On-line Tree Size Prediction using Incremental Models

Computer Science Technical Report 11-01

Ethan Burns   Wheeler Ruml

January 26, 2011

Abstract

It is often useful to know, in advance, how much effort will be required to solve a search problem. Current techniques for estimating the number of nodes that a bounded depth-first search will expand require copious amounts of off-line training and only make predictions in domains with integer-valued heuristic estimates. We present a new technique for estimating the number of nodes that a bounded depth-first search will expand. Our technique has two main benefits over previous approaches: 1) it will work in domains with real-valued heuristic estimates and 2) it can be trained on-line during an iteration of search. We show that our technique can predict as well as previous approaches when trained off-line. We also show that our new approach can be used on-line to accurately predict the number of nodes that an IDA* style search will encounter on subsequent iterations. We demonstrate the usefulness of this technique by controlling an IDA* search: using knowledge learned on previous iterations to set the bound for subsequent iterations. While our technique has more overhead than previous methods for controlling IDA*, it can give more robust performance by accurately doubling the amount of search effort between iterations.

0.1 Introduction

Best-first search is a fundamental tool for automated planning and problem solving. One major drawback of best-first search algorithms such as A* [Hart et al., 1968] is that they may use an excessive amount of memory. This means that for large problems in which many nodes must be generated, A* is infeasible. In many of these domains, iterative deepening A* (IDA*, [Korf, 1985]) can be applied. IDA* performs a series of bounded depth-first searches. Each search expands all nodes whose estimated solution cost, f, falls within the bound. After every iteration that fails to expand a goal, the bound is increased to the minimum f value of any node that was generated but not previously expanded. Because it uses depth-first search, iterative deepening only uses an amount of memory that is linear in the maximum search depth.

Due to the exponential nature of many search spaces, there tends to be a large variability in the running time of search algorithms even across different instances in the same domain. Knuth [1975] points out that making even a slight change to a search space may cause a search that typically requires only seconds to require centuries. This makes it useful to have an estimate of how long a search is going to take.

One possible use for such a prediction is to compare different parameter settings for a problem without having to run a full search. Haslum et al. [2007] use one such technique to select the best among different heuristics for domain independent planning. This approach to estimating search effort, developed by Korf et al. [2001] and later extended by Zahavi et al. [2010], is able to predict the number of expansion with surprising accuracy. However, this method has two drawbacks: 1) it requires copious amounts of off-line training to learn the distribution of heuristic values and 2) it does not extend easily to domains with real-valued heuristic estimates.

Search performance predictions can also be used during search to change the behavior of an algorithm on-line to better suit the characteristics of the current problem. IDA* with Controlled Reexpansion (*IDA*$*_{CR}$, [Sarkar et al., 1991]) uses a simple model of the search tree in between iterations of an IDA* search to predict a good bound for the next iteration. However, as we will see, the simple model used in IDA*$_{CR}$ is fragile because its accuracy relies on properties of a search space that are not always present.

The main contribution of this work is a new type of model that can be used to estimate the number of nodes expanded in an iteration of IDA*. We call our new model an *incremental model* and we show that it is able to predict as accurately as the current state-of-the-art model for the unit-cost 15-puzzle when trained off-line. Unlike the previous approaches, however, our incremental model can also handle domains with real-valued heuristic estimates. Furthermore, while the previous approaches require large amounts of off-line training, our model may be trained on-line during a search. We show that our model can be used to control an IDA* search by using information learned on completed iterations to determine a bound to use in the subsequent iteration. Our results show that our new model accurately predicts IDA* search effort. While IDA* guidance using our model tends to be expensive in terms of CPU time, the gain in accuracy allows the search to remain robust. Unlike the other IDA* variants which occasionally give very poor performance, IDA* using an incremental model is either the best or it remains competitive with the best IDA* variant on the domains used in our experiments.

0.2 Previous Work

Korf et al. [2001] give a formula (henceforth abbreviated *KRE*) for predicting the number of nodes IDA* will expand with a given heuristic when searching to a given cost threshold. The KRE method uses an estimate of the heuristic value distribution in the search space to determine the percentage of nodes at a given depth that are *fertile*. A fertile node is a node within the cost bound of the current search iteration and hence will be expanded by IDA*.

The KRE formula requires two components: 1) the heuristic distribution in the search space and 2) a function for predicting the number of nodes at a given depth in the brute-force search tree. They showed that off-line random sampling can be used to learn the heuristic distribution. For their experiments a sample size of ten billion states was used to estimate the distribution of the 15-puzzle. Additionally, they demonstrate that a set of recurrence relations, based on a feature called the *type* of a node, can be used to find the number of nodes at a given depth in the brute-force search tree for a tiles puzzle or Rubik's cube. The results of the KRE formula using these two techniques gave remarkably accurate predictions when averaged over a large number of initial states for each domain.

Zahavi et al. [2010] provide a further generalization of the KRE formula called Conditional Distribution Prediction, or *CDP*. The CDP formula uses a conditional heuristic distribution to predict the number of nodes within a cost threshold. The formula takes into account more information than KRE such as the heuristic value and node type of the parent and grandparent of each node as conditions on the heuristic distribution. This extra information enables CDP to make predictions for individual initial states and in domains with inconsistent heuristics. Using CDP, [Zahavi et al., 2010] show that substantially more accurate predictions can be made on the sliding tiles puzzle and Rubik's cube given different initial states with the same heuristic value.

While the KRE and CDP formulas are able to give accurate predictions, their main drawback is that they require copious amounts of off-line training to estimate the heuristic distribution in a state space. Not only does this type of training take an excessive amount of time but it also does not allow the model to learn any instance-specific information. In addition, the implementation of these formulas as specified by Zahavi et al. [2010] assumes that the heuristic estimates have integer values so that they can be used to index into a large multi-dimensional array. Many real-world domains have real-valued edge costs and therefore these techniques are not applicable in those domains.

0.2.1   Controlling Iterative Search

There are two reasons why IDA* is able to excel on classic academic benchmarks like the sliding tile puzzle and Rubik's Cube. First, when the appropriate symmetry-breaking and pruning rules are used, both domains form search spaces that are sparsely connected and closely resemble trees. IDA* uses a depth-first search and any domain that has many paths to the same nodes will cause the algorithm to unroll this graph, re-visiting all descendant states, until the cost bound is reached. This causes IDA* to expand many more nodes than a graph search algorithm with duplicate detection such as A*. We will not address this issue in this paper, instead assuming that the search space contains relatively few redundant paths. This is not an unreasonable assumption since many real-world domains (such as temporal planning) have this property.

The second reason that IDA* performs well on both of these domains is that they both have a geometrically increasing number of nodes that fall within the cost bound as it is increased by the minimum possible amount. It has been shown that in a non-geometrically growing domain, IDA* may perform $\mathcal{O}(n^2)$ expansions where n is the number of nodes expanded by A* and in a domain that grows geometrically IDA* will expand $\mathcal{O}(n)$ nodes [Sarkar et al., 1991]. Unfortunately, domains with real-valued edge costs tend to have many distinct f values and therefore do not grow geometrically.

The problem with IDA* in domains with many distinct f values is well known and has been explored in past work. Vempaty et al. [1991] present an algorithm called DFS*. DFS* is a combination of IDA* and depth-first search with branch-and-bound which sets the bounds between iterations more liberally than standard IDA*. While the authors describe a sampling approach to estimate the bound increase between iterations, in their experiments, the bound is simply increased by doubling.

Wah and Shang [1995] present a set of three linear regression models to control an IDA* search. Unfortunately, intimate knowledge of the growth properties of f layers in the desired domain is required before the method can be used. In many settings, such as domain-independent planning for example, this knowledge is not available in advance.

IDA* with Controlled Re-expansion (IDA*$_{CR}$, [Sarkar et al., 1991]) uses a method similar to that of DFS*. IDA*$_{CR}$ uses a simple model of the search space that tracks the f values of the nodes that were pruned during the previous iteration and uses them to find a bound for the next iteration. IDA*$_{CR}$ uses a histogram to count the number of nodes with each out-of-bound f value during each iteration of search. When the iteration is complete, the histogram is used to estimate the f value that will double the number of nodes in the next iteration. The remainder of the search proceeds as in DFS*, by increasing the bound and performing branch-and-bound on the final iteration to guarantee optimality.

While IDA*$_{CR}$ is simple, the model that it uses to estimate search effort relies upon two properties of the search space to achieve good performance. The first is that the number of nodes that are generated outside of the bound must be at least the same as the number of nodes that were expanded. If there are an insufficient number of pruned nodes, IDA*$_{CR}$ sets the bound to the greatest pruned f value that it has seen. This value may be too small to significantly advance the search. The second property is that no nodes that are generated from the pruned successors of one iteration should fall within the bound on the next iteration. If this happens, then the next iteration may be much larger than twice the size of the previous. As we will see, this can cause the search to overshoot the optimal solution cost on its

final iteration, giving rise to excessive search effort.

0.3 Incremental Models of Search Trees

To estimate the number of nodes that IDA* will expand with a given cost threshold we would like to know the distribution of f values in the search space. Assuming a consistent heuristic, all nodes with f values within the threshold will be expanded. If this distribution is given as a histogram that contains the number of nodes with each f value then we can simply sum the weight for f values less than and equal to our cost bound to find the desired number of nodes. Our new incremental model performs this task and has the ability to be trained both off-line with sampling and on-line during a search.

We estimate the distribution of f values using two steps. In the first step we learn a model of how the f values are changing from nodes to their offspring and in the second step we extrapolate from the model of change in f values to estimate the overall distribution of f values. This means that our incremental model manipulates two main distributions: we call the first one the Δf distribution and the second one the f distribution. In the next section, we will describe the Δf distribution and give two techniques for learning it. We will then describe how the Δf distribution can be used to estimate the f distribution.

0.3.1 The Δf Distribution

The goal of learning the Δf distribution is to predict how the f values in the search space change between nodes and their offspring. The advantage of storing Δf values instead of storing the f values themselves is that it enables our model to extrapolate to portions of the search space for which it has no training data, a necessity when using the model on-line or with few training samples. We will use the information from the Δf distribution to build an estimate of the distribution of f values over the search nodes.

The CDP technique of Zahavi et al. [2010] learns a conditional distribution of the heuristic value and node type of a child node c conditioned on the node type and heuristic estimate of the parent node p notated $P(h(c), t(c)|h(p), t(p))$. As described by [Zahavi et al., 2010], this requires indexing into a multi-dimensional array according to $h(p)$ and so the heuristic estimate must be an integer value. Our incremental model also learns a conditional distribution, however in order to handle real-valued heuristic estimates, our incremental model uses the integer valued search-space-steps-to-go estimate d of a node instead of its heuristic estimate, h. In unit-cost domains, d will typically be the same as h, however in domains with real-valued edge costs, d gives an estimate of the number of search nodes on an optimal-cost path to a goal whereas h gives an estimate of the remaining cost. d is typically easy to compute while computing h [Thayer and Ruml, 2009]. The distribution that is learned by the incremental model is $P(\Delta f(c), t(c), \Delta d(c)|d(p), t(p))$, that is the distribution over the change in f value between a parent and child, the child node type and the change in d estimate between a parent and child given the distance estimate of the parent and the type of the parent node.

The only non-integer term used by the incremental model is $\Delta f(c)$. Our implementation uses a large multi-dimensional array of fixed-sized histograms over $\Delta f(c)$ values. Each of the integer-valued features is used to index into the array, resulting in a histogram of the $\Delta f(c)$ values. By storing counts, the model can estimate the branching factor of the search space by dividing the total number of nodes with a given d and t by the total number of their offspring. This branching factor is used to estimate the number of successors of a node when building the f distribution.

Zahavi et al. [2010] found that it is often important to take into account information about the grandparent of a node for the distributions used in CDP. We accomplish this with the incremental model by rolling together the node types of the parent and grandparent into a single type. For example, on the 15-puzzle, if the parent state has the blank in the center and it was generated by a state with the blank on the side then the parent type would be a *side–center* node. This allows us to use an array with the same dimensionality across domains that take different amounts of ancestry into account.

Learning Off-line

We can learn an incremental Δf model off-line using the same method for learning distributions with KRE and CDP. This involves sampling a large number of random states from a domain. The children (or grandchildren) of each sampled state are generated. The change in distance estimate $\Delta d(c) = d(c) - d(p)$, node type $t(c)$ of the child node, node type $t(p)$ of the parent node, and the distance estimate $d(p)$ of the parent node are computed and are used to index into a multi-dimensional array to access the histogram that corresponds to these features. A count of 1 is then added to the histogram for the (possibly real-valued) change in f, $\Delta f(c) = f(c) - f(p)$ between parent and child.

Learning On-line

An incremental Δf model can be learned on-line by looking at each node generation during search. Each time a node is generated, the $\Delta d(c)$, $t(c)$, $t(p)$ and $d(p)$ values are computed for the parent node p and child node c. These values are used to index into an array of histograms, and a count of 1 is added to the corresponding histogram for $\Delta f(c)$, as in the off-line case. In addition to these features, when learning a Δf model on-line the *depth* of the parent node in the search tree is also known. We have found that this feature greatly improves accuracy in some domains (such as the vacuum domain described below) and so we always add it as a conditioning feature when learning an incremental model on-line.

Since an IDA* search will expand some of the same nodes between iterations, much of the data used to learn the Δf distribution on-line will be encountered multiple times, once per iteration. Our implementation tracks the bound used in the previous iteration and the model is only trained when expanding a node that would have been pruned on the previous iteration. This procedure only trains a single time on each distinct node generation. Additionally, the search spaces for many domains form graphs instead of trees. In these domains our implementation of depth-first search does cycle checking by using a hash table of all of the nodes along the current path. In order for our model to take this extra pruning into account we only train the model on the successors of a node that pass the cycle detection.

Learning a Backed-off Model

Due to data sparsity and because the Δf model will be used to extrapolate information about the search space for which it may not have any training data, a backed-off version of the model may be needed. The backed-off model is a more general model that is conditioned on fewer features of each node. When querying the model, if there is no training data for a given set of features, the more general backed-off model is consulted instead.

When learning a model on-line, because the model is learned on instance-specific data, we found that it was only necessary to learn a model that backs off the depth feature. When training off-line, however, we learn a series of two back-off models, first eliminating the parent node distance estimate and then eliminating both the parent distance estimate and type.

0.3.2 The f Distribution

Our incremental model predicts the number of nodes within a cost threshold for a given start state by estimating the distribution of f values among the nodes in the search space. We use the information learned in the Δf model to *incrementally* build our estimation of the f distribution. This is accomplished by using the f value distribution of one search depth and the model of Δf to generate the f value distribution for the next depth. By beginning with the root node, which has a known f value, our procedure simulates the expansions of each depth layer to compute estimates of the f value distribution at the next layer. The accumulation of these depth-based f value distributions can then be used to make our prediction.

To increase accuracy, the distribution of f values at each depth is conditioned on node type t and distance estimate d. This distribution estimates the number of nodes with a given f value at a each depth with each distinct combination of t and d. We begin our simulation with a count of 1 for $f = f(root)$, $t = t(root)$ and $d = d(root)$. Next, the Δf model is used to find a distribution over Δf, t and Δd values for the offspring of each combination of t and d values of the nodes at the current depth. By storing Δ values, we can compute $d(c) = d(p) + \Delta d(c)$ and $f(c) = f(p) + \Delta f(c)$

SIMULATE(*bound, desired, depth, accum, nodes*)
  1. $nodes' = $ SIM-EXPAND(*depth, nodes*)
  2. $accum' = add(accum, nodes - nodes')$
  3. $bound' = find\_bound(accum', bound, desired)$
  4. if $weight\_left\_of(bound', nodes - nodes') > \epsilon$
  5. $depth' = depth + 1$
  6.   SIMULATE(*bound', desired, depth', accum', nodes'*)
  7. else return $accum'$

SIM-EXPAND(*depth, nodes*)
  8. $nodes' = $ new 2d histogram array
  9. for each t and d with $weight(nodes[t, d]) > 0$ do
 10. $fs = nodes[t, d]$
 11.   SIM-GEN(*depth, t, d, fs, nodes'*)
 12. return $nodes'$

SIM-GEN(*depth, t, d, fs, nodes'*)
 13. for each type t' and Δd
 14. $\Delta fs = delta\_f\_model[t', \Delta d, d, t]$
 15.   if $weight(\Delta fs) > 0$ then
 16. $d' = d + \Delta d$
 17. $fs' = $ CONVOLVE($fs, \Delta fs$)
 18. $nodes'[t', d'] = add(nodes'[t', d'], fs')$
 19. done

Figure 1: Pseudo code for the simulation procedure used to estimate the f distribution.

for each parent, p, with a child, c. This gives us the number of nodes with each f, t and d value at the next depth of the search.

Because the Δf values may be real numbers, they are stored as histograms by our Δf model. In order to add $f(p) + \Delta f(c)$, we use a procedure that Ruml [2002] called additive convolution. The convolution of two histograms ω_a and ω_b, where ω_a and ω_b are functions from values to weights, is a histogram ω_c, where $\omega_c(k) = \sum_{i \in Domain(\omega_a)} \omega_a(i) \cdot \omega_b(k - i)$. By convolving the f distribution of a set of nodes with the distribution of the change in f values between these nodes and their offspring, we get the f distribution of the offspring.

Since the maximum depth of a shortest-path search tree is typically unknown, our simulation must use a special criterion to determine when to stop. Ruml [2002] shows that, if a cost bound is known in advance then the distribution of f values at each depth can be pruned above this bound. In shortest path problems, this upper bound can be used to stop the simulation. With a consistent heuristic the f values of nodes will be non-decreasing along a path [Pearl, 1984]. Eventually the simulation will estimate fewer and fewer new nodes within the bound at each depth. When the expected number of new nodes is only a fractional value smaller than some ϵ the simulation can stop. In our experiments we use $\epsilon = 10^{-3}$. Additionally, because the d value of a node can never be negative, we can prune all nodes that would be generated with $d \leq 0$.

If the incremental model is being used to estimate the number of nodes within a cost threshold then the given threshold is used as the upper bound for pruning the histograms. If the incremental model is being used to estimate a bound that will give a specific number of node expansions, when the total distribution of f values contains a weight that is greater than or equal to the desired number of nodes, an upper bound can be found that has the desired weight to its left. This bound will only decrease as the simulation progresses and it can be used to prune the histograms in this setting.

Figure 1 shows the pseudo-code for the procedure that estimates the f distribution. The entry point is the SIM-ULATE function which has the following parameters: the cost threshold, *bound*, desired number of nodes, *desired*,

the current depth, $depth$, a histogram that contains the accumulated distribution of f values so far, $accum$, and a 2-dimensional array of histograms which stores the conditional distribution of f values among the nodes at the current depth. SIMULATE begins by simulating the expansion of the nodes at the current depth (line 1). The result of this is the conditional distribution of f values for the nodes generated as offspring at the next depth. These f values are accumulated into a histogram of all f values seen by the simulation thus far (line 2). An upper bound is determined based on which estimation mode is currently in use: estimating nodes or estimating a threshold (line 3). If enough nodes are expected in the next depth then the simulation continues recursively (lines 4–6), otherwise the accumulation of all f values is returned as the final result. Either $bound$ or $desired$ may be given as ∞ depending on whether the model is being used to estimate the number of nodes or a bound.

The SIM-EXPAND function is used to build the conditional distribution of the f values for the offspring of the nodes at the current simulation-depth. For each node type t and distance estimate d for which there exist nodes at the current depth, the SIM-GEN function is called to estimate the conditional f distribution of their offspring (lines 9–11). SIM-GEN uses the Δf distribution (line 14) to compute the frequency of f values for nodes generated from parents with the specified combination of type and distance-estimate. Because this distribution is over Δf, t and Δd, we have all of the information that is needed to construct the conditional f distribution for the offspring (lines 16–18).

Warm Starting

As an iterative deepening search progresses, some of the shallower depths become *completely expanded*: no nodes are pruned at that depth or any shallower depth. All of the children of nodes in a completely expanded depth are *completely generated*. When learning the Δf distribution on-line, our incremental model has the exact $depth$, d and f values for all of the layers that have been completely generated. We "warm start" the simulation by seeding it with the perfect information for completed layers and beginning at the first depth that has not been completely generated. This can speed up the computation of the f distribution and can increase accuracy.

0.4 Empirical Evaluation

In the following sections we show an empirical study of our new model and some of the related previous approaches. We begin by evaluating the accuracy of the incremental model when trained off-line. We then show the accuracy of the incremental model when used on-line to control an IDA* search.

0.4.1 Off-line Learning

We evaluate the quality of the predictions given by the incremental model when using off-line training by comparing the predictions of the model with the true node expansion counts. For each problem instance the optimal solution cost is used as the cost bound. Because both CDP and the incremental model estimate all of the nodes within a cost bound, the truth values are computed by running a full depth-first search of the tree bounded by the optimal solution cost. This search is equivalent to the final iteration of IDA* if the algorithm were to find the goal node after having expanded every other node that falls within the cost bound.

Estimation Accuracy

We trained both CDP and an incremental model off-line on ten billion random 15-puzzle states. We then compared the predictions given by each model to the true number of nodes within the optimal-solution-cost boundary for each of the standard 100 15-puzzle instances due to Korf [1985]. The left panel of Figure 2 shows the results of this experiment. The x axis is on a log scale; it shows the actual number of nodes within the cost bound. The y axis is also on a log scale; it shows the ratio of the estimated number of nodes to the actual number of nodes. The closer that this ratio is to one (recall that $\log_{10} 1 = 0$) the more accurate the estimation was. The median ratio for the incremental model was 1.435 and the median ratio for CDP was 1.465 on this set of instances. From the plot we can see that, on each instance, the incremental model gave estimations that were nearly equivalent to those given by CDP, the current state-of-the-art predictor for this domain.
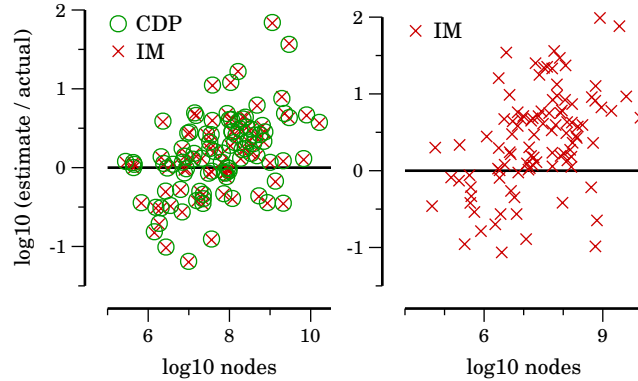
Figure 2: 15-puzzles: off-line sampling. Unit cost tiles (left) and square root cost tiles (right).
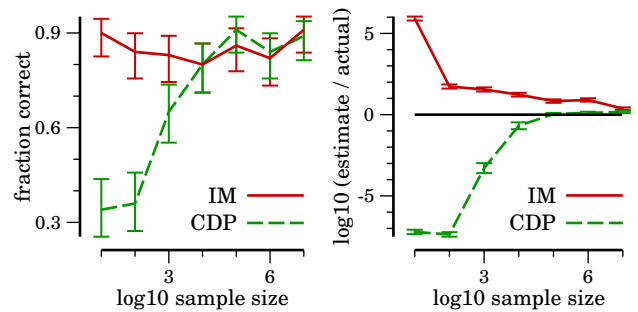

Figure 3: Pair ordering (left) and estimation factors (right).

To demonstrate our incremental model's ability to make predictions in domains with real-valued edge costs and with real-valued heuristic estimates, we created a modified version of the 15-puzzle where each move costs the square root of the tile number that is being moved. We call this problem the square root tiles puzzle. As described by Zahavi et al. [2010], CDP is not able to make predictions on this domain because of the real-valued heuristic estimates. The right panel of Figure 2 shows the estimation to actual ratio for the predictions given by the incremental model trained off-line on fifty billion random square root tiles states. The same 100 puzzle states were used. Again, both axes are on a log scale. The median estimation factor on this set of puzzles was 2.807.

Small Sample Sizes

Haslum et al. [2007] use a technique loosely based on the KRE formula to select between different heuristics for domain independent planning. When given a choice between two heuristics, we would like to select the heuristic that will expand fewer nodes. Using KRE (or CDP) to estimate node expansions requires a very large off-line sample of the heuristic distribution to achieve accurate predictions. Since the incremental model uses Δ values and a backed-off model, however, it is able to make useful predictions with very little training data. To demonstrate this, we created 100 random pairs of instances from Korf's set of 15-puzzles. We used both CDP and the incremental model to estimate the number of expansions required by each instance when given its optimal solution cost. We rated the performance of each model based on the fraction of pairs for which it was able to correctly determine the more difficult of the two instances.

The left plot in Figure 3 shows the fraction of pairs that were ordered correctly by each model for various sample sizes. Error bars represent 95% confidence intervals on the mean. We can see from this plot that the incremental model was able to achieve much higher accuracy when ordering the instances with as few as ten training samples.
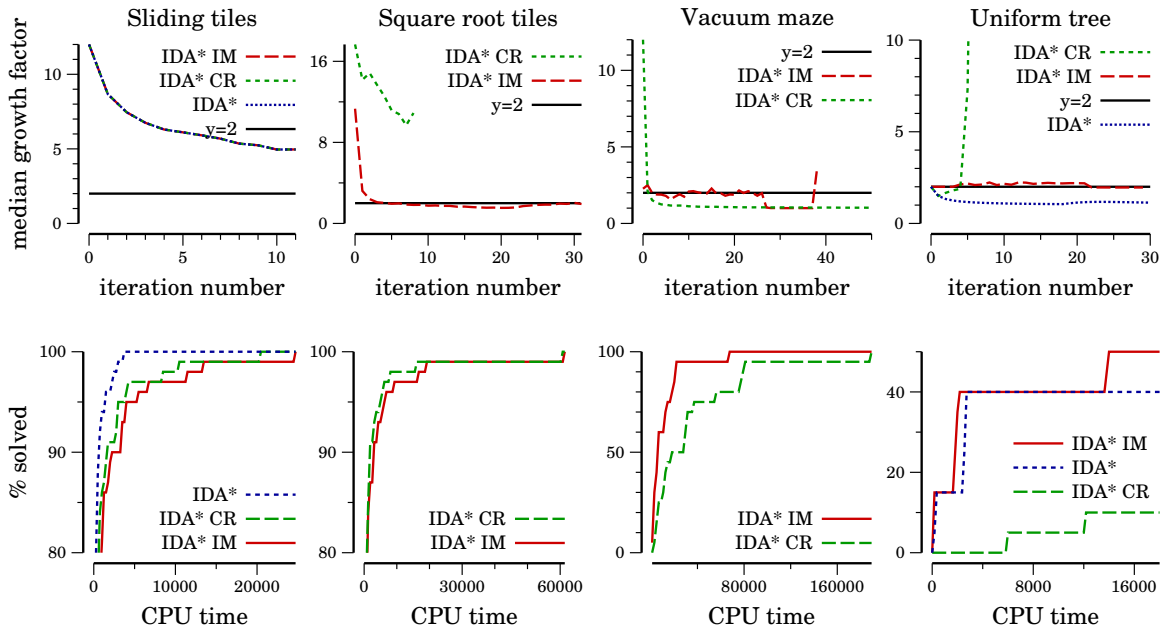
7

Figure 4: IDA*, IDA*$_{CR}$ and IDA*$_{IM}$ growth rates and number of instances solved.

With 10,000+ training samples, CDP was able to improve its accuracy becoming nearly indistinguishable from the incremental model. The right plot in this figure shows the estimation factor over all estimates made by each model. While CDP achieved higher quality estimates with 10,000+ training instances, the incremental model was able to make much more accurate predictions when trained on only 10, 100 and 1,000 samples.

0.4.2 On-line Learning

In this section, we evaluate the incremental model when trained on-line during each iteration of an IDA* search. When it comes time to set the bound for the next iteration, the incremental model is consulted to find a bound that is predicted to double the number of node expansions from that of the previous iteration. We call this algorithm IDA*$_{IM}$. As we will see, because the model is trained on the exact instance for which it will be predicting, the estimations tend to be more accurate than the off-line estimations, even with a much smaller training set. In the following subsections, we evaluate the incremental model by comparing IDA*$_{IM}$ to IDA* and IDA*$_{CR}$..

Sliding Tiles

The unit-cost sliding tiles puzzle is a domain where standard IDA* search works very well. On this set of experiments, we used the Manhattan distance heuristic. The minimum cost increase between iterations is two and it leads to a geometric increase in the number of nodes between subsequent iterations.

The top left panel of Figure 4 shows the median growth factor, the relative size of one iteration compared to the next, on the y axis, for IDA* , IDA*$_{CR}$ and IDA*$_{IM}$. Ideally, all algorithms would have a median growth factor of two. All three of the lines for the algorithms are drawn directly on top of one another in this plot. While both IDA*$_{CR}$ and IDA*$_{IM}$ attempted to double the work done by subsequent iterations, all algorithms still achieved no less than 5x growth. This is because, due to the coarse granularity of f values in this domain, no threshold can actually achieve the target growth factor. However, the median estimation factor (estimated/actual) of the incremental model over all iterations in all instances was 1.029. This is very close to the optimal estimation factor of one. So, while granularity of f values made doubling impossible, the incremental model still predicted the amount of work with great accuracy.

8

The bottom panel shows the percentage of instances solved within the time given on the x axis. Because IDA*$_{IM}$ and IDA*$_{CR}$ must use branch-and-bound on the final iteration of search they are unable to outperform IDA* in this domain.

Square Root Tiles

While IDA* works well on the classic sliding tile puzzle, a trivial modification exposes its fragility: changing the edge costs. In this section, we look at the square root cost variant of the sliding tiles where each action costs the square root of the number of the tile being moved. In these experiments, we used the Manhattan distance heuristic taking into account the square root of the numeric value on each tile. This domain has many distinct f values so when IDA* increases the bound to the smallest out-of-bound f value, it will visit a very small number of new nodes with the same f in the next iteration. We do not show the results for IDA* on this domain because it gave extremely poor performance. IDA* was unable to solve any instances with a one hour timeout and at least one instance requires more than a week to solve. Even with the branch-and-bound requirement, IDA*$_{IM}$ and IDA*$_{CR}$ easily outperform IDA* by increasing the bound more liberally between iterations.

The second column of Figure 4 presents the results for IDA*$_{IM}$ and IDA*$_{CR}$. While IDA*$_{CR}$ gave slightly better performance with respect to CPU time, its model was not able to provide very accurate predictions. The growth factor between iterations for IDA*$_{CR}$ was no smaller than eight times the size of the previous iteration when the goal was to double. The incremental model, however, was able to keep the growth factor very close to doubling. The median estimation factor was 0.871 for the incremental model which is much closer to the optimal estimation factor of 1 than when the model was trained off-line. We conjecture that the model was able to learn features that were specific to the instance for which it was predicting.

One reason why IDA*$_{CR}$ was able to achieve competitive performance in this domain is because, by increasing the bound very quickly, it was able to skip many iterations of search that IDA*$_{IM}$ performed. IDA*$_{CR}$ performed no more than 10 iterations on any instance in this set whereas IDA*$_{IM}$ performed up to 33 iterations on a single instance. Although the rapid bound increase was beneficial in the square root tiles domain, in a subsequent section, we will see that increasing the bound too quickly can severely hinder performance.

Vacuum Maze

The objective of the vacuum maze domain is to navigate a robot through a maze in order for it to vacuum up spots of dirt. In our experiments, we used 20 instances of 500x500 mazes that were built with a depth-first search. Long hallways with no branching were then collapsed into single edges with a cost equivalent to the hallway length. Each maze contained 10 pieces of dirt to be vacuumed and the goal states were all states with no dirt. The median number of states per instance was 56 million and the median optimal solution cost was 28927. The heuristic was the size of the minimum spanning tree of the locations of the dirt and vacuum. The *pathmax* procedure was used to make the f values non-decreasing along a path.

The third column of Figure 4 shows the median growth factor and number of instances solved by each algorithm for a given amount of time. Again, IDA* is not shown due to its very poor performance in this domain. Because there are many dead ends in each maze the branching factor in this domain is very close to one. The model used by IDA*$_{CR}$ gave very inaccurate predictions and the algorithm often increased the bound by too small of an increment between iterations. IDA*$_{CR}$ performed up to 386 iterations on a single instance. With the exception of a dip near iterations 28–38, the incremental model was able to accurately find a bound that doubled the amount of work between iterations. The dip in the growth factors may be attributed to histogram inaccuracy on the later iterations of the search. The median estimation factor of the incremental model was 0.968, which is very close to the perfect factor of one. Because of the poor predictions given by the IDA*$_{CR}$ model it was not able to solve instances as quickly as IDA*$_{IM}$ on this domain.

While our results demonstrate that the incremental model gave very accurate predictions in the vacuum maze domain, it should be noted that, due to the small branching factor, iterative searches are not ideal for this domain. A simple implementation of frontier A* [Korf et al., 2005] was able to solve each instance in this set in no more than 1,887 CPU seconds.

Uniform Trees

We also designed a simple synthetic domain that illustrates the brittleness of IDA*$_{CR}$. We created a set of trees with 3-way branching where each node has outgoing edges of cost of 1, 20 and 100. The goal node lies at a depth of 19 along a random path that is a combination of 1- and 20-cost edge and the heuristic used is $h = 0$ for all nodes. We have found that the model used by IDA*$_{CR}$ will often increase the bound extremely quickly due to the large 100-cost branches. Because of the extremely large searches created by IDA*$_{CR}$ we use a five hour time limit in this domain.

The right-most column Figure 4 shows the growth factors and number of instances solved in a given amount of time for IDA*$_{IM}$, IDA*$_{CR}$ and IDA*. Again, the incremental model was able to achieve very accurate predictions with a median estimation factor of 0.978. IDA*$_{IM}$ was able to solve ten of twenty instances and IDA* solved eight within the time limit. IDA*$_{IM}$ solved every instance in less time than IDA*. IDA*$_{CR}$ was unable to solve more than two instances within the time limit. This was because the model used by IDA*$_{CR}$ grew the bounds in between iterations extremely quickly, as can be seen in the growth factor plot at the upper right in Figure 4.

Although IDA* tended to have reasonable CPU time performance in this domain, its growth factors were very close to one. The only reason that IDA* achieved reasonable performance is because expansions in this synthetic tree domain required virtually no computation. This would not be observed in a domain where expansion required any reasonable computation (i.e. many of the heuristics used for domain independent planning).

0.4.3 Summary

When trained off-line the incremental model was able to make predictions that were nearly indistinguishable from CDP, the current state-of-the art, on the 15-puzzle domain. In addition to this, the incremental model was able to estimate the number of node expansions on a real-valued variant of the sliding tiles puzzle where each move costs the square root of the tile number being moved. When presented with pairs of 15-puzzle instances, the incremental model trained with 10 samples was more accurately able to predict which instance would require fewer expansions than CDP when trained with 10,000 samples.

The incremental model made very accurate predictions across all domains when trained on-line and when used to control the bounds for IDA*, our model made for a robust search. While the alternative approaches occasionally gave extremely poor performance, IDA* controlled by the incremental model achieved the best performance of the IDA* searches in the vacuum maze and uniform tree domains and was competitive with the best search algorithms for both of the sliding tiles domains.

0.5 Discussion

To ensure the termination of the simulation procedure, the incremental model assumes that the heuristic used in the search problem is consistent [Pearl, 1984]. Since the simulation terminates when it predicts that a sufficiently small fraction of nodes will be generated within the current upper bound, it is necessary that the f values consistently increase. If this did not happen, the simulation may continue forever because there is always a chance for a sufficient number of new nodes to be generated within the bound. Adjusting the simulation to handle inconsistent heuristics would be interesting future work.

In search spaces with small branching factors such as the vacuum maze domain, the backed-off model seems to have a greater impact on the accuracy of predictions than in search spaces with larger branching factor such as the sliding tiles domains. Because the branching factor in the vacuum maze domain is small, however, the simulation must extrapolate out to great depths (many of which the model has not been trained on) to accumulate the desired number of expansions. The simple backed-off model used here merely ignored depth. While this tended to give accurate predictions for the vacuum maze domain, a different model may be required for other domains.

0.6 Conclusion

In this paper, we presented a new incremental model for predicting the distribution of f values and hence the number of nodes that bounded depth-first search will visit. Our new model is comparable to state-of-the-art methods in domains

where those methods apply. The three main advantages of our new model are that it works naturally in domains with real-valued heuristic estimates it is accurate with few training samples and it can be trained on-line. We demonstrated that training the model on-line can lead to more accurate predictions. Additionally, we have shown that the incremental model can be used to control an IDA* search, giving a robust algorithm, IDA*$_{IM}$. Given the prevalence of real-valued costs in real-world problems, on-line incremental models are an important step in broadening the applicability of iterative deepening search.

Bibliography

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

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonte, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, July 2007.

Donald E. Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29(129):121–136, January 1975.

Richard E. Korf. Iterative-deepening-A*: An optimal admissible tree search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 1034–1036, 1985.

Richard E. Korf, Michael Reid, and Stefan Edelkamp. Time complexity of iterative-deepening-A*. *Artificial Intelligence*, 129:199–218, 2001.

Richard E. Korf, Weixiong Zhang, Ignacio Thayer, and Heath Hohwald. Frontier search. *Journal of the ACM*, 52(5):715–748, 2005. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/1089023.1089024.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

Wheeler Ruml. *Adaptive Tree Search*. PhD thesis, Harvard University, May 2002.

U.K Sarkar, P.P. Chakrabarti, S. Ghose, and S.C. De Sarkar. Reducing reexpansions in iterative-deepening search by controlling cutoff bounds. *Artificial Intelligence*, 50:207–221, 1991.

Jordan Thayer and Wheeler Ruml. Using distance estimates in heuristic search. In *Proceedings of ICAPS-2009*, 2009.

Nageshwara Rao Vempaty, Vipin Kumar, and Richard E. Korf. Depth-first vs best-first search. In *Proceedings of AAAI-91*, pages 434–440, 1991.

Benjamin W. Wah and Yi Shang. Comparison and evaluation of a class of IDA* algorithms. *International Journal on Artificial Intelligence Tools*, 3(4):493–523, October 1995.

Uzi Zahavi, Ariel Felner, Neil Burch, and Robert C. Holte. Predicting the performance of IDA* using conditional distributions. *Journal of Artificial Intelligence Research*, 37:41–83, 2010.