Predicting the Size of Depth-First Branch and Bound Search Trees

Levi H. S. Lelis
Computing Science Dept.
University of Alberta
Edmonton, Canada T6G 2E8
santanad@cs.ualberta.ca

Lars Otten
Dept. of Computer Science
University of California, Irvine
Irvine, USA 92697-3435
lotten@ics.uci.edu

Rina Dechter
Dept. of Computer Science
University of California, Irvine
Irvine, USA 92697-3435
dechter@ics.uci.edu

Abstract

This paper provides algorithms for predicting the size of the Expanded Search Tree (EST) of Depth-first Branch and Bound algorithms (DFBnB) for optimization tasks. The prediction algorithm is implemented and evaluated in the context of solving combinatorial optimization problems over graphical models such as Bayesian and Markov networks. Our methods extend to DFBnB the approaches provided by Knuth-Chen schemes that were designed and applied for predicting the EST size of backtracking search algorithms. Our empirical results demonstrate good predictions which are superior to competing schemes.

1 Introduction

A frequently used heuristic search algorithm for solving combinatorial optimization problems is Depth-First Branch-and-Bound (DFBnB) [Balas and Toth, 1985]. DFBnB explores the search space in a depth-first manner while keeping track of the current best-known solution cost, denoted c^b. It uses an *admissible* heuristic function $h(\cdot)$, *i.e.,* a function that never overestimates the optimal cost-to-go for every node, and is guided by an evaluation function $f(n) = g(n)+h(n)$, where $g(n)$ is the cost of the path from the root node to n. Since $f(n)$ is an underestimate of the cost of an optimal solution that goes through n, whenever $f(n) \geq c^b$, n is pruned.

In practice DFBnB, especially if guided by an effective heuristic, explores only a small fraction of the usually exponentially large search space, and this varies greatly from one problem instance to the next. However, predicting the size of this Expanded Search Tree, or EST for short, is hard. It depends on intrinsic features of the problem instance that are not visible a priori (*e.g.,* the number of dead ends that may be encountered). The size of the EST may also depend on parameters of the algorithm and in particular on the strength of its guiding heuristic function. Available worst-case complexity analysis is blind to these hidden features and often provides uninformative, even useless, upper bounds.

Predicting the EST size could facilitate the choice of a heuristic on an instance by instance basis. Or, in the context of parallelizing search, a prediction scheme could facilitate load-balancing by partitioning the problem into subproblems of similar EST sizes [Otten and Dechter, 2012b].

Related work. Several methods have been developed for predicting the size of the search tree of backtracking and heuristic search algorithms such as IDA* [Korf, 1985]. See for instance initial work by Knuth [1975], Partial Backtracking by Purdom [1978], Stratified Sampling by Chen [1992], and other contributions [Korf *et al.*, 2001; Zahavi *et al.*, 2010; Burns and Ruml, 2012; Lelis *et al.*, 2013]. These schemes work by sampling a small part of the EST and extrapolating from it. The challenge in applying these sampling techniques to DFBnB lies in their implicit assumption of the "stable children" property. Namely, for every node in the EST, the set of EST children can be determined at the time of sampling. In the case of DFBnB, however, the set of children in the EST depends on c^b , which impacts the pruning but is generally not known at prediction time.

Contributions. In this paper we present *Two-step Stratified Sampling* (`TSS`), a novel algorithm for predicting the EST size of DFBnB that extends the work by Knuth [1975] and Chen [1992]. The algorithm performs multiple "Stratified Sampling runs" followed by a constrained DFBnB execution and exploits memory to cope with the stable children issue. We show that, if given sufficient time and memory, the prediction produced by `TSS` converges to the actual EST size.

We apply our prediction scheme to optimization queries over graphical models, such as finding the most likely explanation in Bayesian networks [Pearl, 1988] (known as MPE or MAP). In particular, we are interested in predicting the size of the search tree expanded by Branch and Bound with mini-bucket heuristic (BBMB) [Kask and Dechter, 2001], which has been extended into a competition-winning solver [Marinescu and Dechter, 2009; Otten and Dechter, 2012a]. In addition to comparing against pure `SS` we compare `TSS` to a prediction method presented by Kilby *et al.* [2006]. Empirical results show that, if memory allows, our prediction is effective and overall far superior to earlier schemes.

2 Formulation and Background

Given a directed, full search tree representing a state-space problem [Nilsson, 1980], we are interested in estimating the size of a subtree which is expanded by a search algorithm

while seeking an optimal solution. We call the former *the underlying search tree* (UST) and the latter *the Expanded Search Tree* (EST).

Problem formulation. Let $S = (N, E)$ be a tree representing an EST where N is its set of nodes and for each $n \in N$ $child(n) = \{n' | (n, n') \in E\}$ is its set of child nodes. Our task is to estimate the size of N without fully generating S.

Definition 1 (General prediction task). *Given any numerical function z over N, the general task is to approximate a function over the EST $S = (N, E)$ of the form*

$$\varphi(S) = \sum_{s \in N} z(s)\,,$$

If $z(s) = 1$ for all $s \in N$, then $\varphi(S)$ is the size of S.

Stratified Sampling. Knuth [1975] showed a method to estimate the size of search tree S by repeatedly performing a random walk from the start state. Under the assumption that all branches have a structure equal to that of the path visited by the random walk, one branch is enough to predict the structure of the entire tree. Knuth observed that his method was not effective when the EST is unbalanced. Chen [1992] addressed this problem with a stratification of the EST through a *type system* to reduce the variance of the sampling process. We call Chen's method Stratified Sampling (SS).

Definition 2 (Type System). *Let $S = (N, E)$ be an EST, $T = \{t_1, \ldots, t_n\}$ is a type system for S if it is a disjoint partitioning of N. If $s \in N$ and $t \in T$ with $s \in t$, we also write $T(s) = t$.*

Definition 3 (Perfect type system). *A type system T is perfect for a tree S if for any two nodes n_1 and n_2 in S, if $T(n_1) = T(n_2)$, then the two subtrees of S rooted at n_1 and n_2 have the same value of φ.*

Definition 4 (Monotonic type system). *[Chen, 1992] A type system is monotonic for S if it is partially ordered such that a node's type must be strictly greater than its parent's type.*

SS's prediction scheme for $\varphi(S)$ generates samples from S, called probes. Each probe p is described by a set A_p of representative/weight pairs $\langle s, w \rangle$, where s is a representative for the type $T(s)$ and w captures the estimated number of nodes of that type in S. For each probe p and its associated set A_p a prediction can be computed as:

$$\hat{\varphi}^{(p)}(S) = \sum_{\langle s,w \rangle \in A_p} w \cdot z(s)\,.$$

Algorithm 1 describes SS for a single probe. The set A is organized into "layers", where $A[i]$ is the subset of node/weight pairs at level i of the search tree – processing level i fully before $i + 1$ forces the type system to be monotonic. $A[0]$ is initialized to contain only the root node with weight 1 (line 1).

In each iteration (Lines 4 through 11), all nodes in $A[i]$ are expanded. The children of each node in $A[i]$ are considered for inclusion in $A[i + 1]$ if they are not to be pruned by the search algorithm based on upper bound c^b (Line 6). If a child s" of node s has a type t that is already represented in $A[i+1]$ by node s', then a *merge* action on s'' and s' is performed. In

Algorithm 1 Stratified Sampling, a single probe

Input: root s^* of a tree, type system T, and initial upper bound c^b.

Output: A sampled tree ST and an array of sets A, where $A[i]$ is the set of pairs $\langle s, w \rangle$ for the nodes $s \in ST$ expanded at level i.

```
1:  initialize A[0] ← {⟨s*, 1⟩}
2:  i ← 0
3:  while i is less then search depth do
4:    for each element ⟨s, w⟩ in A[i] do
5:      for each child s'' of s do
6:        if h(s'') + g(s'') < c^b then
7:          if A[i + 1] contains an element ⟨s', w'⟩ with
            T(s') = T(s'') then
8:            w' ← w' + w
9:            with probability w/w', replace ⟨s', w'⟩ in
              A[i + 1] by ⟨s'', w'⟩
10:         else
11:           insert new element ⟨s'', w⟩ in A[i + 1]
12:   i ← i + 1
```

a merge action we increase the weight in the corresponding representative-weight pair of type t by the weight w of s''. s'' will replace s' according to the probability shown in Line 9. Chen [1992] proved that this scheme reduces the variance of the estimation scheme. The nodes in A form a *sampled subtree* denoted ST.

Clearly, SS using a perfect type system would produce an exact prediction in a single probe. In the absence of that we treat $\hat{\varphi}(S)$ as a random variable; then, if $E[\hat{\varphi}(S)] = \varphi(S)$, we can approximate $\varphi(S)$ by averaging $\hat{\varphi}^{(p)}$ over multiple sampled probes. And indeed, Chen [1992] proved the following theorem.

Theorem 1. *[Chen, 1992] Given a set of independent samples (probes), $p_1, ...p_m$ from a search tree S, and given a monotonic type system T, the average $\frac{1}{m}\sum_{j=1}^{m} \hat{\varphi}^{(p_j)}(S)$ converges to $\varphi(S)$.*

The stable children property. A hidden assumption made by SS is that it can access the child nodes in the EST of every node in EST. SS assumes that child nodes are pruned only if their f-value is greater than or equal to initial upper bound c^b, which is accurate for algorithms such as IDA* [Lelis *et al.*, 2013]. However, as acknowledged by Knuth [1975], this sampling scheme would not produce accurate predictions of the EST generated by DFBnB because the exact child nodes of nodes generated by DFBnB are not known unless we fully run the algorithm.

Definition 5 (stable children property). *Given an EST $S = (N, E)$ implicitly specified. The stable children property is satisfied iff for every $n \in N$ in S along a path leading from the root, the set $child(n)$ in N can be determined based on only the information along the path.*

Proposition 1. *The stable children property for the EST of DFBnB is guaranteed if and only if the initial upper bound c^b is already optimal.*

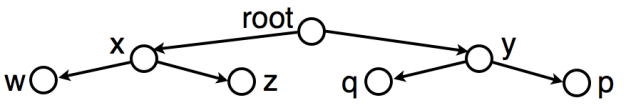

Figure 1: Example of an underlying search tree.

Proof. By counter example. Assume a fixed bound c^b input to SS for estimating the search tree expanded by DFBnB exploring the search space in Figure 1. We assume DFBnB follows a left-first node ordering. Let c_{left} be the cost of the best solution found when exploring the left hand-side of this search tree. When the search algorithm considers child y of the root, then if $f(y) \geq c_{left}$, y will not be expanded, and otherwise it will. In the former, the EST will have only x as a child to the root and in the latter the EST will have both x, y as child nodes to the root; distinguishing between these two cases is not possible without actually executing DFBnB on the left-hand side of the search tree. □

3 Two-Step Stratified Sampling

In order to cope with the lack of the stable children property, we extend SS to approximate the actual set of child nodes in DFBnB's EST using a two-step algorithm. In the **first step** we use SS as in Algorithm 1 to generate a sampled tree ST, assuming a fixed upper-bound c^b which is derived by some preprocessing (*e.g.,* local search); in the **second step** we prune ST by simulating a run of DFBnB restricted to nodes in ST. This yields our two-step Stratified Sampling algorithm (TSS). In contrast to SS, TSS is guaranteed to converge to correct predictions of the size of the EST of DFBnB. For clarity, we describe a naive version of the second step of TSS first, and then present an improved version.

3.1 Naive Second Step

Algorithm 2 describes a naive second step of TSS. It uses an array $SIZE[\cdot]$ indexed by types, where $SIZE[t]$ is the estimated size of the subtree rooted at node s representing type t in ST. Like DFBnB, we explore the UST in a depth-first manner, pruning nodes based on c^b (Line 6) and updating the value of c^b (Line 3). However, in contrast to DFBnB, we restrict the algorithm to only expand node s representing the type of child node s' in ST (Line 7). Hence Algorithm 2 explores only one node of each type in a depth-first fashion while keeping track of the value of the upper bound c^b. If the value of $SIZE[T(s)]$ has not been computed, *i.e.,* $SIZE[T(s)] = 0$, then TSS computes it by calling $SecondStep$ recursively (Line 9).

Limitation of the naive approach. Algorithm 2 corrects (to some extent) for SS's lack of access to the actual child nodes in EST as it searches depth-first and updates the upper bound c^b. However, Algorithm 2 is not sufficiently sensitive to the bound updates when estimating the size of subtrees rooted at nodes of the same type. For example, assume two nodes n_1 and n_2 of type $T(n_1) = T(n_2)$ that root identical subtrees in the UST; given an initial upper bound c^b, DFBnB might expand fewer nodes when exploring n_2's subtree than when exploring n_1's subtree if n_1 is explored first, because it might find a solution that yields a tighter upper bound c'. Thus, when exploring n_2's subtree the value of c' found in n_1's subtree will allow extra pruning.

Algorithm 2 SecondStep(s^*, T, ST, c^b), naive version

Input: root of a tree s^*, type system T, and sampled tree ST, initial upper bound c^b
Output: estimated size of the search tree rooted at s^*

```
1:  Initialize: ∀t ∈ T, SIZE(t) ← 0.
2:  if s* is a solution node with cost better than c^b then
3:      update upper bound c^b
4:  SIZE[T(s*)] ← 1
5:  for each child s' of s* do
6:      if g(s') + h(s') < c^b then
7:          s ← node representing T(s') ∈ ST
8:          if SIZE[T(s)] = 0 then
9:              SecondStep(s, T, ST, c^b)
10:         SIZE[T(s*)] ← SIZE[T(s*)] + SIZE[T(s)]
11: return SIZE[T(s*)]
```

3.2 Using Histograms

In order to handle the above problem and to get a more accurate knowledge we propose to collect more information as follows. We consider the distribution of f-values in the subtree rooted at the nodes representing types in our prediction scheme. The distribution of f-values under a node n is stored as a *histogram*, which will initially be indexed by types.

Definition 6 (Histogram). *Let ST be the sampled tree generated by SS based on type system T and initial upper bound c^b. The histogram of a type t, $hist_t$ is a set of pairs (k, r) for each observed f-value k in the subtree rooted at node $n \in ST$ whose type is t , and r is the estimated number of nodes whose f-value is k in n's subtree.*

Example 1. *Consider the UST shown in Figure 1 and a type system where $T(x) = T(y)$, $f(x) = f(y) = 1$, $f(w) = f(q) = 3$, $f(z) = f(p) = 2$, and an initial upper bound $c^b = 4$. SS produces a sampled tree ST with either the right or the left branch, since $T(x) = T(y)$. Assume SS produces the left branch with nodes x, w, and z. After exploring the subtree rooted at x, we store the histogram: $hist_{T(x)} = \{(1, 1), (2, 1), (3, 1)\}$, as the subtree contains one node whose f-value is 1 (node x), one with f-value of 2 (node z), and one with f-value of 3 (node w). Let us ignore how the histogram is computed for now (this is explained in detail in Algorithm 3 below). We also update the upper bound c^b to 2, which is the value of the solution found in node z.*

When we backtrack to the subtree rooted at y, we use the histogram $hist_{T(x)}$ to estimate the subtree of y because $T(x) = T(y)$. This is done by summing up all the r-values of the entries of $hist_{T(x)}$ whose f-value is less or equal to 2, the current upper bound. Thus, we estimate that the subtree rooted at node y has two nodes (the entry $(3, 1)$ of $hist_{T(x)}$ is pruned), which is exactly the DFBnB's EST size. Using Algorithm 2 would yield $SIZE(T(x))$ of 3, however.

Algorithm 3 SecondStepHist($s^*, T, UnionST, c^b$)

Input: root of a tree s^*, type system T, $UnionST \leftarrow$ union of m sampled trees, initial upper bound c^b
Output: histogram $hist_{s^*}$ and estimate of the EST's size.

```
1: Initialize: ∀s ∈ UnionST, hist_s ← ∅.
2: hist_{s*} ← {(f(s*), 1)}
3: if s* is a solution node with cost better than c^b then
4:    update lower bound c^b
5: for each child s of s* do
6:    if g(s) + h(s) < c^b then
7:       if s is not in UnionST then
8:          s ← random n ∈ UnionST with T(n) = T(s)
9:       if hist_s ≠ ∅ then
10:         histPruned_s ← prune(hist_s, c^b)
11:         hist_{s*} ← hist_{s*} + histPruned_s
12:      else
13:         hist_s ← SecondStepHist(s, T, UnionST, c^b)
14:         hist_{s*} ← hist_{s*} + hist_s
15: return hist_{s*} and the sum of the r-values of hist_{s*}
```

3.3 Union over Multiple Sampled Trees

Before describing the full second step of TSS, we propose another extension of SS that is orthogonal to the use of histograms. Instead of applying the second step to a single subtree generated by SS, we propose to run m independent probes of SS and take the union of the m subtrees into a single tree called $UnionST$. For every path in $UnionST$ there exists a path in at least one of the sampled trees. We are aiming at providing the second step of TSS with a subtree that is closer to the UST. The number of probes of SS is a parameter that controls the first step. In the second step we compute histograms for nodes in $UnionST$. $UnionST$ could have multiple nodes representing the same type. Thus, in contrast with Example 1, in the second step of TSS we are going to index the histograms by nodes in $UnionST$ instead of types and allow multiple histograms for the same type. We now describe the actual second step of TSS.

3.4 Overall TSS

Algorithm 3 describes the second step of TSS. It includes a recursive procedure that computes the histograms for nodes in $UnionST$ that are generated in the first step. The sum of the r-values of the histogram of the start state is the TSS prediction of the size of the EST.

TSS first generates $UnionST$ which is the union of m trees ST generated by SS. It then applies DFBnB restricted to the nodes in $UnionST$ while computing the histograms and doing the appropriate pruning. The histogram of the start state, $hist_{s^*}$, initially contains only one entry with value of one for the f-value of s^* (Line 2) and is computed recursively by computing the histogram of each child s of s^* and combining these histograms' entries into a single histogram (Lines 11 and 14). The combined histogram $hist_3$ of the two histograms $hist_1$ and $hist_2$ has one entry for each k-value found in $hist_1$ and $hist_2$. The r-value of each k in $hist_3$ is the sum of the corresponding r-values of $hist_1$ and $hist_2$.

If the histogram of s was already computed and stored in memory, *i.e.,* $hist_s \neq \emptyset$, then we "prune" $hist_s$ according to the *current* c^b (Line 10), as shown in Example 1. The function $prune(\cdot,\cdot)$ receives as parameters the histogram $hist_s$ and the current upper bound c^b. It returns a histogram, $histPruned_s$, that contains all entries in $hist_s$ for which the k-value is lower than c^b, as illustrated in Example 1.

If TSS generates a node s that is in $UnionST$, then we use the histogram of s itself as an estimate of the size of the subtree rooted at s. If s is not in $UnionST$, we use the histogram of any node representing $T(s)$ in $UnionST$, chosen randomly (line 8).

TSS is efficient because it explores only one node of each type in the search tree sampled by a single probe of SS.

Theorem 2 (Complexity). *The memory complexity of* TSS *is $O(m \times |T|^2)$, where $|T|$ is the size of the type system being employed and also bounds the size of the histograms stored in memory. The time complexity of* TSS *is $O(m \times |T|^2 \times b)$, where b is the branching factor of the UST.*

One could run TSS with $m = 1$ multiple times and average the results, like SS does with its probes. However, this approach is not guaranteed to converge to the correct value. As the number of probes m goes to infinity, TSS would converge properly to the correct value, simply because the subtree accumulated in $UnionST$ approaches a superset of nodes in the actual DFBnB EST in the limit.

Lemma 1. *If $UnionST \supseteq EST$, Algorithm 3 computes the DFBnB EST size exactly.*

Proof. Because $UnionST$ has all nodes in the EST, Algorithm 3 expands the same nodes and in the same order as DFBnB. Thus, Algorithm 3 stores one histogram for each node in the EST and we know exactly the size of the EST. □

Consequently, we have the following.

Theorem 3. *Let S be an EST generated by DFBnB using any heuristic function.* TSS *perfectly predicts the size of S as the number of probes m goes to infinity.*

In practice, however, we are interested in evaluating predictions using a small m for time efficiency reasons.

4 Experimental Results

We evaluate TSS and competing schemes by predicting the EST size of DFBnB when using the mini-bucket heuristic (BBMB) [Kask and Dechter, 2001], on three domains: protein side-chain prediction (pdb), computing haplotypes in genetic analysis (pedigree), and randomly generated grid networks. For each domain we experiment with different mini-bucket i-bounds, yielding different strengths of the heuristic. Every pair of problem instance and i-bound represents a different prediction problem. In total, we have 14, 26, and 54 problems, for pdb, pedigree, and grids, respectively. All experiments are run on 2.6 GHz Intel CPUs with a 10 GB memory limit. We present individual results on selected single problem instances and summary statistics over the entire set.

Weighted Backtrack Estimator. In addition to SS, we also compare TSS to WBE (Weighted Backtrack Estimator) [Kilby

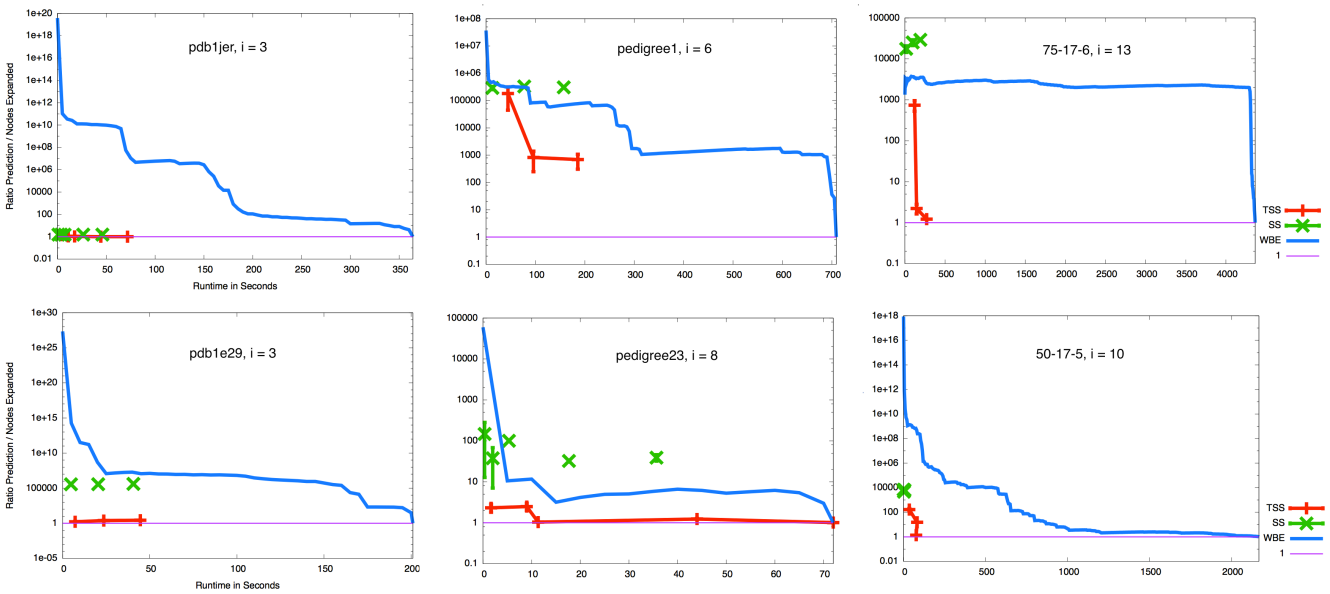

Figure 2: Comparison of TSS, SS, and WBE on six different problem instances. The y-axis shows the ratio between the predicted by the actual number of nodes expanded in log-scale, and the x-axis the runtime in seconds.

et al., 2006]. WBE runs alongside DFBnB search with minimal computational overhead and uses the explored branches to predict unvisited ones and through that the EST's size. WBE produces perfect predictions when the search finishes. We implemented WBE in the context of BBMB. Kilby *et al.* presented another prediction algorithm, the Recursive Estimator (RE), whose performance was similar to WBE's in their experiments. Both WBE and RE were developed to predict the size of binary trees, but in contrast to WBE it is not clear how to generalize RE to non-binary search trees.

Type Systems. A type system can be defined based on any feature of the nodes. In our experiments we use the f-value of the nodes as types, as well as their depth level (cf. Algorithm 1). Namely, nodes n and n' are of the same type if they are at the same depth and have the same f-value.

Secondly, it is worth noting that optimization problems over graphical models present an additional difficulty not commonly considered in the context of SS and other IDA*-related prediction schemes [Lelis *et al.*, 2013]. Namely, the cost function input and the derived heuristic bounds are real-valued. As a result, a type system built on floating point equality might be far too large for producing efficient predictions. We address this issue by multiplying a heuristic value by a constant C and truncating it to the integer portion to compute a node's type; different values of C yield type systems of different sizes. Finally, we use the same strategy to control the size of the histograms, which are also determined by the number of different f-values.

Results on Single Instances. Figure 2 shows the prediction results for TSS, SS, and WBE on six problem instances. These instances are representative in the sense that they highlight different aspects of the prediction methods. The x-axis represents the runtime in seconds required to produce the predictions and the y-axis, in log-scale, the ratio between the predicted and the actual EST size. A perfect prediction has a ratio of 1, indicated in each plot by a horizontal line. For TSS and SS we show the average ratio over five independent runs, *i.e.,* we run Algorithm 3 or Algorithm 1 five independent times and compute the average ratio as $\frac{predicted}{actual}$ if $predicted > actual$ and as $\frac{actual}{predicted}$, otherwise – that way we avoid overestimations and underestimations canceling each other out when averaging. Assuming the error follows a normal distribution, we show the 95% confidence interval with error bars (note that these are hardly noticeable in the plots). For each problem we first run a limited-discrepancy search [Harvey and Ginsberg, 1995] with a maximum discrepancy of 1 to find an initial bound c^b which is provided to both DFBnB and to the prediction algorithms. We use $C = 50$ for pdb and $C = 100$ for pedigree and grid instances. We control the prediction runtime and accuracy for TSS and SS with different number of probes. WBE was set to produce a prediction every 5 seconds.

Results on the Entire Set of Problems. Table 1 presents summary results across the entire set of benchmarks. We display the average ratio of the results of TSS using different number of probes m and different values of C averaged across different problems. Runtime is hard to show directly, so instead we report *the average percentage of the DFBnB search time that the algorithm covered (column %).* For instance, a %-value of 10 means that the prediction was produced in 10% of the full DFBnB runtime. For each prediction produced by TSS in x seconds we report one prediction produced by WBE in y seconds such that y is the lowest value greater than x (thereby giving WBE a slight advantage).

Column n shows TSS's coverage, *i.e.,* the number of problems for which TSS is able to produce predictions. Namely,

pdb (total of 14 problems)																
	TSS ($C = 100$)		WBE			TSS ($C = 50$)		WBE			TSS ($C = 10$)		WBE		
m	ratio	%	ratio	%	n	ratio	%	ratio	%	n	ratio	%	ratio	%	n
1	**66.4**	**3.07**	8.05e+25	3.57	13	**7.34e+03**	**2.6**	2.31e+27	3.4	14	**1.73e+04**	**0.383**	9.02e+30	1.19	14
5	**14.3**	**8.26**	4.37e+20	9.21	12	**779**	**4.9**	8.21e+25	5.6	13	**5.19e+03**	**1.11**	1.55e+30	1.49	14
10	**12.1**	**13.5**	2.98e+20	14.3	10	**78.3**	**7.57**	4.39e+20	7.9	12	**4.34e+03**	**1.87**	1.1e+29	2.69	14
50	**1.1**	**12.2**	8.12e+07	12.5	2	**1.43**	**29.3**	7.59e+13	29.8	4	**174**	**4.96**	9.75e+25	5.37	13
100	-	-	-	-	0	**1.26**	**9.78**	1.34e+14	10.1	3	**3.88**	**8.55**	2.21e+25	9.31	11
pedigree (total of 26 problems)																
	TSS ($C = 1000$)		WBE			TSS ($C = 100$)		WBE			TSS ($C = 10$)		WBE		
m	ratio	%	ratio	%	n	ratio	%	ratio	%	n	ratio	%	ratio	%	n
1	**1.21e+04**	**15.9**	2.88e+04	29.8	12	**4.2e+06**	**2.38**	2.49e+08	14.5	15	**7.45e+06**	**8.47**	1.41e+09	11.4	21
5	**128**	**33.5**	8.58e+03	47.3	10	**1.34e+03**	**18.8**	4.83e+04	28.3	12	**3.17e+06**	**7.09**	7.22e+08	17.3	19
10	**80.4**	**30.4**	9.92e+03	36.7	8	**1.07e+03**	**22.3**	3.17e+04	28.8	12	**3.92e+06**	**5.67**	7.14e+08	14.7	18
50	**1.47**	**58.6**	3.94	81.2	4	**58.8**	**23.6**	3.87e+04	27.1	8	**4.37e+06**	**10.4**	1.3e+08	21.4	15
100	**1.52**	**72.7**	3.46	93.5	3	**72.4**	**42.3**	8.28e+03	54.3	8	6.46e+04	18.9	4.98e+04	32.2	13
grids (total of 54 problems)																
	TSS ($C = 100$)		WBE			TSS ($C = 50$)		WBE			TSS ($C = 10$)		WBE		
m	ratio	%	ratio	%	n	ratio	%	ratio	%	n	ratio	%	ratio	%	n
1	**373**	**20.4**	3.02e+07	34.2	50	**9.95e+03**	**12.7**	4.07e+07	29.6	51	**2.61e+06**	**1.61**	1.41e+10	14.9	53
5	**12.3**	**38.5**	5.53e+06	46.1	45	**196**	**25.4**	1.85e+07	37.8	49	**8.51e+05**	**4.39**	4.81e+08	19.3	53
10	**3.47**	**41.2**	1.23e+07	47.3	40	**24.9**	**31.1**	9.25e+06	40.7	46	**3.02e+05**	**7.89**	1.09e+08	26	53
50	**1.94**	**61.1**	9.75e+04	66.2	20	**3.09**	**44.8**	3.93e+05	49.6	33	**416**	**17.3**	4.49e+07	30.1	50
100	**2.22**	**69.6**	7.21e+04	71.4	12	**1.93**	**57.3**	1.26e+05	63	25	**134**	**23**	4.16e+07	33.4	48

Table 1: Summary of experiments on three domains. "ratio" denotes the average ratios between the predicted and the actual EST size, % is the corresponding average prediction time relative to the runtime of complete DFBnB.

TSS does not produce results either if it takes more time than the actual DFBnB search or if it runs out of memory. Instances for which TSS did not produce results are not included in the averages in Table 1, since neither SS nor WBE produced reasonable results on those instances either.

Finally, we highlight in Table 1 those entries of TSS where TSS produces more accurate predictions in less time than WBE. SS is not included in the table because it produced unreasonable predictions even when granted more time than the actual DFBnB search.

Discussion. The results in Figure 2 suggest that TSS is far superior to both WBE and SS. WBE often produces accurate predictions only when DFBnB is about to finish (DFBnB search time is implied by the plots' rightmost x-value). For instance, on the 75-17-6 grid instance with $i = 13$ WBE is accurate only after more than one hour of computation time, while TSS is able to quickly produce accurate predictions. As anticipated, SS also tends to produce poor predictions. However, if the initial upper bound c^b is in fact the optimal cost, then SS will also produce accurate predictions – in this case the stable children property is satisfied because a full DFBnB run would never update c^b . Although rare, this is depicted in the plot of pdb1jer with $i = 3$. Finally, there are a few instances for which no method was able to produce accurate predictions. An example is pedigree1 when DFBnB is used with $i = 6$.

Overall, we see that TSS is the only robust method which is able to produce reasonable predictions in a timely fashion. In particular, Table 1 shows that TSS can produce predictions orders of magnitude more accurate than WBE on average. WBE yields accurate predictions on the pedigree domain, providing average ratios of 3.94 and 2.79. These are produced, however, when DFBnB is about to finish exploring the whole EST: 81% and 94% of the search is complete, respectively. Table 1 also demonstrates the tradeoff between quality and runtime of TSS: as expected predictions are more accurate with time, namely they get more accurate when m and C grow. A similar expected tradeoff is also observed between accuracy and coverage.

The main difference between WBE and TSS is that the former is restricted to sample the branches of the EST currently explored by DFBnB, while the latter has the freedom to sample different parts of the EST guided by the type system. In fact, WBE and TSS could have almost opposite sampling behaviors if DFBnB is implemented to explore the subtrees rooted at nodes with lower f-value first (which is standard since nodes with lower f-value tend to be more promising). Thus, WBE tends to explore first the subtrees rooted at nodes of same or "similar" type, while TSS explores subtrees rooted at nodes of different types. Our experimental results suggest that TSS's diversity in sampling can be effective in practice.

In summary, our empirical results show that, if memory allows, TSS produces very reasonable predictions, far superior to competing schemes. Yet, TSS is a memory-intensive method. With more time and memory it produces better results, with the memory bound effectively limiting its prediction power. Secondly, the UST sampled by SS in the first step of TSS is sometimes far larger than the actual DFBnB EST. In such a case even the first step could take a prohibitive amount of time. We observed this phenomenon especially when no initial upper bound c^b was provided. Clearly, TSS thus works best in domains with high solution density.

5 Conclusion

We presented *Two-Step Stratified Sampling* or TSS, a prediction algorithm able to produce good estimates of the size of the Expanded Search Tree (EST) of Depth-first Branch and Bound (DFBnB) for optimization problems over graphical models. Building upon Chen's Stratified Sampling (SS) [Chen, 1992], TSS modifies SS so it can handle the lack of the stable children property in the EST of DFBnB.

6 Acknowledgements

We thank the anonymous reviewers for their thoughtful comments and suggestions. This work was supported by LCD at the University of Regina, AICML at the University of Alberta, and by NSF grant IIS-1065618.

References

[Balas and Toth, 1985] E. Balas and P. Toth. Branch and bound methods. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kart, and D. B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, New York, 1985.

[Burns and Ruml, 2012] E. Burns and W. Ruml. Iterative-deepening search with on-line tree size prediction. In *Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION 2012)*, pages 1–15, 2012.

[Chen, 1992] P.-C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM J. Comp.*, 21:295–315, 1992.

[Harvey and Ginsberg, 1995] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 607–613, 1995.

[Kask and Dechter, 2001] K. Kask and R. Dechter. A general scheme for automatic search heuristics from specification dependencies. *Artificial Intelligence*, pages 91–131, 2001.

[Kilby *et al.*, 2006] P. Kilby, J. K. Slaney, S. Thiébaux, and T. Walsh. Estimating search tree size. In *Proceedings of the 21st Conference on Artificial Intelligence (AAAI 2006)*, pages 1014–1019, 2006.

[Knuth, 1975] D. E. Knuth. Estimating the efficiency of backtrack programs. *Math. Comp.*, 29:121–136, 1975.

[Korf *et al.*, 2001] R. E. Korf, M. Reid, and S. Edelkamp. Time complexity of Iterative-Deepening-A*. *Artificial Intelligence*, 129(1–2):199–218, 2001.

[Korf, 1985] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence.*, 27(1):97–109, 1985.

[Lelis *et al.*, 2013] L. H. S. Lelis, S. Zilles, and R. C. Holte. Predicting the size of IDA*'s search tree. *Artificial Intelligence*, 196:53–76, 2013.

[Marinescu and Dechter, 2009] R. Marinescu and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16–17):1457–1491, 2009.

[Nilsson, 1980] N. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1980.

[Otten and Dechter, 2012a] L. Otten and R. Dechter. Anytime AND/OR depth-first search for combinatorial optimization. *AI Communications*, 25(3):211–227, 2012.

[Otten and Dechter, 2012b] L. Otten and R. Dechter. A case study in complexity estimation: Towards parallel branch-and-bound over graphical models. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, pages 665–674, 2012.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

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

[Zahavi *et al.*, 2010] U. Zahavi, A. Felner, N. Burch, and R. C. Holte. Predicting the performance of IDA* using conditional distributions. *Journal of Artificial Intelligence Research*, 37:41–83, 2010.