Active Stratified Sampling with Clustering-Based Type Systems for Predicting the Search Tree Size of Problems with Real-Valued Heuristics

Levi H. S. Lelis
Computing Science Department
University of Alberta
Edmonton, Canada T6G 2E8

Abstract

In this paper we advance the line of research launched by Knuth which was later improved by Chen for predicting the size of the search tree expanded by heuristic search algorithms such as IDA*. Chen's Stratified Sampling (SS) uses a partition of the nodes in the search tree called type system to guide its sampling. Recent work has shown that SS using type systems based on integer-valued heuristic functions can be quite effective. However, type systems based on real-valued heuristic functions are often too large to be practical. We use the k-means clustering algorithm for creating effective type systems for domains with real-valued heuristics. Orthogonal to the type systems, another contribution of this paper is the introduction of an algorithm called Active SS. SS allocates the same number of samples for each type. Active SS is the application of the idea of active sampling to search trees. Active SS allocates more samples to the types with higher uncertainty. Our empirical results show that (i) SS using clustering-based type systems tends to produce better predictions than competing schemes that do not use a type system, and that (ii) Active SS can produce better predictions than the regular version of SS.

Introduction

Often one does not know a priori the time required by heuristic search algorithms to solve a particular state-space problem. The inability to predict the search runtime may prevent the application of such algorithms in practical scenarios.

One approach for predicting the runtime of search algorithms is to estimate the size of the algorithm's Expanded Search Tree (EST) (Knuth 1975). In addition to the merit of estimating the search runtime, fast and accurate predictions of the EST size could also be used to fairly divide the search workload among different processors in a parallel computing setting. Such predictions could also be used to select the most suitable heuristic function to guide the search algorithm: Is it better to use a slow and accurate heuristic, or a fast and inaccurate one?

Related work. Knuth (1975) launched a line of research which was later improved by Chen (1992) for predicting the EST size of search algorithms such as IDA* (Korf 1985). Chen's Stratified Sampling (SS) samples the EST guided by a partitioning of the nodes in the EST called *type system*. Chen assumed that nodes of the same *type* root subtrees of the same size. Thus, SS only expands one node of each type for estimating the EST size. Recent work has shown that SS guided by a type system based on the heuristic function used to guide the search algorithm produces good predictions of the IDA* EST size in domains with integer-valued heuristics (Lelis, Zilles, and Holte 2013). However, in domains with real-valued heuristics, such type systems are often too large to be practical.

Contributions. In this paper we study a method that uses the k-means clustering algorithm (McQueen 1967) for deriving effective *clustering-based* type systems for domains with real-valued heuristics. Orthogonal to the clustering-based type systems, we also introduce *Active* SS, an algorithm that uses the idea of active sampling (Etore and Jourdain 2010) in search trees. With the aim of producing better predictions of the EST size, Active SS allocates more samples to the types with higher variance. In contrast with other active sampling methods, it is not immediately clear that Active SS will perform better than SS. This is because Active SS deals with search trees and it requires a relatively expensive procedure to keep track of the variance of the types.

We run experiments on optimization problems over graphical models such as finding the most likely explanation in Bayesian networks (Pearl 1988). Specifically, we are interested in predicting the EST size of Depth-First Branch and Bound (DFBnB) with mini-bucket heuristic (Kask and Dechter 2001) while proving a given solution to be optimal. Because we provide the optimal solution cost to DFBnB our experimental setup is the same as the one used by others for predicting the EST size of IDA* (Korf, Reid, and Edelkamp 2001). In our empirical results (i) SS using clustering-based type systems tends to produce better predictions than prediction methods that do not use a type system, and (ii) Active SS produces better predictions than regular SS.

Problem Formulation

Given a directed, full search tree representing a state-space problem (Nilsson 1980), we want to estimate the size of the subtree expanded by a search algorithm exploring the search

tree while proving a given solution cost c^b to be optimal. We call the nodes expanded by the search algorithm the *Expanded Search Tree* (EST).

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 (The 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 EST size 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. Namely, if $s \in N$ and $t \in T$ with $s \in t$, we also write $T(s) = t$.*

Chen assumed that nodes of the same type root subtrees of the same size. Thus, SS expands only one node of each type to predict the EST size.

Definition 3 (perfect type system)**.** *A type system T is perfect for a tree S iff 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.*

Chen required the type systems to be monotonic.

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).

Algorithm 1 Stratified Sampling, a single probe

Input: root s^* of a tree, type system T, and upper bound c^b.
Output: array of sets A, where $A[i]$ is the set of pairs $\langle s, w \rangle$ for the nodes s expanded at level i.

```
1:  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
```

In each iteration (lines 4 through 7), 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 7). 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 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* of the EST.

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 $\varphi(S)$.*

Type Systems

A type system could be defined based on any feature of the nodes in the search tree. Lelis, Zilles and Holte (2013) showed that SS produces accurate predictions of the IDA* EST size when using type systems based on the heuristic function $h(\cdot)$ used to guide IDA*. The function $h(n)$ provides an estimate of the cost-to-go of a solution going through node n in the EST. They used a type system defined as $T_h(n) = (h(n))$.[1] According to T_h, if nodes n and m have the same h-value, then they have the same type.

The optimization problems that arise in Probabilistic Graphical Models present a problem that prevents the direct use of heuristic-based type systems in practice. Namely,

[1] A variation of T_h was first introduced by Zahavi et al. (2010).

Algorithm 2 k-means clustering for nodes in the search tree

Input: set of nodes $\mathcal{X}$ and number of clusters $k > 0$.
Output: partitions $\mathcal{X}$ into k disjoint groups $\mathcal{X}_k$.

1: create k different centroids.
2: **while** centroid assignment has not converged **do**
3: assign each node $n \in \mathcal{X}$ to the closest centroid.
4: adjust each centroid to be the mean of the value of their assigned nodes.

both the cost and the heuristic functions are real-valued and a type system based on the comparison of floating-point heuristics might be too large to be practical. Lelis, Otten and Dechter (2013) deal with this issue by multiplying the h-values by a constant C and using the integer part of the resulting number to define the type system. Different values of C result in type systems of different sizes. Although such approach might work well in some cases, it has a high experimental cost as it can be hard to find suitable values of C. Moreover, such approach usually requires different values of C for different domains (Lelis, Otten, and Dechter 2013).

We study a domain-independent method based on the k-means clustering algorithm (McQueen 1967) for creating type systems. In contrast with the constant C approach, *clustering-based* type systems allow one to control exactly the number of nodes SS expands at every level of search, independently of the domain. Although the constant C approach might be useful in some real-world scenarios, in this paper we are interested in an automated solution and we only use clustering-based type systems in our experiments.

Clustering-Based Type Systems

We treat the problem of creating type systems as a clustering problem. A clustering algorithm groups together objects that are more alike according to a measure of similarity. In a *clustering-based* type system, nodes n and m are of the same type if they are in the same cluster. We use the words cluster and type interchangeably hereafter.

Algorithm 2 shows the k-means clustering algorithm for nodes in the search tree. It receives as input a set of nodes $\mathcal{X}$ encountered at a given level of a SS probe and a number of clusters $k > 0$. Algorithm 2 partitions $\mathcal{X}$ into k disjoint clusters $\mathcal{X}_k$. The k-means algorithm requires a measure of similarity to compute the distance between the nodes in $\mathcal{X}$ and the k centroids (line 3). We use the absolute difference between the nodes's f-values as the measure of similarity. The f-value of node n is computed as $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the start state to node n in the search tree. For nodes $m, n, v \in \mathcal{X}$, according to our measure of similarity, m is closer to n than to v if $|f(m) - f(n)| < |f(m) - f(v)|$. By using a clustering-based type system one is able to control the number of node expansions at every level of the search tree. Specifically, SS expands at most k nodes at every level of search.

Our clustering-based type systems are not the first to account for the f-value of the nodes. Lelis, Zilles and Holte (2013) implicitly considered the node's f-value in their type system, although only the h-value appeared in their definition. This is because they ran experiments on domains with unit-edge costs and by considering the level of the search tree as part of the type systems they were implicitly considering the node's f-value.

Sampling with Multiple Representatives

In this paper we use a variation of SS introduced by Chen (1989) that explores multiple nodes of the same type in a single probe. In order to expand multiple nodes of the same type Chen suggested to augment the types in a given type system T with a random number. For instance, in the multiple-representative setting, the T_h type system is augmented with a random integer ∇ drawn from the interval $[1, M]$, for some integer M. SS using T_h in the multiple-representative setting expands at most M nodes of each T_h type. We define $T_r(s) = (T(s), \nabla)$ as the multiple representative version of a base type system T.

Purdom's Algorithm

The idea of sampling multiple representatives was first introduced by Purdom (1978) as an improvement to Knuth's algorithm. Random walks are unlikely to sample deep branches of tall and skinny search trees as a probe is likely to quickly encounter a leaf node. As a result, one has to perform a large number of probes with Knuth's algorithm to produce accurate predictions. Instead of expanding one node at random at every level of search, Purdom's algorithm expands at most $M > 1$ nodes at random at every level of search. Purdom explained that by sampling multiple nodes one would increase the probability of reaching deeper branches of the search tree. Note that Purdom's strategy is different than performing M independent probes with Knuth's algorithm. This is because one could sample the same branch multiple times across different probes. By sampling multiple branches at once, one does not sample the same branch more than once within a probe, which increases the chances of a probe reaching deeper branches of the EST. The same argument applies to SS.

Note that the strategy of expanding multiple representative nodes does not preclude SS nor Purdom's algorithm from performing multiple probes.

Unified Framework

Algorithm 3 describes a single probe of SS using a multiple-representative clustering-based type system. The differences between Algorithm 1 and Algorithm 3 are the following. In the latter we do not insert one node of each type directly into A, but we first insert all nodes into set $\mathcal{X}$ with the weight of their parents (lines 5–8). We then use Algorithm **??** to cluster the nodes in $\mathcal{X}$ into k different types (line 9). In line 10 we select the maximum number of representative nodes m_t of each type t that will be expanded; for now we define $m_t = q/k$ for all t. Here, q is an input parameter representing the total number of extra nodes SS is allowed to expand at every level in addition to one node of each type. The type of a node s is defined by the cluster $\mathcal{X}_t$ s belongs to, and also by the random number between 1 and m_t (line 12).

Algorithm 3 Stratified Sampling with Clustering-Based Types and Multiple Representatives, a single probe

Input: start state s^*, number of types k, number of samples q, cost bound c^b, and a type system T.
Output: array of sets A, where $A[i]$ is the set of pairs $\langle s, w\rangle$ for the nodes s expanded at level i. A sampled subtree A of the EST.

1: $A[0] \leftarrow \{\langle s^*, 1\rangle\}$
2: $\mathcal{X} \leftarrow \emptyset$
3: $i \leftarrow 0$
4: **while** i is less then search depth **do**
5: **for** each element $\langle s, w\rangle$ in $A[i]$ **do**
6: **for** each child s'' of s **do**
7: **if** $h(s'') + g(s'') \leq c^b$ **then**
8: insert $\langle s'', w\rangle$ in $\mathcal{X}$
9: partition nodes in $\mathcal{X}$ into k types $\mathcal{X}_t$ (Algorithm 2)
10: $m_t \leftarrow$ number of representative nodes for each type t
11: **for** each element $\langle s, w\rangle$ in $\mathcal{X}$ **do**
12: $T(s) \leftarrow \big(t, \text{random } (1, m_t)\big)$, for $s \in \mathcal{X}_t$
13: **if** $A[i+1]$ contains an element $\langle s', w'\rangle$ with $T(s') = T(s)$ **then**
14: $w' \leftarrow w' + w$
15: with probability w/w', replace $\langle s', w'\rangle$ in $A[i+1]$ by $\langle s, w'\rangle$
16: **else**
17: insert new element $\langle s, w\rangle$ in $A[i+1]$
18: $i \leftarrow i + 1$
19: clear $\mathcal{X}$

This version of SS generalizes all three algorithms: Knuth's, Purdom's and Chen's. If $k = 1$ and $q = 0$, then Algorithm 3 behaves like Knuth's algorithm as it expands a single node at every level of search. If $k = 1$ and $q > 0$, then Algorithm 3 behaves like Purdom's algorithm as it expands at most $q + 1$ nodes at every level of search. If $k > 1$ and $q > 0$, then Algorithm 3 behaves like SS with multiple representatives employing a clustering-based type system.

Active Stratified Sampling for Search Trees

Active SS is also described by Algorithm 3. The differences between regular SS, which we call Passive SS hereafter, and Active SS are the following. First, instead of sampling q/k extra nodes per type, Active SS uses the Adaptive Allocation of Samples algorithm (AA) (Etore and Jourdain 2010) for allocating the q extra samples among different types, with the aim of reducing the variance of the estimates. Second, Active SS calls a bookkeeping procedure after each probe to collect the statistics required by AA about the types in the EST. We now describe the AA and the bookkeeping procedures.

Preliminaries

Let $\hat{Y}_t^i$ be the estimated size of the subtree rooted at a node of type t at level i of the EST. SS approximates the value of $\hat{Y}_t^i$ for all i and t encountered in the EST as a byproduct of its estimation of the EST's size, $\hat{\varphi}(S)$. The value of $\hat{Y}_t^i$ estimated by SS depends on the representative node n of type t it expands at level i. Thus, we treat $\hat{Y}_t^i$ as a random variable. Clearly, $E[\hat{\varphi}(S)] = E[\hat{Y}_{t_{s^*}}^1]$, where t_{s^*} is the type of the start state s^* at the first level of the EST.

Intuitively, due to the tree structure, the variance of the estimated value of $E[\hat{Y}_{t_{s^*}}^1]$ produced by SS correlates with the variance of the estimated values of $E[\hat{Y}_t^i]$ for all i and t encountered in the EST. Thus, we are interested in reducing the variance of the estimates of $E[\hat{Y}_t^i]$ for all i and t.

Algorithm 4 Adaptive Allocation of Samples

Input: number of samples q, a type system T, and a collection of types $U \subseteq T$ found at level i of the EST
Output: number of samples m_t for each $t \in U$

1: **for** each t in U **do**
2: compute the empirical standard deviation $\hat{\sigma}_t$:

$$\hat{\sigma}_t \leftarrow \sqrt{\frac{1}{B_{t,p}^i}\sum_{a=1}^{B_{t,p}^i}(\hat{Y}_{t,a}^i)^2 - \left(\frac{1}{B_{t,p}^i}\sum_{a=1}^{B_{t,p}^i}\hat{Y}_{t,a}^i\right)^2}$$

3: compute the approximated probability mass pr_t:

$$pr_t \leftarrow \frac{w_t}{\sum_{u\in U} w_u}$$

4: **for** each t in U **do**
5: compute the number of samples m_t:

$$m_t \leftarrow \begin{cases} q/|U| & \text{if } \sum_{u\in U} pr_u\hat{\sigma}_u = 0\,, \\ \frac{pr_t\hat{\sigma}_t}{\sum_{u\in U} pr_u\hat{\sigma}_u}\cdot q & \text{otherwise} \end{cases}$$

Adaptive Allocation of Samples

Given a collection of nodes $\mathcal{X}$ at level i of the EST and a k-disjoint partition of $\mathcal{X}$, we are interested in allocating the q samples among the k types in $\mathcal{X}$ in a way that reduces the variance of the estimates of $E[\hat{Y}_t^i]$.

AA, described in Algorithm 4, allocates the number of samples for each type t proportionally to $\hat{Y}_t^i$'s variance (we describe how to get the samples on which $\hat{Y}_t^i$ is based in our bookkeeping procedure below). Intuitively, types with higher variance should be sampled more often to reduce the variance of the predictions.

AA takes as input a collection of types U and a number of extra samples q to be allocated among each type in U. AA returns the number of samples m_t that will be allocated for each type t in U. First, AA computes the empirical standard deviation $\hat{\sigma}_t$ for each type $t \in U$ (line 2). The value of $\hat{\sigma}_t$ is computed based on different samples of $\hat{Y}_t^i$. Here, $\hat{Y}_{t,a}^i$ is the a-th sample and $B_{t,p}^i$ is the total number of samples of type t at level i observed until probe p.

AA assumes that the probability mass of each type $t \in U$, pr_t, is known. Although we do not know the pr_t-values exactly, we have estimates of their values given by the weight associated with each type (line 3). Recall that the weight as-

Algorithm 5 Active SS Bookkeeping

Input: sampled tree A, upper bound c^b, and type systems T and T_r, where T_r is T augmented with a random integer.

Output: a collection Z_u^i for each $u \in T$ of $\hat{Y}_t^i$-values with $t \in T_r$ encountered in A.

```
1: for i ← tree depth to 1 do
2:   for each node s in A[i] do
3:     Ŷ^i_{T_r(s)} ← 1
4:     for each child s'' of s do
5:       if h(s'') + g(s'') < c^b then
6:         Ŷ^i_{T_r(s)} ← Ŷ^i_{T_r(s)} + Ŷ^{i+1}_{T_r(s'')}
7:     insert Ŷ^i_{T_r(s)} in Z^i_{T(s)}
8:   i ← i + 1
```

sociated with each type t is the estimated number of nodes of type t (see Algorithm 1). Once the values of $\hat{\sigma}_t$ and pr_t are computed, AA determines the number of samples m_t according to the equation shown in line 5.

The m_t-values in Algorithm 4 are not defined as integers. Etore and Jourdain (2010) show a complex scheme to compute integer values of m_t that sum up to q. Grover (2008) conjectured that rounding down the m_t-values and allocating the remaining samples to the options with the largest allocation ratios would perform as well as the original AA. We round up the m_t-values to the lowest integer greater than m_t because our samples are relatively cheap to obtain.

Bookkeeping

We now show how to compute the values of $\hat{Y}_{t,a}^i$ used in Algorithm 4. SS estimates the size of the EST by summing up the w-values in A. We are able to compute the values of $\hat{Y}_t^i$ for all i and t with some bookkeeping. This is done with dynamic programming as shown in Algorithm 5.

Algorithm 5 receives as input the sampled subtree A produced in a single probe of SS, the upper bound c^b and two type systems: T and T_r. Algorithm 5 returns one collection Z_u^i for each $u \in T$ of $\hat{Y}_t^i$-values with $t \in T_r$ encountered in A. We iterate over the different levels of A in reverse order, i.e., we go from the deepest level of the sampled subtree to the root (line 1). Then, we compute the values of $\hat{Y}_t^i$ based on the values of $\hat{Y}_t^{i+1}$ already computed (line 6).

Note that the $\hat{Y}_t^i$-values produced by Algorithm 5 are for types $t \in T_r$, but in Algorithm 4 we are interested in the $\hat{Y}_u^i$-values for types $u \in T$. That is why Z_u^i is indexed by types $u \in T$ and not by types $t \in T_r$.

Overall Active Stratified Sampling

In summary, Active SS is described by Algorithms 3, 4 and 5. Active SS uses Algorithm 4 to define the values of m_t (line 10 of Algorithm 3). Note that like Passive SS, Active SS also uses the value of q/k to set the values of m_t during its first probe. Once the first probe is finished and the first $\hat{Y}_t^i$-values are stored in memory, Active SS calls Algorithm 4 to define the m_t-values. Active SS calls Algorithm 5 after each probe to store the new $\hat{Y}_t^i$-values in memory.

Runtime Behavior of Active SS

Passive SS expands one node of each type at every level of the EST. In addition to that, in our implementation, Active SS iterates once again over the nodes in the sampled tree A in Algorithms 4 and 5. Thus, an Active SS probe takes about twice as long as a Passive SS probe. In the next section we verify empirically that Active SS's computational overhead can result in better predictions.

Experimental Results

In our experiments we predict the size of the search tree expanded by Depth-First Branch and Bound (DFBnB) with mini-bucket heuristic (Kask and Dechter 2001) while proving a given solution to be optimal. Because we provide the optimal solution cost and DFBnB does not detect transpositions, our experimental setup is the same as the one used by others when predicting the size of the IDA* search tree for a given cost bound (Korf, Reid, and Edelkamp 2001; Zahavi et al. 2010; Lelis, Zilles, and Holte 2013). All experiments are run on 2.6 GHz Intel CPUs.

Our experiments are run on three domains: protein side-chain prediction (pdb), randomly generated grid networks (grids), and computing haplotypes in genetic analysis (pedigree). In total, we have 164, 56 and 13 problems, for pdb, grids and pedigree, respectively.

In our experiments we measure the error and the runtime of different prediction schemes. The prediction error is measured with the *relative unsigned error*, which is computed as $\frac{|predicted - actual|}{actual}$. We show the percentage of the relative unsigned error, which is the relative unsigned error multiplied by 100. We repeat each prediction task 10 times and, in our plots, we show the average relative unsigned prediction error on the y-axis and runtime in seconds on the x-axis for different numbers of probes. In addition to the average error, assuming the error follows a normal distribution, we also show the 95% confidence interval with error bars. Note that in some cases the error bars can hardly be noticed.

We compare three algorithms: Active SS, Passive SS, and Purdom's algorithm. In preliminary results Purdom's algorithm produced much better predictions than Knuth's algorithm. Thus, we compare Active and Passive SS using clustering-based type systems to Purdom's algorithm, the best prediction algorithm, to the best of our knowledge, that does not use a type system.

We use the following set of input values: $k = \{2, 3, 4, 5, 15, 25\}$ and $q = \{10, 100, 150\}$. Note that for Purdom's algorithm k always equals to 1. The number of probes p used in our experiments depends on the algorithm as the runtime of each probe differs from algorithm to algorithm. We choose the number of probes so that we can compare the accuracy and the runtime of the different prediction methods. First, we show average results for the different values of q and k (Figure 1). Then, we show results for different values of k and number of probes p, for $q = 100$ (Figure 2). Finally, we show results for different values of q and p, for $k = 25$ (Figure 3). The results for $q = 100$ and $k = 25$ are

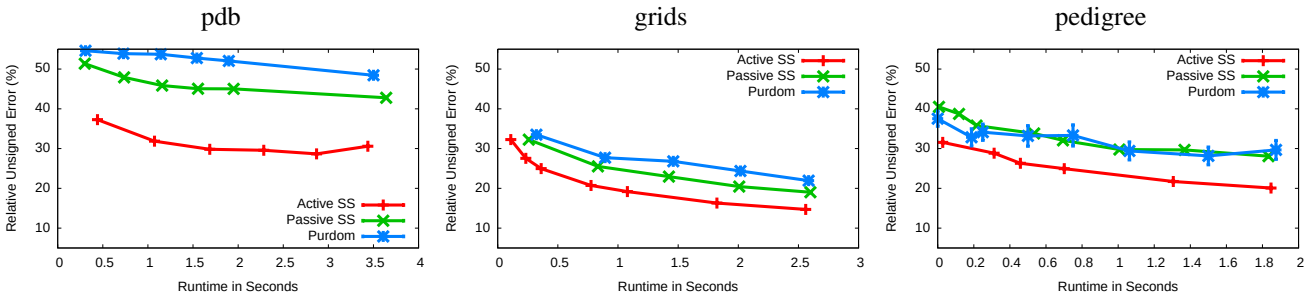

Figure 1: Average results for Active SS, Passive SS and Purdom's algorithm.

representative in the sense that they are qualitatively similar to the prediction results when q and k assume other values.

We are interested in predictions that can be produced much quicker than the time required to run the actual search. Thus, we choose the input parameters p, q and k in a way that the algorithms can quickly produce predictions. The average search times (the search runtime to prove a solution to be optimal) for the pdb, grids and pedigree domains is 11,008, 25,591, and 5,710,850 seconds, respectively. The predictions are produced within 8 seconds—a small fraction of the actual search runtime.

Averaged Results

Figure 1 presents the average results. Each point in the plots in Figure 1 represents the prediction error and runtime averaged over all the different values of q and k for each value of p (number of probes). Active SS performs best on average in all three domains: It produces more accurate predictions in less time than Passive SS and Purdom's algorithm. In the pdb domain the difference between Active SS and other schemes is larger. Passive SS using clustering-based type systems performs better than Purdom's algorithm in two out of three domains. In the pedigree domain Passive SS and Purdom's algorithm produce predictions of similar quality.

Empirical Study of the Parameter q

Figure 2 shows the prediction results for q in $\{10, 100, 150\}$ and $k = 25$. As we increase q, the predictions tend to become more accurate at the cost of increasing their runtime. Active SS outperforms Passive SS and Purdom's algorithm except in the pedigree domain with q-values of 100 and 150, where the predictions produced by Active SS and Purdom's algorithm are of similar quality.

The results in Figure 2 suggest that Passive SS is competitive with Purdom's algorithm for lower values of q, but is outperformed for larger values of q. As an illustrative example, consider the following case. For very large values of q, all three algorithms will expand a number of nodes close to the actual EST size, and Purdom's algorithm should be preferred in such cases as it does not have the computational overhead the SS algorithms have for computing the types. However, in practice, in order to obtain fast predictions, one should use manageable values of q. Our results suggest that Active SS is the algorithm of choice for lower values of q.

Empirical Study of the Parameter k

Figure 3 shows the empirical results for the k in $\{2, 5, 15\}$ and $q = 100$. Since k always equals 1 in Purdom's algorithm, we repeat the curve for Purdom's algorithm in plots of the same domain.

The results in Figure 3 suggest that Active SS is more robust than Passive SS to variations in the value of k. Although Active SS presents some improvement when increasing the value of k, the gain is not substantial (e.g., in the pdb domain the prediction error goes from 27% to 22% when increasing k from 2 to 15 in predictions produced in approximately 2 seconds). Note that for larger values of k the runtime of the k-means algorithm used for defining the type system increases. We conjecture that, despite the increase in runtime for computing the types, Active SS is able to use the extra knowledge provided by the larger number of clusters to allocate more samples in important parts of the search tree, i.e., the parts in which uncertainty is higher. The same phenomenon is not observed with Passive SS. For instance, in the case of the grids domain, Passive SS tends to produce worse predictions as we increase the value of k.

Discussion

Although Purdom's algorithm tended to perform worse than the SS algorithms, it produced reasonable predictions and one could consider using it in practice due to its simplicity.

The clustering-based type system is an alternative to the approach in which one uses the integer part of the resulting multiplication of the heuristic values by a constant C as the type system. The clustering-based type systems are domain-independent in the sense that one knows exactly the number of nodes expanded at every level of search, independently of the problem. This is not true for the C multiplier approach as one does not know a priori the number of nodes expanded at every level of search for a given value of C. The C-multiplier approach might have a high experimental cost as it can be hard to find a suitable value of C.

We compared SS using clustering-based type systems to Purdom's algorithm, the best method, to the best of our knowledge, that does not use a type system. Our results showed that Passive SS using clustering-based type systems produced better predictions than Purdom's algorithm in two out of three domains in the average case (see Figure 1).

We applied the idea of active sampling (Etore and Jour-

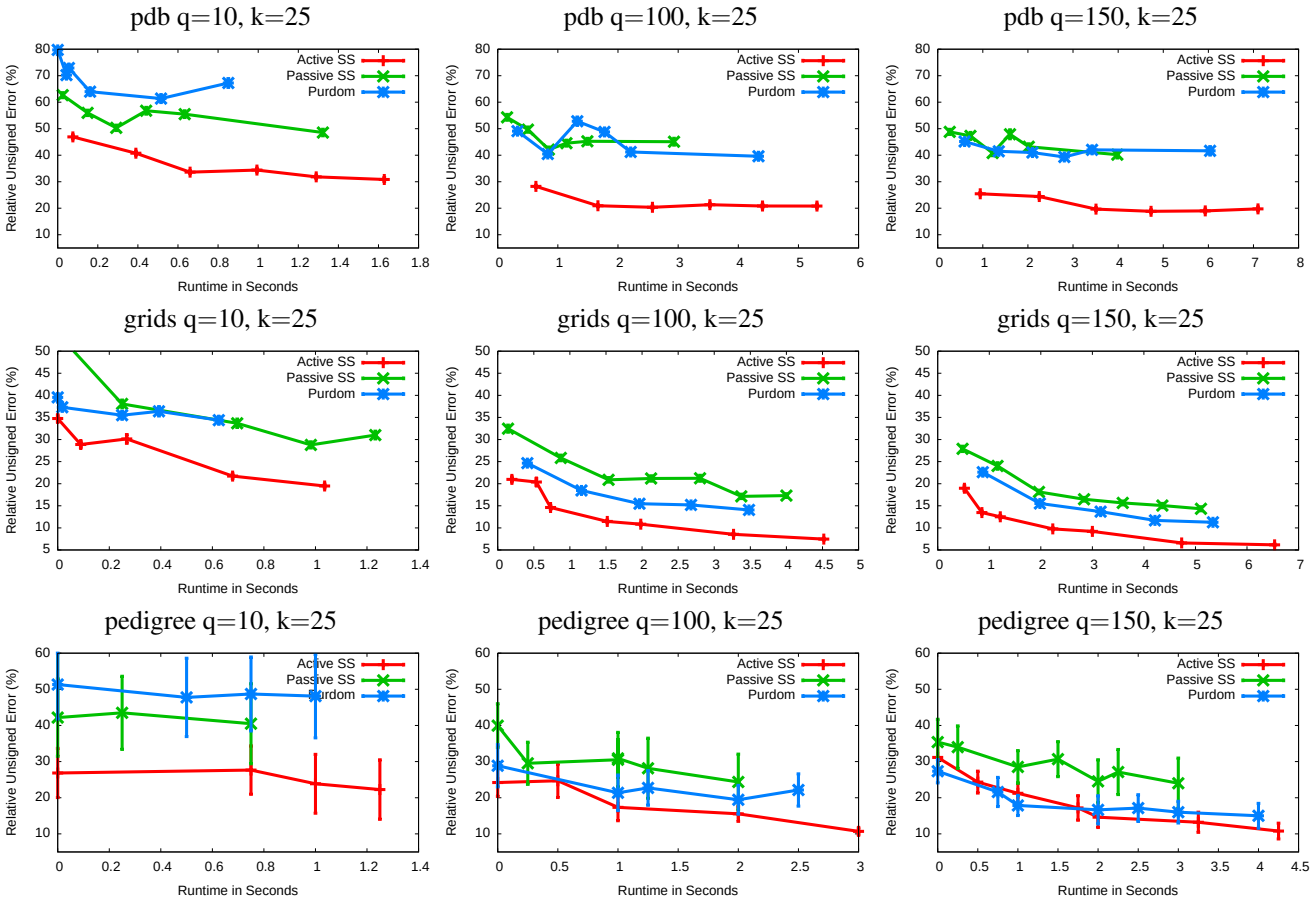

Figure 2: Active SS, Passive SS and Purdom's algorithm for different values of q.

dain 2010) in the context of search trees. The bookkeeping required by the AA algorithm is not expensive for the general case of stratified sampling because one has immediate access to the value of a sample (Antos, Grover, and Szepesvári 2008). By contrast, the bookkeeping required by AA in the context of search trees is relatively expensive as one has to re-expand the nodes expanded by SS to compute the value of the samples. Thus, one probe of Active SS is about twice as slow as a probe of Passive SS. Our experimental results showed that although the bookkeeping of Active SS is somewhat expensive, because it carefully chooses the parts of the search tree that should be sampled more often, it produced better predictions than Passive SS.

Passive SS is expected to produce better predictions than Active SS when "very good" type systems are available. As an illustrative example, Passive SS certainly performs better than Active SS when the type system employed is perfect, i.e., nodes of the same type root subtrees of the same size. Clearly, with a perfect type system a single probe of Active or Passive SS suffices to produce a perfect prediction. Therefore, one will be better off with the fastest algorithm. As a practical example, Passive SS is able to produce very accurate predictions on puzzle domains with integer-valued heuristic (Lelis, Zilles, and Holte 2013). We do not expect Active SS to present gains over Passive SS in those domains.

Active SS is general and could be applied to other problems. For instance, Bresina, Drummond and Swanson (1995) used Knuth's algorithm to measure the expected solution quality of scheduling problems. One could also use Active SS to reduce the variance of Bresina et al.'s algorithm. Assuming a type system is available, instead of measuring the variance of the size of the subtree rooted at nodes of the same type, one would measure the variance of the quality of the solution found in the subtrees rooted at nodes of the same type.

Related Work

Independently of Knuth's line of research, Korf, Reid and Edelkamp (2001) developed a method for predicting the size of the IDA* search tree for a given cost bound. Their method works for the special case of consistent heuristics. Later, Zahavi et al. (2010) presented CDP, a prediction method based on Korf et al.'s ideas that works with both consistent and inconsistent heuristics. Burns and Ruml (2012) presented IM, a prediction method that generalizes CDP for domains with real-valued edge costs. Burns and Ruml's goal was to avoid the poor performance of IDA* in domains with real-valued edge costs by setting a cost bound d that would expand an

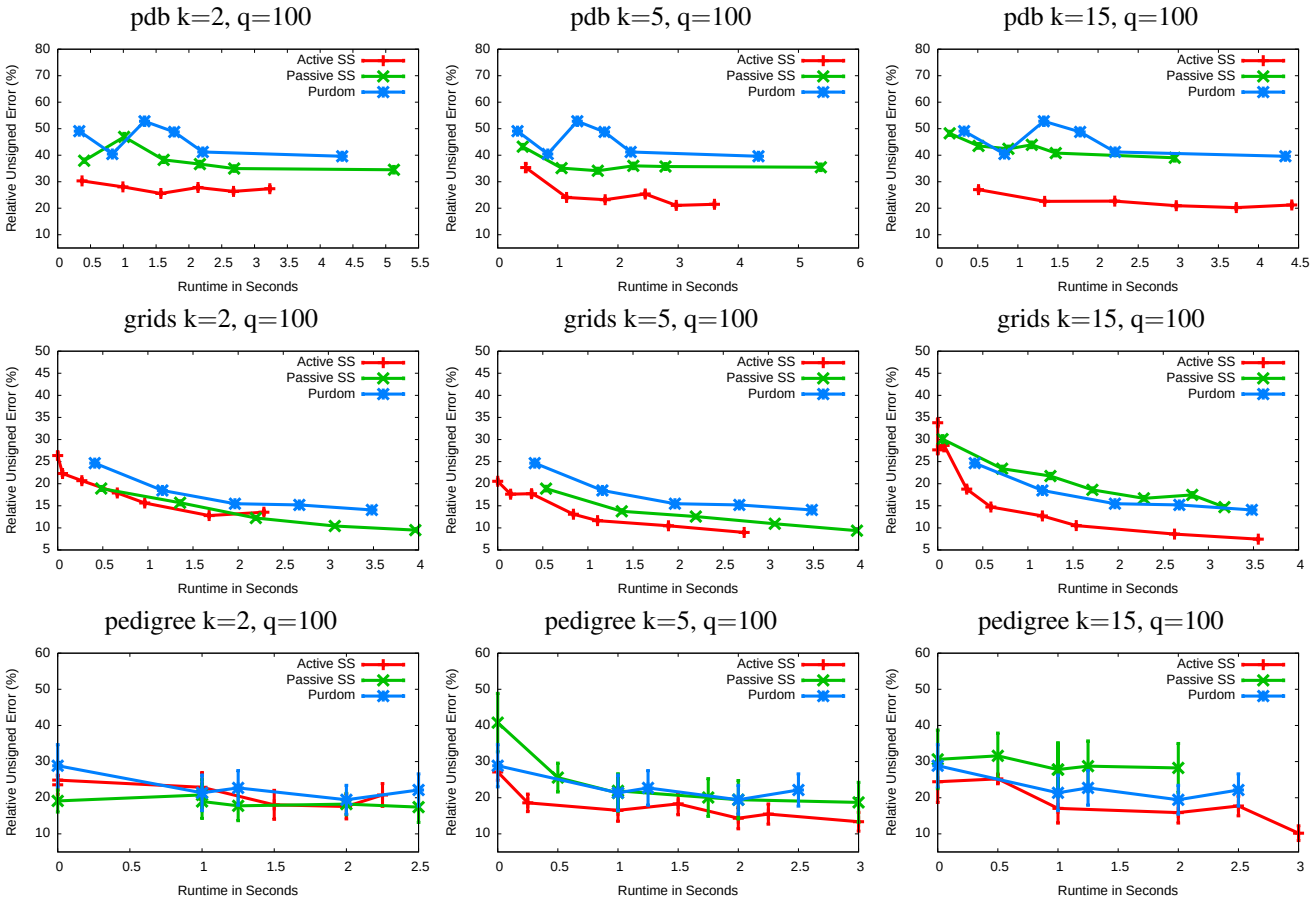

Figure 3: Active SS, Passive SS and Purdom's algorithm for different values of k.

exponentially larger number of nodes in each iteration.

Lelis, Zilles and Holte (2013) connected and compared the methods developed in Knuth's and Korf et al.'s lines of research: SS and CDP. They concluded that CDP is the algorithm of choice if preprocessing is allowed and one is interested in very fast predictions. This is because CDP samples the state-space as a preprocessing step and stores the prediction results in a lookup table. SS, on the other hand, is the algorithm of choice if one is interested in accurate predictions, which is the case we studied in this paper.

A different prediction approach is taken by Kilby et al. (2006) and Thayer, Stern and Lelis (2012). In contrast with SS that samples the EST independently of the search algorithm, their methods use the information observed by the search algorithm to infer the number of nodes yet to be expanded. The advantage of such approach is that the prediction method has access to the bound updates in branch and bound methods (Kilby et al. 2006) and to the transposition detection in best-first search algorithms (Thayer, Stern, and Lelis 2012). The disadvantage of using the information observed by the search algorithm is that the prediction method has access to a biased sample of the EST, which could lead to poor estimates. In fact, Lelis, Otten and Dechter (2013) presented TSS, a variation of SS equipped with a method for approximating the bound updates of branch and bound methods. They showed that TSS is able to produce better predictions than Kilby et al.'s algorithm of the EST size of the Depth-First Branch and Bound. They conjectured that TSS produces better predictions because it does not "follow" the search algorithm and is able to collect a better sample of the EST. Applying our type systems and active sampling to TSS are interesting directions of future work.

Conclusions

In this paper we used the k-means clustering algorithm for creating type systems for domains with real-valued heuristic functions. Our empirical results showed that SS using a clustering-based type system tends to produce better predictions than Purdom's algorithm, which is, to the best of our knowledge, the best prediction method that does not use a type system.

We also presented Active SS, a prediction algorithm that uses the ideas of active sampling in the context of search trees. In contrast with other active sampling approaches, it is not immediately clear that Active SS can perform better than Passive SS due to its relatively expensive bookkeeping procedure. Our empirical results showed that Active SS can perform better than Passive SS.

Acknowledgements

The author would like to thank Rick Valenzano, Rob Holte and Sandra Zilles for helpful discussions on an earlier draft of this paper, and also Lars Otten and Rina Dechter for making their DFBnB code available. This work was supported by the Laboratory for Computational Discovery at the University of Regina, AICML, and NSERC.

References

Antos, A.; Grover, V.; and Szepesvári, C. 2008. Active learning in multi-armed bandits. In *Algorithmic Learning Theory, 19th International Conference, ALT 2008, Budapest, Hungary, October 2008, Proceedings*, volume 5254 of *Lecture Notes in Artificial Intelligence*, 287–302. Berlin: Springer.

Bresina, J. L.; Drummond, M.; and Swanson, K. 1995. Expected solution quality. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*, 1583–1591.

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

Chen, P.-C. 1989. *Heuristic Sampling on Backtrack Trees*. Ph.D. Dissertation, Stanford University.

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

Etore, P., and Jourdain, B. 2010. Adaptive optimal allocation in stratified sampling methods. *Methodology and Computing in Applied Probability* 12(3):335–360.

Grover, V. 2008. Active learning and its application to heteroscedastic problems. Master's thesis, University of Alberta.

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

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

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

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

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

Lelis, L. H. S.; Otten, L.; and Dechter, R. 2013. Predicting the size of depth-first branch and bound search trees. In *International Joint Conference on Artificial Intelligence*, to appear. AAAI Press.

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

McQueen, J. B. 1967. Some methods of classification and analysis of multivariate observations. In Cam, L. M. L., and Neyman, J., eds., *Proceedings of Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 281–297.

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

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

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

Thayer, J. T.; Stern, R.; and Lelis, L. H. S. 2012. Are we there yet? - estimating search progress. In Borrajo, D.; Felner, A.; Korf, R. E.; Likhachev, M.; López, C. L.; Ruml, W.; and Sturtevant, N. R., eds., *SOCS*. AAAI Press.

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