Learning Inadmissible Heuristics During Search

Jordan T. Thayer and **Austin Dionne** and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
jtd7, ajv2, ruml at cs.unh.edu

Abstract

Suboptimal search algorithms offer shorter solving times by sacrificing guaranteed solution optimality. While optimal search algorithms like A* and IDA* require admissible heuristics, suboptimal search algorithms need not constrain their guidance in this way. Previous work has explored using off-line training to transform admissible heuristics into more effective inadmissible ones. In this paper we demonstrate that this transformation can be performed on-line, during search. In addition to not requiring training instances and extensive pre-computation, an on-line approach allows the learned heuristic to be tailored to a specific problem instance. We evaluate our techniques in four different benchmark domains using both greedy best-first search and bounded suboptimal search. We find that heuristics learned on-line result in both faster search and better solutions while relying only on information readily available in any best-first search.

Introduction

Heuristic search is a widespread approach to automated planning and problem solving. If time and memory permit, we can use algorithms such as A* (Hart, Nilsson, and Raphael 1968) with an admissible (non-overestimating) heuristic to find cost-optimal solutions. Unfortunately, there are problems too large and deadlines too short for optimal search. Here, suboptimal search is a practical alternative. Such algorithms are not constrained by optimality requirements and may consult inadmissible heuristics for guidance.

As we later discuss in detail, several authors have proposed learning informed inadmissible heuristics by recording for many states the true cost-to-go, h^*, and a set of features. They then learn a function from features to an estimate of the cost-to-go, $\widehat{h}$. Such an approach makes the limiting assumption that we have access to a representative training set, or that we can generate one automatically. It further assumes that the training instances and test instances are similar enough to one another for the learning to transfer effectively between instances.

In this paper, we demonstrate that learning heuristics during search itself is a practical and effective alternative to off-line training. We present a technique for improving heuristics based on temporal difference learning (Sutton 1988) that avoids the limitations of previous approaches. Using temporal difference learning to improve heuristics has been suggested (Nilsson 1998, pages 172-175), but never, to our knowledge, actually implemented and evaluated. Further, an on-line technique allows for instance-specific learning, which we demonstrate is beneficial. We compare these heuristics to off-line learning in terms of accuracy and search guidance. We find that although heuristics learned on-line are less accurate than heuristics learned off-line, they provide better guidance. In greedy best-first search (Doran and Michie 1966), the learned heuristic produces better solutions faster than other approaches, including greedy search using a powerful pre-computed pattern database heuristic. In bounded suboptimal search, our technique improves upon the previous state of the art, optimistic search (Thayer and Ruml 2008) and eliminates the need for parameter tuning.

Previous Work

Many techniques for automatically generating heuristics have been proposed. This paper focuses on learning heuristics for general single-agent best-first search. In particular, we allow ourselves to consider heuristics that may occasionally over-estimate the true cost-to-go from a node to the goal. We do not directly consider techniques for enforcing admissibility in this paper, although we could use learned admissible heuristics as features for learning.

Samadi, Felner, and Schaeffer (2008) present a technique for combining an arbitrary number of features into a single cost-to-go estimate. In their implementation, these features are pre-computed pattern databases. They train an artificial neural network (ANN) to map these values to an estimate of the cost-to-go using h^* as the target value. For large problems, they substitute the optimal solution of a relaxed problem for h^*. The target values and even the features assume that we have access to a training set. It further assumes that all instances are similar to one another so that what we learn on the training set will transfer to new instances. These assumptions limit the applicability of the approach to domains where instances are similar enough for learning to transfer between instances.

An alternative approach is to interleave learning and solving rather than performing all of its learning before performing any search. Bramanti-Gregor and Davis (1993) propose a technique that iteratively improves a heuristic used for

solving a batch of problems. Using the current heuristic, they attempt to solve all of the problems in a set of instances within a given expansion bound using A* search. Any instances that are solved are used to train a new heuristic using linear regression against h^*. The process then repeats until all instances are solved. If all of the remaining instances are too difficult to solve using the current heuristic, it applies a weight to the current heuristic. Again, we must be able to assume that all of the instances we are trying to solve are similar enough to one another to allow learning to transfer across instances. Fink (2007) proposes a similar technique that learns an ordering over the nodes rather than modeling h^*.

Jabarri Arfaee, Zilles, and Holte (2010) show that the process of solving a set of instances can be shortened by interleaving learning with solving. Their bootstrapping method attempts to solve all of the instances in a set within a time bound using a base heuristic. It then uses information from the solved instances to train a new heuristic using an ANN. If none of the instances are solved, new easy-to-solve instances are automatically generated by random walks backwards from the goal. The process repeats with the new heuristic and the unsolved instances until all instances are solved. While bootstrapping avoids the need for a set of training instances, it still assumes that the instances are similar enough for the learning to transfer effectively.

None of these techniques learn during search itself. Learning is either performed offline before using the algorithm to solve problems, or it is done between batches of instances. Both require us to assume that we have a set of similar instances to learn from and solve. Much of the work in learning heuristics (everything we have presented with the exception of Fink, 2007) has focused on learning heuristics for permutation puzzles. Every instance of a given permutation puzzle shares the same state space and goal state, and so the learned heuristics always generalize perfectly for these domains. Compare this to domain-independent planning, where two instances may not even discuss the same type of problem. In such situations, single instances can differ substantially from one another and it may not be obvious how to determine if one instance would yield information useful for solving another.

Learning During Search

Our goal is to learn an improved cost-to-go estimate, $\widehat{h}$, during a search. First we discuss how to adapt the learning techniques used by previous approaches to work during search. Then we present a new model for improving heuristics using the error experienced during a single expansion. Finally, we compare these techniques in terms of heuristic accuracy and guidance when used in greedy best-first search and in bounded suboptimal search.

Adapting Previous Techniques Although the previously-proposed techniques do no learning during search itself, the learning algorithms they rely on can be trained on-line. Unlike the off-line or interleaved cases, when learning during search we no longer have the optimal cost-to-go from a node to use as a target value and so we must devise a way to approximate h^* on-line. We begin by noting that the optimal completion of a node p involves going through its best child, $bc(p)$. Furthermore, the optimal cost-to-go from p relates directly to $h^*(bc(p))$ and the transition cost $c(p, bc(p))$:

$$h^*(p) = c(p, bc(p)) + h^*(bc(p)) \quad (1)$$

Equation 1 suggests that we can approximate $h^*(p)$ as $h(bc(p)) + c(p, bc(p))$, and use this to learn an improved heuristic on-line. This is a slight generalization of move invariance (Christensen and Korf 1986), which holds that the entire node evaluation function should not vary between a parent and its best child. In a manner similar to temporal difference learning, we approximate the true cost-to-go based on the heuristic of the best child and then learn a function from features of the parent to this estimate.

We use the following four features: $g(n)$, the cost of arriving at n from the root, $h(n)$, an estimate of the cost-to-go from n to the goal, $depth(n)$, the number of actions between the root and n, and $d(n)$, an estimate of the distance-to-go along a cost-optimal path from n to the goal, as features for the learning algorithms. Several previous proposals have used distance estimates for search guidance (see Thayer and Ruml, 2009). In many domains, actions can have varying costs. Here, one can usually construct a distance-to-go heuristic using methods very similar to those for the cost-to-go heuristic, for example by tracking the number of actions required rather than the cost of the actions required to solve a problem. While $h(n)$ estimates the cost of the cheapest solution beneath n, $d(n)$ estimates the length *of that solution.* Similarly, in these domains we can make a distinction between the cost of arriving at a node, $g(n)$, and the number of actions along a path from the root to a node, $depth(n)$.

While both LMS and ANNs should converge in the limit of infinite training data given the right learning rate and independent examples, they will do so faster with reasonable initializations. For LMS, we set the weight on $h(n)$ to be 1 and all others to 0, so that our initial heuristic is equal to the base heuristic. For the ANN, we perform the first one hundred expansions as we would in a greedy search on $h(n)$, recording the features and target. We then batch train the network on these examples for 1000 epochs for initialization. Convergence is only guaranteed for these techniques if the training examples are sampled independently. When we are getting our training examples from the nodes expanded by a search algorithm, successive examples may be strongly related to one another. So it is an empirical question whether these techniques will work well in practice.

Observing Heuristic Error Rather than using the relationship between parent and child to approximate h^*, an alternative approach is to instead use this relationship to measure and correct for the error in the heuristic in a single expansion. With an estimate of the error across a single step, we can attempt to correct for this heuristic error by estimating the number of steps to go.

Since our estimator of search distance-to-go is also likely to be inaccurate, we begin by showing how to correct d using observed single-step error. A key feature of search distance-to-go for a particular node p is that its best child should have

a true distance-to-go $d^*(bc(p))$ of exactly one less than the true distance to go of its parent. We can define the one-step distance error ϵ_d as:

$$\epsilon_{dp} = (1 + d(bc(p))) - d(p) \quad (2)$$

We require that the best child selected for this calculation not represent the parent state of p. Thus, states with no children other than the inverse action back to their parent have no associated ϵ_d. Using Equation 2, we derive:

Theorem 1 *For any node p with a goal beneath it:*

$$d^*(p) = d(p) + \sum_{n \in p \rightsquigarrow goal} \epsilon_{dn} \quad (3)$$

where $p \rightsquigarrow goal$ is the set of nodes along the path between the state p and the goal, including p and excluding the goal.

Proof: The proof is by induction over the nodes in $p \rightsquigarrow goal$. For our base case, we show that when $bc(p)$ is the goal, Equation 3 holds:

$$\begin{aligned} d^*(p) &= d(p) + \textstyle\sum_{n \in p \rightsquigarrow goal} \epsilon_{dn} \\ &= d(p) + \epsilon_{dp} \text{ because } p \rightsquigarrow goal = \{p\} \\ &= d(p) + 1 + d(bc(p)) - d(p) \text{ by Eq. 2} \\ &= d(p) + 1 - d(p) \text{ because } d(bc(p)) = 0 \\ &= 1 \end{aligned}$$

As this is true, the base case holds.

Now for an arbitrary node p, by assuming that Equation 3 holds for $bc(p)$, we show that it holds for p as well:

$$\begin{aligned} d^*(p) = \;& 1 + d^*(bc(p)) \text{ by definition of } bc \\ = \;& 1 + d(bc(p)) + \textstyle\sum_{n \in bc(p) \rightsquigarrow goal} \epsilon_{dn} \text{ assumption} \\ = \;& d(p) + \epsilon_{dp} + \textstyle\sum_{n \in bc(p) \rightsquigarrow goal} \epsilon_{dn} \text{ by Eq. 2} \\ = \;& d(p) + \textstyle\sum_{n \in p \rightsquigarrow goal} \epsilon_{dn} \end{aligned}$$

□

We define the mean one-step error $\bar{\epsilon}_d$ along the path from p to the goal as:

$$\bar{\epsilon}_d = \frac{\sum_{n \in p \rightsquigarrow goal} \epsilon_{dn}}{d^*(p)} \quad (4)$$

Using Equations 3 and 4, we can define $d^*(p)$ in terms of $\bar{\epsilon}_d$.

$$d^*(p) = d(p) + d^*(p) \cdot \bar{\epsilon}_d \quad (5)$$

Solving Equation 5 for $d^*(p)$ yields:

$$d^*(p) = \frac{d(p)}{1 - \bar{\epsilon}_d} \quad (6)$$

Another way to think of Equation 6 is as the closed form of the following infinite geometric series that recursively accounts for error in $d(p)$:

$$\begin{aligned} d^*(p) &= d(p) + d(p) \cdot \bar{\epsilon}_d + (d(p) \cdot \bar{\epsilon}_d) \cdot \bar{\epsilon}_d + \ldots \quad (7) \\ &= d(p) \cdot \textstyle\sum_{i=1}^{\infty} (\bar{\epsilon}_d)^i \quad (8) \end{aligned}$$

We now turn to estimating cost-to-go. Analogously to $d(n)$, there is a relationship between the cost-to-go estimates of a parent and its best child. This allows us to define the single-step error in h as:

$$\epsilon_h = (h(bc(p)) + c(p, bc(p))) - h(p) \quad (9)$$

As in Equation 3, the sum of the cost-to-go heuristic and the single-step errors from a node p to the goal equals the true cost-to-go:

$$h^*(p) = h(p) + \textstyle\sum_{n \in p \rightsquigarrow goal} \epsilon_{hn} \quad (10)$$

We define the mean one-step error $\bar{\epsilon}_h$ along the path from p to the goal as:

$$\bar{\epsilon}_h = \frac{\sum_{n \in p \rightsquigarrow goal} \epsilon_{hn}}{d^*(p)} \quad (11)$$

Solving for $\sum_{n \in p \rightsquigarrow goal} \epsilon_{hn}$ and substituting into Equation 10,

$$h^*(p) = h(p) + d^*(p) \cdot \bar{\epsilon}_h \quad (12)$$

Using Equation 6 we have:

$$h^*(p) = h(p) + \frac{d(p)}{1 - \bar{\epsilon}_d} \cdot \bar{\epsilon}_h \quad (13)$$

The quantities $\bar{\epsilon_d}$ and $\bar{\epsilon_h}$ are the mean one-step errors along an optimal path to the goal. During a search, these values are unknown and must be estimated. We now discuss two techniques for estimating $\bar{\epsilon_d}$ and $\bar{\epsilon_h}$.

The *Global Error Model* assumes that the distribution of one-step errors across the entire search space is uniform and can be estimated by a global average of all observed single step errors. To do this, we must estimate which node is $bc(p)$. We assume it is the node with minimum $f(n) = g(n) + h(n)$ among all of p's children, breaking ties on $f(n)$ in favor of low $d(n)$. The search maintains a running average of the mean one-step errors observed so far. We then calculate $\widehat{h}$ using Equation 13:

$$\widehat{h}^{global}(n) = h(n) + \frac{d(n)}{1 - \bar{\epsilon}_d^{global}} \cdot \bar{\epsilon}_h^{global} \quad (14)$$

The *Path Based Error Model* calculates the mean one-step error only along the current search path, $\bar{\epsilon}_d^{path}$ and $\bar{\epsilon}_h^{path}$. This model maintains a separate average for each partial solution being considered by the search. This is done by passing the cumulative single-step error experienced by a parent node down to all of its children. We can then use the depth of the node to determine the average single-step error along this path. $\widehat{h}^{path}$ is then computed analogously to Equation 14. In either model, if our estimate of $\bar{\epsilon}_d$ is ever as large as one, we assume we have infinite distance and cost-to-go.

Comparing On-line and Off-line Learning

We now evaluate these learned heuristics. First, we consider their absolute accuracy. The different heuristics we compare are:

Baseline: A standard admissible heuristic.

ANN Off-Line: We trained a three layer neural network with three hidden nodes (the same architecture used by Jabarri Arfaee, Zilles, and Holte (2010)) and used it to compute $\widehat{h}$. We used $h^*(n)$ as the target value and used $g^*(n)$, the optimal cost of arriving at a node from the starting position, as a feature in addition to $d(n)$, $h(n)$, $depth(n)$,

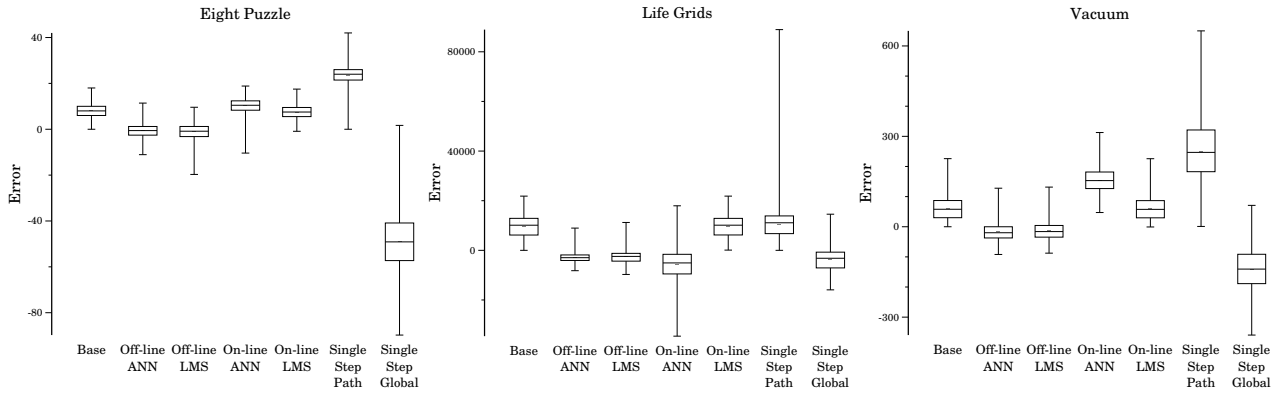

Figure 1: Absolute accuracy of learned heuristics versus the true cost-to-go

and a constant. We used a learning rate of 0.01 and at least 500,000 feature-target pairs taken from 10 random instances, with the exact number of pairs varying by domain. We trained the network for 10,000 epochs or until it converged.

ANN On-Line: We trained the same kind of neural network on-line. The features were very similar, with $g^*(n)$ replaced by $g(n)$. Since we didn't have access to the true cost-to-go, $h(bc(p)) + c(p, bc(p))$ was used. To initialize the network, we collected 100 training pairs and performed a batch regression for 1000 epochs or until the network converged. Doing the batched regression any shorter or longer had a negative impact on performance. After this initial period, we began streaming the features and target values to the learner.

LMS Off-Line: Using the same data as we did when training the off-line ANN, we optimally solved a least mean squared linear regression using $h^*(n)$ as the target value and $g^*(n)$, $d(n)$, $h(n)$, $depth(n)$ and a constant as features.

LMS On-Line: We stream examples past an LMS learner to estimate the true cost-to-go on-line. We used the same features and target values as we did in the on-line neural network. The learning rate was the same as the off-line ANN learner. We initialized the weights of the regression so that initially our heuristic was identical to the base heuristic.

Single-Step Global: We averaged the observed single-step errors in h and d globally, and used these to construct the estimated cost-to-go as in Equation 14.

Single-Step Path: We recorded an A* search of each problem using the base heuristic. We then averaged the observed single-step errors in h and d along the path to each node, and used these to construct an estimated cost-to-go.

Cost Step Global: In order to assess the impact of distance estimates on heuristic performance, we altered the single-step error model to work using only cost-to-go estimates, removing the need for distance-to-go estimates. Rather than measuring the error in $h(n)$ per-step, we measure it per-cost: $\epsilon_h = \frac{(h(bc(p))+c(p,bc(p)))-h(p)}{c(p,bc(p))}$. We compute $\widehat{h}(n)$ as $\frac{h(n)}{1-\bar{\epsilon}_h^{cost}}$, and estimate $\bar{\epsilon}_h^{cost}$ using a global average.

Cost Path Adapt: We calculate the estimated cost-to-go as in the cost step global model except that $\bar{\epsilon}_h^{cost}$ is estimated using the path based model.

For the accuracy study we considered three benchmark domains. We used relatively small instances because we needed to find optimal cost-to-go values for all states. The domains were:

Sliding Tiles Puzzles We examined 100 random 8-puzzle instances.

Grid-world Navigation Following Thayer and Ruml (2008) we tested on grid pathfinding problems using the "life" cost function. This cost function produces problems where actions have a large range of costs, short solutions are more costly than longer ones, and the search space includes several large g-value plateaus. These properties have recently seen significant interest (Benton et al. 2010). We examined 200 by 200 grids, allowing for movement in each of the cardinal directions. The grids were generated by blocking 35% of the cells at random.

Vacuum World In this domain, which follows the first state space presented in Russell and Norvig (2003)(page 34), a robot is charged with cleaning up a grid world. Movement is in the cardinal directions, and when the robot is on top of a pile of dirt, it may vacuum. Cleaning and movement have unit cost. We used 100 instances that are 200 by 200 with 5 piles of dirt and 35% of cells blocked randomly. The robot and dirt are placed randomly in unblocked cells.

All algorithms were implemented in Objective Caml and compiled to native code on 64-bit Intel Linux systems with 3.16 GHz Core2 duo processors and 8 GB of RAM.

Figure 1 shows the relative accuracy of the learned heuristics. The y-axis shows the error present in the heuristic, $h^*(n) - \widehat{h}(n)$. Each box represents the middle 50% of the data, with a line at the median, and whiskers extend to the minimum and maximum values. Cost step corrections are omitted for life cost grids because they had estimates that were infinitely inaccurate. The estimators we learned off-line are, as expected, more accurate in all three domains than the on-line estimates. We see that they have lower absolute error values, and the bulk of the error they experience is close to 0. We also see that the path based corrections have

	Eight Puzzle		Life Grids		Vacuum	
	msec	cost	msec	$\frac{cost}{1000}$	secs	$\frac{cost}{1000}$
Baseline	1.07	128	47	74	8.11	1.0
Off-line ANN	*2.07*	32	*196*	71	0.72	*1.1*
Off-line LMS	*53.06*	**22**	26	72	0.64	1.4
On-line ANN	*4.11*	*159*	*231*	*82*	1.37	*1.3*
On-line LMS	0.91	47	*56*	73	0.56	0.8
S.S. Path	**0.03**	34	**24**	**64**	**0.19**	**0.6**
Cost Path			*79*	67		
S.S. Global	*3.55*	43	45	69	0.26	0.8
Cost Global			*116*	71		

Table 1: Search using off-line versus on-line learning

a wide variance, and aren't particularly accurate. Therefore, we would expect that the off-line techniques would dominate the on-line techniques when used to guide search.

Table 1 presents the results of using these heuristics within a greedy best-first search, showing the mean CPU time required to find a solution and the mean cost of the solution. Entries worse than the base heuristic are *italicized*, and the best value in each column is **bolded**. It reveals that, surprisingly, neither of our expectations is confirmed. Both ANN heuristics occasionally perform worse than the base heuristic despite being more accurate. The single-step heuristics tend to produce better quality solutions than other approaches, and the path based heuristic is always faster. The path based corrections, which end up producing the best results when used in search, can only be performed on-line as they depend on the path from the root to a node which is determined by search order. For the eight puzzle, where it produces a worse solution than off-line LMS, we note that it is nearly two thousand times faster. For permutation puzzles like the 8-puzzle, the state space for all problems is identical and a heuristic learned on one instance of the problem transfers perfectly to new instances of that problem. Here, the off-line techniques benefit by knowing the correct answer at the beginning of search while the on-line technique must learn the improved heuristic on the fly. When we compare the cost-based and single-step models we observe that using distance estimates is beneficial.

These results are especially surprising considering how well these learning techniques have performed in previous work on learning in heuristic search. Our explanation is that previous work has mostly focused on learning heuristics for iterative deepening A*. The role, and therefore the desired properties, of the heuristic in IDA* and greedy best-first search differ substantially. IDA* uses heuristics primarily for pruning, and in many implementations only pruning, while greedy best-first search uses the heuristic solely for guidance. IDA* works by expanding all nodes within a cost boundary, and iteratively increasing this cost boundary until a solution is contained within it. In all but the final iteration, the relative ordering of nodes is of no consequence and many implementations ignore child ordering as a result. [1] Accurate cost estimates allow IDA* to prune unpromising nodes early, dramatically reducing the size of the search tree.

Performance On Larger Problems

We have presented a new technique for learning inadmissible cost-to-go estimates during a search and compared it to the learning techniques used in previous work, both off-line and on-line. We saw that the off-line techniques did not provide guide the search as effectively as the on-line techniques when used in greedy best-first search. We now focus on the relative performance of the on-line heuristics only. This allows us to examine much larger problems in which optimal search is infeasible, problems where we would consider using suboptimal search in practice. We begin by looking at the learned heuristics' performance in greedy best-first search algorithms that provide no guarantees on the quality of solution. We then examine bounded suboptimal search algorithms which guarantee returned solutions are within a user-specified factor of optimal. The instances for this study are larger than those in the accuracy study:

Sliding Tiles Puzzles We used the 100 instances of the 15-puzzle presented by Korf (1985).

Grid-world Navigation We show results over 20 instances of 2000 by 1200 grids.

Vacuum World We used 100 instances that are 500 cells tall by 500 cells wide with twenty piles of dirt.

Dynamic Robot Navigation This domains follows that used by Likhachev, Gordon, and Thrun (2003). The goal is to find the fastest path from the starting location of the robot to some goal location and heading, taking momentum into account. We perform this search in worlds that are 500 by 500 cells in size. We scatter 75 lines, up to 70 cells in length, with random orientations across the domain and present results averaged over 100 instance. (This domain was absent from the study of heuristic accuracy because enumerating h^* for all states is difficult even for grids of modest size.)

Greedy Best-First Search

In our study of greedy best-first search, we include an additional technique. While we can use multiple heuristics as features for a machine learning algorithm in greedy best-first search, Röger and Helmert (2010) showed that very good performance can be attained by alternating through a set of heuristics. Alternation maintains several copies of the open list, each sorted on a different heuristic, and selects nodes from each in turn. We alternate between the cost-to-go estimate $h(n)$ and the distance-to-go estimate $d(n)$ for domains where these differ.

Table 2 shows the performance of these heuristics in greedy best-first search. For domains with unit cost actions, the single-step and cost-based models are identical because $h(n) = d(n)$ and $c(p, bc(p)) = 1$, and therefore the results are omitted from the table. Similarly, alternating is omitted for unit cost domains. We see that path based corrections produce better solutions than all other techniques across all four search domains. They are the fastest technique in 3 domains. The next best technique, on-line LMS regression, is the fastest algorithm for the 15 puzzle, but is

[1] The current state-of-the-art is to run IDA* with multiple action orderings in parallel (Valenzano et al. 2010),which takes advantage of child ordering, but doesn't use the heuristic to order the children.

	15 Puzzle		Life Grids		Vacuum World		Dynamic Robots	
	msecs	cost	msecs	$\frac{cost}{1e6}$	secs	$\frac{cost}{1000}$	msecs	cost
Baseline	30	395	170	3.0	22.9	12.6	65	537
Alternating			114	3.0			*110*	*643*
On-line ANN	*69*	*300*	*571*	*5.1*	*34.1*	*17.7*	*7921*	*6829*
On-line LMS	**15**	174	*219*	2.9	20.5	6.1	22	63
Single-Step Path	16	**107**	**70**	**2.8**	**6.4**	**3.4**	**11**	48
Cost Path Adapt			*2573*	*3.2*			12	**47**
Single-Step Global	36	159	745	3.0	10.4	4.5	2563	*871*
Cost Global Adapt			*5487*	*9.8*			*unknown*	*unknown*

Table 2: Greedy search in life, sliding tiles, dynamic robot navigation, and vacuum world

several times slower than the path based corrections for all other domains we examined. The cost-based global correction failed to solve all instances within a five minute cutoff in the dynamic robot domain. For the LMS heuristic and global single-step models, the weights and errors differed substantially between instances, sometimes by orders of magnitude on all domains save the 15-puzzle, where the numbers were, as one would expect, quite similar across instances.

One might wonder if the accuracy of the base heuristic had any influence on the performance of the single-step model. In Table 3 we compare our best learning method with a modern pattern database for the 15-puzzle, the 7-8 PDB (Korf and Felner 2002). We see that using the pattern database heuristic substantially improves the performance of greedy search when compared to the Manhattan distance heuristic that we used as a baseline. However, our path based heuristic finds solutions faster than the PDB heuristic and those solutions are not much worse on average, and on some instances our heuristic can find better solutions. This is accomplished without the benefit of the pre-computation needed to construct the pattern databases. If we add our path based correction to the PDB heuristic, it further improves performance, finding better solutions faster than ether the PDB alone or path based corrections on top of Manhattan distance. From this we conclude that single-step correction can improve the performance of even strong heuristics.

We have not yet demonstrated that the single-step model learns anything instance-specific. Table 4 shows the performance of our global single-step model when learned on-line and when using a previously learned model with learning turned off, either from the same instance or a random instance. We use the global model because it is clear how to transfer the information learned from one instance to another. In this table, we present results in terms of nodes generated in order to focus on search guidance and ignore the overhead of learning. As we saw before, we see that the heuristic corrected on-line produces better results than the base heuristic without learning. Additionally, both static off-line models are better than the on-line correction. This shows us that the improved performance is not due to some fortuitous synergy of the learning process with search. If it were, the on-line model would out-perform both static models. We're actually learning something meaningful. Finally, the heuristic learned from the same instance performs better than one from a random instance. This indicates that the technique is learning, on-line, an instance-specific model and that instance specific information is beneficial. Although Table 4 only reports results in vacuum world, similar results were obtained for all domains except for the 15-puzzle, where all instances share the same search space.

Bounded Suboptimal Search

Occasionally we need guarantees on the quality of a solution returned by a search. In this setting we use a bounded suboptimal search. These algorithms are guaranteed to return a solution within a fixed factor of optimal. We refer to this bound as w in homage to weighted A* (Pohl 1973).

Algorithms like weighted A* rely on the admissibility of their base heuristic to obtain their suboptimality bound. However, there are algorithms that can use arbitrary heuristics for at least a portion of their search. Optimistic search (Thayer and Ruml 2008), the current state of the art in bounded suboptimal search, is one such algorithm. As proposed, optimistic search works by running weighted A* with a weight higher than the desired suboptimality bound, and then after that search finds a goal, some additional nodes are expanded using an admissible heuristic to prove the solution found was within the desired suboptimality bound.

However, the first phase of optimistic search can use any inadmissible heuristic and still retain its guarantees of bounded suboptimality as long as an admissible heuristic is available for the second phase. We can replace the weighted admissible heuristic from the first phase of optimistic search with $\widehat{f}(n) = g(n) + w \cdot \widehat{h}(n)$ where $\widehat{h}$ is any learned heuristic. We call this modification of optimistic search *skeptical search*. It is skeptical in that it does not place absolute trust in the base heuristic.

Figure 2 compares weighted A*, several parameter settings for the original optimistic search and skeptical search. The x-axis of the plot is the suboptimality bound, the desired guarantee on solution quality. The y-axis represents the amount of time needed to solve problems for the given bound. We show only the results for skeptical with path-based correction, the solid line in all four plots, as this learned heuristic performed best.

Although many of the algorithms are often difficult to distinguish in detail, what is clear is that skeptical search is always at least competitive with the current state-of-the-art, optimistic search, for any of the optimism settings exam-

Heuristic	msecs	cost
M.D.	30	395
7-8 PDB	*44*	101
M.D. S.S. Path	16	107
7-8 PDB S.S. Path	**10**	**65**

Table 3: Greedy on the 15 puzzle

Learning	$\frac{nodes}{1000}$	$\frac{cost}{1000}$
None	217	12.57
On-line	41	4.49
Random Instance	19	3.89
Same Instance	**16**	**3.85**

Table 4: Global single-step on vacuum

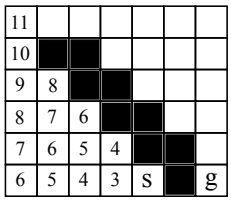

Figure 3: single-step corrections can fail

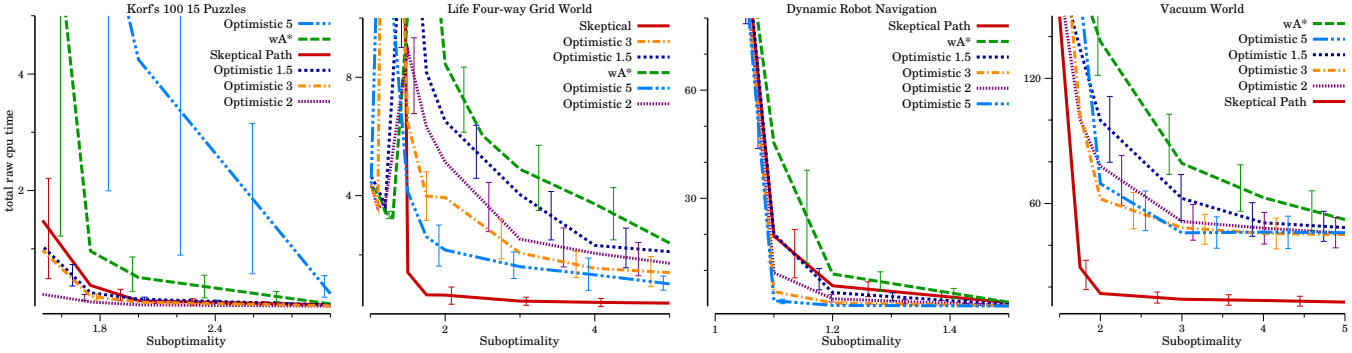

Figure 2: Skeptical search versus various parameter settings for optimistic search

ined. We can't tell the difference between the two algorithms for the fifteen puzzle or for dynamic robot navigation because the confidence intervals on the time required overlap. For life cost grids, skeptical search is up to thirteen times faster than optimistic search, and up to 20 times faster in vacuum world. Overall, skeptical search provides state-of-the-art performance for bounded suboptimal heuristic search.

In addition to out-performing optimistic search, skeptical search removes the need for parameter tuning. Optimistic search requires an ad hoc parameter, an optimism factor, in addition to the desired suboptimality bound. The optimism factor tells optimistic search how aggressive it should be in pursuing the initial solution. If it is set too high, the incumbent solution will be outside of the desired bound, and the performance of the algorithm will suffer. If it is set too low, finding the initial solution will take too long, pulling down overall algorithm performance. Skeptical search has only the desired suboptimality bound as a parameter. Rather than requiring an explicit optimism factor, skeptical search constructs $\widehat{h}$ using its experience during problem solving.

Discussion

Our evaluation revealed that the heuristics learned on-line result in finding better solutions faster than heuristics learned off-line, and is faster than the base heuristics as well. Additionally, an on-line approach allows us to learn instance-specific heuristic corrections, something that is not practical with previous approaches. We can now effectively use learned heuristics for a larger range of problems than was previously possible because we don't have to worry about poor generalization or constructing representative training sets. The best feature of this learning technique is that it only relies on information that any best-first search already has available. The single-step corrections we presented appear to be quite general, working with poorly informed heuristics and well informed heuristics, as we saw in the sliding tile puzzle (Table 3). It also works in domains with inconsistent base heuristics, such as vacuum worlds. Admissibility of the underlying heuristic is not a requirement. The new learning technique resulted in state-of-the-art performance for both greedy and bounded suboptimal search.

The technique does have limitations. Figure 3 shows a problem where single-step corrections perform poorly. The start state is marked with 's', and the goal is marked with a 'g'. The grid is 4-connected. The numbers in the cells show the value of $d(n)$, the distance-to-go estimate. Each move that could take us out of the beginning section into the half of the grid with the goal is a move that will increase the estimated distance-to-go. For any search to escape the beginning of the problem, it must experience a single-step error of 2 repeatedly. When we reach the state with a distance-to-go of 11, the estimated single-step error will be 2 for both the global and path-based methods. Until the estimate is lowered below one by expanding many additional nodes with no single-step error, $\widehat{h}(n) = \widehat{d}(n) = \infty$. In our implementation of the search algorithms presented here, we break ties in favor of low $g(n)$. Thus, if the cost-to-go estimates become infinitely large, we will perform a uniform cost search due to tie-breaking. If we had just been doing a greedy search on the base heuristic in this example, we would go straight to the goal from the state marked 11 rather than laboriously performing uniform-cost-search.

For the single-step corrections we present to work well, they require that the estimates of $\bar{\epsilon}_d$ to be reasonable. If

this estimate ever gets too large, we will estimate that the distance-to-go and cost-to-go are both infinite. Single-step corrections can fail if $d(bc(p)) \geq d(p)$, especially if this happens early, when our estimates of the average error can be easily influenced. In domains with either large heuristic plateaus in the estimated distance-to-go or in domains where the search must frequently go in the 'wrong' direction as defined by the heuristic, we will over-estimate the value of $\bar{\epsilon}_d$, resulting in poor performance. This does not appear to be a characteristic of any of the benchmark domains we have examined.

For the heuristic corrections that we learn on-line, with the exception of path-based heuristic correction, the cost-to-go estimate for all nodes changes with every expansion. This means that the order of nodes being considered by the search could change with every expansion. We found that it is better to ignore this problem than to update cost-to-go estimates and re-sort the open list.

An additional limitation of the approach is that, unlike alternating search or regression-based approaches, it cannot currently make use of multiple cost-to-go estimates. They would need to be combined into a single heuristic before being used in our new model. Similarly, although we find an improved estimate of the number of search steps-to-go, we never directly use it to determine search order. We suspect both are likely beneficial.

While many previously-proposed search algorithms can use inadmissible heuristics for guidance, few of them have actually been used that way in the literature. Typically a weighted admissible heuristic is used instead. A weighted admissible heuristic will often be inadmissible, however it isn't any more informed than the base heuristic because it doesn't have any new information. A greedy search on the weighted heuristic would be no different than a greedy search on the base heuristic. Our work has shown that by using inadmissible heuristics that rely on additional sources of information such as distance-to-go estimates and measurements of heuristic error, we can improve the performance of suboptimal heuristic search.

Conclusion

We presented and evaluated a new technique for learning inadmissible heuristics on-line during a search. Our correction works by observing the error in the cost-to-go heuristic in a single transition between a parent node and its best child. We can use this observed error along with an estimate of the remaining number of search steps to construct an improved heuristic on-line. This on-line learning obviates the need for a training set or pre-computation and allows us to learn instance-specific heuristics. We showed that our improved heuristic works well in both greedy best-first search and bounded suboptimal search across a variety of domains. In greedy best-first search, it leads to solving problems faster and to finding higher quality solutions. In bounded suboptimal search, it provided state of the art performance while allowing us to remove a parameter present in optimistic search, safeguarding us against poor parameter selection. The simplicity of the learning technique combined with the wide availability of the features it uses suggest that single-step heuristic error correction can be applied in a wide variety of application domains as an easy way to boost the performance of a variety of suboptimal search algorithms.

Acknowledgments

We gratefully acknowledge support from NSF (grant IIS-0812141) and the DARPA CSSG program (grant N10AP20029).

References

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmueller, R.; and Kambhampati, S. 2010. G-value plateuas: A challenge for planning. In *Proceedings of ICAPS 2010*.

Bramanti-Gregor, A., and Davis, H. 1993. The statistical learning of accurate heuristics. In *IJCAI-93*, 1079–1085.

Christensen, J., and Korf, R. 1986. A unified theory of heuristic evaluation functions and its application to learning. In *AAAI-86*, 148–152.

Doran, J. E., and Michie, D. 1966. Experiments with the graph traverser program. In *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 235–259.

Fink, M. 2007. Online learning of search heuristics. In *AISTATS*.

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.

Jabarri Arfaee, S.; Zilles, S.; and Holte, R. 2010. Bootstrap learning of heuristic functions. In *SoCS-10*.

Korf, R., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134:9–22.

Korf, R. E. 1985. Iterative-deepening-A*: An optimal admissible tree search. In *IJCAI-85*, 1034–1036.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *NIPS-03*.

Nilsson, N. J. 1998. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, Inc. 172–175.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computation issues in heuristic problem solving. In *Proceedings of IJCAI-73*, 12–17.

Röger, G., and Helmert, M. 2010. The more the merrier: Combining heuristic estimators for satisficing planning. In *ICAPS-10*, 246–249.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey: Prentice Hall, second edition.

Samadi, M.; Felner, A.; and Schaeffer, J. 2008. Learning from multiple heuristics. In *Proceedings of AAAI-08*.

Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine Learning* 3:9–44.

Thayer, J. T., and Ruml, W. 2008. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *ICAPS-08*.

Thayer, J., and Ruml, W. 2009. Using distance estimates in heuristic search. In *Proceedings of ICAPS-09*.

Valenzano, R.; Sturtevant, N.; Schaeffer, J.; Buro, K.; and Kishimoto, A. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *ICAPS-10*.