Predicting Optimal Solution Cost with Bidirectional Stratified Sampling (Abstract)

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

Roni Stern, Ariel Felner
Information Systems Engineering
Ben Gurion University
Beer-Sheva, Israel 85104
(roni.stern@gmail.com)
(felner@bgu.ac.il)

Sandra Zilles
Computer Science Dept.
University of Regina
Regina, Canada S4S 0A2
(zilles@cs.uregina.ca)

Robert C. Holte
Computing Science Dept.
University of Alberta
Edmonton, Canada T6G 2E8
(holte@cs.ualberta.ca)

Abstract

Optimal planning and heuristic search systems solve state-space search problems by finding a least-cost path from start to goal. As a byproduct of having an optimal path they also determine the optimal solution cost. In this paper we focus on the problem of determining the optimal solution cost for a state-space search problem directly, i.e., without actually finding a solution path of that cost. We present an efficient algorithm, BiSS, based on ideas of bidirectional search and stratified sampling that produces accurate estimates of the optimal solution cost. Our method is guaranteed to return the optimal solution cost in the limit as the sample size goes to infinity.

Introduction

Given an admissible heuristic function, search algorithms find the least-cost path from the start state to a goal state. A byproduct of finding such a path is that the optimal solution cost becomes known. Lelis et al. (2011) pointed out that there are situations in which one is interested only in the optimal solution cost, the solution path is not required. For instance, for the purpose of bidding on a project, a construction company does not need to know the entire plan for the project, but only an accurate estimate of the project's cost. This allows the time and expenses required to calculate a complete plan to be saved in the event that the company's bid is not successful. Lelis et al. (2011) developed SCP, the current state-of-the-art solution cost predictor.

Heuristic functions themselves provide estimates of the solution cost of a given problem instance. However, they are built to guide search algorithms and are applied to a large number of nodes during the course of solving an instance. It is absolutely essential, therefore, that a heuristic function be very quick to calculate. Often heuristics must trade accuracy for speed of computation so that they are useful in practice. By contrast, a solution cost predictor is applied only once for a given problem instance and it is expected to be as accurate as possible. Obviously, it should be faster than a search algorithm that finds a least-cost path. However, it is not required to be nearly as fast as a heuristic function.

In this paper we present a new algorithm for predicting the optimal solution cost of a problem instance. We call our method BiSS, for Bidirectional Stratified Sampling. BiSS has two advantages over SCP: (1) BiSS entirely avoids the time-consuming preprocessing required by SCP; and (2) unlike SCP, BiSS is guaranteed to return the optimal solution cost in the limit as its sample size goes to infinity.

BiSS is based on a method by Chen (1992), Stratified Sampling (SS). SS uses a partition of the state space, named *type system*, to efficiently estimate the size of search trees. Chen assumed that nodes of the same type at a level of the search tree would root subtrees of the same size. Thus, by sampling only one node of each type SS efficiently estimates the size of a search tree.

SS can be used to approximate any function of the form

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

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

Given a node s^* and a type system T, 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 during sampling. In the pair $\langle s, w\rangle$ in A for type $t \in T$, s 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 search tree rooted at s^*. $\varphi(s^*)$ is then approximated by $\hat{\varphi}(s^*, T)$, defined as

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

Algorithm 1 shows SS in detail. The set A is divided into subsets, one for every layer in the search tree; hence $A[i]$ is the set of representative-weight pairs for the types encountered at level i. Representative nodes from $A[i]$ are expanded to get representative nodes for $A[i+1]$ as follows. All nodes

Algorithm 1 Stratified Sampling

1: *input:* root s^* of a tree and a type system T
2: *output:* an array of sets A, where $A[i]$ is the set of pairs $\langle s, w\rangle$ at level i.
3: $A[0] \leftarrow \{\langle s^*, 1\rangle\}; i \leftarrow 0$
4: **while stopping condition** is false **do**
5: **for** each element $\langle s, w\rangle$ in $A[i]$ **do**
6: **for** each child $\hat{s}$ of s **do**
7: **if** $A[i+1]$ contains an element $\langle s', w'\rangle$ with $T(s') = T(\hat{s})$ **then**
8: $w' \leftarrow w' + w$
9: with probability w/w', replace $\langle s', w'\rangle$ in $A[i+1]$ by $\langle \hat{s}, w'\rangle$
10: **else**
11: insert new element $\langle \hat{s}, w\rangle$ in $A[i+1]$.
12: **end if**
13: **end for**
14: **end for**
15: $i \leftarrow i + 1$
16: **end while**

in $A[i]$ are expanded and the children of each node in $A[i]$ are considered for inclusion in $A[i+1]$. If a child $\hat{s}$ has a type t that is already represented in $A[i+1]$ by another node s', then a merge action on $\hat{s}$ and s' is performed. In a merge action we increase the weight in the corresponding representative-weight pair of type t by the weight $w(s)$ of $\hat{s}$'s parent s. $\hat{s}$ will replace the s' according to the probability shown in line 9.

BiSS is a bidirectional variant of SS for predicting optimal solution costs. It interleaves the execution of two copies of SS, one proceeding forwards from the start state, the other proceeding backwards from the goal state. BiSS switches between the two searches when one iteration of the loop in lines 4 to 15 of Algorithm 1 has finished. One "step" in a particular direction thus corresponds to the expansion of all the representative nodes at a given level. When referring to the array A in the SS algorithm, we will use A^F for the forward search and A^B for the backward search.

BiSS's stopping condition is based on the set of types that have occurred at each level of the searches. We define $\mathcal{T}^F[n] = \{T(s) | \langle s, w\rangle \in A^F[n]\}$, the set of types of the nodes expanded at level n by the copy of the SS algorithm searching forward from the start state, and $\mathcal{T}^B[m] = \{T(s) | \langle s, w\rangle \in A^B[m]\}$. BiSS stops the bidirectional sampling with an estimate of the optimal solution cost when the type sets at the frontiers of the two searches overlap for several consecutive levels. We called this stopping condition a "match" between the two searches, defined as follows.

Definition 1. ***(match)*** *– For any n and m we say that $\mathcal{T}^F[n]$ and $\mathcal{T}^B[m]$ match if $\mathcal{T}^F[n+r] \cap \mathcal{T}^B[m-r] \neq \emptyset$ for all $r \in \{0, 1, \cdots, K\}$ where $K = \max\{\lfloor \gamma \cdot m \rfloor, 1\}$. Here $\gamma \in [0, 1]$ is an input parameter.*

After each step in each direction BiSS tests if the same type occurs in both $\mathcal{T}^F[n]$ and $\mathcal{T}^B[m]$, where n and m are the most recently generated levels in the respective search directions. If this happens, BiSS extends the forward search up to level $n+K$ so that a match, as defined in Definition 1, can be fully tested. If a match occurs at step n from the start state and at step m from the goal state, then the searches terminate and $n+m$ is returned as an estimate of the optimal solution cost.

Experimental Results

Our experiments are run in three domains: the Blocks World, the Sliding-Tile puzzle, and the Pancake puzzle. However, due to space constraints we show results only on the (5x5) 24 sliding-tile puzzle (24-puzzle). For more empirical results see the extended version of this paper (Lelis et al. 2012).

Solution cost predictions are compared using *relative absolute error* (Lelis, Stern, and Jabbari Arfaee 2011) for a set of optimal solution costs. For all start states with optimal solution cost X one computes the absolute difference of the predicted solution cost and X, adds these up, divides by the number of start states with optimal solution cost X and then divides by X. A system that makes perfect predictions will have a relative absolute error of 0.00.

Cost	Relative Absolute Error	
	BiSS	h
96	**0.05**	0.25
98	**0.03**	0.26
100	**0.03**	0.26
102	**0.04**	0.25
104	**0.03**	0.25

Table 1: 24-puzzle

Table 1 shows the relative absolute error of the predictions of BiSS and Manhattan Distance (the heuristic we use to build BiSS's type system) on the 24-puzzle for start states with different solution costs. As can be observed, BiSS makes very accurate predictions — the most inaccurate predictions shown in Table 1 are for start states with optimal solution cost of 96, and even then BiSS makes predictions that are only 5% longer (or shorter) than the optimal solution cost on average. Equally accurate predictions were made in the other two domains we tested.

Conclusion

We presented BiSS, an efficient algorithm that accurately predicts the optimal solution cost without finding a least-cost path from start to goal. BiSS does not require preprocessing and is guaranteed to return the optimal solution cost in the limit as the number of its probes goes to infinity.

Acknowledgements

This work was supported by the LCD at the University of Regina, AITF, AICML, and NSERC.

References

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

Lelis, L.; Stern, R.; Felner, A.; Zilles, S.; and Holte, R. C. 2012. Predicting optimal solution cost with bidirectional stratified sampling. In *ICAPS*.

Lelis, L.; Stern, R.; and Jabbari Arfaee, S. 2011. Predicting optimal solution costs with conditional probabilities. In *SoCS*, 100–107.