Memory-Efficient Tree Size Prediction for Depth-First Search in Graphical Models

Levi H. S. Lelis[1], Lars Otten[2], and Rina Dechter[3]

[1] Departamento de Informática, Universidade Federal de Viçosa, Brazil
[2] Google Inc., USA
[3] Department of Computer Science, University of California, Irvine, USA

Abstract. We address the problem of predicting the size of the search tree explored by Depth-First Branch and Bound (DFBnB) while solving optimization problems over graphical models. Building upon methodology introduced by Knuth and his student Chen, this paper presents a memory-efficient scheme called Retentive Stratified Sampling (RSS). Through empirical evaluation on probabilistic graphical models from various problem domains we show impressive prediction power that is far superior to recent competing schemes.

1 Introduction

The most common search scheme for Graphical Models optimization tasks, such as MAP/MPE or Weighted CSP, is Depth-First Branch-and-Bound (DFBnB). Its use for finding both exact and approximate solutions was extensively studied in recent years [1–4]. Our paper addresses the general question of predicting the size of the DFBnB explored search tree, focusing on graphical models optimization tasks.

DFBnB [5] explores the search space in a depth-first manner while keeping track of the current best-known solution cost, denoted $cbound$, which can be initialized with the value of a solution derived by some preprocessing (e.g., local search). DFBnB 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 node n. Since $f(n)$ is an underestimate of the cost of an optimal solution that goes through n, whenever $f(n) \geq cbound$, n is pruned.

Often the user of search algorithms such as DFBnB does not know a priori how long the search will take to finish solving a problem instance: it could take seconds, hours or years. This is due to a series of factors, including the strength of the heuristic guiding the search. Prediction is particularly elusive for graphical models where solvers originate in diverse communities (e.g., CP, UAI, OR) and employ different principles for (a) traversing the search space, (b) for generating the heuristic lower bound function, and (c) for pruning nodes. In addition to estimating the algorithm's running time, estimates of expanded search tree size could be used to decide which heuristic function to use to solve a particular problem instance: should one use the slow but accurate heuristic or the fast but inaccurate one? (e.g., by controlling the i-bound in the case of the mini-bucket heuristics [1]). Or, in the context of parallelizing search, a prediction scheme could facilitate load-balancing by partitioning the problem into subproblems of similar EST sizes [6].

Our approach in this paper builds upon the Stratified Sampling (SS) scheme [7, 8]. Knuth [7] proposed a method for estimating the running time of tree search methods by quickly estimating the size of the *expanded search tree* (EST). Under the reasonable assumption that the time required to expand a node is constant throughout the EST, an estimate of the EST's size provides an estimate of the algorithm's running time.

Prediction schemes were investigated in the past primarily in the context of path-finding problems. Specifically, various methods have been developed for estimating EST size of search algorithms such as IDA* [9]. Examples include Partial Backtracking by Purdom [10] and SS by Chen [8], both based on the seminal work of Knuth [7]; other related methods include [11–14]. All of the above work by sampling a small portion of the EST and extrapolating from it. None of those earlier works addressed graphical models tasks which unlike path-finding problems all their solution nodes appear are at a fixed finite depth (i.e., the number of variables). Also, none of these earlier works considered branch and bound search schemes.

Recently Lelis et al. [15] initiated investigating the usage of SS for estimating the DFBnB EST size and evaluated its effectiveness on graphical models. They observed that methods such as SS make the implicit *stable children* assumption, namely that the set of children of node n in an EST can be determined given only the path from the root of the EST to n. Crucially, however, this property does not hold in the context of DFBnB where pruning depends on the upper bound $cbound$ that is updated dynamically throughout the search. Lelis et al. thus introduced a new SS scheme called Two-Step Stratified Sampling (TSS), described in more detail later, that mitigates this problem [15]. They also provided an empirical evaluation of their approach by looking at a specific DFBnB solver applied to a collection of typical graphical models benchmarks from the probabilistic domain.

Contributions. TSS presented a substantial advance to the DFBnB search space prediction task, but it was also shown to be limited by its memory requirements. As a result, TSS can produce poor estimates or, in some cases, no estimates at all. In this paper we introduce *Retentive Stratified Sampling* (RSS) that addresses differently the stable children property of DFBnB, resulting in a far more memory-efficient scheme. Namely, instead of memorizing every node expanded during sampling, RSS retains only the encountered solution paths. We show that this scheme is asymptotically unbiased.

We test RSS empirically on optimization benchmarks over probabilistic graphical models [16] using DFBnB guided by the mini-bucket heuristic [1, 17] (BBMB), which has been extended into a competition-winning solver [18, 19]. We compare RSS with TSS and WBE [20], over prediction tasks from 3 problem domains in graphical models. Our empirical results show that RSS overcomes the memory limitation of TSS and yields estimates far superior to any of the currently competing methods of its kind.

2 Background

Given a directed and implicitly defined full search tree representing a state-space problem [21], we are interested in estimating the size of the subtree expanded by a search algorithm seeking an optimal solution. We call the former the *underlying search tree* (UST) and the latter the *Expanded Search Tree* (EST). Let $S(s^*) = (N, E)$ be a tree representing such an EST rooted at s^*. For each $n \in N$ *child*$(n) = \{n' | (n, n') \in E\}$ defines the node-child relationship in the EST. The prediction task is to estimate the size of N without fully expanding the EST.

2.1 The Knuth-Chen Method

Knuth [7] presented a method to estimate the size of a tree by repeatedly performing a random walk from the root. Under the assumption that all branches have a structure equal to the path visited by the random walk, one branch is enough to estimate the size of the tree. Knuth observed that his method, while guaranteed to converge to the right value, is not effective when the tree is unbalanced. Chen [8] proposed *Stratified Sampling* (SS), which improves upon Knuth's method with a stratification of the tree through a *type system* to reduce the variance of the sampling process.

Definition 1 (Type System). *Let $S(s^*) = (N, E)$ be a tree rooted at s^*, and T a function from N to a finite set of numerical types $\{t_1, \ldots, t_n\}$. We call T a type system, and it yields a partition of N into $T = \{t_1, \ldots, t_n\}$ where $t_i = \{s \in N | T(s) = t_i\}$. We abuse notation: t_i denotes a type and also the set of nodes in N that map to type t_i.*

A type system can be based on any property of the nodes in the search tree. For example, Zahavi et al. [12] used a type system that accounts for the f-value of the nodes to make predictions of the size of the IDA* EST. That is, nodes n and n' have the same type if they have the same f-value. Still in the context of IDA* predictions, Lelis et al. [22] used variations of Zahavi et al.'s type system in which they also account for the f-value of the nodes in the neighborhood of n when computing n's type. In this paper we use the type system introduced by Lelis et al. [15], which is also based on the f-value; we describe such type system in Section 4.1 below.

Chen's Stratified Sampling (SS) is a general method for approximating any function of the form

$$\varphi(s^*) = \sum_{n \in S(s^*)} z(n) \,,$$

where z is any function assigning a numerical value to a node. $\varphi(s^*)$ represents a numerical property of the search tree rooted at s^*. For instance, if $z(n) = 1$ for all $n \in S(s^*)$, then $\varphi(s^*)$ is the size of the tree.

Instead of traversing the entire tree and summing all z-values, SS assumes subtrees rooted at nodes of the same type will have equal values of φ and so only one node of each type, chosen randomly, is expanded. In practice, a type system is good for a function $\varphi(s^*)$ if nodes having identical type root subtrees with similar values of φ. Clearly, we wish to have a good type system with a small number of types. If we have a type for each node, then the type system will be good in the above sense, yet completely ineffective.

SS estimates $\varphi(s^*)$ as follows. First, it samples the tree rooted at s^* and returns a set A of *representative-weight* pairs, with one such pair for every unique type seen

Algorithm 1 Stratified Sampling, a single probe

```
Input: tree root $s^*$, type system $T$, initial upper bound $cbound$.
Output: a sampled tree $ST$ specified by a set $A$ which is divided into subsets, where $A[i]$ is the
    set of pairs $\langle s, w\rangle$ for the nodes $s \in ST$ expanded at level $i$.
 1: initialize $A[0] \leftarrow \{\langle s^*, 1\rangle\}$
 2: $i \leftarrow 0$
 3: while $i$ is less then search depth do
 4:   for each element $\langle s, w\rangle$ in $A[i]$ do
 5:     for each child $s''$ of $s$ do
 6:       if $h(s'') + g(s'') < cbound$ then
 7:         if $A[i+1]$ contains an element $\langle s', w'\rangle$ with $T(s') = T(s'')$ then
 8:           $w' \leftarrow w' + w$
 9:           with probability $w/w'$, replace $\langle s', w'\rangle$ in $A[i+1]$ by $\langle s'', w'\rangle$
10:         else
11:           insert the new element $\langle s'', w\rangle$ into $A[i+1]$
12:   $i \leftarrow i + 1$
```

during sampling. In the pair $\langle n, w\rangle$ in A for type $t \in T$, n is the unique node of type t that was expanded during search and w is an estimate of the number of nodes of type t in the tree rooted at s^*. $\varphi(s^*)$ is then approximated by $\hat{\varphi}(s^*)$

$$\hat{\varphi}(s^*) = \sum_{\langle n,w\rangle \in A} w \cdot z(n)\,, \tag{1}$$

`SS` (see Algorithm 1) receives as input a start state s^*, a type system T, and an initial upper bound *cbound* which is derived by some preprocessing (e.g., local search). `SS` returns a set A which is indexed by depth in the search tree, where $A[i]$ is the set of representative-weight pairs for the types encountered at depth i.

In `SS` types are required to be partially ordered: a node's type must be strictly greater than the type of its parent. Chen suggests that this can be guaranteed by adding the depth of a node to the type system and then sorting the types lexicographically. In our implementation of `SS`, types at one level are treated separately from types at another level by the division of A into the $A[i]$. If the same type occurs on different levels the occurrences will be treated as though they were different types – the depth of search is implicitly added to the type system.

The algorithm works as follows: $A[0]$ is initialized to contain only the root of the tree to be probed, 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 a child s'' has a type t that is already represented in $A[i+1]$ by node s' with weight w', then a *merge* of s'' and s' is performed: increase weight w' of s' by the weight w of s'''s parent s (since there were w nodes at level i that are assumed to have children of type t at level $i+1$). With a certain probability (line 9) s'' will replace the s'. Chen [8] proved that this stochastic choice of type representatives reduces the variance of the estimation. Once all the states in $A[i]$ are expanded, we move to the next iteration. In Chen's `SS`, this process continues until $A[i]$ is empty. The set of nodes in A represents a subtree of the tree `SS` samples, we call this subtree the *sampled tree*.

A single run of SS is called a *probe*. We denote as $\hat{\varphi}^{(p)}(s^*)$ the estimate produced by SS's p-th probe.

Theorem 1. *[8] Given a set of independent probes $p_1, \cdots, p_m$ produced by* SS *using type system T from tree $S(s^*)$, the average $\frac{1}{m}\sum_{j=1}^{m}\hat{\varphi}^{(p_j)}(s^*)$ converges to $\varphi(s^*)$ as m goes to infinity. Namely,*

$$lim_{m\to\infty}\frac{1}{m}\sum_{j=1}^{m}\hat{\varphi}^{(p_j)}(s^*) = \varphi(s^*)$$

2.2 Two-Step Stratified Sampling (TSS)

Stable Children Property. Lelis et al. [15] observed the implicit assumption in SS that it has access to the generative process of the node-child relationship for every node. In particular, SS prunes child nodes only if their f-value is greater than or equal to the initial upper bound *cbound* (line 6 in Algorithm 1). While such pruning scheme holds for predicting EST size of algorithms such as IDA* [22], it does not hold in the case of DFBnB, where pruning is based on an upper bound that is updated throughout search. As a result, the exact child nodes generated by DFBnB are not available to the sampling algorithm.

Definition 2 (Stable Children Property). *[15] Given an EST $S(s^*) = (N, E)$, the stable children property is satisfied iff for every path π leading from the root s^* to a node n, the set $child(n)$ in EST can be determined based on π alone.*

Lelis et al. [15] overcome the lack of the stable children property in the EST of DFBnB by producing the estimate in two steps. In the first step, their TSS algorithm generates m independent SS probes assuming that the search tree is bounded by the initial upper bound *cbound*. Each SS probe produces a sampled tree, and TSS stores in memory the union of all m sampled trees, denoted $UnionST$. In the second step, TSS emulates DFBnB restricted to the nodes in $UnionST$. $UnionST$ gets larger as we increase the value of m. In particular, as m goes to infinity, $UnionST$ converges to the search tree bounded by *cbound*. In this theoretical scenario, TSS's second step expands exactly the same nodes that DFBnB expands and TSS is able to determine the set $child(n)$ exactly and thus produces perfect estimates.

Although in theory the TSS scheme overcomes the lack of the stable child property, it can have high memory requirement, as it stores every node expanded in the first step of each of the m probes. Therefore, TSS is often limited to only a few probes, frequently producing poor predictions or no predictions at all.

Theorem 2 (TSS's Time and Memory Complexity). *[15] The memory complexity of* TSS *is $O(m \times |T|^2)$, where $|T|$ is the size of the type system being employed and m is the number of* TSS *probes.* TSS *time complexity is $O(m \times |T|^2 \times b)$, where b is the branching factor of $UnionST$.*

2.3 Graphical models

A *graphical model* is given as a set of variables $X = \{X_1, \ldots, X_n\}$, their respective finite domains $D = \{D_1, \ldots, D_n\}$, a set of functions $F = \{f_1, \ldots, f_m\}$, each defined over a subset of X (the function's *scope*), and a combination operator (typically sum, product, or join) over functions. Together with a marginalization operator such as $\min_X$ and $\max_X$ we obtain a *reasoning problem*. For instance, a *weighted constraint satisfaction* problem is typically expressed through a set of cost functions over the variables, with the goal of finding the minimum of the sum over these costs (i.e., we seek $argmin_X \sum_i f_i$) . In the area of probabilistic reasoning, the *most probable explanation* task over a Bayesian network is defined as maximizing the product of the probabilities ($argmin_X \prod_i f_i$). The set of function scopes imply a primal graph, an induced width or tree width that is known to control the complexity of variable-elimination and search algorithms for solving a variety of graphical models tasks [23].

The search tree of a graphical model. The most successful schemes for solving optimization tasks over graphical models is by DFBnB search. In its simplest formulation the nodes in UST are consistent partial assignment of values to the variables along a fixed variable ordering $X_1, \cdots, X_n$. The root is the empty assignment, and a node at depth d is $n = (x_1, \cdots, x_d)$ where x_i is a value from the domain of X_i. Child nodes of n extend it by assigning values to the next variable in the ordering. Solutions correspond to full assignments and all appear at depth n. Leaves of the UST correspond either to partial assignments that cannot be extended consistency or to full assignments representing solutions. DFBnB prunes the search tree in the usual manner, comparing its heuristic evaluation function to the current upper-bound.

A popular heuristic function that guides search schemes for graphical models is the mini-bucket heuristic. It is based on mini-bucket elimination, an approximate variant of variable elimination that computes approximations to reasoning problems over graphical models [1]. A control parameter, denoted as i-bound, allows a trade-off between accuracy of the heuristic and its computational requirements: higher values of i yield a more accurate heuristic but take more time and space to compute.

3 Retentive Stratified Sampling

In this section we present Retentive Stratified Sampling (RSS). The central idea is that it is sufficient to have available the full set of solutions subsumed in the DFBnB EST in order to allow SS to determine exactly the set $child(n)$ in the EST.

DFBnB defines a complete ordering on the nodes in the EST, implied by the order in which the child nodes of each parent node are expanded. This expansion ordering also induces an order on the solution leaf nodes.

Definition 3 (Solution Search Tree). *Given that the DFBnB EST is an ordered search tree, the subtree of EST that is restricted to only solution paths is called* Solution Search Tree *(SST). The leaves in the SST are ordered, from left to right, reflecting their discovery order by DFBnB. For each node, its child nodes are ordered from left to right as well. If we have k solutions we assume they are ordered by $s_1, \cdots, s_k$.*

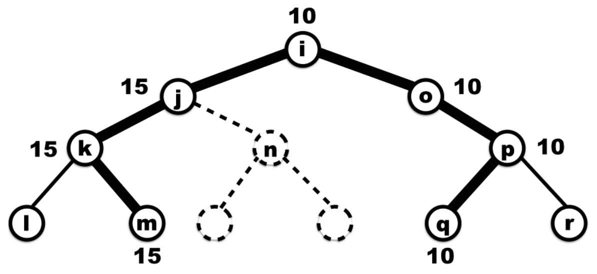

Fig. 1. Example of a UST. The dashed nodes and arcs do not belong to the EST; the arcs in bold represent the SST, with solution nodes m and q, with solution costs of 15 and 10 respectively. The numbers by the nodes a in the SST show the lowest cost solution $l(a)$ going through a.

The assumption that DFBnB has a deterministic ordering of child-node expansion is common since DFBnB usually expands first the subtree rooted at the most promising child (i.e., the child with lowest f-value). By definition the leaves of SST are ordered in decreasing cost from left to right.

Lemma 1. *If the ordered Solution Search Tree SST is available to* `SS`*, then for every node n in the DFBnB EST,* `SS` *can determine the set $child(n)$ in the EST.*

Proof (sketch). We prove the theorem constructively, by providing an algorithm (see Algorithm 3) for the task. Given an SST, we will associate each node m in SST with the lowest cost solution in SST that goes through m, denoted $l(m)$. This is easy to compute by a depth-first search traversal on the SST in time linear in $|SST|$. It is also easy to update the $l(\cdot)$-values whenever new solutions are added to SST: after a new solution is added to the SST, its cost is propagated upwards, namely the minimal costs $l(m)$ for all m along the solution path are updated in the obvious way.

Given a partial path $\pi = n_0, n_1, \ldots, n_d$ in UST, from the root n_0 to a node $n = n_d$, we wish to determine the correct upper bound that would be used by DFBnB. This is done as follows: let node n_j be the closest ancestor of $n = n_d$ on the path π going in reverse order from n_d backwards towards n_0 that (1) appears in SST, and (2) has a child node m in SST which is not on π, such that m immediately precedes n_{j+1} (which is the child node of n_j's on π) according to the child-node ordering. Clearly m can be identified in time linear on the depth of SST. It is easy to see that if m exists, then the lowest cost solution encountered by DFBnB prior to n is $l(m)$, which is thus the upper bound available to DFBnB when it visits n.

Example 1. The tree shown in Figure 1 represents a hypothetical UST where the dashed nodes are pruned by DFBnB, and the arcs in bold represent the SST. We are assuming that DFBnB visits the nodes in lexicographical order. If `RSS` encounters node n during sampling, it identifies $l(k) = 15$ as the relevant upper bound as follows. `RSS` identifies j as the first ancestor of n along the path π going from n towards the root that appears

Algorithm 2 Retentive Stratified Sampling, a single probe

```
Input: tree root $s^*$, type system $T$, solution branches $B$, initial upper bound $cbound$.
Output: a sampled tree $ST$ represented by 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 $i$, and solutions $B$ to be reused in next probe.
 1: initialize $A[0] \leftarrow \{\langle s^*, 1 \rangle\}$
 2: $i \leftarrow 0$
 3: while $i$ is less then search depth do
 4:     for each element $\langle s, w \rangle$ in $A[i]$ do
 5:         if $s$ is a solution node ending a solution path $\pi_s$ and $\pi_s$ is not in $B$ then
 6:             $B \leftarrow Insert(B, \pi_S)$
 7:         for each child $s''$ of $s$ do
 8:             $curr_b \leftarrow VerifyBound(s'', B, cbound)$ // cf. Algorithm 3
 9:             if $h(s'') + g(s'') < curr_b$ then
10:                 if $A[i+1]$ contains an element $\langle s', w' \rangle$ with $T(s') = T(s'')$ then
11:                     $w' \leftarrow w' + w$
12:                     with probability $w/w'$, replace $\langle s', w' \rangle$ in $A[i+1]$ by $\langle s'', w' \rangle$
13:                 else
14:                     insert new element $\langle s'', w \rangle$ in $A[i+1]$
15:     $i \leftarrow i + 1$
```

Algorithm 3 VerifyBound

```
Input: node $s \in UST$ along path $\pi$, ordered tree-structure $B$ whose arcs are labeled $l(n, m)$
    denoting the lowest solution cost below $m$, and initial upper bound $cbound$.
Output: upper bound for $s$ according to $B$
 1: $(n', m') \leftarrow$ identify $n' \in B$ as the closest ancestor to $s$ along $\pi$ that has a child node $m'$ on
    $B$ which immediately precedes $m$", which is $n'$'s child on $\pi$.
 2: If $n'$ exists, then return $l(m')$, return $cbound$ otherwise.
```

in SST, and has a child node k in SST which is not on π, which immediately precedes j's child node on π (which in this case is n itself) according to the child-node ordering. If $f(n) = 16$ (which is greater than $l(k)$), then RSS correctly prunes n. As another example, if RSS encounters node p during sampling, it identifies $l(j) = 15$ as the relevant upper bound as follows. i is the first ancestor of p along the path π going from p towards the root that appears in SST, and has a child node j in SST which is not on π, and which immediately precedes i's child node on π (node o) according to the child-node ordering. In this case, if $f(p) = 10$, then RSS correctly expands p.

Since the SST is generally far smaller than the EST, we are likely to get a memory-efficient algorithm, which is obviously superior to TSS. Algorithm RSS implements the scheme described in Lemma 1 in its pseudo code shown in Algorithms 2 and 3. RSS can be viewed as SS with the following two extensions:

1. Algorithm 2 approximates SST in the initially empty tree structure B, which is updated throughout probes. Specifically, whenever a solution sol is generated, it is inserted into B (respecting the parent-child ordering induced by DFBnB). In doing so, pruning within the B structure can be applied by removing from B any solution that succeeds sol in B and has a higher cost (note that solutions preceding sol in

B will always have higher cost due to the pruning in Algorithm 2, line 9). Thus, at all times we maintain in B an ordered tree where leaves have decreasing cost going from the first to the last solution expanded by DFBnB. The function $Insert(B, \pi_S)$ (line 6 of Algorithm 2) accomplishes this task and can work in time linear in $|B|$ (the function is not formally introduced). The tree B that `RSS` outputs in the i-th probe is used as input for the $(i+1)$-th probe.
2. `RSS` does not insert child s'' into $A[i+1]$ if there is a solution in B that appears before s'', and $h(s'') + g(s'') \geq curr_b$ (lines 8 and 9 of Algorithm 2).

3.1 Asymptotically Perfect Predictions

Since every branch in the EST has a non-zero probability of being sampled, it is quite immediate that:

Lemma 2. *The tree structure B converges to SST as the number of probes goes to infinity.*

From Lemma 2 and from Theorem 1 it follows that `RSS` has the asymptotic guarantee to generate perfect predictions of the size of the DFBnB EST. Formally,

Theorem 3. *Given a set of independent probes $p_1, \cdots, p_m$ produced by* `RSS` *using type system T from a search tree $S(s^*)$ representing a DFBnB EST, there exists $j_0 \leq m$ such that, the average $\frac{1}{m-j_0}\sum_{j \geq j_0}^{m} \hat{\varphi}^{(p_j)}(S)$ converges to $\varphi(s^*)$ as $m \to \infty$.*

Proof. Eventually, after a finite number of probes j_0, B will be equal (or close) to SST (Lemma 2), allowing `RSS` to determine the exact set $child(n)$ for any node n in the DFBnB EST (Lemma 1), which in turn allows us to apply Theorem 1.

In practice j_0 is unknown and we use a heuristic estimate as described below.

3.2 Time and Space Complexity of Retentive Stratified Sampling

In each probe, in addition to the B structure, `RSS` with type system T stores in memory at most $|T|$ nodes. Across multiple probes B converges to SST. Thus,

Theorem 4. `RSS`*'s time complexity after m probes is $O(m \times |T| \times d)$, where $|T|$ is the size of the type system and d is the EST depth (i.e., the number of variables in the graphical model). The space complexity is $O(|T| + |SST|)$.*

Although `RSS`'s time and space complexities depend on parameters for which we might not know the values in advance, they allow us to contrast, for example, `RSS` and `TSS` memory requirements. In particular, we observe that the amount of memory `TSS` requires is bounded by $|EST|$, while the amount of memory `RSS` requires is bounded by $|SST|$. In the worst case $|SST|$ is bounded by the number of solutions of the original problem, in the best case SST is a single branch. In practice $|SST|$ tends to be much smaller than the $|EST|$.

4 Experiments

4.1 Empirical Methodology

We evaluate RSS by predicting EST sizes of DFBnB using the mini-bucket heuristic (BBMB) [1, 17]. For a given problem instance we experiment with different mini-bucket i-bounds for producing different heuristic strengths. RSS is currently not able to account for AND/OR search spaces and caching, techniques used in the more advanced solvers such as AOBB [18, 19].

We consider three problem domains, protein side-chain prediction (pdb), computing haplotypes in genetic analysis (pedigree), and randomly generated grid networks, with 14, 4, and 14 problem instances, respectively. We use the following i-bounds values: 3 for pdb; 6, 7, 8, 10, 11, 12, and 13 for pedigree; and 10, 11, 12, and 13 for grids. Thus, we use 14, 28, and 56 prediction tasks (pairs problem instance and i-bound) for pdb, pedigree, and grids, respectively. We remove from our test set the tasks that DFBnB is able to solve in less than a second, and the tasks that DFBnB is not able to solve after several days of running time. After removing such tasks our test set contains 14, 26, and 54 prediction tasks for pdb, pedigree, and grids, respectively. We remove the easy instances from our test set because they are uninteresting and the instances that DFBnB is not able to solve after several days of running time because we are not able to verify the algorithms' prediction accuracy on those instances. The average running time of DFBnB on the tasks of each domain is 89.3 minutes, 72.2 hours, and 10.9 minutes, for pdb, pedigree and grids, respectively, on a 2.6 GHz CPU (10GB RAM).

We compare the performance of RSS against TSS and WBE (described below). We leave SS out of our experiments because Lelis et al. [15] have already shown that SS is not able to produce good predictions of DFBnB EST size even when granted more computation time than the time required by DFBnB to solve the problem. Since TSS and RSS are stochastic algorithms, we consider the average result over five independent runs for each prediction task. In each case we use the following ratio as a measure of accuracy: $\frac{predicted}{actual}$ if $predicted > actual$ and $\frac{actual}{predicted}$, otherwise — this prevents over- and underestimations canceling out when averaging results. Perfect predictions yield a ratio of 1.0.

Weighted Backtrack Estimator The *Weighted Backtrack Estimator* (WBE) [20] runs alongside DFBnB search with minimal computational overhead. It uses explored branches to predict unvisited ones and thereby the EST size. WBE produces perfect predictions when the search finishes. We implemented WBE in the context of BBMB, yielding an updated prediction every 5 seconds. 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 The use of the f-value to define a node's type has proven effective in other heuristic search tree size estimation problems [22]. In our experiments, in addition to a node's f-value, we also use its depth level (cf. Algorithm 2) for its type, namely

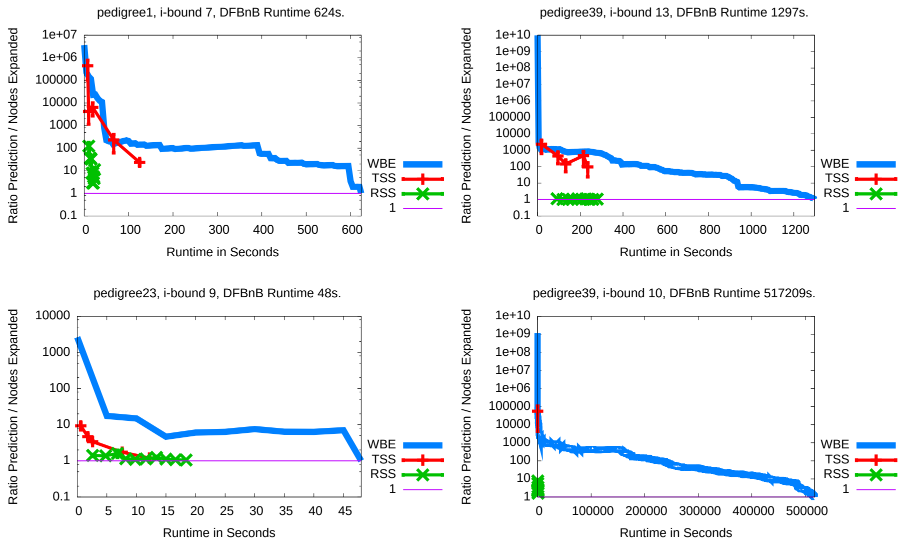

Fig. 2. Prediction accuracy over time of `RSS`, `TSS`, and `WBE` on select representative prediction tasks of the pedigree domain.

nodes n and n' have the same type if they are at the same level of the UST and if $f(n) = f(n')$. We note that the cost function and accordingly, the derived heuristic in graphical model problems are often real-valued and a type system based on floating point equality might be far too large. To mitigate this we apply the technique used by Lelis et al. [15], multiplying $f(n)$ by a constant C and truncating to the integer portion. The constant C allows us to control to some extent the size of the type system. That is, larger values of C result in larger type systems, which implies in slower but possibly more accurate predictions. This is because larger range of types yields a larger coverage of the search space.

Warmstarting RSS Theorem 3 showed that `RSS` is unbiased in the limit, in particular because `RSS` is eventually able to determine exactly the sets $child(n)$ in the EST (i.e., $B = SST$) and the number of probes based on this will eventually outweigh the earlier ones. From Theorem 3 we know that the initial set of probes are skewed and should not be included in the estimate. The rule we used is that upon termination of the probes, we compute the `RSS` estimation by only averaging over probes obtained since the last addition of a new solution to B.

4.2 Select Individual Results

We begin by showing results of `RSS`, `TSS`, and `WBE` on select individual prediction tasks in Figures 2, 3, and 4. These instances are representative in that they highlight different aspects of the prediction methods. For each scheme, we plot in log-scale the

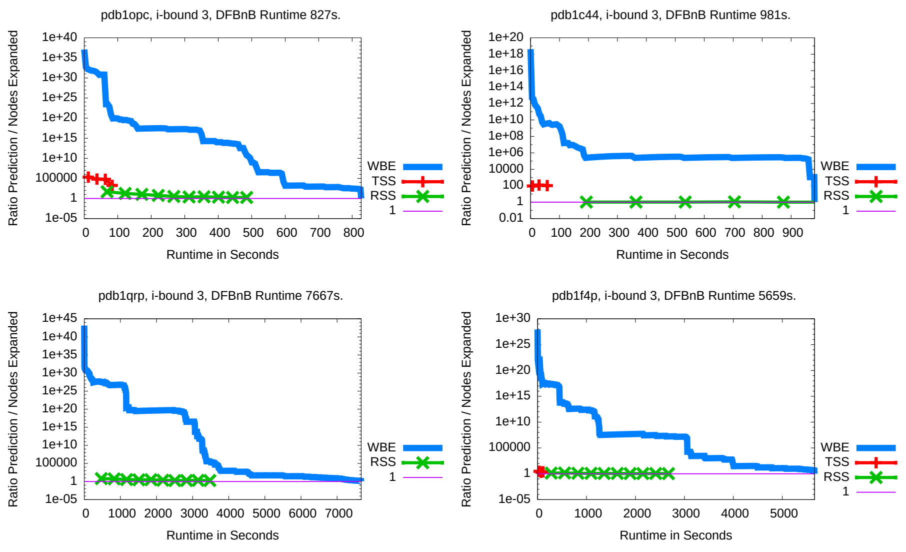

Fig. 3. Prediction accuracy over time of `RSS`, `TSS`, and `WBE` on select representative prediction tasks of the pdb domain.

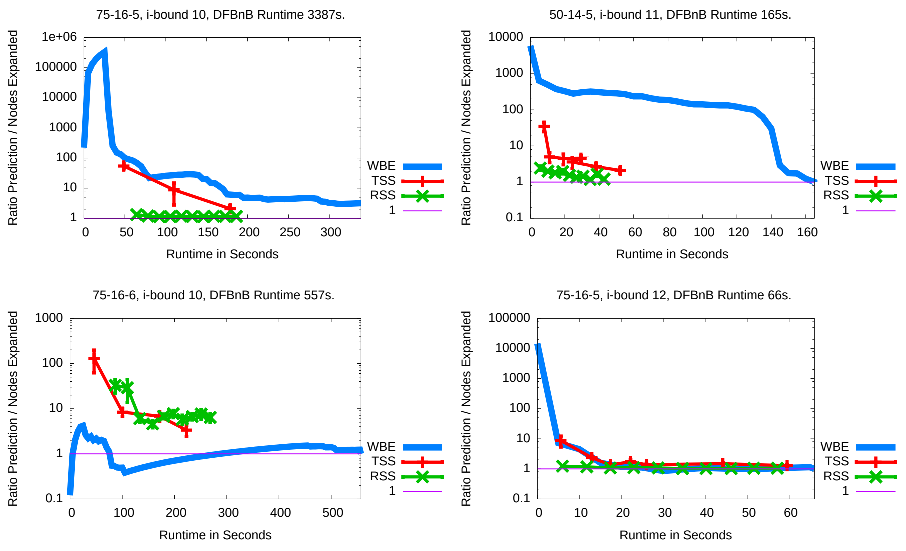

Fig. 4. Prediction accuracy over time of `RSS`, `TSS`, and `WBE` on select representative prediction tasks of the grids domain.

ratio of predicted and actual EST size, as defined above, as a function of running time in seconds. We run RSS with different number of probes and use the warmstarting strategy to generate predictions with different running time. Results for RSS and TSS are averaged over 5 independent runs. There is very little variance in the prediction accuracy over different runs of RSS as the 95% confidence interval is shown but is hardly noticeable. Note that because we vary the number of probes, RSS is oblivious to the DFBnB total running time. That is why in some plots we do not present the RSS results for larger DFBnB running times. The DFBnB total running time is shown on the top of each plot. For each problem we first run a limited-discrepancy search [24] with a maximum discrepancy of 1 to quickly find an initial bound $cbound$ which is provided to both DFBnB and to the prediction algorithms. We use $C = 100$ in this experiment for both RSS and TSS (see Section 4.1 on type systems).

The prediction results in Figures 2, 3, and 4 suggest that RSS is far superior to both TSS and WBE. For instance, RSS quickly produces almost perfect estimates of EST size of pedigree39 with $i = 10$ (Figure 2); TSS is unable to yield good estimates and quickly runs out of memory. WBE is able to produce acceptable predictions only towards the end of DFBnB execution—approximately 6 days on this instance. For pdb1qrp with $i = 3$ (Figure 3) TSS is unable to produce predictions at all—RSS, however, quickly produces near-perfect estimates. Similarly, RSS outperforms the other methods on almost all instances. The pdb1c44 problem instance with i-bound of 3 (Figure 3) shows another situation we would like to highlight. In that instance RSS starts producing estimates after 200 seconds of computation time. This is because in the first 200 seconds RSS is constantly updating B and according to our warmstarting strategy described above RSS does not produce estimates while updating B. 75-16-6 with $i = 10$ (Figure 4) is one of the rare cases in which WBE performs better than both RSS and TSS.

4.3 Comparison with TSS

Table 1 shows a summary of prediction results of TSS and RSS for $C = 10$ and $C = 100$. Here "%" is the average prediction time relative to DFBnB. For instance, a value of 20 means that the prediction was produced in 20% of the DFBnB running time. Given a number of probes for TSS and the resulting %-value, the number of RSS probes is chosen so that it has %-values smaller than TSS, thereby giving the latter a small advantage. In each case we observe that n, the number of instances where TSS does not run out of memory, decreases with the number of probes m. For instance, for pedigrees with $C = 100$ and $m = 50$ probes, TSS is able to produce predictions only for 8 out of 26 prediction tasks (the comparison is performed only on these 8 instances). We also observe in Table 1 the trade-off between prediction accuracy and running time provided by parameter C. RSS using $C = 100$, and thus a larger type system, produces more accurate predictions, but it requires more time to produce such predictions.

Table 1 suggests that RSS produces substantially more accurate predictions than TSS, in less time. In many cases, its average ratio of predicted and actual EST size is orders of magnitude better than TSS. For instance, for grids with $C = 10$ the ratios with $m = 10$ probes for TSS and RSS are over 300,000 and 15.1, respectively, which drops to 416 and 2.17 with $m = 50$. Overall, its accuracy results in Table 1 and its

pedigree ($C = 10$)						pedigree ($C = 100$)					
		TSS		RSS				TSS		RSS	
m	n	ratio	%	ratio	%	m	n	ratio	%	ratio	%
1	21	7.45e+06	8.47	**3.5**	**7.04**	1	15	4.2e+06	2.38	**181**	**1.54**
10	18	3.92e+06	5.67	3.2	6.83	10	12	1.07e+03	22.3	**1.6**	**21.4**
50	15	4.37e+06	10.4	**2.21**	**6.62**	50	8	58.8	23.6	**1.71**	**21.5**
100	13	6.46e+04	18.9	**2.3**	**16.7**	100	8	72.4	42.3	**1.7**	**42**

pdb ($C = 10$)						pdb ($C = 100$)					
		TSS		RSS				TSS		RSS	
m	n	ratio	%	ratio	%	m	n	ratio	%	ratio	%
1	14	1.73e+04	0.383	9.83	2.14	1	13	66.4	3.07	**6.66**	**2.7**
10	14	4.34e+03	1.87	9.83	2.14	10	10	12.1	13.5	**1.2**	**12.3**
50	13	174	4.96	**9.44**	**2.05**	50	2	1.1	12.2	**1.02**	**7.47**
100	11	3.88	8.55	**1.37**	**7.35**	100	0	-	-	-	-

grids ($C = 10$)						grids ($C = 100$)					
		TSS		RSS				TSS		RSS	
m	n	ratio	%	ratio	%	m	n	ratio	%	ratio	%
1	53	2.61e+06	1.61	**1.54e+04**	**1.47**	1	50	373	20.4	**4.18**	**17.5**
10	53	3.02e+05	7.89	**15.1**	**7.49**	10	40	3.47	41.2	**2.61**	**37.3**
50	50	416	17.3	**2.17**	**16.8**	50	20	1.94	61.1	**1.4**	**57.3**
100	48	134	23	**1.87**	**21.8**	100	12	2.22	69.6	**1.24**	**67.3**

Table 1. Prediction results of RSS and TSS for $C = 10$ and $C = 100$. For each number of TSS probes m, n is the number of tasks that TSS is able to produce predictions for without running out of memory and which the algorithms are compared on. Ratio of predicted and actual EST size is computed as above, "%" is the average percentage of the full DFBnB search time. Bold results indicate that a scheme produced more accurate predictions in shorter time.

more modest memory requirement suggest that RSS decisively outperforms and thus supersedes TSS.

4.4 Comparison with WBE

Lastly, we compare the performance of RSS and WBE. Evaluation can occur on the entire set of prediction tasks, since neither of the two schemes had issues running out of memory. The results in Table 2 are again averaged per problem domain, but this time organized by choosing predictions with similar %-value (see table caption for details). Note that the %-values for WBE are by design larger than the ones for RSS, thus giving WBE a slight advantage in terms of computation time.

In this prediction setting, RSS performs substantially better than WBE, in all cases being several orders of magnitude more accurate than WBE while taking the same or less amount of time. Most significantly, in case of pdb tasks RSS average ratio in Table 2 never exceeds 2, while WBE overestimates EST size by 18 or more orders of magnitude. Secondly, RSS is able to provide estimations within a factor of 2 (on average) after only 10-15% of DFBnB search time. To the best of our knowledge, this is the first time a sampling algorithm is able to produce such accurate predictions of DFBnB EST size without having memory issues.

	pedigree (26 tasks)				pdb (14 tasks)				grids (54 tasks)			
	WBE		RSS		WBE		RSS		WBE		RSS	
% range	ratio	%	ratio	%	ratio	%	ratio	%	ratio	%	ratio	%
0 - 5	6.92e+06	1.61	**20**	**1.44**	1.7e+25	3.16	**1.78**	**3.14**	8.09e+07	3.48	**142**	**3.27**
5 - 10	6.34e+03	8.82	**12**	**7.16**	2e+26	7.4	**1.97**	**7.36**	2.95e+05	7.65	**85.1**	**7.25**
10 - 15	7.65e+03	15.2	**2.45**	**12.4**	5.74e+25	12.3	**1.86**	**12.2**	2.08e+05	12.6	**1.68**	**11.9**
15 - 20	545	19	**1.06**	**18**	3.11e+19	17.1	**1.72**	**17.1**	1.17e+07	19.5	**5.55**	**17**
20 - 25	111	43.1	**1.41**	**22.4**	6.91e+18	22.8	**1.46**	**22.7**	450	25.4	**1.83**	**22.3**

Table 2. Prediction results of WBE and RSS ($C = 100$), averaged per problem domain and arranged by %-value: for RSS we average the results obtained within 0-5%, 5-10%, ..., 20-25% of DFBnB runtime. In each case we then pick the next-highest (in terms of %) WBE result to compare. Bold results indicate that a scheme produced more accurate predictions in shorter time.

5 Related and Future Work

Another approach for DFBnB EST size prediction lies in off-line learning of regression models based on features extracted from the problem instance, the search space, and possibly candidate solvers. These techniques have been applied to satisfiability problems [25], combinatorial auctions [26], and graphical models [6]. These methods generally require collecting a large set of solved training instances, which can take a substantial amount of time. By contrast, RSS does not rely on training data; its output, however, could be used as an input feature for a regression-based approach, an interesting direction we hope to investigate in the future.

RSS has two limitations we hope to address in the future. First, RSS is not able to produce estimates of the size of AND/OR search trees [18]. Second, our sampling scheme does not account for DFBnB implementations that use caching to avoid expanding duplicated nodes.

6 Conclusion

We have introduced *Retentive Stratified Sampling* (RSS), a scheme for estimating the size of DFBnB search trees. RSS repeatedly probes the search tree and remembers solution nodes it encounters in the process, which are used to apply pruning in subsequent probes. We have demonstrated the superiority of RSS over competing schemes like TSS and WBE, namely its ability to produce estimates with high accuracy in relatively little time. In addition, unlike other schemes RSS does not suffer from memory issues, further adding to its attractiveness.

Acknowledgements

This work was sponsored by FAPEMIG, in part by NSF grants IIS-1065618 and IIS-1254071, and in part by the United States Air Force under Contract No. FA8750-14-C-0011 under the DARPA PPAML program.

References

1. Kask, K., Dechter, R.: A general scheme for automatic search heuristics from specification dependencies. Artificial Intelligence (2001) 91–131
2. Marinescu, R., Dechter, R.: Memory intensive AND/OR search for combinatorial optimization in graphical models. Artificial Intelligence **173** (2009) 1492–1524
3. Otten, L., Dechter, R.: Anytime AND/OR depth first search for combinatorial optimization. In: Proceedings of the Symposium on Combinatorial Search, AAAI Press (2011) 117–124
4. de Givry, S., Schiex, T., Verfaillie, G.: Exploiting tree decomposition and soft local consistency in Weighted CSP. In: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press (2006) 22–27
5. Balas, E., Toth, P.: Branch and bound methods. In Lawler, E.L., Lenstra, J.K., Kart, A.H.G.R., Shmoys, D.B., eds.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. John Wiley & Sons, New York (1985)
6. Otten, L., Dechter, R.: A case study in complexity estimation: Towards parallel branch-and-bound over graphical models. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence. (2012) 665–674
7. Knuth, D.E.: Estimating the efficiency of backtrack programs. Math. Comp. **29** (1975) 121–136
8. Chen, P.C.: Heuristic sampling: A method for predicting the performance of tree searching programs. SIAM Journal on Computing **21** (1992) 295–315
9. Korf, R.E.: Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence **27** (1985) 97–109
10. Purdom, P.W.: Tree size by partial backtracking. SIAM Journal of Computing **7** (1978) 481–491
11. Korf, R.E., Reid, M., Edelkamp, S.: Time complexity of Iterative-Deepening-A*. Artificial Intelligence **129** (2001) 199–218
12. Zahavi, U., Felner, A., Burch, N., Holte, R.C.: Predicting the performance of IDA* using conditional distributions. Journal of Artificial Intelligence Research **37** (2010) 41–83
13. Burns, E., Ruml, W.: Iterative-deepening search with on-line tree size prediction. In: Proceedings of the International Conference on Learning and Intelligent Optimization. (2012) 1–15
14. Lelis, L.H.S.: Active stratified sampling with clustering-based type systems for predicting the search tree size of problems with real-valued heuristics. In: Proceedings of the Symposium on Combinatorial Search, AAAI Press (2013) 123–131
15. Lelis, L.H.S., Otten, L., Dechter, R.: Predicting the size of depth-first branch and bound search trees. In: International Joint Conference on Artificial Intelligence. (2013) 594–600
16. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann (1988)
17. Dechter, R., Rish, I.: Mini-buckets: a general scheme for bounded inference. Journal of the ACM **50** (2003) 107–153
18. Marinescu, R., Dechter, R.: AND/OR Branch-and-Bound search for combinatorial optimization in graphical models. Artificial Intelligence **173** (2009) 1457–1491
19. Otten, L., Dechter, R.: Anytime AND/OR depth-first search for combinatorial optimization. AI Communications **25** (2012) 211–227
20. Kilby, P., Slaney, J.K., Thiébaux, S., Walsh, T.: Estimating search tree size. In: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press (2006) 1014–1019
21. Nilsson, N.: Principles of Artificial Intelligence. Morgan Kaufmann (1980)
22. Lelis, L.H.S., Zilles, S., Holte, R.C.: Predicting the Size of IDA*'s Search Tree. Artificial Intelligence (2013) 53–76

23. Dechter, R.: Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers (2013)
24. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. In: Proceedings of the International Joint Conference on Artificial Intelligence. (1995) 607–613
25. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based Algorithm Selection for SAT. J. Artif. Intell. Res. (JAIR) **32** (2008) 565–606
26. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Empirical hardness models: Methodology and a case study on combinatorial auctions. Journal of the ACM **56** (2009) 1–52