Are We There Yet? – Estimating Search Progress

Jordan T. Thayer
University of New Hampshire
Department of Computer Science
jtd7@cs.unh.edu

Roni Stern
Ben-Gurion University of the Negev
ISE Department
roni.stern@gmail.com

Levi H. S. Lelis
University of Alberta
Department of Computing Science
santanad@cs.ualberta.ca

Abstract

Heuristic search is a general problem solving technique. While most evaluations of heuristic search focus on the speed of search, there are relatively few techniques for predicting when search will end. This paper provides a study of progress estimating techniques for optimal, suboptimal, and bounded suboptimal heuristic search algorithms. We examine two previously proposed techniques, search velocity and search vacillation, as well as two new approaches, path-based estimation and distribution-based estimation. We find that both new approaches are better at estimating the remaining amount of search effort than previous work in all three varieties of search, occasionally erring by less than 5%.

Introduction

Many problems can be modeled as search problem where one is required to find a path in a state space. There are many varieties of search problems and corresponding search algorithms that are designed to handle them. For example, search algorithms such as A* (Hart, Nilsson, and Raphael 1968) are used when we require solutions with the least possible cost, while algorithms like greedy best-first search are used when we simply need to solve a problem quickly. In this paper we investigate methods to monitor the *progress* of the search process for finding solution paths.

End-users greatly value having progress indicators for long tasks (Myers 1985). Further, being able to determine how much time remains before a search algorithm will return a solution has a number of applications. Estimates of time remaining let us know if we should wait for the current algorithm to finish or if we need to change our requirements to get something that will finish within our lifetimes. In the context of anytime search algorithms, it can also help decide whether we should interrupt a running anytime search algorithm now, or if we should wait for an improved solution.

In this paper we propose three approaches for estimating the search progress. The first approach, called the *speed-based* approach, uses existing techniques from time-constrained search (Hiraishi, Ohwada, and Mizoguchi 1998; Dionne, Thayer, and Ruml 2011) to estimate the speed of search progression. The search speed is combined with an estimate of the distance to the goal to provide an estimate of the future search effort.

The second approach, called the *path-based* approach, considers the relation between cost of arriving at an expanded node and the estimated cost of a solution through the same node. The ratio of these values estimates progress.

Finally we present an approach called *distribution based progress estimator*, or *DBP*, which is based on a novel concept called the d-*distribution*. The d-*distribution* of a search is the number of nodes that were generated with a specific estimate of actions remaining, or d-value. *DBP* estimates the search progress by estimating the final d-distribution of search by fitting a curve to the current d-distribution.

We evaluate the proposed techniques for estimating the search progress on two domains, the 15-puzzle and Life-cost Grids (Thayer and Ruml 2011). We compare the accuracy of the progress estimations for A*, weighted A*, and greedy best-first search. Results show that the path-based and distribution-based are superior to the search-speed techniques. It is not conclusive if the distribution-based approach is significantly better than the path-based estimator.

Progress Estimation

In this section we describe formally what search progress is, and how it relates to estimating future search effort. Throughout this paper we assume the time until a search algorithm returns a solution is directly related to the number of nodes yet to be generated. This is reasonable for domains where the cost of node generation is the same for all the nodes. In domains where this isn't true, it can be corrected for so long as we know how the cost of generating nodes changes during search.

Search progress is a number between 0 and 1, representing how near to completion a search is. More formally, let $Gen_A(P)$ be the number of nodes generated by a given search algorithm A while attempting to solve a search problem P. Let $Rem^*_A(P, Gen_A(P))$ be the number of remaining number of nodes that are going to be generated by A when solving P after A has already generated Gen nodes.[1] Search *progress* is the ratio between the already generated

[1]We omit P and A from both Gen and Rem^* when A and P are clear from the context.

nodes and the total number of nodes generated by A when solving P.

Definition 1 (Search Progress) *The search progress of A solving P after generating Gen nodes is:*

$$Prog^*(Gen) = \frac{Gen}{Gen + Rem^*(Gen)}$$

The progress is composed of two components: Gen and $Rem^*(Gen)$. The number of nodes generated so far, Gen, is known. Generally the number of remaining nodes, $Rem^*(Gen)$, is usually not known a priori.[2]

Using future search effort ($Rem^*(Gen)$), one can derive the search progress ($Prog^*(Gen)$) and vice versa. Frequently both values are unknown during search. Some techniques presented in this paper directly estimate search progress, while other techniques estimate future search effort and derive progress from that.

Speed-Based Progress Estimator

We begin by considering *speed-based progress estimators*, or *SBP* for short. Let h_{min} be the lowest h-value of any expanded node. The value of h_{min} gives a heuristic notion of how far the search is from the goal, starting from $h_{min} = h(start)$ and reaching $h_{min} = 0$ when the goal is found. Clearly, h_{min} monotonically decreases to zero throughout the search. The *speed-based progress estimator* described in this section are based on estimating the rate of change in h_{min} with respect to the number of nodes expanded by search. We call this the *search speed*. The search speed and h_{min} are then used to estimate the future search effort, which is then used to estimate the search progress.

Next, we describe two methods for estimating the search-speed, which are a based on previous work on time-constrained search (Hiraishi, Ohwada, and Mizoguchi 1998) and deadline-aware search (Dionne, Thayer, and Ruml 2011).

Velocity-Based Search Speed Estimator

Time-constrained search aims at finding the best possible solution given a time constraint. Hiraishi et al. (1998) proposed a time-constrained search algorithm that is based on Weighted A* (Pohl 1970). Weighted A* uses the node evaluation function $f'(n) = g(n) + w \cdot h(n)$, where w is a parameter. Setting $w = 1$ results in behavior identical to A*, and as w is increased, the algorithm behaves more and more like greedy best first search.

The time-constrained search algorithm of Hiraishi et al. starts by running Weighted A* with w=1. During the search, their algorithm considers two search values that relate to search-speed: the *target search velocity* and *search velocity*. The *target search velocity* is calculated at the beginning of the search, as the initial h_{min}, which is the heuristic value of the start state (h_{start}), divided by the time constraint t. Then, during the search, after every node is expanded, the *search velocity*, denoted by V is calculated as the difference between h_{start} and h_{min}, divided by time that has passed from the beginning of the search. Let the time passed from the beginning of the search be measured by the number of nodes generated so far. Search velocity is $V = \frac{h_{start} - h_{min}}{Gen}$. The time constrained search then adjusts w such that the observed velocity and desired velocity match.

It is easy to see that the *search velocity* is defined to exactly estimate the *search speed* mentioned above. Hence, we can use the *search velocity* to estimate the future search effort, denoted by SE_V as follows:

$$SE_V = \frac{h_{min}}{V}$$

This search effort estimation can then be used to estimate progress, as explained previously. We call the resulting search progress estimator the *velocity-based search progress estimator* (*VeSP*):

$$VeSP(Gen) = \frac{Gen}{Gen + SE_V}$$

Vacillation-Based Search Speed Estimator

Deadline-Aware Search (DAS) is an alternate technique to search under a deadline (Dionne, Thayer, and Ruml 2011). DAS builds on the understanding that generally many nodes are expanded between the time a node is generated and its expansion. This so called *expansion delay* is measured by the DAS algorithm. The average *expansion delay*, $\overline{\Delta e}$, can be used to estimate the *speed* with which search advances towards a goal. We use it to estimate future search effort estimation, SE_e, as follows:

$$SE_e = \overline{\Delta e} \cdot h_{min}$$

We call the resulting search progress estimator the *vacillation-based search progress estimator*, or *VaSP* for short. The progress estimation formula of *VaSP*:

$$VaSP(Gen) = \frac{Gen}{Gen + SE_e}$$

Note that in order to calculate the average expansion delay for *VaSP*, the search algorithm must maintain for every generated node the time when it was generated. This incurs some overhead not required by *VeSP*. Also, note that in *VaSP* the average expansion delay is multiplied by h_{min}. This corresponds to unit edge cost domains. For non-unit edge-cost domains, multiply the average expansion delay by d_{min}.

Both *VeSP* and *VaSP* generalize naturally to estimate the progress of other search algorithms. Clearly, in the first several hundred nodes the speed estimates are expected to be inaccurate, but, at least intuitively, after enough nodes have been expanded both speed estimations can adjust to the speed of the search algorithm that is used.

The above techniques for estimating search progress are adaptations of techniques from time-constrained search to estimate search progress. In the next section we propose simple search progress estimator methods that directly estimate the search progress. These simple estimators are found to be more accurate in our experimental results.

[2]The number of remaining nodes is also known as the *future search effort* (Dionne, Thayer, and Ruml 2011)

Path-Based Progress Estimator

We now introduce the *path-based progress estimator*, or *PBP* in short. First, we describe *PBP* for estimating the progress of an A* search. Then we describe how *PBP* can be adapted to estimate the progress of greedy search.

Consider the components of $f(n)$. Part of the evaluation function, $g(n)$, is in terms of expended cost, while a portion of it, $h(n)$, represents cost-to-go. Let n be the last node generated (i.e., n is the Gen-th node that was generated). The first progress estimator that we consider, called the *naive path-based progress estimator* (*NPBP*) is :

$$\mathit{NPBP}(Gen) = \frac{g(n)}{f(n)}$$

Initially, *NPBP* returns zero, since the g-value of the root is zero. When the goal node is expanded, *NPBP* returns one, since the h-value of the goal is also zero. If the cost-to-go estimate is perfectly accurate ($h(n) = h^*(n)$) and there is a single path to the goal, *NPBP* will return perfect progress estimations, in unit edge cost domains, and A* expands only the nodes in the optimal path. In unit-cost domains $g(n)$ is the number of steps taken between the root and n, $h^*(n)$ is the number of steps between n and a goal, and $f(n)$ is the total number of steps along the optimal path. In this case $\frac{g(n)}{f(n)}$ is an accurate estimate of the progress of search.

Non-Uniform Edge Cost

The above reasoning is incorrect for domains with non-uniform costs. Consider this example: there is a start node s, an intermediate node i, and a goal node g. The cost of going from s to i is $c(s,i) = 1$, and the cost of going from i to g is $c(i,g) = 2$. Assuming perfect information, the previous estimate of progress will tell us that we are only a third of the way done at i, when we are halfway to the goal.

To adapt *NPBP* to non-uniform edge cost domains, we make the progress estimate insensitive to cost as follows: let $d(n)$ be an estimate of the number of actions in the optimal path from n to a goal and $D(n)$ be the depth of node n. We define $L(n) = D(n) + d(n)$. Then the adapted *NPBP* is:

$$\mathit{NPBPL}(n) = \frac{D(n)}{L(n)}$$

If we have perfect information on D and d as well as h, then *NPBPL* will return the exact search progress for non-uniform edge cost as well. For simplicity, we assume uniform edge-cost domain, unless stated otherwise.

Non-Perfect Heuristics

In reality, $h(n)$ is not always equal to $h^*(n)$ and A* will expand nodes off the optimal path. Here the progress estimation of *NPBP* is imperfect. Furthermore, a perfect progress estimation will increase after every node that is expanded. Since *NPBP* is based on the g and h of the node that is currently expanded by A*, its progress estimation may decrease when a new node is expanded. This is because an A* search may expand a node that is close to the goal (high g, low h) and subsequently one far from the goal (low g, high h).

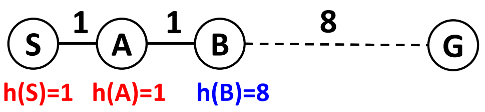


Figure 1: Example of PBP and fPBP

To overcome this we propose the following modification of *NPBP*: always return the highest value that was returned so far by *NPBP*. This ensures that the progress estimation will be monotonic non-decreasing. We call this progress estimator the *path-based progress estimator*, or *PBP* in short. *PBP* shares with *NPBP* the following properties: *PBP* initially returns zero, it returns one when the goal is found, and it is perfect when we have a perfect heuristic. In non-uniform edge cost domains, one can use the same logic as *PBP*, but use $\frac{D(n)}{L(n)}$ instead of $\frac{g(n)}{f(n)}$ in a variant called *PBPL*.

Note that one can the same logic of *PBP* and *PBPL* to construct a progress estimator for Weighted A* (Pohl 1970). Weighted A* uses an evaluation function $f'(n) = g(n) + w \cdot h(n)$, where w is a parameter, instead of the $f(n)$ evaluation function of A*. Thus, we can estimate the progress of Weighted A* by $\frac{g(n)}{f'(n)}$. We denote by *wPBP* this simple adaptation of *PBP* to Weighted A*.

Using Optimal Solution Cost

Sometimes an accurate estimate of the cost of the optimal solution is available (Lelis, Stern, and Arfaee 2011; Lelis et al. 2012). This information could be used to better estimate the progress of search. Assume for a moment that the cost of the optimal solution, OPT, is known. This new progress estimator is based on OPT, and on the minimal f-value in the open list, denoted as f_{min}. Assuming either a consistent heuristic or the use of Pathmax (Mero 1984), f_{min} is monotonic non-decreasing. When the search starts f_{min} is the f-value of the start state $f(start)$ (this is equivalent to $h(start)$). During search f_{min} increases, and when the goal is found $f_{min} = OPT$.

f_{min} can be used to define a new progress estimator, which we call the *f-value path-based progress estimator*, or *fPBP*, that is defined next.

$$\mathit{fPBP}(Gen) = \frac{f_{min} - f(start)}{OPT - f(start)}$$

Here, the denominator $OPT - f(start)$ denotes how much progress must be made since the beginning of search until the optimal solution is found, and the numerator $f_{min} - f(start)$ denotes the search effort completed thus far. Note that for the extreme case where the heuristic is perfect, we have that $f_{start} = OPT$ and $fPBP$ is undefined (since it incurs zero divided by zero).

Naturally, OPT is often unavailable. However, we can estimate OPT with an accurate inadmissible heuristic (Jabbari Arfaee, Zilles, and Holte 2011; Thayer, Dionne, and Ruml 2011) or with solution cost predictors (Lelis, Stern, and Arfaee 2011; Lelis et al. 2012).

Clearly, it is not always easy to obtain good estimates of OPT, and thus it is easier in some cases to implement

PBP. However, *fPBP* is less sensitive to an overly optimistic heuristic function, as explained next. Consider the following extreme example, depicted in Figure 1. There is only a single path from the initial node S to the goal G, that is composed of 10 nodes, i.e., OPT=10, and all the edges have unit cost. Assume that the heuristic of the first two nodes on the path is one, i.e., $h(S) = h(A) = 1$. This value of $h(A)$ and $h(S)$ is admissible, as $h^*(S) = 10$, and $h^*(A) = 9$. However, h is clearly very misleading with respect to node A. This has a great effect on *PBP*, since once A is expanded, *PBP* will return $\frac{g(A)}{f(A)} = \frac{1}{2}$, and will keep this value until a node with a higher value of $\frac{g}{h}$ is expanded. Clearly, the search is far from being half done, since the optimal solution cost is 10. Furthermore, *PBP* will still return $\frac{1}{2}$ even after node B is expanded, which has a perfect heuristic $h(B) = h^*(B) = 8$. By contrast, when A is expanded, *fPBP* will return $\frac{2-1}{10-1} = \frac{1}{9}$, which is clearly more accurate than $\frac{1}{2}$ that was returned by *PBP*.

PBP for Greedy Best-First Search

Next, we describe how to use the concept of *PBP* to estimate the progress of greedy best-first search (GBFS). GBFS, also known as Pure Heuristic Search, is a best-first search that orders the nodes in the open-list according to their h-value, always expanding the node in the open list with the lowest h-value. GBFS is commonly used when the task is to find a solution to a search problem as fast as possible.

One way to define *PBP* for GBFS is exactly the same as *PBP* for A*. However, GBFS behaves differently from A*. GBFS accounts only for the heuristic value of the nodes in the open list to decide which node to expand next. Therefore, we propose the following variation of *PBP* for GFBS, denoted by *hPBP*, that also only accounts for the heuristic value of the nodes seen during search.

$$\textit{hPBP}(Gen) = \frac{h(root) - h_{min}}{h(root)}$$

As before, it is also possible to define an equivalent estimator that considers distance-to-go (i.e., estimated number of actions to reach the goal), for non-unit-cost domains:

$$\textit{dPBP}(Gen) = \frac{d(root) - d_{min}}{d(root)}$$

Both the path-based and the speed-based search progress estimations described so far heavily depend on the value of h_{min}. This means that if a node is generated with very low h_{min} in a early stage of the search, e.g., due to a gross under-estimation of the heuristic, both search speed and path-based progress estimators will be inaccurate. The next progress estimator, called the *distribution based progress estimator* or *DBP* in short, is based on a completely different approach that is robust to such sporadic heuristic errors.

Distribution-Based Progress Estimator

DBP estimates the search progress by learning the "structure" of the search space.[3] Specifically, *DBP* tries to predict

[3]We use the term "search space" to denote the set of nodes that will be visited by the search algorithm that is executed.

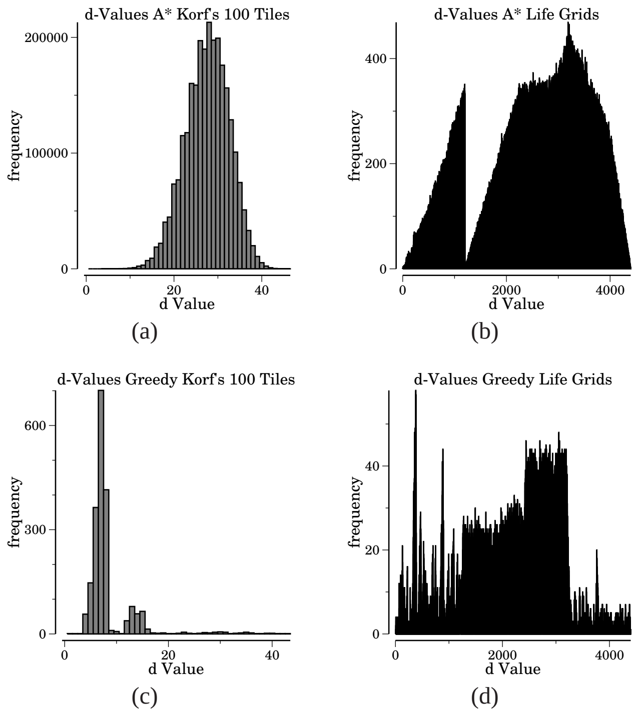


Figure 2: d-distributions for A* and GBFS

how many nodes will be generated during the search for every value of d.[4] This is done by accumulating throughout the search the number of nodes generated so far for every possible d-value, and interpolating the number of nodes that will be generated in the future for every possible d-value.

Let $d_0, .., d_m$ be all the possible d-values a node can have. Throughout the search, *DBP* maintains for every d-value d_i a counter $c[d_i]$ that counts the number of nodes generated with d-value equal to d_i. Initially, $c[d_i] = 0$ for all d values, except for the d-value of the start state, d_s, for which $c[d_s] = 1$. When a node n is generated, then $c[d(n)]$ is incremented by one. As the search progresses, the values of $c[d_1], .., c[d_m]$ are updated, resulting in a distribution of the frequency of the different heuristic values. We call this distribution the d-distribution of the search at a given time, or simply the *current* d-distribution. The d-distribution when the search ends is called a *complete* d-distribution. Figure 2 shows the complete d-distribution for A* and GBFS of a random instance of the 15-puzzle and of the life-grid domains.

Let c^* denote the *complete* d-distribution, and correspondingly, let $c^*[d_i]$ be the number of nodes generated throughout the search with d-value equal to d_i. Given the complete d-distribution, one can easily compute the exact search progress as follows:

$$\textit{Prog}^*(Gen) = \frac{Gen}{\sum_{i=1}^{m} c^*[d_i]}$$

The complete d-distribution, c^*, is not available until the search is finished. To overcome this, *DBP* estimates the

[4]Recall that in uniform edge cost domains d is equivalent to h.

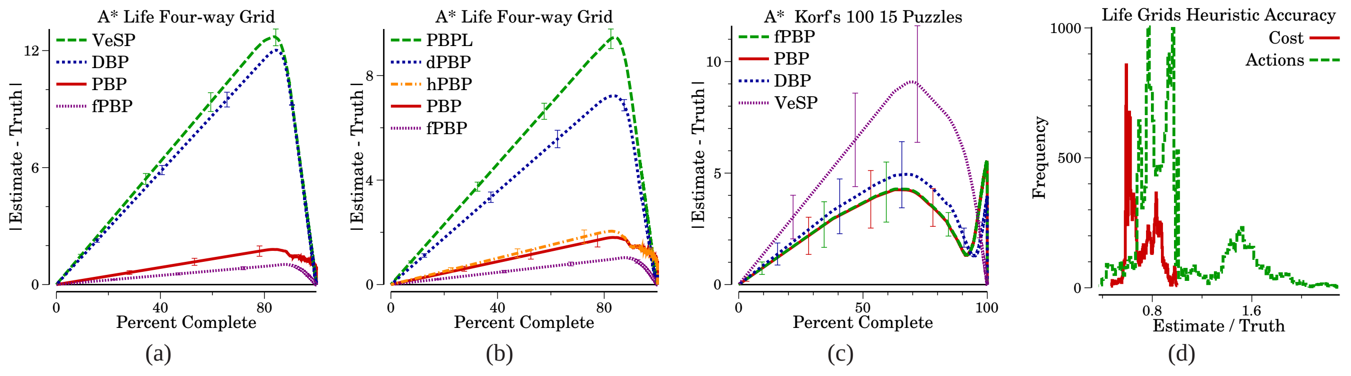


Figure 3: Study of Progress Estimation in A* and the accuracy of d vs. h.

complete distribution from current distribution by using a second degree polynomial fit. Let $\hat{c}$ denote the resulting estimation of the complete d-distribution, and correspondingly, let $\hat{c}[d_i]$ be the number of nodes that will have a d-value of d_i according to $\hat{c}$. The search progress estimated by *DBP* with $\hat{c}$ is given by:

$$\mathit{DBP}(Gen) = \frac{Gen}{\sum_{i=1}^{m} \hat{c}[d_i]}$$

In the beginning of the search the curve fitting required by *DBP* is not given enough data to provide accurate estimations of the complete d-distribution. However, as the search progresses, more nodes are generated, and $\hat{c}$ is expected to provide a better estimation of the complete d-distribution.

Empirical Evaluation

In this section we evaluate accuracy of the proposed progress estimators techniques for three search algorithms: A*, Greedy Best First Search (GBFS) and Weighted A* (Pohl 1970). Recall that *progress* is measured by the ratio between the number of nodes generated and the total number of nodes generated by the search algorithm when the goal is found (Definition 1). Thus, the accuracy of a progress estimator is measured by the absolute difference between the *estimated progress* returned by the progress estimator, and the *real progress*, computed after the problem was solved. In the plots shown in this section, the y-axis is the average accuracy of a progress estimator for a set of problem instances. The x-axis in the plots is the *real progress*. As it is impractical to write down the estimated progress for every expanded node for these problems, we take 500 samples uniformly at random from all data points generated by the search.

We test the estimators on Life-cost grids and on the 15 puzzle. Life-cost grids were first proposed by Ruml and Do (2007). These are a standard 4-connected grid with a slightly different cost function, moving out of a cell has cost equal to the y-coordinate of the cell. We use 100 grids that are 2000x1200, with a starting state in the lower left-hand corner of the grid and the goal in the lower right. For the 15-puzzle, we look at the 100 instances used by Korf (1985).

A*

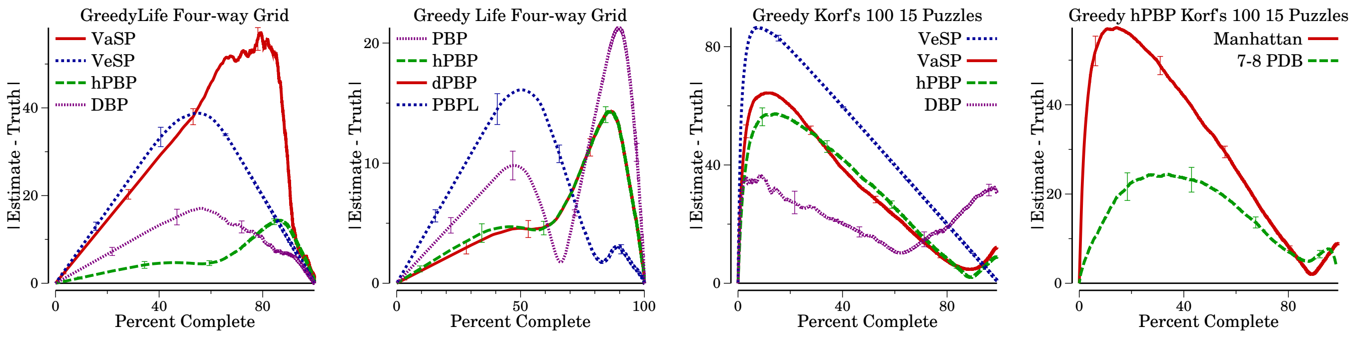
Figure 3(a) shows the relative performance of *VeSP*, *DBP*, *PBP*, and *fPBP*. We exclude the results of VaSP from this plot because they were substantially worse than the other approaches. The two best estimators in this case are clearly the path-based estimators *fPBP* and *PBP*. Even though less accurate than the path-based estimators, DBP is also fairly accurate with errors no greater than 10%. We believe the accuracy of *DBP* could be improved in this domain, however. Figure 2(b) shows the d-distribution for this domain (life grids). Clearly the second-order polynomial used by *DBP* to estimate the real d-distribution does not capture the actual distribution of d-values, explaining *DBP*'s performance.

Since the path-based estimators performed best in this domain, we compared several variants of the path-based estimators. Specifically, we consider three path-based estimator variants: *PBP*, *PBPL* and *fPBP*. Recall that *PBPL* is *PBP* that uses a distance-to-go heuristic (d) and depth (D) instead of a cost-to-go heuristic (h) and cost-so-far (g).

The results are shown in Figure 3(b). First, we can see that *PBPL* is outperformed by both *fPBP* and *PBP*. This is counter intuitive, as the actions-to-go heuristics (i.e, d) corresponds more accurately than cost-to-go heuristics (h) to the number of expansions required to reach a goal. This phenomenon can be explained by considering the accuracy of d and h in this domain, shown in Figure 3(d). In this plot, we show the relative error in the cost-to-go and actions-to-go heuristics computed as $\frac{h(n)}{h^*(n)}$ and $\frac{d(n)}{d^*(n)}$ respectively. The plot shows clearly that the d heuristic in this domain is far less accurate than the h heuristic, explaining the improved performance of h-based estimators (*fPBP* and *PBP*) over the d-based estimator *PBPL*.

The second observation that can be seen from Figure 3(b) is that *fPBP* outperforms both *PBP*, and *PBPL*. This supports the analysis given in the path-based section and exemplified in Figure 1. However, note that *fPBP* uses additional information that is not available to *PBP*: optimal solution cost. Interestingly, the difference in accuracy between *fPBP* and *PBP* is very small, where *fPBP* is only about 1% more accurate than *PBP*.

The relative performance of the estimators on the 15-



(a) (b) (c) (d)

Figure 4: Study of Progress Estimation in Greedy Search

puzzle problem is shown in Figure 3(c). In this domain both *DBP* and *PBP* have similar accuracy. Figure 2(a) shows the d-distribution seen during an A* search for a random start state of the 15-puzzle. The second-order polynomial fit used by *DBP* is clearly a good choice in this case, which explains the improved performance of *DBP* in this domain.

In contrast to the life-grids domain, *PBP* is more accurate than *fPBP* in the 15-puzzle domain. Recall that *fPBP* measures search progress according to the function $\frac{f_{min}-f(start)}{OPT-f(start)}$. As OPT and $f(start)$ remain fixed throughout the search, we have that *fPBP* measures search progress based on f_{min}, i.e., the value of the largest expanded f-value. Thus, *fPBP* assumes that the search effort required to increase the f-value is the same throughout search. In the Grid domain the number of nodes in different f-layers is a relatively fixed number as the f-values increases. Thus, this assumption of *fPBP* holds. However, in the tiles domain the number of nodes in a f-layer grows exponentially as the f-value increases. Thus, in the tiles domain the assumption that the search effort required to increase f_{min} is the same throughout search clearly does not hold. *PBP*, on the other hand, measures search progress according to the maximal $\frac{g(n)}{f(n)}$ observed. Thus, *PBP* also considered the advance in g-values in addition to the advance in f-values seen during the search. Hence, *PBP* is able to estimate different search effort for different f-layers, which results in more accurate progress estimations in this domain.

Greedy Search

We now discuss the search progress estimators in Greedy Best First Search. First, we compare the three progress estimation approaches presented in this paper: speed-based (*VaSP* and *VeSP*), path-based and histogram-based (*DBP*). As explained in the path-based estimation section, for GBFS we modified *fPBP* to consider either h or d instead of f, resulting in *hPBP* and *dPBP* progress estimators, respectively.

First, consider the y-axis scale used for the GBFS progress estimations in Figure 4 in comparison to the y-axis scale used for the A* plots in Figure 3. Clearly, all progress estimators perform poorly for estimating the progress of GBFS, with errors ranging up to in comparison with their performance in estimating the progress of an A* search.

Next, consider the life grid results shown in Figure 4(a). Similar to the A* results described above, both *hPBP* and *DBP* significantly outperform the speed-based methods *VaSP* and *VeSP*. *VaSP* and *VeSP* both frequently err in their estimates by more than 30%, while *PBP* and *DBP* have estimates that do not err by more than 20%, and frequently have less than 10% absolute error. *hPBP* and *DBP* differentiate themselves by when they are most accurate, with *hPBP* being more accurate early on, and *DBP* having better estimates near the end of the search.

Following, consider the performance of the different variants of *PBP*, namely *hPBP*, *dPBP*, *PBP* and *PBPL*. Accuracy results are shown in Figure 4(b) In this domain we see small difference between these variants, where some of them are more accurate at the beginning of the search (*dPBP* and *hPBP*), while others path-based predictors are are better near the end of the search (e.g., *PBPL*). It is hard to see a significant variant outperforming the other.

Figure 4(c) presents these same estimation techniques in greedy search on the 15-puzzle. Results in the tiles problem are similar to those reported on grids in that *DBP* and *hPBP* are still the best predictors overall, with *DBP* being more accurate in most of the search, except near the end.

While *hPBP* produces accurate estimates with error frequently below 10% on the Grid domain, its estimates on the tiles domain are inaccurate. We conjecture that the inaccuracy of *hPBP* on the 15-puzzle is explained by the combination of two facts. First, usually in the beginning of a greedy search the value of h_{min} decreases rapidly (Imai and Kishimoto 2011). This will mislead *hPBP* into thinking that the search progressed more than it actually have. Second, the range of heuristic values for the 15-puzzle is much smaller than the range for the Grid domain. Thus, the rapid decrease in h_{min} in the beginning of search will have a major impact on the *hPBP* estimations for the 15 puzzle. For instance, if h_{min} decreases from 40 to 20 with just a few nodes expanded, *hPBP* will mistakenly assert that the search is about half-way done. On the other hand, on a domain with much deeper solutions and larger range of heuristic values, a quick decrease of 20 in the h_{min} will not represent a large change in the estimated search progress.

Another factor that contributes to the difference in accuracy of *hPBP* between the two domains is the accuracy of the heuristic functions used. Figure 4(d) shows the accuracy of *hPBP* when using heuristic functions of different accuracy. Here we make *hPBP* estimations when employing Manhattan Distance and the 7-8 additive pattern database (7-8 PDB) (Felner, Korf, and Hanan 2004). Korf and Felner showed that the 7-8 PDB is far more accurate than Manhattan Distance, and we see that the progress estimations when using the 7-8 PDB are also more accurate. The heuristic used in grid is relatively more accurate than the heuristic used in the 15-puzzle. Thus, as observed in the results, *hPBP* is often more accurate for life grids.

Even though *DBP* is the best estimator on the 15-puzzle as shown in Figure 4(c), its estimations have an average error of about 25%, reaching values of almost 40% in the beginning and in the end of search. These are inaccurate estimations, especially if compared to the DBP estimations for A* (where the error of *DBP* is at most 10%). The inaccuracy of *DBP* for greedy search is explained by the histograms in Figure 2(d) and 2(c); the second-order polynomial used by *DBP* doesn't capture the shape of the d-distribution of greedy search.

Weighted A*

Now that we have discussed A* search and greedy search, we turn our attentions to Weighted A* (Pohl 1970), a best-first search algorithm that uses an evaluation function $f_w(n) = g(n) + w \cdot h(n)$, where w is a parameter. The behavior of Weighted A* (WA*) scales smoothly between the A* and GBFS extremes, according to the value of w, where $w = 1$ results in A*, and $w = \infty$ results in GBFS.

First, we compare the accuracy of the speed-based, path-based and distribution-based approaches, when used to estimate the progress of Weighted A* with $w = 1.5$. Results for life grid and 15-puzzle are shown in Figure 5(a) and 5(b). As before, we see that *PBP* and *DBP* have, in general, the best performance, and are more accurate than either *VeSP* and *VaSP* when estimating search progress (*VaSP* is not shown as it was much worse than other estimators).

Next, consider the effect of different values of w on the performance of *PBP*. Figure 5(c) compares the performance of *PBP* with $w = 1.1, 1.5, 1.75$ and 2. As can clearly be seen, increasing w results in less accurate search progress estimations. This is understandable, as increasing w results in Weighted A* behaving more like GBFS. Since the accuracy of *PBP* was shown to be significantly worse for GBFS than for A*, it is clear that *PBP* with higher w will be less accurate that *PBP* for lower values w.

Lastly, we consider the performance of variants of the *PBP* technique in Weighted A* search. Results are shown in Figure 5(d). As the *dPBP* and *PBPL* estimates did not work in either A* or GBFS, we omit them from the evaluation here. Instead, we focus on *hPBP*, *PBP*, and *wPBP*. As the results show, *PBP* outperforms both *hPBP* and *wPBP*. We conjecture that *PBP* outperforms *wPBP* because $f'(n)$ provides a pessimistic estimate of the search effort required to complete search. Thayer and Ruml (2008) have shown that WA* typically finds solutions with cost much lower than $w * OPT$, and thus wPBP is misled.

Discussion and Summary

fPBP assumes that the search effort required to increase f_{min} is the same across search; *PBP* assumes that the search effort required to increase the value of $\frac{g(n)}{f(n)}$ is also the same throughout search. These assumptions might be problematic in domains with shallow solutions and large branching factor. In such domains any change in f_{min} and in the largest g-value seen might represent a large change in the estimated percentage of search completed. For instance, in a domain with average solution depth of 5, after expanding the root node *PBP* might estimate that about 20% of search was completed, while in reality very little of the search was completed after expanding the root node. In such domains we expect *DBP* to perform better, as long as the fitting function being used is able to capture the actual distribution of d-values.

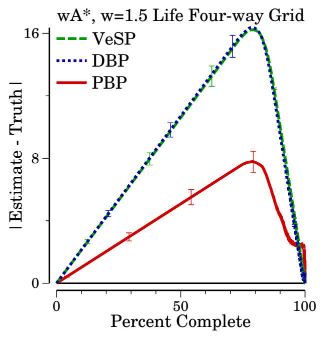
We observed the following trends in the results. First, the novel *PBP* and *DBP* progress estimators are able to predict with high accuracy the progress of an A* search, differing from the actual progress by less than 5%. Second, the proposed progress estimators are in general less accurate for GBFS. Results for Weighted A* lie in the continuum between these two extremes. Third, it is often the case that the simpler *PBP* variants are more accurate than even the more complex *DBP*, as well as *VaSP* and *VeSP*.

Related Work

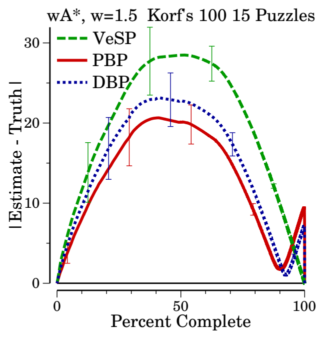
A considerable amount of research was devoted to estimating the search effort of tree-based methods (Korf, Reid, and Edelkamp 2001; Zahavi et al. 2010; Lelis, Zilles, and Holte 2012; Knuth 1975; Burns and Ruml 2012; Kilby et al. 2006). The largest difference between the methods we presented in this paper and the ones mentioned above is that we estimate search progress for best-first searches instead of tree-based searches. Breyer and Korf (2008) predicted the number of nodes expanded by A* for the 15-puzzle, but they used the information learned from a complete breadth-first search in the state space. Hernadvolgyi and Holte (2004) also made estimations for the number of nodes expanded by A*, but they ignored the transposition detection the algorithm does. None of the methods cited above account for transposition detection other than transpositions on a single path. To the best of our knowledge we are the first to make predictions of the search effort of algorithms that detect transpositions. In addition, the methods presented in this paper are online estimators in the sense that they estimate search effort while using the information learned by a search algorithm, while most tree-based methods do not consult search performance.

Future Work

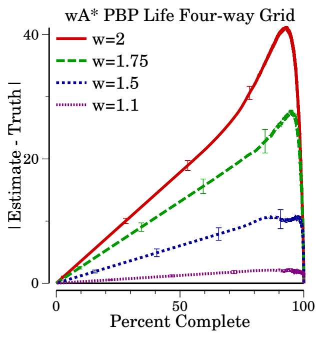
While having good indications of when our searches will end is important, the most exciting potential application of progress estimation techniques is in the realm of search under time constraints. Our evaluation showed that we have more accurate estimators than those previously used in deadline search algorithms, and that is likely to lead to better performance for search under a deadline.



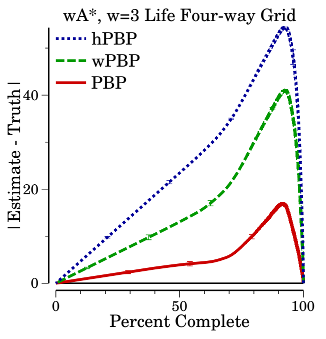
(a)



(b)



(c)



(d)

Figure 5: Study of Progress Estimation in WA*

In this paper, we do not consider adapting techniques for estimating the size of an IDA* search tree to estimating best-first search progress. The largest challenge here is learning how to account for duplicates in tightly connected search spaces like grids. Future work will investigation of how to adapt the methods mentioned above so that they can be used to measure search progress of best-first search.

Conclusion

In this paper we have presented and evaluated several techniques for estimating the *search progress* during the execution of a search algorithm. Estimating the search progress can improve the applicability of search algorithms in real applications, as end-users strongly prefer having progress indicators such as progress bars when applications perform long tasks (Myers 1985). This is to our knowledge the first work on estimating the *search progress*. The proposed techniques work well for A* and Weighted A* with low-weights. Future work will study how to estimate the progress of GBFS and other search algorithms.

References

Breyer, T., and Korf, R. 2008. Results results in analyzing the performance of heuristic search. In *Proceedings of the first international workshop on search in artificial intelligence and robotics*.

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

Dionne, A. J.; Thayer, J. T.; and Ruml, W. 2011. Deadline-aware search using on-line measures of behavior. In *SOCS*.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR)* 22:279–318.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

Hernádvölgyi, I. T., and Holte, R. C. 2004. Steps towards the automatic creation of search heuristics. Technical Report TR-04-02, University of Alberta.

Hiraishi, H.; Ohwada, H.; and Mizoguchi, F. 1998. Time-constrained heuristicsearch for practical route finding. In *Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, 389–398.

Imai, T., and Kishimoto, A. 2011. A novel technique for avoiding plateaus of greedy best-first search in satisficing planning. In *AAAI*.

Jabbari Arfaee, S.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17):2075–2098.

Kilby, P.; Slaney, J. K.; Thiébaux, S.; and Walsh, T. 2006. Estimating search tree size. In *AAAI*.

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

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. Iterative-deepening-A*: An optimal admissible tree search. In *IJCAI*, 1034–1036.

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

Lelis, L.; Stern, R.; and Arfaee, S. J. 2011. Predicting solution cost with conditional probabilities. In *SOCS*.

Lelis, L. H. S.; Zilles, S.; and Holte, R. C. 2012. Fast and accurate predictions of IDA*'s performance. In *AAAI*.

Mero, L. 1984. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence* 23:13–27.

Myers, B. A. 1985. The importance of percent-done progress indicators for computer-human interfaces. In *the Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '85, 11–17. ACM.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193–204.

Ruml, W., and Do, M. B. 2007. Best-first utility-guided search. In *Proceedings of IJCAI-07*, 2378–2384.

Thayer, J. T., and Ruml, W. 2008. Faster than Weighted A*: An optimistic approach to bounded suboptimal search. In *ICAPS*, 355–362.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, 674–679.

Thayer, J.; Dionne, A.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *ICAPS*.

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