Better Parameter-Free Anytime Search by Minimizing Time Between Solutions

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

J. Benton
Arizona State University
Department of Computer Science
j.benton@asu.edu

Malte Helmert
Universität Basel
Departement Mathematik und Informatik
4056 Basel, Switzerland
malte.helmert@unibas.ch

Abstract

This paper presents a new anytime search algorithm, anytime explicit estimation search (AEES). AEES is an anytime search algorithm which attempts to minimize the time between improvements to its incumbent solution by taking advantage of the differences between solution cost and length. We provide an argument that minimizing the time between solutions is the right thing to do for an anytime search algorithm and show that when actions have differing costs, many state-of-the-art search algorithms, including the search strategy of LAMA11 and anytime nonparametric A*, do not minimize the time between solutions. An empirical evaluation on seven domains shows that AEES often has both the shortest time between incumbent solutions and the best solution in hand for a wide variety of cutoffs.

Introduction

Anytime search strategies have gained an ever-wider popularity for problem solving. They arguably appeal to users because they gradually return solutions with monotonically improving quality, allowing one to arbitrarily stop computation after outputting a "good enough" satisficing (i.e., potentially suboptimal) solution. This contrasts with typical solving strategies where we must choose between a suboptimal solver to find a single, quickly found but potentially costly solution or an optimal solver for a slowly (or never) found optimal solution. Despite the increased use of anytime algorithms in, for example, the International Planning Competitions (IPCs), most work in the area defines anytime algorithms without considering what might constitute a 'reasonable behavior' from the user's perspective. The main consideration is how to best balance the time it takes to improve solutions over time. Ideally, we would like to find the first solution as quickly as possible, even if it is costly. Then, we would like to find the next fastest-to-find solution that has better quality than the first, and so forth. This definition of *"ideal" performance* matches suggestions by other investigators, who argue that solutions that are only a few steps away from discovery should never be ignored in favor of solutions that may exist further away but take much longer to find (Cushing, Benton, and Kambhampati 2011).

In this paper, we propose Anytime Explicit Estimation Search, which attempts to optimize this performance objective. AEES is an anytime variant of the Explicit Estimation Search (EES) algorithm (Thayer and Ruml 2011), which was designed for bounded suboptimal search. It combines inadmissible estimates of cost-to-go and distance-to-go to determine which node is likely to lead to a better solution quickest, and expands that node, directly optimizing the previously introduced notion of *ideal performance*. To avoid needing a parameter schedule (of suboptimality bounds for EES), AEES uses an admissible cost-to-go estimate to compute a dynamic bound on solution quality.

Though the design of AEES focuses on minimizing time between improving solutions, our empirical evaluation reveals that it also finds good solutions quickly and is competitive with modern anytime search strategies such as those used by LAMA11 (Richter, Westphal, and Helmert 2011; Richter and Westphal 2010) and anytime nonparametric A* (ANA*) (van den Berg et al. 2011). In contrast to these approaches, AEES always considers estimates of solution length in addition to estimates of solution cost. Hence, it often performs best in domains with non-uniform costs. Its performance is consistently good, unlike the LAMA11 search strategy, anytime repairing A* (Likhachev, Gordon, and Thrun 2003), or ANA*. In 5 out of the 7 benchmark domains we investigate, AEES both improves the incumbent solution the quickest and has the best solution in hand for a wide variety of potential cutoffs.

Related Work

Although anytime algorithms can take any form, they tend to be based on best-first heuristic search algorithms and can loosely be classified into one of three frameworks: the continued search framework, the repairing search framework, and the restarting framework. A best-first heuristic search algorithm is one that maintains a list of all generated, but not yet expanded, search nodes in sorted order so that it can easily determine which node to expand next (i.e., the "best" node). For example, Weighted A* (Pohl 1973) is a best-first search on $f'(n) = g(n) + w \cdot h(n)$.

Continued Search runs a best-first search until the open list, the set of all nodes generated but not yet expanded, has been exhausted (Hansen and Zhou 2007). The largest drawback of the continued framework is that it does not reconsider its

parameters as new incumbents are found. The search strategy used to find the first solution is the same as that used to find the last solution, and this can be inefficient (Thayer and Ruml 2010).

Repairing Search addresses this shortcoming of the continued search framework, and it's designed to work well in domains with many duplicates (Likhachev, Gordon, and Thrun 2003). When a repairing search encounters a solution, parameters used by the underlying search are set to new values. This allows these algorithms to pursue solutions of higher quality as the quality of the incumbent improves. Repairing search algorithms also delay the reexpansion of duplicate nodes (states previously encountered by a more expensive path) until the next iteration (until the next goal is found). This generally improves performance.

Restarting Search is one of the simplest frameworks for anytime search. Restarting weighted A* (RwA*) (Richter, Thayer, and Ruml 2010), the search strategy at the center of the award winning LAMA planner (Richter and Westphal 2008; Richter, Westphal, and Helmert 2011), is an example of an algorithm in the restarting framework. RwA* runs a sequence of weighted A* searches, each with a parameter picked from a hand-crafted parameter schedule. The subsequent searches do not throw away all of the effort of previous searches; they may share information in the form of the incumbent solution, cached heuristic values, and stored paths from the root to states. This way, when a new iteration of search encounters a node previously explored, it need not recompute the heuristic (which may be expensive) and it can replace the current path to the node with a better one found in a previous search iteration.

Contract Search algorithms are related to anytime algorithms and can use many of the same techniques discussed here (Zilberstein, Charpillet, and Chassaing 1999; Dionne, Thayer, and Ruml 2011). However, contract algorithms take a deadline as input and can use it to its advantage. Anytime algorithms are appropriate when the computational deadline remains unknown or is ambiguous. Despite this difference, anytime algorithms are often used even when computational deadlines are known (e.g., the International Planning Competitions (Do et al. 2008; Olaya et al. 2011)).

Anytime Nonparametric A*

Anytime Nonparametric A* (ANA*) (van den Berg et al. 2011) is a continued search that can be seen as an anytime variant of potential search (Stern, Puzis, and Felner 2011). Anytime nonparametric A* expands the node with maximal $e(n) = \frac{G-g(n)}{h(n)}$, which is equivalent to expanding the node with minimal $e'(n) = \frac{h(n)}{G-g(n)}$, where G is the cost of the current incumbent solution, initially ∞. Despite the fact that ANA* changes its strategy as search progresses, it does so without any input from the user, an important capability. The incumbent solution sets G for the ongoing search algorithm. This avoids having to find a good schedule of weights for each domain, a task which is time consuming and prone to human error. We later show a similar technique for setting the suboptimality bound of anytime search algorithms.

BEES and BEEPS

Bounded Cost Explicit Estimation Search and Bounded Cost Explicit Estimation Search with Potential (BEES and BEEPS respectively) (Thayer et al. 2012) are strongly related to the technique presented in this paper, in that all three algorithms are based upon the Explicit Estimation Search (EES) algorithm (Thayer and Ruml 2011). BEES and BEEPS adapt Explicit Estimation Search algorithm to be used in the setting of bounded cost search (Stern, Puzis, and Felner 2011), where the goal is to find some solution within a user supplied cost-bound as quickly as possible.

BEES and BEEPS can be adapted to work in an anytime fashion by supplying an appropriate set of cost-bounds to the algorithms. Specifically this can be achieved by initially setting the cost-bound to be ∞ and then decreasing it to be ϵ smaller than the current incumbent solution. This allows both BEES and BEEPS to find a stream of solutions, eventually converging on the optimal cost solution, giving them anytime behavior.

LAMA11

LAMA11 is a planner that uses a restarting search. It has two modes of operation, one for domains with unit-cost actions and one for domains where actions have differing costs. LAMA11 runs greedy search on distance-to-go (this step is omitted for unit-cost domains), then greedy search on augmented cost-to-go, then a sequence of weighted A* searches (on inadmissible heuristics). The cost-to-go heuristic is augmented to take advantage of actions-to-go information as well. The final iteration is an A* search on an inadmissible heuristic which runs until the search space has been exhausted. LAMA11 also leverages planning-specific search enhancements such as delayed heuristic evaluation and helpful actions. These help LAMA11 cope with incredibly large branching factors and expensive heuristic functions which are unique to domain independent planning.

Anytime Explicit Estimation Search

Anytime explicit estimation search (AEES) uses EES (Thayer and Ruml 2011) as its main component, trying to minimize the time it takes to improve the quality of the incumbent solutions in anytime search. EES is a bounded suboptimal best-first search where 'best' is the node estimated to both lead to a solution within the desired suboptimality bound and have the fewest remaining actions between it and a goal. EES can only expand nodes that can be shown to lead to a solution whose cost is no more than some bounded factor w of optimal. In order to avoid relying on hand-tuned parameter schedules, a new suboptimality bound is computed every time a new goal is encountered.

EES keeps track of three values for every node. The first is $f(n) = g(n) + h(n)$. This is the same node evaluation function used by A* (Pohl 1970). The second value is $\widehat{f}(n) = g(n) + \widehat{h}(n)$, an inadmissible doppelganger of $f(n)$. $\widehat{f}$ is EES's best estimate of the cost of an optimal solution passing through n. $\widehat{h}$ could be hand-crafted by an expert, learned offline (Xu, Fern, and Yoon 2007), learned over the course of many similar searches (Jabbari Arfaee, Zilles, and

AEES(root)

1. $open \leftarrow \{root\}, cost \leftarrow \infty, w \leftarrow \infty$
2. **while** $open \neq \{\}$
3. let $n = selectNode(open, w)$ in
4. **if** $f(n) \geq cost$ **then** continue
5. **else if** $goal_p(n)$ **then** $onGoal(n, w, cost, open)$
6. **else** $expand(n, open, cost)$
7. $open \leftarrow open - \{n\}$
8. **for each** child c of n
9. **if** $f(c) < cost$ **then** $open \leftarrow open \cup \{c\}$

onGoal(n, w, cost, open)

1. **if** $g(n) < cost$
2. **then** let $best_f = \text{argmin}_{n \in open} f(n)$ in
3. $cost \leftarrow g(n)$
4. $w \leftarrow \frac{cost}{f(best_f)}$

selectNode(open, w)

1. **if** $\widehat{f}(best_{\widehat{d}}) \leq w \cdot f(best_f)$ **then** $best_{\widehat{d}}$
2. **else if** $\widehat{f}(best_{\widehat{f}}) \leq w \cdot f(best_f)$ **then** $best_{\widehat{f}}$
3. **else** $best_f$

Figure 1: Anytime Explicit Estimation Search

Holte 2011), or learned during the solving of a single instance (Thayer, Dionne, and Ruml 2011) as in the following evaluation. The third value is $\widehat{d}(n)$. This is an estimate of the number of actions between n and a goal. For domains where all actions have identical cost, $\widehat{h}(n) = \widehat{d}(n)$. However, many domains have actions of differing cost. Such domains are often challenging (Benton et al. 2010). $\widehat{d}$ can often be computed alongside $\widehat{h}$ by keeping track of the number of actions being taken as well as their cost. Like $\widehat{h}$, $\widehat{d}$, an inadmissible estimate of actions-to-go, often provides better guidance. $\widehat{d}$ can be constructed analogously to $\widehat{h}$.

EES keeps track of three special nodes using these three values. $best_f$, the node which provides a lower bound on the cost of an optimal solution, $best_{\widehat{f}}$, the node which provides EES with an estimate of the cost of an optimal solution to the problem, and $best_{\widehat{d}}$, the node estimated to be within the suboptimality bound and nearest to a goal. Formally, if $open$ contains all nodes generated, but not yet expanded, then $best_f = \text{argmin}_{n \in open} f(n)$, $best_{\widehat{f}} = \text{argmin}_{n \in open} \widehat{f}(n)$, and $best_{\widehat{d}} = \text{argmin}_{n \in open \wedge \widehat{f}(n) \leq w \cdot \widehat{f}(best_{\widehat{f}})} \widehat{d}(n)$. Note that $best_{\widehat{d}}$ is chosen from a *focal* list based on $best_{\widehat{f}}$. EES suspects that nodes with $\widehat{f}(n) \leq w \cdot \widehat{f}(best_{\widehat{f}})$ are those nodes that lead to solutions within the suboptimality bound. At every expansion, EES chooses one of these nodes using $selectNode$ (Figure 1).

Pseudo code for AEES is provided in Figure 1. In line 3 of $AEES$ in Figure 1 we see AEES and EES have the same definition of best, and thus expand nodes in the same order. $selectNode$ pursues the nearest solution estimated to be within the suboptimality bound, provided we can currently prove this node is actually within the bound (line 1 of $selectNode$). Selecting $best_{\widehat{d}}$ is pursuing the next fastest-to-find solution. $best_{\widehat{d}}$ is estimated to both be within bound and have the fewest actions (and thus node expansions) between it and a goal. All other nodes are selected in an effort to make $best_{\widehat{d}}$ pursuable, either by raising our lower-bound on optimal solution cost or by adding new nodes to the pool from which $best_{\widehat{d}}$ is selected.

EES and AEES differ in what happens when a goal node is encountered (line 5 of $AEES$). EES would simply return the solution. AEES is an anytime search algorithm that must eventually converge on an optimal solution. When AEES finds a goal, it updates the cost of the incumbent solution and lowers the suboptimality bound w before continuing search. Setting w effectively is important because it determines which nodes can be expanded. We could use hand-crafted parameter schedules, but these are problematic. What is a good weight schedule on one domain will likely be inappropriate for others. We must choose between taking the time to find an ideal parameter schedule or using one we know was not tailored to the domain of interest.

Rather than supplying a schedule of suboptimality bounds, we compute one online. During search, we can compute a dynamic bound on the suboptimality of the incumbent solution. Rather than supplying a sequence of suboptimality bounds, we need only compute the dynamic bound when the algorithm needs the next parameter, when a new solution is encountered. In AEES, a dynamic bound can be computed as $\frac{g(incumbent)}{f(best_f)}$. $f(best_f)$ provides a lower bound on the cost of an optimal solution to our problem, and so this equation computes an upper bound on the suboptimality of the current incumbent solution. We use this dynamic bound to set w for the next iteration of AEES. This technique has also been used to augment parameter schedules used by anytime search (Likhachev, Gordon, and Thrun 2003; Hansen and Zhou 2007; Thayer and Ruml 2010).

The largest difference between anytime BEES and BEEPS and AEES is how these algorithm determine if a solution is likely to improve upon the current incumbent. While all three approaches use inadmissible estimates of cost-to-go to guess if a node will improve, AEES will make this determination in a relative manner, while BEES and BEEPS do so in an absolute fashion. That is, if $\widehat{f}(n) \geq g(inc)$ then neither BEES nor BEEPS will expand that node while pursuing an improved incumbent; that node will not be expanded until we are trying to prove that the optimal solution is in hand. However, AEES may well expand this node as $w \cdot f(best_f)$ is often larger than $g(inc)$. Thus, AEES can expand nodes where $\widehat{f}(n) \geq g(inc)$.

At first glance, this appears incorrect; $\widehat{f}$ tells us that it cannot improve the incumbent solution, assuming $\widehat{h}$ is correct. However by construction, all nodes on focal ($\widehat{f}(n) \leq w \cdot \widehat{f}(best_f H)$) are capable of improving upon the incumbent given the current bound if $\widehat{f}$ is correct. When we compare $\widehat{f}(n)$ and $g(inc)$, we are comparing an estimate to truth, however looking at $\widehat{f}(n)$ and $\widehat{f}(best_{\widehat{f}})$ compares two esti-

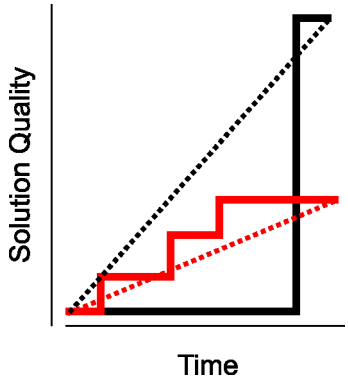

Figure 2: Maximizing $\frac{\Delta q}{\Delta t}$ is not ideal.

mated values. While we would like the inadmissible heuristics to be very accurate, frequently they are not. If the heuristic is consistent in its errors (ie magnitude, direction) the relative comparisons of AEES are more likely to correctly identify a node as leading to an improved solution than BEES or BEEPS. We will see this leads to a significant difference in performance in the empirical evaluation.

Ideal Performance vs. Dominance

We present a theoretical analysis of anytime search algorithms. In doing so, we argue for a definition of *ideal performance* of anytime algorithms and demonstrate that when actions have varying costs, previous state-of-the-art approaches to anytime heuristic search can be arbitrarily worse than AEES. This occurs even if we are willing to assume perfect heuristic estimators and tie breaking.

Maximizing $\frac{\Delta q}{\Delta t}$ is not ideal

When discussing the ideal performance of an anytime algorithm, we intuitively think of an algorithm that quickly improves the quality of a solution it has in hand. We might even go so far as to posit that the algorithm that has the largest positive change in solution quality over time ($\frac{\Delta q}{\Delta t}$) is the best performing anytime search algorithm.

Figure 2 shows why this intuition leads to an ignoratio elenchi – it fails to address the issue at hand for anytime algorithms. The figure shows the performance of two hypothetical anytime algorithms, red and black. The x-axis represents time, and the y-axis reports the quality of solutions the algorithms have in hand at a particular time. We also show the slope of the performance curves ($\frac{\Delta q}{\Delta t}$) as dotted lines of the same color. We see in the figure that although the black algorithm has the higher $\frac{\Delta q}{\Delta t}$, the red algorithm has the better solution in hand for many points in time. This means that if we were to stop either of the algorithms at an arbitrary time point, the black algorithm would have a worse solution than the red one.

As it turns out, A* guarantees the largest $\frac{\Delta q}{\Delta t}$ among all algorithms (in terms of node expansions, and assuming all algorithms to have access to the same information). This is because A* is a provably optimal algorithm for optimal heuristic search (Dechter and Pearl 1985), and thus it has the largest possible change in solution quality in the shortest possible time, giving it a dominating $\frac{\Delta q}{\Delta t}$. It is well-known that A* provides very poor anytime performance, confirming the suspicion that maximizing $\frac{\Delta q}{\Delta t}$ will not lead to ideal anytime performance.

Ideal Performance and Dominance

We can now define our notion *ideal performance*. Given an anytime algorithm χ, let the ith solution found by χ be π_i^χ, the time to find π_i^χ, $t(\pi_i^\chi)$ and the cost of π_i^χ, $c(\pi_i^\chi)$. We assume that at $i = 0$ (i.e., the null solution), $t(\pi_0^\chi) = 0$ and $c(\pi_0^\chi) = \infty$. An ideally performing anytime algorithm minimizes $t(\pi_{i+1}^\chi) - t(\pi_i^\chi)$, or the time between solutions. In other words, it will find the next improving solution in the least amount of time. Our definition contrasts with dominance. Let A and B be two anytime algorithms. If we were to stop each algorithm at an arbitrary time cutoff τ, we say that the best solution of A at τ, π_τ^A dominates the best solution of B at τ, π_τ^B, iff $c(\pi_\tau^A) < c(\pi_\tau^B)$.

Note that our definition of ideal says little about the quality of those solutions, other than the typical assumption that $c(\pi_{i+1}^\chi) < c(\pi_i^\chi)$. It is certainly possible to construct examples where the algorithm with ideal performance will be dominated for certain cutoffs; however, our notion of ideal hinges on the idea that we do not know the cutoff before hand. In the face of unknown deadlines, we argue that the best practice is to try to find a new improving solution.

Naturally, we desire an algorithm with both of these qualities (ideal performance and dominance); that is, we want an algorithm that gives the best solution of all algorithms at any given cutoff and the smallest times between improving the incumbent solutions. Though it is difficult to measure the *optimal* ideal performance in general (i.e., across all possible solution methods), we discuss how AEES works toward ideal performance in the context of anytime search. It does this by taking steps toward minimizing the time between incumbent solutions while also giving the best solution of all state-of-the-art anytime algorithms for many time cutoffs, as the empirical evaluation reveals.

Intuitively, AEES uses an estimate on the time required to find solutions. During the search for the first solution, AEES greedily searches on $\widehat{d}(n)$, the estimated distance to a goal. For ideal performance, we would like to search in order of minimal search effort, but how to estimate this is unclear. Luckily the estimated distance to a solution is strongly related to the depth of that solution in a search tree. Therefore we can use $\widehat{d}$ as a reasonable proxy for search effort. The strategy of AEES is to run EES with a suboptimality bound computed as the bound on the suboptimality over the current incumbent solution (which is ∞ initially). EES estimates which nodes will lead to a solution within this bound. Indeed, these are exactly the nodes on the focal list of EES. Further, pruning on the current incumbent ensures that we find only improving solutions.

Example Figure 3 shows a single instance of a family of graphs where the previously proposed anytime search algorithms will not minimize the time between solutions. These

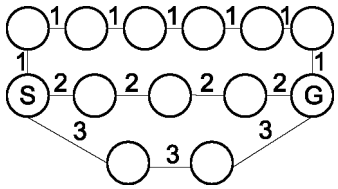

Figure 3: A difficult graph for anytime search algorithms.

graphs all have a start node s, a goal node g, and three paths connecting them, a short but costly path, a long cost-optimal path, and a path of medium length that is neither the most nor the least expensive. Though we cannot cover all anytime search algorithms, we note that any algorithm that does not incorporate d in determining search order will fail in the same manner that ANA* fails. We assume that we have perfect information about cost and distance-to-go and begin by discussing *ideal performance* on this graph.

Ideal performance: Using our definition of ideal performance, in this graph we want to find the shortest path, then next shortest path, then finally the longest, but cost-optimal, path. This performance is ideal in the sense that we previously described: it minimizes the amount of time the algorithm is without solution and then minimizes time to improve the incumbent. For a given graph of this family and algorithm to compare to (excepting AEES), there will always be a range of cutoffs for which ideal performance has a better solution than the algorithm, and no algorithm improves the quality of the incumbent faster than ideal on any graph in this family.

LAMA11 Search Strategy: LAMA will find the shortest (bottom) path in its initial greedy search on d. Then it searches greedily on h. Since h is perfect ($h = h^*$), this search will pursue the longest optimal cost plan next, and thus take longer to improve the incumbent than ideal. We can specify a cutoff which allows for finding the shortest and second shortest paths, but not the shortest and longest path. In these situations, the LAMA11 search strategy will have a worse than ideal solution.

ANA*: Before it has an incumbent, ANA* will perform a best-first search on cost-to-go. Since h is perfect ($h = h^*$), this search will pursue the longest optimal cost plan, and thus take longer to find the first solution than ideal. We can then specify a cutoff that will allow for finding the shortest solution, and no other solution. ANA* will not have any solution, while an ideal algorithm finds the shortest solution.

EES Based Algorithms: We have perfect information, and thus $h = \widehat{h} = h^*$ and $d = \widehat{d} = d^*$. Initially, AEES and ABEES will find the shortest solution, as all solutions are within a suboptimality bound of ∞. Then, we need to look at which solutions are within the new suboptimality bound. $f(best_f)$ is equal to $f^*(opt)$, so we get exactly the suboptimality bound of the current incumbent, and since both other solutions have cost less than the current incumbent, they must be within the bound. Since both are within the bound, we pursue the one with least d, the next shortest solution. Thus, the EES based algorithms find the solutions in the same order as the ideal above, and so we cannot choose a cutoff for which the ideal algorithm has a better solution than these algorithms.

Experiments

Although AEES does try to minimize the time between improvements to the incumbent solution, and thus its behavior is ideal in that sense, we do not know if it will have better solutions in hand for a given cutoff than other anytime search algorithms. This evaluation reveals that the two correlate: the algorithm with the smallest time between solutions tends to have the best solution in hand at any given time for the domains examined here. For 5 of the 7 benchmarks considered, this is AEES. On the remaining two benchmarks, AEES is competitive with the best performing algorithm.

We performed an evaluation on 4 domain specific benchmarks and 3 benchmarks from domain independent planning. All experiments were run on 64-bit Intel Linux systems with 3.16 GHz Core2 duo processors and 8 GB of RAM. Domain specific solvers were implemented in OCaml, and algorithms were cut off when they exhausted memory or 10 minutes had passed. For the domain independent benchmarks, we evaluate the algorithms in the Fast Downward planner which is implemented in C++. The time cutoff was changed from 10 minutes 30 to match settings used in the International Planning Competition.

Our evaluation covers five anytime search algorithms: Anytime repairing A* (**ARA***), using a weight schedule of 5, 3, 2, 1.5, 1 following Richter, Thayer, and Ruml (2010), Anytime nonparametric A* (**ANA***), **AEES** as described in this paper, and **LAMA11-PSS**, an implementation of LAMA11 which makes no use of planning-specific enhancements. This works as a greedy search on $d(n)$, followed by a greedy search on $h(n)$, followed by a sequence of weighted A* searches using the same weight schedule used by ARA*. Finally, we compare against an anytime version of the bounded-cost explicit estimation search algorithm (ABEES in the plots). We do not compare to an anytime variant of BEEPS, as BEES and BEEPS did not have substantially different performance (Thayer et al. 2012).

Sliding Tiles Puzzles We use the 100 instances of the 15-puzzle presented by Korf (1985). h and d are the Manhattan distance heuristic, and $\widehat{h}$ and $\widehat{d}$ are computed using the online learning techniques described in Thayer, Dionne, and Ruml (2011). The performance of the anytime algorithms in this domain is shown in the first panel of Figure 4. The x-axis represents the time at which the algorithm was halted on a log scale. On the y-axis, we report the mean quality (and 95% confidence intervals) of the solution the algorithm had in hand at the time it was halted. Solution quality is the cost of the best for any algorithm on this problem divided by the cost of the algorithm's current solution.

The 15-puzzle is a worst-case scenario for AEES. Distinguishing between solution cost and length provides no advantage when actions have uniform cost. Node generation and heuristic computation are very cheap. We see in the leftmost panel of Figure 4 that despite its handicap in this domain AEES is competitive with the best performing search algorithms, LAMA11-PSS and ANA*. ANA* also

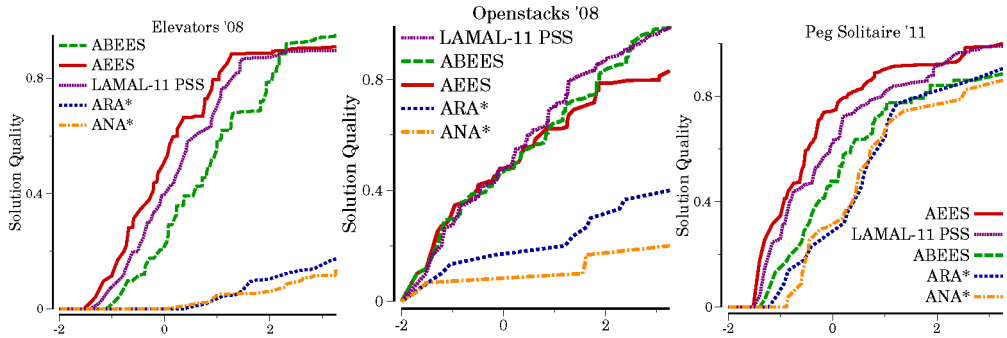

Figure 4: Performance of Anytime Algorithms in Domain Specific Solvers

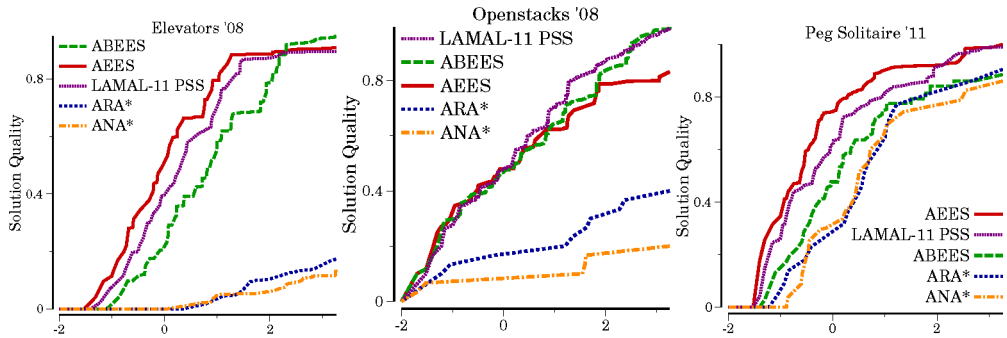

Figure 5: Performance in Domain Independent Planning

has the shortest time between solutions, as we later discuss.

We also note that ABEES is one of the worst performing algorithms in this domain. Although BEES and EES have somewhat similar approaches to search, they differ in the nature of the bounds they use for determining if a node is suitable for search. EES relies on relative cost bounds, while BEES focuses on absolute cost bounds. BEES is thus more sensitive to the accuracy of its heuristics, as we previously noted. Thus, if $\widehat{h}$ is wildly inaccurate, so long as it values that are accurate relative to one another, AEES will continue to work, while BEES and BEEPS will not.

Inverse Tiles We study the same 100 15-puzzle instances as above, but we replace the standard cost function with one where the cost of moving a tile is the inverse of its face value, $\frac{1}{face}$ as suggested by Thayer, Dionne, and Ruml (2011). This separates the cost and length of a solution without altering other properties of the domain. h is computed as the weighted sum of Manhattan distances for each tile, that is $h(n) = \frac{MD(n)}{face(n)}$. d is the unadulterated Manhattan distance.

Both AEES and ABEES search strategy take advantage of distance-to-go estimates, and as we see in the plot (second panel, Figure 4), they are the best performing algorithms in this domain. LAMA11 also uses actions-to-go estimates in search, but it does not consult them directly beyond the first iteration. Rather, they are used to augment the cost-to-go function. We see in the plot that not relying on action-to-go estimates directly causes a sharp drop-off in LAMA11-PSS performance for this domain when compared to the unit tiles problem. ARA* and ANA* never use actions-to-go and fail to solve over a third of the instances while AEES and LAMA11-PSS solve all instances. AEES has the shortest average time between solutions.

Dock Robot We implemented a dockyard robot domain inspired by Ghallab, Nau, and Traverso (2004) where a robot puts containers in their desired stack using a crane. A robot may drive between stacks and load or unload itself. We tested on 150 random problems with 3 locations on a unit square and 15 containers with random start and goal locations. Driving has a cost of the distance traveled, loading and unloading the robot costs 0.1 plus 0.05 times the number of containers stacked at the depot. h is the cost of driving between all depots with out of place containers plus the cost of moving the deepest out of place container in the stack to the robot. d is computed by substituting 1 for the action cost.

The performance of the algorithms on this domain is presented in the third panel of Figure 4. We see that AEES, which has the shortest time between improving solutions on average, also dominates other anytime algorithms. ARA* beats LAMA11-PSS and ANA* because it is not being too greedy. Of the 150 instances, greedy best-first search on h fails to solve 55 of them within 10 minutes. ANA* runs a greedy search on h early on, resulting in bad performance. ARA*'s performance in this domain is partially due to duplicate-delaying, as this domain contains many cycles, and partially the result of a good weight schedule. If the ini-

tial weight were very large (i. e.∞), ARA* would perform much like ANA*.

Vacuum World This domain mirrors the first state space presented in Russell and Norvig (2003). A robot must clean a grid. Movement is in the cardinal directions. Whenever the robot is on a dirty cell, it may vacuum. The cost of motion is one plus the number of dirty cells already cleaned. We use 100 solvable instances that are 500x500. Each cell has a 35% chance of being blocked, and there are twenty piles of dirt, placed randomly in unblocked cells. For h we compute the minimum spanning tree of the robot and dirt, order the edges by greatest length first, and then multiply the edge weights by the current weight of the robot plus the number of edges already considered. d is the length of greedily visiting all dirty cells if there were no obstacles.

The final panel of Figure 4 shows the performance of the algorithms in this domain. This domain has a substantial difference between actions and cost-to-go, and again we see that AEES has dominant performance in this domain. This once again shows the importance of considering $\widehat{d}$ when searching.

Planning The three panels of Figure 5 show the performance of the anytime search algorithms on three planning benchmarks. For these plots, we omit confidence intervals. The instances are designed to be of increasing difficulty, they are not random, and so displaying confidence intervals makes no sense. The domains are the elevators domain from the 2008 IPC, the open stacks domain from the 2008 IPC, and the Peg Solitaire domain from the 2011 IPC. Note we are still discussing the performance of the underlying search strategy of LAMA11, which does not take advantage of features particular to domain independent planning (helpful actions, lazy evaluation). We use LM-Cut (Helmert and Domshlak 2009) for h, the LM-Count heuristic for $\widehat{h}$, and the FF-heuristic (ignoring action costs) for $\widehat{d}$.

In all three domains, AEES and LAMA11-PSS dominate ANA* and ARA*. Peg Solitaire is the only domain of those presented where ANA* and ARA* are at all competitive with these two approaches. ANA* and ARA* perform poorly because neither take advantage of inadmissible heuristics. AEES and LAMA11-PSS take advantage of both inadmissible heuristics and actions-to-go estimates and both have similar performance for all of the planning domains presented. In elevators '08 and Peg Solitaire '11, AEES has slightly better performance in terms of dominance than LAMA11-PSS. In openstacks, LAMA11-PSS finds solutions nearly three times faster than AEES and also has slightly better solutions for cutoffs longer than 10 seconds.

Dominance and Ideal Performance

Table 1 shows the average time between solutions and the average number of solutions returned for the algorithms under consideration on all benchmarks used in our evaluation. To average time between solutions, we add an additional data point at the time cutoff for all algorithms that did not find an optimal solution, and then compute the difference in time between all reported solutions for all instances, taking the mean of these values ($\overline{\Delta T}$). In the table the smallest mean differences are **bolded** and the largest are *italicized*.

	AEES	ABEES	LAMA11	ANA*	ARA*
Tiles	86	*127*	220	**71**	118
Inv. Tile	104	**75**	211	277	*343*
Dock	**127**	354	*588*	379	342
Vacuum	**13**	18	283	*495*	435
Elev.	**500**	663	621	1059	*1200*
O. Stacks	462	557	**162**	1500	1286
PegSol	356	*960*	**288**	837	900

Table 1: Average time between solutions in seconds

Comparing the results in Table 1 with the mean solution quality results in Figures 4 and 5 reveals an interesting phenomena. The algorithm with the smallest $\overline{\Delta T}$'s is almost always the best performing algorithm for that domain; it has the best mean solution quality across most cutoffs. This suggests that the previously proposed notion of ideal performance, minimizing the time between improvements to the incumbent solution, and having the best solution for a given cutoff are strongly correlated. We note AEES often has the smallest $\overline{\Delta T}$, and it never has the largest. Similarly, AEES often has the best solution in hand at any time, and it is never the worst performing algorithm in these benchmarks.

The Peg Solitaire and tiles domain seem to provide an exception to this trend. Here ANA* and LAMA11-PSS have small $\overline{\Delta T}$, but do not report the best mean quality across all times. This is because the solutions are not encountered uniformly across all times. Most solutions do not occur in the first few seconds of search, they happen later on. In these cases, the algorithm reporting the lowest $\overline{\Delta T}$ is among the best performing algorithms for long cutoffs. If the plots were not on a log scale, it would always appear that the dominating algorithm also had the smallest $\overline{\Delta T}$.

Summary

AEES was consistently among the best performing anytime search algorithm for domains where actions could have differing costs, and often it had the best solution in hand for many cutoffs. This is unsurprising because AEES takes advantage of actions-to-go estimates constantly, a quantity which other algorithms use sparingly (LAMA11) or not at all (ANA*, ARA*). As a result, AEES performs better, in general, than other previously proposed algorithms for anytime search. Although it was not always the dominating algorithm, it was never completely dominated for any domain, as all other algorithms under consideration were.

For short deadlines (i. e.within the first few seconds) we saw that AEES and ABEES had very similar performance. However, as time marched on, AEES often began to perform far better than ABEES. As we discussed previously, this is because ABEES is incorrectly estimating many of the solutions to have cost beyond that of the current incumbent. As time progresses, the cost of the incumbent solution is reduced, but the quality of the inadmissible heuristic learned by ABEES and AEES does not neccessarily improve. Thus, ABEES often incorrectly assumes all nodes will have cost higher than the current incumbent, and reverts to A* search, while AEES, which relies on relative bounds, will not.

For domain independent planning, the results were mixed. When we compare AEES to other search strategies, it is often the best performing. However, if we evaluate LAMA11 with planning-specific enhancements, we see that it has better coverage, and thus better aggregate performance. AEES almost always outperformed LAMA11-PSS. This suggests that once we find a way to incorporate the search enhancements used by LAMA11, AEES ought to consistently outperform LAMA11 in planning.

Finally, we showed that there was a strong correlation between our notion of ideal anytime performance and that of algorithm dominance. Algorithms that have small delays between improvements to the solution also tend to have the best solution given a wide range of cutoffs.

Conclusion and Future Work

This work presented AEES, a new state of the art anytime search algorithm that draws on recent advances in bounded suboptimal search and anytime search to provide a better performing, more robust algorithm. The algorithm works toward providing a search with *ideal performance* by using inadmissible estimates of solution cost and length. It also uses admissible estimates of solution cost to set its own parameters during search, obviating the need for parameter schedules and tuning. Our evaluation showed that minimizing the time between improving solutions appears strongly related to having the best incumbent solution for a wide variety of cutoffs. Future work will explore the theoretical properties of ideal performance and the correlation between ideal performance and dominance is causal.

Acknowledgments

This research is supported in part by the Ofce of Naval Research grants N00014-09-1-0017 and N00014-07-1-1049, the National Science Foundation grants IIS-0905672 and grant IIS-0812141, the DARPA CSSG program grant N10AP20029 and the German Research Council (DFG) as part of the project "Kontrollwissen für domänenunabhängige Planungssysteme" (KontWiss).

References

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmueller, R.; and Kambhampati, S. 2010. G-value plateuas: A challenge for planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*.

Cushing, W.; Benton, J.; and Kambhampati, S. 2011. Cost-based satisficing search considered harmful. In *Proceedings of the Third Workshop on Heuristics for Domain-independent Planning*, 43–52.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM* 32(3):505–536.

Dionne, A. J.; Thayer, J. T.; and Ruml, W. 2011. Deadline-aware search using on-line measures of behavior. In *Proc. SoCS 2011*, 39–46.

Do, M.; Helmert, M.; Refanidis, I.; Buffet, O.; Bryce, D.; Fern, A.; Khardon, R.; and Tepalli, P. 2008. The 2008 international planning competition. IPC 2008 Domains, http://ipc.informatik.uni-freiburg.de/Planners.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers.

Hansen, E. A., and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research* 28:267–297.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 162–169.

Jabbari Arfaee, S.; Zilles, S.; and Holte, R. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence*.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proceedings of the Seventeenth Annual Conference on Neural Information Processing Systems*.

Olaya, A. G.; Lopez, C. L.; Jimenez, S.; Coles, A.; Coles, A.; Sanner, S.; and Youn, S. 2011. The 2011 international planning competition. IPC 2011 Domains, http://ipc.icaps-conference.org.

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

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.

Richter, S., and Westphal, M. 2008. The LAMA planner — Using landmark counting in heuristic search. IPC 2008 short papers, http://ipc.informatik.uni-freiburg.de/Planners.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, 137–144.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). IPC 2011 planner abstracts.

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

Stern, R.; Puzis, R.; and Felner, A. 2011. Potential search: A bounded-cost search algorithm. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*.

Thayer, J. T., and Ruml, W. 2010. Anytime heuristic search: Frameworks and algorithms. In *Symposium on Combinatorial Search*.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal

search: A direct approach using inadmissible estimates. In *Proc. IJCAI 2011*, 674–679.

Thayer, J. T.; Stern, R.; Felner, A.; and Ruml, W. 2012. Faster bounded-cost search using inadmissible estimates. In *Proceedings of the Twenty-second International Conference on Automated Planning and Scheduling*.

Thayer, J. T.; Dionne, A.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*.

van den Berg, J.; Shah, R.; Huang, A.; and Goldberg, K. Y. 2011. ANA*: Anytime nonparametric A*. In *Proc. AAAI 2011*, 105–111.

Xu, Y.; Fern, A.; and Yoon, S. 2007. Discriminative learning of beam-search heuristics for planning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*.

Zilberstein, S.; Charpillet, F.; and Chassaing, P. 1999. Real-time problem-solving with contract algorithms. In *Proceedings of IJCAI-99*, 1008–1013.