

Finding Acceptable Solutions Faster Using  
Inadmissible Information  
Computer Science Technical Report 10-01

Jordan Thayer      Wheeler Ruml

April 7, 2010

## **Abstract**

Bounded suboptimal search algorithms attempt to find a solution quickly while guaranteeing that its cost does not exceed optimal by more than a desired factor. Typically these algorithms use a single admissible heuristic evaluation function for both guiding search and bounding solution quality. In this paper, we present a new approach to bounded suboptimal search that separates these roles, consulting inadmissible information to determine search order and using admissible information to guarantee quality. Unlike previous proposals, it explicitly estimates expected solution cost and search distance in an attempt to reach a solution within the suboptimality bound as quickly as possible. We show how to construct these estimates during search using information that is readily available yet often overlooked. In an empirical evaluation across six diverse benchmark domains, the new techniques have better overall performance than previous approaches, including weighted A\* and optimistic search.

## 0.1 Introduction

When resources are plentiful or an optimal solution is required, A\* search [1] using a consistent heuristic will find an optimal solution as fast as any equally informed search [2]. However, in many practical settings we must accept suboptimal solutions in order to reduce the time or memory required for the search. In this paper we focus on bounded suboptimal search, algorithms that find solutions whose cost is within a specified factor  $w$  of optimal. We say such algorithms are  $w$ -admissible.

As we discuss below, all of the previously proposed algorithms that we are aware of fail to directly address the problem of finding solutions within a bound as quickly as possible. In this paper we introduce explicit estimation search (EES), a bounded suboptimal search algorithm that uses unbiased cost and distance estimates rather than lower bounds to find solutions of bounded quality as quickly as possible. We introduce skeptical search, a simplification of EES that has reduced overhead and better performance in some domains. We then illustrate how to construct the unbiased estimates that EES and skeptical search rely on. We conduct a comprehensive empirical analysis of these and other bounded suboptimal algorithms on grid pathfinding, the sliding tile puzzle, the pancake puzzle, dynamic robot pathfinding, the TSP, and vacuum planning problems. We find that explicit estimation search has the strongest overall performance, surpassing the current state of the art including weighted A\* [3] and optimistic search [4].

## 0.2 Previous Approaches

We now describe three previously proposed algorithms that demonstrate the most common approaches to bounded suboptimal search, and their strengths and flaws.

**Weighted A\*** is a simple and effective bounded suboptimal search. In weighted A\* the traditional node evaluation function of A\* is modified to place additional emphasis on the heuristic evaluation function;  $f(n) = g(n) + h(n)$  becomes  $f'(n) = g(n) + w \cdot h(n)$ . The weight,  $w$ , increases the importance of  $h$  (estimated cost of reaching a goal from  $n$ ) relative to  $g$  (cost of reaching  $n$ ), making the search greedier.

Placing additional emphasis on the heuristic rewards nodes that have low  $h$ . The higher the  $h$  value, the more unattractive weighting makes a node look. Skewing the evaluation function in this way encourages progress towards areas with low  $h$ . If the heuristic is well informed and admissible, nodes near a goal must have relatively low  $h$  values. We would hope to find solutions quickly by forcing the search into such areas of the space. If all actions have the same cost, then we directly optimize the length of a solution as well. If there are actions with varying cost, we may fail to find the nearest  $w$ -admissible solution, which can increase solving time.

Weighted searches may expand many nodes that, strictly speaking, they do not have to expand in order to find a  $w$ -admissible solution. We use weighted A\* as an example. Weighted A\* will expand all nodes that it generated with  $f'(n) < g(s)$  where  $s$  is the suboptimal goal returned by the search. This is a result of the search order. However, if we obtain  $s$  by some method other than best first search on  $f'$ , we would only need to expand those nodes with  $w \cdot f(n) < g(s)$  in order to show that  $s$

represents a  $w$ -admissible solution. It is obvious that fewer nodes satisfy the inequality  $w \cdot f(n) = w \cdot (g(n) + h(n)) < g(s)$  than do  $f'(n) = g(n) + w \cdot h(n) < g(s)$ , as the former scales both  $g$  and  $h$  while the latter only increases the size of  $h$ . Weighted searches will always be at risk of expanding nodes beyond those needed to prove the bound.

**Optimistic search** attempts to improve upon weighted A\* by addressing the problem of potentially expanding more nodes than needed in order to prove a suboptimality bound. In optimistic search, weighted A\* is run with a weight higher than the desired suboptimality bound. This weight is determined by scaling the desired bound up by an optimism factor which must also be specified by the user. Thayer and Ruml use a value of 2 [4]. The node with the smallest  $f$ -value of all open nodes, called  $f_{min}$ , serves as a lower bound on the cost of an optimal solution to the problem [5]. The quality bound of the incumbent solution can be calculated by dividing its cost by  $f(f_{min})$ . Optimistic search expands  $f_{min}$  until this dynamic bound is at least as tight as the desired bound. This stage of the search is called the cleanup phase.

Although the cleanup phase of optimistic search expands only those nodes necessary to prove the bound, the initial phase during which the solution is found is flawed. Optimistic search works by increasing the weight used in the initial search, and while higher weights generally lead to faster weighted A\* searches, this is not universally true. It has been shown that single-minded focus on cost to go can lead to poor search performance in domains where the lower bound requirements on  $h$  prevent it from effectively discriminating between nodes [6]. Here, an over-reliance on  $h$  can actually lead to decreased performance, potentially causing optimistic search to perform worse than weighted A\*. These problems can be avoided by incorporating multiple sources of information.

Additionally we only suspect that the initial solution will be within the user's desired bound based on the past performance of weighted A\*. There is nothing about the search order that suggests that this solution will be within the bound, and if we set the optimism factor too high, it frequently won't be (see [4] for details regarding this case). This is especially problematic for new domains where the performance of weighted A\* is unknown.

## 0.2.1 Distance-Oriented Searches

Not all bounded suboptimal search algorithms operate by focusing solely on a cost to go heuristic.  $A_\epsilon^*$  [7] maintains two orderings on its nodes, open and focal. In the first ordering nodes are sorted in order of  $f$ . The node at the front of this list is  $f_{min}$ , and it is used to form the focal list. The focal list contains nodes whose  $f$  value is within a factor  $w$  of  $f(f_{min})$  and is sorted on  $d$ , an estimate of the distance from a node to the goal. That is, focal contains those nodes that can currently be shown to be within the  $w$ -admissibility bound, sorted in order of their estimated goal proximity,  $d$ . The node from the front of focal,  $d_{min}$ , is chosen for expansion.

$A_\epsilon^*$  explicitly chases the nearest solution that can be shown to be within the bound for all problems, something previous approaches did not attempt. Other algorithms focus solely on the cost of a solution. This will map directly to the nearest solution for domains with unit cost actions, for example the sliding tile puzzle, but this will not

work for domains where actions have varying cost. Unfortunately  $A_\epsilon^*$  is still relying on lower bounds to guide the search. While  $d$  need not be admissible, the focal list is formed by consulting  $f_{min}$ , a lower bound on the cost of an optimal solution. The lower bound is guiding  $A_\epsilon^*$  to a large extent, and it frequently contradicts the order suggested by  $d$ . When using an admissible  $h$  function, the  $f$  values of nodes typically increase as one descends from the root while  $d$  tends to decrease. Nodes with low  $d$  will often have relatively high  $f$  values and  $d_{min}$  is often the node with the highest  $f$  in focal. Children of  $d_{min}$  are thus not likely to be included in focal. This causes a constant emptying and refilling of the focal list, which results in terrible performance for  $A_\epsilon^*$  in many domains [8]. We now turn to explicit estimation search, which addresses these flaws in previous approaches to bounded suboptimal search.

### 0.3 Explicit Estimation Search

The objective of bounded suboptimal search is to find solutions within the given suboptimality bound as quickly as possible. This suggests the following search order: For all nodes that appear to be on a path to a  $w$ -admissible solution, expand the node that seems closest to a goal. Explicit estimation search (EES) follows this principle as directly as possible while strictly guaranteeing bounded suboptimality. In addition to  $g(n)$ ,  $h(n)$ , and  $d(n)$ , EES uses  $\hat{h}$ , a potentially inadmissible but more accurate version of  $h$ , and  $\hat{d}$ , a more informed version of  $d$ . They can be supplied by the user, and we will discuss one way in which they may be constructed during the search below. Using these values, we construct two cost functions,  $f$  and  $\hat{f}$ .  $f$  is the traditional cost function of A\* and provides a lower bound on the cost of an optimal solution through a node.  $\hat{f}(n) = g(n) + \hat{h}(n)$  attempts to be an unbiased estimate of the cost of the best solution through  $n$ .

EES selects one of the following nodes to expand:

$$\begin{aligned} f_{min} &= \operatorname{argmin}_{n \in open} f(n) \\ best_{\hat{f}} &= \operatorname{argmin}_{n \in open} \hat{f}(n) \\ best_{\hat{d}} &= \operatorname{argmin}_{n \in open \wedge \hat{f}(n) \leq w \cdot \hat{f}(best_{\hat{f}})} \hat{d}(n) \end{aligned}$$

As before,  $f_{min}$  is the node with the lowest  $f$  value among all unexpanded nodes, and acts as a lower bound on the cost of an optimal solution, a value we will need to prove that the final solution lies within the desired suboptimality bound.  $best_{\hat{f}}$  is the node with the lowest predicted solution cost. Rather than being chosen from all open nodes as  $f_{min}$  and  $best_{\hat{f}}$  are,  $best_{\hat{d}}$  is selected from a restricted set of nodes. Specifically, it must be a member of the set of nodes whose  $\hat{f}$  value is within a factor  $w$  of  $\hat{f}(best_{\hat{f}})$ .  $\hat{f}(best_{\hat{f}})$  represents our best estimate of the cost of an optimal solution, so  $best_{\hat{d}}$  is selected from those nodes we suspect lead to a  $w$ -admissible solution. Of these,  $best_{\hat{d}}$

is the node nearest to a goal. At every expansion, EES chooses from among these three nodes using the rules:

$$selectNode = \begin{cases} best_{\hat{d}} & \text{if } \hat{f}(best_{\hat{d}}) \leq w \cdot f(f_{min}) \\ best_{\hat{f}} & \text{if } \hat{f}(best_{\hat{f}}) \leq w \cdot f(f_{min}) \\ f_{min} & \text{otherwise} \end{cases}$$

We first consider  $best_{\hat{d}}$ , as pursuing nearer goals should lead to a goal fastest.  $best_{\hat{d}}$  is returned if the solution it will lead to can be shown to be within the suboptimality bound, seen in the first rule of  $selectNode$ . Specifically, we only return  $best_{\hat{d}}$  if the estimated cost of a solution through it,  $\hat{f}(best_{\hat{d}})$ , is within a factor  $w$  of a lower bound of the cost of an optimal solution,  $f(f_{min})$ . If  $best_{\hat{d}}$  is unsuitable,  $best_{\hat{f}}$  is examined. We suspect that this node lies along a path to an optimal solution, as it has the smallest  $\hat{f}$  value. Pursuing paths of the highest quality is desirable, as proving that they are within the suboptimality bound is easier. Expanding this node may enlarge the set that  $best_{\hat{d}}$  is selected from because it potentially replaces  $best_{\hat{f}}$  with a node whose  $\hat{f}$  is larger. We only expand  $best_{\hat{f}}$  if it can be shown to be within the bound. If neither  $best_{\hat{f}}$  nor  $best_{\hat{d}}$  were within the bound, we return  $f_{min}$ . Expanding it could raise our lower bound by enlarging  $f(f_{min})$ , allowing us to consider  $best_{\hat{d}}$  or  $best_{\hat{f}}$  in the next expansion.

**Theorem 1** *if  $\hat{h}(n) \geq h(n)$  and  $g(opt)$  is the cost of an optimal solution, then for every node  $n$  expanded by EES, it is true that  $f(n) \leq w \cdot g(opt)$*

**Proof:**  $selectNode$  will always return one of  $best_{\hat{d}}$ ,  $best_{\hat{f}}$  or  $f_{min}$ . No matter what node we select we have

$f(n) \leq w \cdot f(f_{min})$ . This is trivial for the third case in  $selectNode$ , where  $f_{min}$  is chosen. For the other two cases, we must rely on the fact that  $h(n) \leq \hat{h}(n)$ . So long as this is true, when  $best_{\hat{d}}$  is selected:

$$\begin{aligned} \hat{f}(best_{\hat{d}}) &\leq w \cdot f(f_{min}) \\ g(best_{\hat{d}}) + \hat{h}(best_{\hat{d}}) &\leq w \cdot f(f_{min}) \\ g(best_{\hat{d}}) + h(best_{\hat{d}}) &\leq w \cdot f(f_{min}) \\ f(best_{\hat{d}}) &\leq w \cdot f(f_{min}) \\ f(best_{\hat{d}}) &\leq w \cdot g(opt) \end{aligned}$$

Whenever  $best_{\hat{d}}$  is selected for expansion, its  $f$  value is within a bounded factor  $w$  of the cost of an optimal solution. If  $best_{\hat{d}}$  is a solution,  $h(best_{\hat{d}}) = 0$  and  $f(best_{\hat{d}}) = g(best_{\hat{d}})$ , in which case the cost of the solution represented by  $best_{\hat{d}}$  is within a bounded factor  $w$  of the cost of an optimal solution. The  $best_{\hat{f}}$  case is identical.  $\square$

### 0.3.1 Behavior

Explicit estimation search fixes the problems we have just highlighted with previous approaches. It takes both the distance to go estimate as well as the cost to go estimate

into consideration when deciding what node to expand next. This allows it to prefer finding solutions as quickly as possible in all domains instead of just unit cost domains. It does this by operating in the same spirit as  $A_\epsilon^*$ , using both an open and a focal list. We will see that the orderings used for EES do not conflict with one another, allowing EES to perform well across all suboptimality bounds. Like optimistic search, EES uses a cleanup list to avoid unnecessary expansions when proving a suboptimality bound. Rather than doing all of these expansions after having found a solution, EES interleaves cleanup expansions with those directed towards finding a solution. As a result, it can never run into the problem of having an incumbent solution that falls outside of the desired suboptimality bound. Further, EES relies on unbiased estimates of the cost to go, rather than the past performance of an underlying algorithm, when determining if a node is likely to be within the bound. Not only is this a more principled approach, it neatly avoids the problems optimistic search has when running on novel domains.

Much like  $A_\epsilon^*$  or weighted A\*, EES will become greedier as the bound is loosened. Like  $A_\epsilon^*$  EES becomes a greedy search on  $\hat{d}$ , unlike than weighted A\* which focuses almost exclusively on cost estimates. The greedy behavior of weighted A\* will always be tempered by  $f$ 's inclusion of  $g$ . Searches like  $A_\epsilon^*$  and EES, on the other hand, can become even greedier. When  $w$  becomes sufficiently large, they perform a purely greedy search on their estimates of distance-to-go.

For tighter suboptimality bounds, EES behaves much more like optimistic search. It will frequently expand  $best_{\hat{f}}$ , pursuing what appears to be the highest quality solution, and  $f_{min}$  in order to prove the suboptimality bound. This is much like a variation on optimistic search that interleaves its initial and cleanup phases. EES has the significant advantage of never being able to find a solution outside of the desired suboptimality bound, as per the previous theorem. At a suboptimality bound of 1, it expands nodes in A\* order, breaking ties in favor of low  $\hat{d}$ .

### 0.3.2 Implementation

EES is structured like a classic best-first search. We insert the initial node into *open*, and at each step, we select the next node for expansion using *selectNode*. To efficiently access  $best_{\hat{f}}$ ,  $best_{\hat{d}}$ , and  $f_{min}$ , EES maintains three queues, the *open* list, *focal* list, and *cleanup* list respectively. *open* and *focal* are strongly related to one another. The *open* list contains all generated but unexpanded nodes sorted on  $\hat{f}(n)$ . The node at the front of the *open* list is  $best_{\hat{f}}$ . *focal* is a prefix of the *open* list ordered on  $\hat{d}$ . *focal* contains all of those nodes that have  $\hat{f}$  values within a factor  $w$  of  $\hat{f}(best_{\hat{f}})$  which estimates the cost of the optimal solution. The node at the front of *focal* is  $best_{\hat{d}}$ . *cleanup* contains all nodes from *open*, but  $f(n)$  instead of  $\hat{f}(n)$ . The node at the front of *cleanup* is  $f_{min}$ , and it provides a lower bound on the cost of an optimal solution.

Efficiently performing EES requires fast access to  $f_{min}$ ,  $best_{\hat{f}}$ , and  $best_{\hat{d}}$ . We need to be able to select one of these nodes, remove it from all relevant data structures, and reinsert its children efficiently. To accomplish this we implement *cleanup* as a binary heap, *open* as a red-black tree, and *focal* as a heap synchronized with a left prefix of *open*. This lets us perform all insertions and removals in logarithmic time (except for transferring nodes from *open* onto *focal*).

### 0.3.3 Simplified approaches

With three queues to manage, EES has significant overhead, so we also introduce two simplified techniques for incorporating inadmissible heuristic information.

**Clamping** [9] is a simple technique for using an inadmissible cost function  $\hat{h}$  to guide search while maintaining bounded suboptimality. We merely restrict  $\hat{f}$  to never be larger than  $w \cdot f$  as in  $f_{ca}(n) = \min((w \cdot f), \hat{f}(n))$ .  $f_{ca}$  has several drawbacks: it cannot incorporate search distance estimates effectively, it fails to become greedy at high weights, and if  $\hat{h}$  is much greater than  $h$ , the search will devolve into A\*, but without proving optimality. In this case,  $w \cdot f(n)$  is frequently less than  $\hat{f}(n)$ , causing the nodes to be sorted and expanded in the same order they would be in a A\* search.

**Skeptical Search** attempts to scale gracefully from a search on  $f$  at low suboptimality bounds to a search on  $\hat{h}$  for large bounds. This results in a search on  $\hat{f}'(n) = g(n) + w \cdot \hat{h}(n)$ , where  $w$  is the desired suboptimality bound. We call this Skeptical Search because it learns to be mistrusting of the overly optimistic heuristic  $h$ . It is likely to produce very high quality solutions if  $\hat{h}$  is accurate, and we expect it to decrease in solving time as  $w$  increases, like weighted A\*.

The main advantage of this approach is in reduced overhead. It never needs to calculate  $\hat{d}$ , and it is never concerned with  $best_{\hat{d}}$ . As a result, it maintains fewer sorted lists and performs fewer heuristic calculations and thus has significantly less overhead. Search on  $\hat{f}'$  is obviously not going to be guaranteed to return  $w$ -admissible solutions, so we implement skeptical search as a more principled version of optimistic search.

Skeptical search is preferable to optimistic search because the user need only supply one parameter, the desired suboptimality bound. The optimism factor required by optimistic search could be charitably characterized as correcting  $h$  into  $\hat{h}$ , which skeptical search does explicitly. The major drawback of skeptical search is that it has poor performance for low weights. Unlike EES or optimistic search, skeptical search will not converge to A\* performance at a bound of 1, it will instead expand nodes in  $\hat{f}$  order, which will only mimic A\* when  $\hat{h} = h^*$ , the true cost to go.

## 0.4 Deriving Inadmissible Heuristics

EES uses the unbiased estimations  $\hat{h}$  and  $\hat{d}$  in addition to lower bounds, but where do they come from? We may either create them based on insight into the domain or derive them automatically. One idea that has been mentioned in passing by several authors[10] [11], but never (to our knowledge) actually pursued, is to learn an inadmissible heuristic function during search using temporal differences.

If  $h$  were perfect, then the  $f$  value of a parent node would be the same as the lowest  $f$  among its children. However, admissible heuristics usually underestimate the cost to the goal and  $f$  tends to rise in value along any path. The rise in value from a parent to its best child is a measurement of the error in the heuristic. As nodes are expanded during search, one can calculate the average one step error,  $e_h$ , in  $h$ . One can estimate a corrected value as  $\hat{h}(n) = h(n) + e_h \cdot d(n)$ . One can also measure one step error in



$d$ , resulting in  $\widehat{d}(n) = d(n) + e_d \cdot d(n)$ . We can use  $\widehat{d}$  to improve the estimate of  $\widehat{h}$ , as in  $\widehat{h} = h(n) + e_h \widehat{d}(n)$ .

When estimating  $e_h$  on-line its value is changing over time and so are the  $\widehat{h}$  and  $\widehat{f}$  of every node. Resorting the open list after every expansion is too costly. Two possible approximations are 1) to re-sort the open list only occasionally, perhaps at a geometrically growing interval and 2) to not re-sort at all and have each node keep forever the  $\widehat{f}$  value computed when it was generated. Not resorting the open list has no affect on the  $w$ -admissibility of the solution for any of the proposed algorithms and produces slightly better results.

## 0.5 Empirical Evaluation

In addition to the algorithms discussed above, we implemented and tested against all other bounded suboptimal searches in the literature, namely  $A_\epsilon$  [12], AlphaA\* [13], and revised dynamically weighted A\* [8]. We now describe each algorithm briefly.

$A_\epsilon$  is very similar in form to  $A_\epsilon^*$ . It starts by expanding  $n$  such that  $\text{argmin}_n d(n) : f(n) \leq w \cdot f(f_{min})$ , exactly the node that  $A_\epsilon^*$  expands every iteration.  $A_\epsilon$  commits to this node, not unlike the way realtime algorithms commit to a node, and repeatedly follows the best child from each expansion until a goal is found or until the best child could not be shown to be  $w$ -admissible. If the best child would be outside of the bound,  $A_\epsilon$  must either abandon its commitment, expanding  $n$  such that  $\text{argmin}_n d(n) : f(n) \leq w \cdot f(f_{min})$ , or it may instead choose to persevere, expanding  $f_{min}$  until the desired child is within the bound, continuing along this path once the lower bound on optimal solution cost has been sufficiently raised. Many different persevering strategies exist, we choose to press onward if it appears the solution is at least 90% complete.

AlphaA\* also uses the idea of maintaining multiple orderings over the nodes. Nodes are stored using one of two cost functions. Either they are stored with their  $f$  values or their  $f'$  value. If a node's parent has an  $h$  value greater than the last node expanded, it is stored with  $f_{wA^*}$ , otherwise it is stored with  $f_{A^*}$ . If a node appears to be closer to the solution than its parent, it receives a favorable node evaluation. However, if the heuristic appears to be misleading us, or if the node truly represents a step in the wrong direction, it receives the worse value.

Revised dynamically weighted A\* attempts to extend this approach by using two measures of goal proximity:  $h$ , and  $d$ , the estimated number of steps to the goal. As nodes progress towards the goal, the weight is steadily decreased. Revised dynamically weighted A\* sorts nodes in order of  $f_{rdwA^*} = g(n) + \max(1, w \frac{d(n)}{d(\text{root})})h(n)$ . Revised dynamically weighted A\* is identical to or significantly better than dynamically weighted A\* [14] in all domains.

All algorithms were implemented in Objective Caml and compiled to native binaries on 64-bit Intel Linux systems. All the algorithms were sampled at the following suboptimality bounds: 1, 1.005, 1.001, 1.01, 1.05, 1.1, 1.15, 1.2, 1.3, 1.5, 1.75, 2, 2.5, 3, and 5. We show 95% confidence intervals averaged over all instances for each domain.

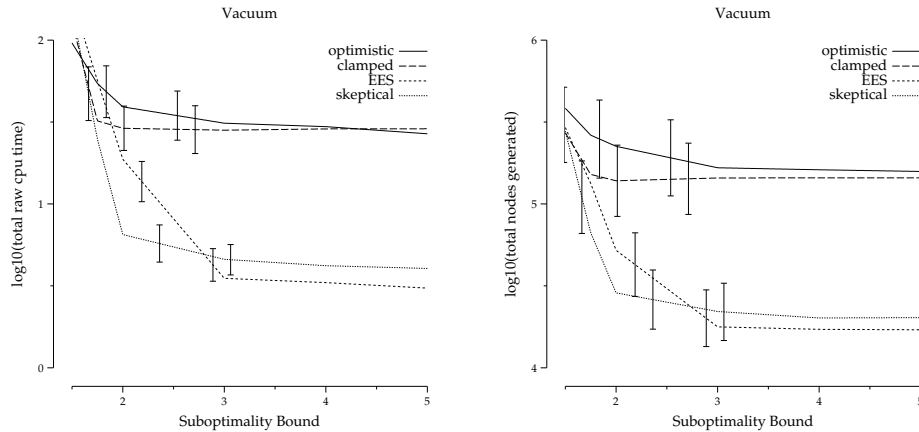


Figure 1: Vacuum Planning

### 0.5.1 Vacuum World

In this domain, inspired by the first state space depicted in [15], 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 remove it. Cleaning and movement have unit cost. We use the minimum spanning tree of the robot and dirt locations plus the number of piles of dirt as an admissible  $h$ . Search distance is estimated by finding the length of a greedy solution on a board with no obstacles. We used instances that are 500 cells tall by 500 cells wide, each cell having a 35% probability of being blocked. We place twenty piles of dirt randomly in unblocked cells. The robot's start location is selected randomly from the unblocked cells. We averaged over 100 instances.

The results are presented in Figure 1. The suboptimality bound of the algorithms is on the x-axis, with a bound of 1 requiring an optimal solution, and a bound of 3 means that the solution returned has cost within a factor of 3 of the optimal solution. The y-axis shows either the total amount of nodes generated, or the time consumed by the algorithm. The y-axis is always presented in log scale for readability. We display the best four algorithms per domain for the same reason. The legend is sorted from worst to best in terms of nodes generated or cpu time consumed. All algorithms were run with a five minute time limit. Not all algorithms solve all problems across all suboptimality bounds. If an algorithm failed to solve a problem, it was charged for the full amount of time and for as many nodes as it could generate within that time limit. As such, the plots represent lower bounds on the actual performance of the algorithms. All of the plots follow this layout. There is a very clear separation between algorithms that take advantage of learning and distance heuristics and those based around weighted  $A^*$ , as we see by the large separation between EES and Skeptical Search and Optimistic Search.  $A^*$  performed almost as well as these algorithms for high bounds, but failed to solve many instances at weights lower than 2. Alpha\* performed very poorly. Clamped search performs poorly at high weights because it cannot become sufficiently

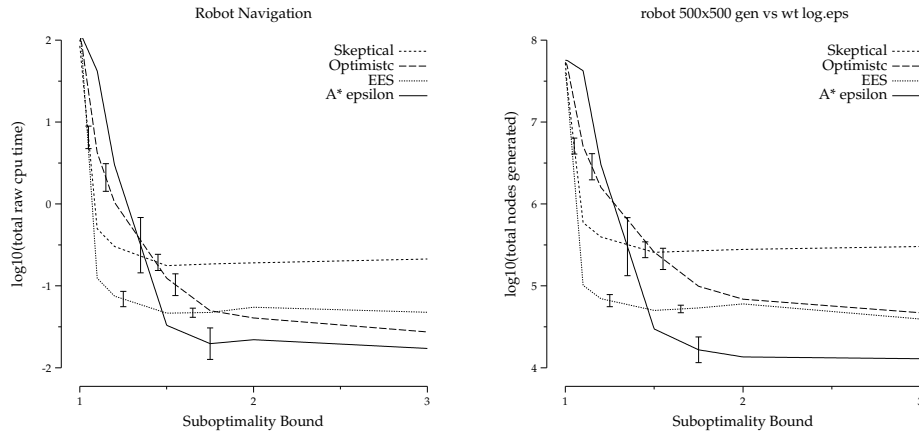


Figure 2: Dynamic Robot Navigation

greedy.

### 0.5.2 Dynamic Robot Navigation

This domain follows that used by [16]. The goal is to find the fastest path from the starting location of the robot to some goal location and heading. We perform this search in worlds that are 500 by 500 cells in size. To add challenge to these problems, we scatter 75 lines with random orientations across the domain. Each line is up to 70 cells in length. Both  $h$  and  $d$  are found by searching the problem without dynamics exhaustively from the root. For  $d$ , we return the shortest distance between that point and the goal, assuming the robot can stop, start, and turn on a dime.  $h$  is similar to  $d$ , but we assume that the robot is constantly moving at maximum velocity to obtain a lower bound on the time to arrive at the goal from the current state.

Results for the dynamic robot navigation problem are shown in Figure 2. We see again in the timing results that the three fastest algorithms all exploit inadmissible heuristics to improve their search order. Skeptical search and EES are outperforming all other algorithms until a weight of about 1.5, where  $A_\epsilon^*$  starts performing very well. At this point, it performs a greedy search on  $d$ , which is very well informed for this domain. Before weights of 1.5, approaches using inadmissible information are orders of magnitude faster. We suspect EES would be just as fast if we did not correct  $d$  into  $\hat{d}$  on this domain.

### 0.5.3 Macro Sliding Tile Puzzle

We examined algorithm performance on the macro 15-puzzle [17], using Korf’s original instances [18]. The macro 15-puzzle is identical to the physical implementation of the 15-puzzle in the sense that moving multiple tiles towards the blank requires the same amount of effort as sliding the blank only one tile in a give direction. Since the

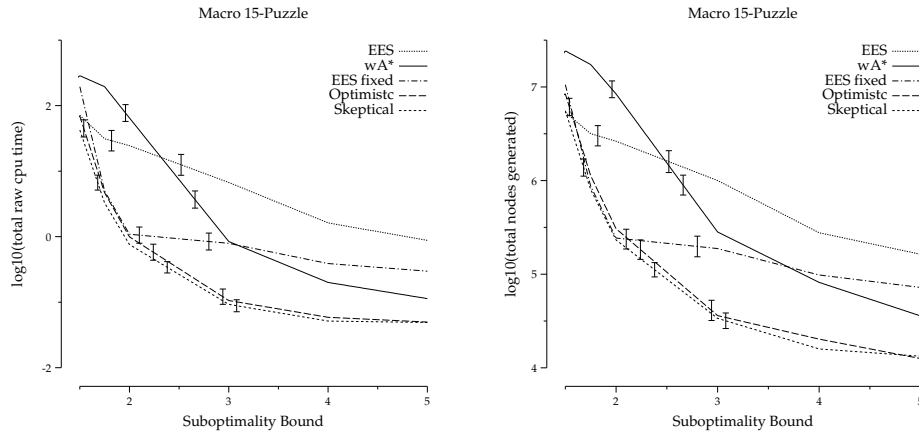


Figure 3: Macro 15-Puzzle

blank may move up to three spaces at once, we use manhattan distance divided by three for both  $h$  and  $d$ . This is the only domain where the learned inadmissible heuristic was outperformed by the hand crafted one, in this case we used the undivided manhattan distance.

Results for macro tiles are presented in Figure 3. EES using the hand crafted heuristic is called “EES Fixed” in the figures. Skeptical search and Optimistic Search are the two best algorithms by both metrics in this domain. Optimistic does so well in this domain because it is accidentally correcting for heuristic error. While an optimism of 2 is typically just a very good rule of thumb, for macro tile puzzles it is almost exactly covering for the introduction of the macro moves. This explains why Skeptical search and optimistic search are nearly identical here.

### 0.5.4 Grid Pathfinding

Following [4] we tested on grid pathfinding problems using the “life” cost function. We show results over 20 instances of 2000 by 1200 grids, allowing for movement in each of the cardinal directions. The grids were generated by blocking 35% of the cells at random. The start is in the lower left of the grid, with the goal appearing in the lower right.

Figure 4 shows the results for grid pathfinding problems. We immediately notice that all of the algorithms, save weighted A\* with duplicate dropping, are having extreme difficulty for weights less than 1.5. In this region, the other algorithms are reopening a large number of states that they have previously visited with a suboptimal  $g$  value. Weighted A\* may avoid reopening these nodes because the heuristic for this domain is consistent [19].

As the weight increases, we see that EES gains an edge in terms of the number of nodes generated during the search. When we examine the CPU plot, EES is no longer clearly superior to wA\* dd. In this domain node expansion is extremely inexpensive.

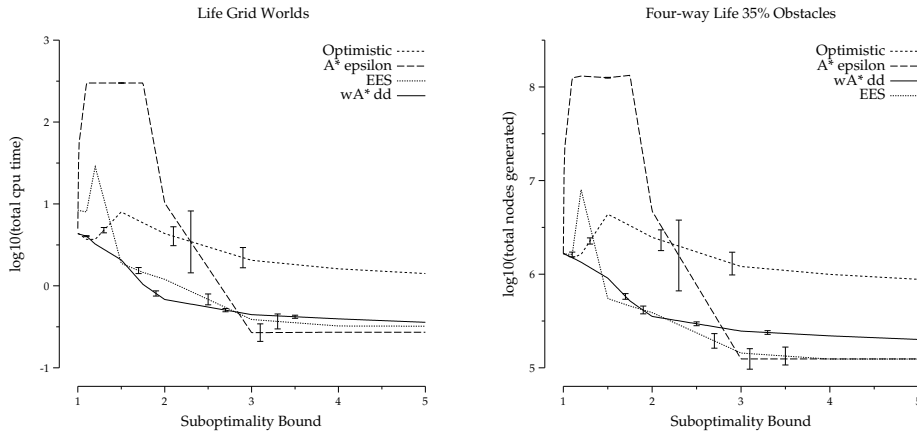


Figure 4: Grid World Pathfinding

Thus the increased overhead of EES is more noticeable. When we take away the large difference between  $h$  and  $d$  by running on boards with 8-way movement (not shown) the results change slightly. Duplicates are no longer a concern and the slight edge that distance information provided at high suboptimality bounds goes away. Weighted A\* dd becomes the search of choice, although EES is still very competitive in both CPU time and nodes generated.

### 0.5.5 Traveling Salesman

Following Pearl and Kim[7], we test on a straightforward encoding of the traveling salesman problem. Each node represents a partial tour with each action representing the choice of which city to visit next. We used the minimum spanning tree heuristic for  $h$  and the number of cities remaining for  $d$ . We test on two types of instances, 100 cities placed either uniformly in a unit square or with distance chosen uniformly at random between 0.75 and 1.25 (called “hard” by Pearl & Kim). All problems are symmetric. Results are averaged over 40 instances.

Figure 5 shows the results in unit square versions of the traveling salesman problem. There was little difference between the algorithms on the “hard” Pearl & Kim instances so we omit them.

### 0.5.6 Pancake Puzzle

We also performed experiments on the 10 Heavy Pancake puzzle. Like the original puzzle [17], the goal is to arrange a permutation of numbers from 1 to  $N$  into an ascending sequence. Successor states are generated by selecting a prefix of the current sequence and reversing that prefix. Each state in the problem has  $N-1$  successors. Each pancake has a weight, equal to its index. The cost of a move is the sum of the pancakes being moved. We liken this to rearranging a stack of free-weights from a careless pile

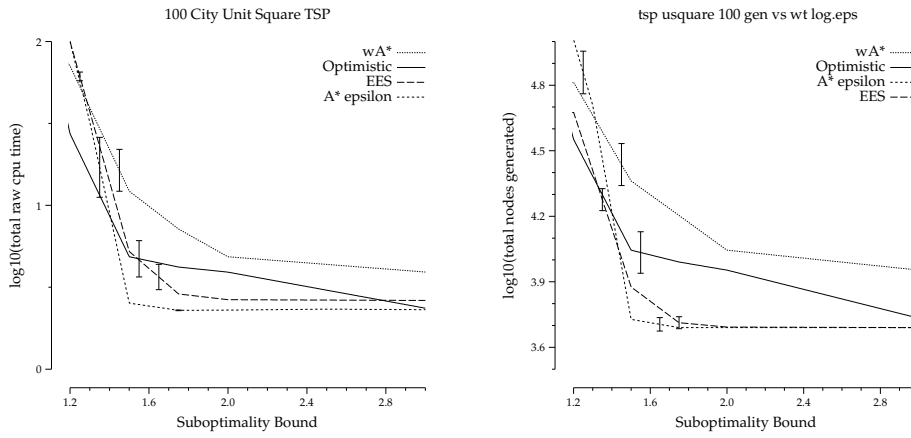


Figure 5: TSP

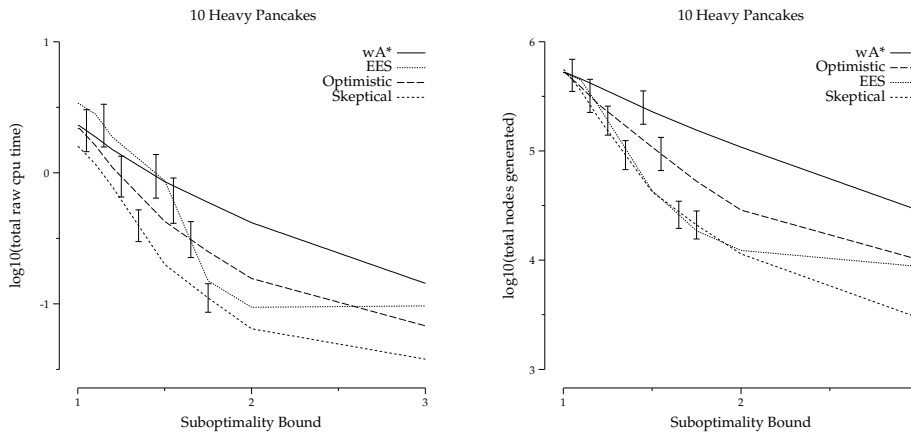


Figure 6: Pancake Puzzle

<b>Nodes Generated</b>	1st	2nd	3rd	4th	> 4th
EES	2	3	1	0	0
Optimistic	0	1	2	3	0
Skeptical	3	0	0	1	2
$A_\epsilon^*$	1	1	1	0	3
wA*	1	0	0	2	3
Alpha*	0	0	0	0	6
$A_\epsilon$	0	0	0	0	6
Clamped	0	0	1	0	5
$A_\epsilon^*$ hhat	0	0	0	0	6
rdwA*	0	0	0	0	6
<b>CPU Time</b>	1st	2nd	3rd	4th	> 4th
EES	0	3	3	0	0
Optimistic	0	2	1	3	0
Skeptical	3	0	0	1	2
$A_\epsilon^*$	2	1	1	0	3
wA*	1	0	0	2	3
$A_\epsilon$	0	0	0	0	6
Alpha*	0	0	0	0	6
Clamped	0	0	1	0	5
$A_\epsilon^*$ hhat	0	0	0	0	6
rdwA*	0	0	0	0	6

Table 1: Rank across all benchmarks

into a neat stack, with the largest weight at the base and the smallest weight at the top. We used pattern database for both  $h$  and  $d$ . In Figure 6 we see that EES and Skeptical search are the best search in terms of nodes generated and solving time.

### 0.5.7 Summary of Results

Table 0.5.7 summarizes the number of times each algorithm achieved each ranking across all six benchmark domains when ranked by the number of nodes generated and the amount of CPU time consumed. Of all the algorithms we tested, those incorporating inadmissible information such as  $d$  and  $\hat{h}$  were always among the top four performers in every domain we looked at. EES and Optimistic Search are the only algorithms that appear in every plot. This is because they makes very informed decisions about expansion order. Both are separating the goal of finding solutions and that of proving their quality, and EES is deciding whether or not to expand a node based on unbiased estimates of solution cost and length. The price for this stability is overhead, resulting in slightly longer run times than that of Skeptical search. Optimistic search is also a strong performer, primarily because weighted A\* never performs exceptionally poorly on many of these domains, however it is slower than both EES and Skeptical Search.

Table 0.5.7 provides a different perspective on the results in aggregate than Ta-

CPU	1.5	1.75	2.	3.	4.	5.
optimistic	1.6	1.5	1.6	2.1	2.4	2.1
wA*	4.1	3.4	2.8	3.7	3.4	2.4
skeptical	2.6	4.7	4.9	5.1	11.4	13.9
$A^*_\epsilon$	50.4	44.8	28.5	1.8	1.1	0.6
Clamped	8.3	10.1	11.6	67.0	85.6	85.8
Alpha*	126.6	140.1	181.6	282.2	309.3	315.0
rdwA*	374.1	316.9	245.1	101.0	84.8	128.1
$A_\epsilon$	911.4	857.7	683.2	624.9	597.4	614.3
Generated	1.5	1.75	2.	3.	4.	5.
optimistic	3.1	2.4	2.5	3.3	3.4	3.2
wA*	6.6	5.5	4.5	5.5	5.0	4.0
skeptical	3.2	3.0	2.8	3.8	11.5	15.4
$A^*_\epsilon$	58.4	44.9	17.9	1.8	1.1	0.8
Clamped	6.8	5.6	7.1	76.5	95.9	97.4
Alpha*	1.2	1.5	2.2	4.4	5.6	5.7
rdwA*	187.3	171.7	150.2	86.3	78.1	163.1
$A_\epsilon$	1514	1415	1122	994	918	979

Table 2: CPU time and nodes generated relative to EES

ble 0.5.7. Here, rather than measuring the relative performance on single domains, we look at the relative performance of the algorithms at different suboptimality bounds across all domains. We present the number of nodes generated and CPU time consumed relative to that used by EES, averaged over all domains. Such figures give us a quantitative sense of the relative performance of the algorithms. We see that, with a single exception of  $A^*_\epsilon$  run with a bound of 5, explicit estimation search consumes less time and generates fewer nodes than any other approach.

This may seem surprising, since in our previous evaluations, EES was infrequently the best algorithm; it was often in second place. Although EES frequently comes in second place, it is never extremely outperformed by the other algorithms. The other bounded suboptimal algorithms, on the other hand, manage to fail spectacularly in some domains. This drags their average performance down. Thus explicit estimation search is the superior choice when running on a set of problems with diverse properties or when running on a problem that we know very little about.

## 0.6 Discussion

There are two ideas driving these new search algorithms: inadmissible sources of information are readily available and helpful, and of all solutions within the bound, we prefer the one we can find the fastest. Combining pattern databases [20], using linear programming to correct for heuristic error [21], or using offline regression to find better heuristic models are all possible approaches to constructing  $\hat{h}$  and  $\hat{d}$ , but they are fundamentally different from our current approach in that they all require training data,



where as we gather all of our experience during the search.

The results presented in the evaluation clearly show that there are some limitations to EES despite its generally good performance. If the cost of expanding nodes is very low, a better expansion order may not expedite search because of explicit estimation's overhead. In domains with consistent heuristics and many duplicates, weighted A\* can avoid re-expanding the duplicate states, and has remarkably good performance as a result. If  $\hat{h}$  is very inaccurate, EES and skeptical search can perform poorly, as we saw in the macro-tiles domain.

## 0.7 Conclusions

We showed how additional information can be exploited in bounded suboptimal search, resulting in better search orders and shorter solving times across a wide variety of benchmark domains. Unlike previous approaches, explicit estimation search (EES) converts the stated goal of bounded suboptimal search rather directly into an expansion order by taking advantage of inadmissible cost to go and search distance estimators that attempt to be unbiased rather than lower bounds. We also showed how to generate these inadmissible estimates during search without requiring training data. Skeptical search makes use of only the cost-to-go estimates to provide a search order almost as principled as that of EES with less overhead.

## 0.8 Acknowledgments

We gratefully acknowledge support from both the NSF grant IIS-0812141 and the DARPA CSSG program (grant HR0011-09-1-0021).

# Bibliography

- [1] 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.
- [2] Rina Dechter and Judea Pearl. The optimality of A\*. In Laveen Kanal and Vipin Kumar, editors, *Search in Artificial Intelligence*, pages 166–199. Springer-Verlag, 1988.
- [3] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1:193–204, 1970.
- [4] Jordan T. Thayer and Wheeler Ruml. Faster than weighted A\*: An optimistic approach to bounded suboptimal search. In *Proceedings of ICAPS-2008*, Fall 2008.
- [5] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.
- [6] J. Benton, Kartik Talamadupula, Patrik Eyerich, Roburt Mattmueller, and Subbaro Kambhampati. G-value plateaus: A challenge for planning. In *Proceedings of ICAPS 2010*, 2010.
- [7] Judea Pearl and Jin H. Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4):391–399, July 1982.
- [8] Jordan Thayer and Wheeler Ruml. Using distance estimates in heuristic search. In *Proceedings of ICAPS-2009*, 2009.
- [9] Jordan T. Thayer, Wheeler Ruml, and Ephrat Bitton. Fast and loose in bounded suboptimal heuristic search. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, Summer 2008.
- [10] Donald Michie and Robert Ross. Experiments with the adaptive graph traverser. In *Machine Intelligence 5*, pages 301–318, 1969.
- [11] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, San Francisco, CA, 1998.

- [12] M. Ghallab and D.G. Allard.  $A_\epsilon$ : An efficient near admissible heuristic search algorithm. In *Proceedings 8th IJCAI*, 1983.
- [13] B. Reese. AlphaA\*: An  $\epsilon$ -admissible heuristic search algorithm. Unpublished, retrieved from <http://home1.stofanet.dk/breese/papers.html>, 1999.
- [14] Ira Pohl. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computation issues in heuristic problem solving. In *Proceedings of IJCAI-73*, pages 12–17, 1973.
- [15] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Third edition, 2010.
- [16] Maxim Likhachev, Geoff Gordon, and Sebastian Thrun. ARA\*: Anytime A\* with provable bounds on sub-optimality. In *Proceedings of NIPS 16*, 2004.
- [17] R. Holte, J. Grajkowski, and B. Tanner. Hierarchical heuristic search revisited. In *Symposium on Abstract Reformulation and Approximation*, pages 121–133, 2005.
- [18] Richard E. Korf. Iterative-deepening-A\*: An optimal admissible tree search. In *Proceedings of IJCAI-85*, pages 1034–1036, 1985.
- [19] Maxim Likhachev, Geoff Gordon, and Sebastian Thrun. ARA\*: Formal analysis. Technical Report CMU-CS-03-148, Carnegie Mellon University School of Computer Science, July 2003.
- [20] Mehdi Samadi, Ariel Felner, and Jonathan Schaeffer. Learning from multiple heuristics. In *Proceedings of AAAI-08*, Summer 2008.
- [21] Marek Petrik and Shlomo Zilberstein. Learning heuristic functions through approximate linear programming. In *Proceedings of ICAPS-2008*, Fall 2008.