r > 2l,$$
$$4l \geq V > r \Rightarrow r < 4l.$$

Plugging $r \in (2l, 4l)$ into (4), which is monotonic, yields bounds $\varphi^{-l} \in (0.61, 0.73)$ (and thus $\varphi^{-r} \in (0.27, 0.39)$). Row 2 (resp. 3) of Table 3 shows the variance of a set of samples if one uses 0.61 (resp. 0.73) as the probability to go left and 0.39 (resp. 0.27) as the probability to go right at every inner node. Row 2 shows that using probabilities $(0.61, 0.39)$, the variance is worse than for probabilities $\approx (0.70, 0.30)$ (row one), but better than for probabilities $(0.5, 0.5)$ (Table 1), which is a good sign that using the ratio φ to get probabilities for SVB trees reduces the variance compared to the simple sampling method. Perhaps surprisingly, row 3 shows improvement in variance over all previous examples (except the "perfect" case). The reason for this is that even though $(0.7, 0.3)$ are the probabilities that perfectly describe the node distribution at the root r of $\mathcal{T}$, this does not hold at the root of every subtree of $\mathcal{T}$. Indeed, 0.73 is closer to $5/6$ and $3/4$ than 0.7 is (refer to Figure 1).

5 Numerical Experiments on MIP B&B Trees

5.1 Data Generation

We have solved the 87 MIP instances of the MIPLIB 2010 "benchmark" test set [Koch *et al.*, 2011] with the MIP solver SCIP 3.2.1 [Achterberg, 2009], using default settings. Each B&B tree is then recorded in a file by SCIP using the VBC format (traditionally used to plot the B&B tree structure). Of the 87 instances, 8 timed out (after 24 hours), 3 exceeded the memory limit (4GB), 9 were too small (under 100 leaves), and 4 were too large (the VBC file several tens of gigabytes large), leaving 63 instances.

	p_k	p_u
aa(T)	0.14	0.24
wa(T)	0.17	0.32

Table 4: Precision error of p_k and p_u (data is rounded).

	Offline		Online	
Method	p_k	p_u	p_k	p_u
Wins	509	121	43	20
Median error	5.74%	12.20%	34.11%	38.94%
Geometric error	5.45%	13.82	43.01%	1839%

Table 5: Results for p_k and p_u (data is rounded).

5.2 Comparing $\mathbf{p_k}$ to the Node Distribution in Trees

We first measure how precisely p_k reflects the distribution of nodes in trees compared to p_u. At each inner node $v \in I$ we measure the absolute error

$$e(v) = \left| p(vw) - \frac{size(w)}{size(v) - 1} \right|, \qquad vw \in A$$

where $size()$ returns the size of the subtree rooted at a node, and p can be either p_k or p_u. We use two error measures for a given tree, the first is the arithmetic average $aa(T)$, the second is an average weighted by the number of leaves in the subtree rooted at each inner node:

$$wa(T) = \frac{\sum_{v \in I}(size(v)+1)e(v)/2}{\sum_{v \in I}(size(v)+1)/2}.$$

This measure reflects the fact that an error at an inner node that is the root of a large subtree propagates to each of its leaves in Algorithm 1. Table 4 gives the average of the error measures $aa(T)$ and $wa(T)$ over the 63 instances for p_k and p_u. Note that the values provided are by definition in $[0, 1]$ for p_k and in $[0, .5]$ for p_u. Table 4 clearly shows that p_k better describes the distribution of nodes in (sub)trees than p_u. We now inquire if and how this increase in probability precision allows for better tree-size estimates.

5.3 Offline Tree-Size Estimation Results

We simulate the sampling of the leaves of a tree T with uniform probability and without replacement by placing all the leaves of T in a list, shuffling the list, and running Algorithm 1 in that order on each item of the list. This is done with 10 different seeds for each of the 63 trees to produce different permutations of the sampling order, providing a total test set of 630 *leaf lists*. For each leaf list, Algorithm 1 is run using p_k, then p_u. A leaf's individual estimate is given by the value C, and its weight by $\frac{1}{D}$, as described in Algorithm 1. The running estimate after k leaves have been drawn is given by

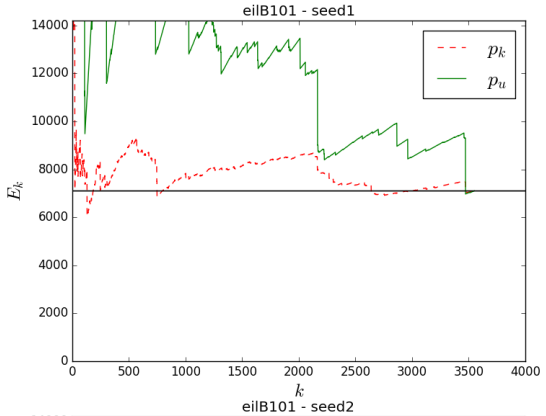
$$E_k = \frac{\sum_{s \in S} \frac{C_s}{D_s}}{\sum_{s \in S} \frac{1}{D_s}},$$

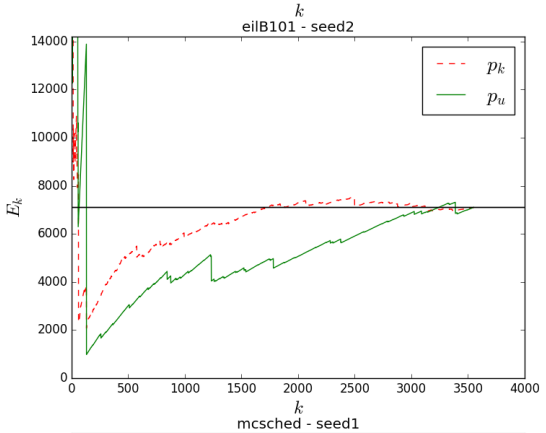
where S is the set of the first k leaves. Note that E_k with p_u is called "weighted backtrack estimator" in [Kilby *et al.*, 2006]. Figure 4a (on the left) provides the plot of a few instances. From top to bottom, the first two figures correspond to the same instance, eilB101, for two different seeds. The differences between the two plots reflect the high variability of the sampling method. Nevertheless, p_k clearly wins over p_u in both cases. The third plot shows an example where p_u outperforms p_k. The fourth plot shows an instance with a large tree where the opposite happens. In all four plots and in general, the trend is that the estimation error typically decreases non-monotonically and (by design) systematically reaches the exact tree-size when all samples are taken into account.

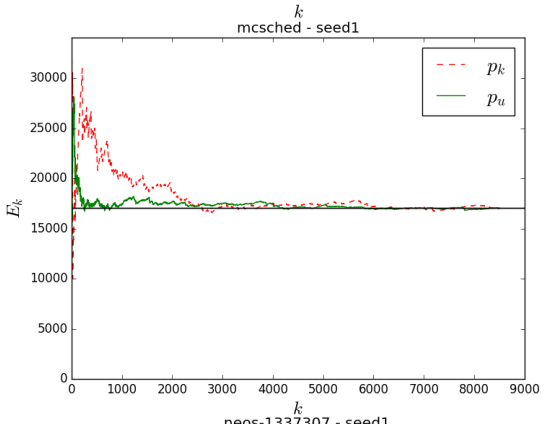
We now present aggregated results over the 63 instances. As a measure of estimation precision, we use the mean absolute percentage error (MAPE), which is a measure of the area-under-curve between the E_k's and exact tree-size. Note that we use a variation of MAPE, as we do not take E_k into account for $k \leq 100$, i.e. until we have at least 100 leaves, as the high variance inherent to the sampling method can cause leaves with extreme overestimates drawn at the very beginning to have a dominating effect on the MAPE, making comparisons essentially meaningless. Table 5, column "Offline", presents results for each method in number of wins, average over the 63 instances of the median MAPE error over the 10 seeds, and geometric mean of the MAPE across all 630 instances. The number of wins shows that using p_k rather than p_u is preferable on most instances, but not systematically. The other two measures both show that using p_k in Algorithm 1 more than *halves* the imprecision incurred by the use of p_u.

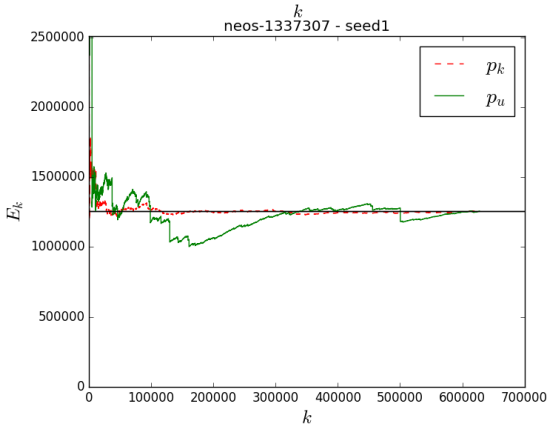
5.4 Online Tree-Size Estimation Results

We have applied the method described in Section 5.3 to the online setting, where instead of shuffling the leaf list, it is sorted according to the order in which the leaves are found (i.e. traversed) by the B&B algorithm. Figure 4b shows some examples (side-by-side with their offline counterparts). The first plot shows that p_u follows a very peculiar pattern, whereas p_k is quite similar to the offline version. The second plot shows how both methods can degrade drastically in the online case, one of them overestimating, the other underestimating. In the third plot, we can see that both methods essentially predict that the size of the tree will be twice the number of leaves found so far, which is a trivial lower bound for a binary tree. Unfortunately, many instances are similar to the third plot, which renders both methods impractical in their current form. The results are presented in Table 5, column "Online", with the same setup as the offline results, on the 63 non-permutated instances. Preliminary work has not allowed us to address this phenomenon satisfactorily. Further work is thus required to obtain reliable online estimates. One possible direction is a method that would in essence allow the online case to be treated similarly to the offline case (e.g. a resampling within the traversed leaves). However, even though a purely statistical method may be applied successfully, it would be beneficial both in theory and practice if one could obtain knowledge that would explain why the sampling method behaves so differently online and offline.

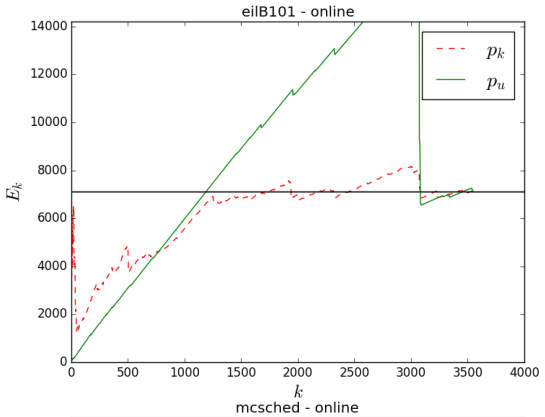


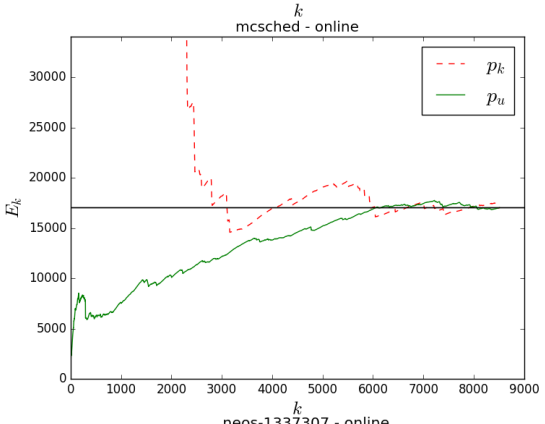


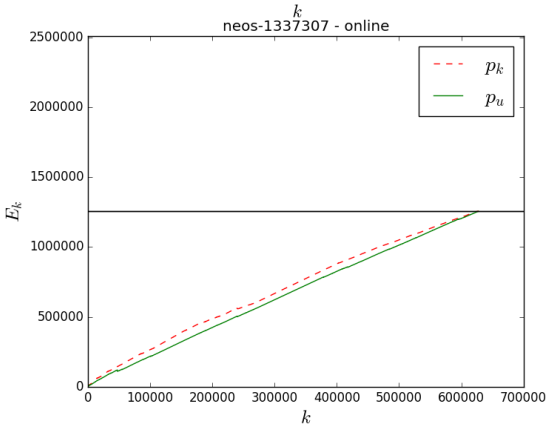




(a) Offline tree-size prediction







(b) Online tree-size prediction

Figure 4: Tree-size predictions using Algorithm 1 on some instances. The horizontal line indicates the actual tree-size.

References

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

[Achterberg, 2009] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.

[Chen, 1992] P. C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21(2):295–315, 1992.

[Cornuéjols *et al.*, 2006] G. Cornuéjols, M. Karamanov, and Y. Li. Early estimates of the size of branch-and-bound trees. *INFORMS Journal on Computing*, 18:86–96, 2006.

[Kilby *et al.*, 2006] P. Kilby, J. Slaney, S. Thiébaux, and T. Walsh. Estimating search tree size. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, pages 1014–1019. AAAI Press, 2006.

[Knuth, 1975] D. E. Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29:122–136, 1975.

[Koch *et al.*, 2011] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[Le Bodic and Nemhauser, 2017] P. Le Bodic and G. Nemhauser. An abstract model for branching and its application to mixed integer programming. *Mathematical Programming*, pages 1–37, 2017.

[Lelis *et al.*, 2013] L. H. S. Lelis, L. Otten, and R. Dechter. Predicting the size of depth-first branch and bound search trees. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 594–600. AAAI Press, 2013.

[Lobjois and Lemaître, 1998] L. Lobjois and M. Lemaître. Branch and bound algorithm selection by performance prediction. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 353–358, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[Marcos Alvarez *et al.*, 2014] A. Marcos Alvarez, Q. Louveaux, and L. Wehenkel. A supervised machine learning approach to variable branching in branch-and-bound. Technical report, Université de Liège, 2014.

[Nemhauser and Wolsey, 1988] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[Özaltın *et al.*, 2011] O. Y. Özaltın, B. Hunsaker, and A. J. Schaefer. Predicting the solution time of branch-and-bound algorithms for mixed-integer programs. *INFORMS Journal on Computing*, 23(3):392–403, July 2011.

[Purdom, 1978] P. W. Purdom. Tree size by partial backtracking. *SIAM Journal on Computing*, 7(4):481–491, 1978.

[Thayer *et al.*, 2012] J. T. Thayer, R. Stern, and L. H. S. Lelis. Are we there yet? - estimating search progress. In D. Borrajo, A. Felner, R. E. Korf, M. Likhachev, C. Linares López, W. Ruml, and N. R. Sturtevant, editors, *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*. AAAI Press, 2012.