Simpler Bounded Suboptimal Search

Matthew Hatem and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
mhatem and ruml at cs.unh.edu

Abstract

It is commonly appreciated that solving search problems optimally can take too long. Bounded suboptimal search algorithms trade increased solution cost in a principled way for reduced solving time. Explicit Estimation Search (EES) is a recent state-of-the-art algorithm for bounded suboptimal search. Although it tends to expand fewer nodes than alternative algorithms, its per-node expansion overhead is much higher, causing it to sometimes take longer. In this paper, we present simplified variants of EES (SEES) and an earlier algorithm, A^*_ϵ (SA^*_ϵ), that use different implementations of the same motivating ideas to significantly reduce search overhead and implementation complexity. In an empirical evaluation, we find that SEES matches or outperforms classic bounded suboptimal search algorithms, such as WA*, on all domains tested. We also confirm that, while SEES and SA^*_ϵ expand roughly the same number of nodes as their progenitors, they solve problems significantly faster and are much easier to implement. This work widens the applicability of state-of-the-art bounded suboptimal search by making it easier to deploy.

Introduction

Heuristic search is a fundamental problem-solving technique in artificial intelligence. Algorithms such as A* [Hart, Nilsson, and Raphael, 1968] have been developed to find optimal (lowest-cost) solutions. A* uses an admissible heuristic function to avoid exploring much of the search space. However, verifying that a solution is optimal requires expanding every node whose f value is less than the optimal solution cost, which we will denote by C^*. For many problems of practical interest, there are too many such nodes to allow the search to complete within a reasonable amount of time [Helmert and Röger, 2008]. A* maintains an *open list*, containing nodes that have been generated but not yet expanded, and a *closed list*, containing all generated nodes, in order to prevent duplicated search effort. On problems with a moderate number of duplicates, the memory required by A* is proportional to its running time, also making it impractical for large problems.

These concerns have motivated the development of bounded suboptimal search algorithms. These algorithms trade increased solution cost for decreased solving time. Given a user-defined suboptimality bound w, they are guaranteed to return a solution of cost $C \leq w \cdot C^*$. A range of bounded suboptimal search algorithms have been proposed, of which the best-known is Weighted A* (WA*, Pohl [1973]). WA* is a best-first search using the $f'(n) = g(n) + w \cdot h(n)$ evaluation function.

WA* returns solutions with bounded suboptimality only when using an admissible cost-to-go heuristic $h(n)$. There has been much previous work over the past decade demonstrating that inadmissible but accurate heuristics can be used effectively to guide search algorithms [Jabbari Arfaee, Zilles, and Holte, 2011; Samadi, Felner, and Schaeffer, 2008]. Inadmissible heuristics can even be learned online [Thayer, Dionne, and Ruml, 2011] and used in bounded suboptimal search [Thayer and Ruml, 2011]. Furthermore, recent work has shown that WA* can perform very poorly in domains where the costs of the edges are not uniform [Wilt and Ruml, 2012]. In such domains, an estimate of the minimal number of actions needed to reach a goal can be utilized as an additional heuristic to effectively guide the search to finding solutions quickly. This is known as the *distance-to-go* heuristic of a node, and denoted by $\widehat{d}(n)$.

A^*_ϵ [Pearl and Kim, 1982] is a bounded suboptimal search algorithm that incorporates distance-to-go estimates by maintaining two orderings of the search frontier. While A^*_ϵ has been shown to perform better than WA* in some domains, it is unable to enjoy the benefits of inadmissible heuristics. Moreover, A^*_ϵ must synchronize its two orderings during search and can suffer significant overhead for each node expansion.

Explicit Estimation Search (EES) [Thayer and Ruml, 2011] is a recent state-of-the-art bounded suboptimal search algorithm that incorporates inadmissible heuristics as well as distance-to-go estimates to guide its search and has been shown to expand fewer nodes than WA* and A^*_ϵ across a wide variety of domains. Unfortunately, like A^*_ϵ, EES must synchronize multiple priority queues, resulting in lower node expansion rates than single queue algorithms such as WA*. As we explain further below, it is possible to reduce some of the overhead by employing complex data structures. However, as our results show, substantial overhead remains

and it is possible for WA* to outperform A^*_ϵ and EES when distance-to-go estimates and inadmissible heuristics do not provide enough of an advantage to overcome this inefficiency.

The main contribution of this paper is a simpler approach to implementing the ideas behind A^*_ϵ and EES that preserves their intention while allowing us to avoid maintaining multiple orderings of the search frontier. We call these new algorithms Simplified A^*_ϵ (SA^*_ϵ) and Simplified EES (SEES). While they are merely approximations to the originals and can offer different search behavior, they are both easier to implement and have significantly less overhead per node expansion. In an empirical evaluation, we compare SA^*_ϵ and SEES with optimized versions of A^*_ϵ, EES and WA* on a variety of benchmark domains. The results show that while the simplified implementations expand roughly the same number of nodes, they have much higher node expansion rates and thus solve problems significantly faster, achieving a new state-of-the-art in our benchmark domains and they are significantly easier to implement. This work generalizes the ideas inherent in contemporary bounded suboptimal search by presenting an alternative implementation methodology, and widens their applicability by simplifying their deployment.

Previous Work

WA* is a simple modification of A* and is perhaps the most popular bounded suboptimal search algorithm in use today. Unfortunately, WA* is far from state-of-the-art: it does not enjoy the benefits of inadmissible heuristics or distance-to-go estimates which have been shown to dramatically improve performance for bounded suboptimal search. This has led to the development of algorithms like A^*_ϵ and EES.

Bounded suboptimal search attempts to find solutions that satisfy the user-specified bound on solution cost as quickly as possible. Solving time is directly related to the number of nodes that are expanded during search. Finding shorter solutions typically requires fewer node expansions. Intuitively, we can speed up search by prioritizing nodes that are estimated to have shorter paths to the goal.

Like A*, A^*_ϵ orders the open list using an admissible evaluation function $f(n)$. A second priority queue, the *focal* list contains a prefix of the open list: those nodes n for which $f(n) \leq w \cdot f(best_f)$, where $best_f$ is the node at the front of the open list. The focal list is ordered by a potentially inadmissible estimate of distance-to-go $\widehat{d}(n)$ and is used to prioritize nodes that are estimated to have shorter paths to the goal. The node at the front of the focal list is denoted by $best_{\widehat{d}}$.

With an admissible heuristic, the value $f(n)$ will tend to increase along a path. Newly generated nodes, including those where $\widehat{d}$ has gone down, will often not qualify for entry into the focal list. Shallower nodes with $f(n) = f(best_f)$ will tend to have higher estimates of distance-to-go and be pushed to the back of the focal list. As a result, A^*_ϵ suffers from a thrashing effect where the focal list is required to empty almost completely before $best_f$ is expanded, finally raising the value of $f(best_f)$ and refilling the focal

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: Pseudo-code for node selection in EES.

list [Thayer, Ruml, and Kreis, 2009]. While A^*_ϵ has been shown to perform better than WA* in some domains, it is often worse, and it is also unable to take advantage of accurate but inadmissible heuristics for search guidance.

Explicit Estimation Search (EES) is a recent state-of-the-art bounded suboptimal search algorithm that uses an inadmissible estimate of cost-to-go $\widehat{h}$ and an inadmissible distance-to-go estimate $\widehat{d}$. To incorporate multiple heuristics, EES maintains three different orderings of the search frontier. The open list is ordered using an inadmissible estimate of solution cost $\widehat{f}(n) = g(n) + \widehat{h}(n)$. The node at the front of the open list is denoted $best_{\widehat{f}}$. A focal list containing all nodes n for which $\widehat{f}(n) < w \cdot \widehat{f}(best_{\widehat{f}})$ is ordered using $\widehat{d}(n)$. EES also maintains a *cleanup* list, containing all open nodes, that is ordered using the admissible estimate of solution cost $f(n) = g(n) + h(n)$.

Details of the EES search strategy are given in Figure 1. EES pursues nodes that appear to be near the goal by checking the focal list first (line 1). If the front of the focal list can not guarantee admissibility, then EES tries to expand the node that appears to be on a cheapest path to the goal according to the accurate but potentially inadmissible heuristic by checking the front of the open list (line 2). If this node cannot guarantee admissibility, then EES falls back to just expanding the node on the frontier with the lowest admissible estimate of solution cost f (line 3) which helps the tests in lines 1 and 2 succeed in the future.

Both A^*_ϵ and EES suffer from overhead in having to synchronize the focal list each time the f or $\widehat{f}$ of the node at the front of the open list changes. A naive implementation of A^*_ϵ or EES would maintain a single list of nodes ordered by f and, for each node expansion, perform a linear scan of the list to find an admissible node with minimum $\widehat{d}$. With a loose upper bound on solution cost, this implementation could visit the entire open list for every node expansion.

To make A^*_ϵ and EES practical, a more efficient implementation is required [Thayer, Ruml, and Kreis, 2009]. The open list can be represented by a specialized red-black tree and the focal list can be represented by a binary heap based priority queue. These data structures allow us to identify the node to expand in constant time, remove it from the focal list and the open list in logarithmic time, and insert new nodes in logarithmic time.

When the node with minimum f or $\widehat{f}$ changes, nodes may need to be added or removed from the focal list. To do this efficiently, we do a top-down traversal of the red-black tree, visiting only the nodes that need to be added or removed. We denote the upper bound ($w \cdot f(best_f)$ in the case of A^*_ϵ and $w \cdot \widehat{f}(best_{\widehat{f}})$ in the case of EES) for the focal list with b. For each node expansion we assume that b will not change, and all child nodes that qualify for the focal list according to the

current value of b are added. The value b is updated only after processing all children. If in fact b does not change, then the focal list is already synchronized and no further processing is necessary. If b decreases to b', then a traversal of the red-black tree is performed, visiting only nodes with a value that is within the interval of b' and b. Each visited node is removed from the focal list. If b increases to b', then a traversal is performed, visiting only nodes within the interval b and b', and adding visited nodes to the focal list.

Unfortunately, even with this optimization, A^*_ϵ and EES can have significant node expansion overhead compared to single queue based algorithms like WA*, especially when the upper bound for the focal list changes frequently. In the worst case, the entire open list might need to be visited on every node expansion. These algorithms only have a chance of performing better than WA* when distance-to-go estimates and inadmissible heuristics provide a significant advantage over heuristic guidance alone.

Simplified Bounded Suboptimal Search

The main contribution of this paper is an alternative approach to implementing multi-queue based search algorithms without incurring the overhead of maintaining multiple queues. In this section we present two new algorithms that can be viewed as simplified variants of A^*_ϵ and EES. We call these new algorithms Simplified A^*_ϵ (SA^*_ϵ) and Simplified EES (SEES). We will start with a discussion of SA^*_ϵ and then extend the ideas to SEES by incorporating inadmissible heuristics.

Simplified A^*_ϵ

A^*_ϵ expands the node with lowest $\widehat{d}(n)$ among those whose $f(n)$ is within a factor w of the lowest $f(n)$ on open. Intuitively, A^*_ϵ can be viewed as an algorithm that is doing Speedy search (a best-first search on $\widehat{d}$ [Ruml and Do, 2007]) within an adaptive upper bound on solution cost. We can approximate this search strategy by using a fixed upper bound that is updated if the Speedy search fails. This approach, which is less adaptive but has far less overhead, can be viewed as a combination of iterative deepening [Korf, 1985a] and Speedy search.

SA^*_ϵ simplifies implementation by eliminating one of the priority queues, namely the open list, replacing it with iterative deepening on f. Like IDA*, SA^*_ϵ maintains a threshold t_f that is initialized to f of the initial state. Search proceeds in iterations, expanding all nodes with $f(n) \leq w \cdot t_f$ in each iteration. However, unlike IDA*, within each iteration nodes are expanded best-first in lowest $\widehat{d}(n)$ order. Nodes that exceed the current upper bound on solution cost are pruned. At the start of each iteration the threshold is updated to the minimum cost of all pruned nodes. The completeness of SA^*_ϵ follows easily from the increasing bound.

The pseudo code for SA^*_ϵ is given in Figure 2, ignoring the code in brackets. SA^*_ϵ begins by computing h of the initial state to initialize the t_f threshold (line 1). Next, search proceeds by performing a bounded speedy search (lines 2–5). The initial state is added to the open list, a priority queue that is sorted by $\widehat{d}(n)$ (line 7). The speedy search proceeds by expanding the best node on open, pruning all child nodes where $f(n)$ exceeds the current t_f threshold (lines 14-19). The minimum $f(n)$ of all pruned nodes is remembered at each iteration and is used as the threshold for the next iteration (lines 3, 5 and 16). At each iteration, SA^*_ϵ prunes all nodes with $f(n) > w \cdot t_f$ (line 15). If the heuristic is admissible, SA^*_ϵ can terminate when a goal node is expanded and guarantee that the cost of the solution is w-admissible (lines 10-11). If a goal is not found in one iteration, the threshold is updated (line 5) and the speedy search repeats.

Theorem 1 *If SA^*_ϵ terminates with a solution, it is w-admissible if the heuristic is admissible.*

Proof: Let C be the cost of the goal returned by SA^*_ϵ and assume for the sake of contradiction that $C > w \cdot C^*$. Let t_i be the threshold used in the iteration when the goal node was expanded. Since the goal node was expanded, it holds that $C \leq w \cdot t_i$. On the other hand, the goal was not expanded in the previous iteration, where the threshold was lower than t_i. Since t_i is updated to be the minimum f value of all nodes that exceeded the previous bound, then at least one node p that is on the optimal path to the goal has $f(p) \geq w \cdot t_i$. Therefore:

$$\begin{aligned} C &\leq w \cdot t_i \leq w \cdot f(p) \\ &\leq w \cdot (g(p) + h(p)) \leq w \cdot C^* \end{aligned}$$

This contradicts the assumption that $C > w \cdot C^*$. □

SA^*_ϵ approximates the search behavior of A^*_ϵ by expanding nodes that appear to be closer to the goal first, among those nodes that guarantee admissibility. Because SA^*_ϵ expands only nodes that guarantee admissibility, it can terminate as soon as it expands a goal, just like A^*_ϵ. Unlike A^*_ϵ, which adaptively sets the upper bound for the focal list, potentially at each node expansion, SA^*_ϵ uses a fixed upper bound at each iteration and must exhaust all nodes in the focal list before increasing this bound.

Simplified EES

A^*_ϵ and EES have similar motivations, and both algorithms perform a speedy search within an adaptive upper bound on solution cost. We refer to the nodes that are within the upper bound as the search envelope. EES differs from A^*_ϵ in that it incorporates an accurate but potentially inadmissible heuristic $\widehat{h}$ to help shape the search envelope. This inadmissible heuristic helps by pruning unpromising nodes from the search envelope that would otherwise be expanded by A^*_ϵ. To approximate the search strategy of EES, we can apply the same iterative deepening technique of SA^*_ϵ. However, this time we need to perform iterative deepening on two upper bounds: the upper bound on solution cost f and the upper bound on the inadmissible estimate of solution cost $\widehat{f}$.

SEES orders node expansions according to $\widehat{d}(n)$ and prunes nodes with high $f(n)$ and $\widehat{f}(n)$. This emulates the same criteria used to determine which nodes form the focal list in EES and which nodes are selected for expansion (Figure 1). However, one critical difference is that SEES only ever expands nodes from the focal list, because all

SA$^*_\epsilon$ or SEES($init$)
1. $solution \leftarrow nil$; $t_f \leftarrow h(init)$; [$t_{\widehat{f}} \leftarrow \widehat{h}(init)$]
2. while $t_f < \infty$ AND $solution = nil$
3. $\quad t_{f_{next}} \leftarrow \infty$; [$t_{\widehat{f}_{next}} \leftarrow \infty$]
4. $\quad$ Speedy($init$)
5. $\quad t_f \leftarrow t_{f_{next}}$; [$t_{\widehat{f}} \leftarrow t_{\widehat{f}_{next}}$]
6. return solution

Speedy($init$)
7. $open \leftarrow \{init\}$; $closed \leftarrow \emptyset$
8. while $open$ is not empty
9. $\quad n \leftarrow$ remove node from $open$ with min $\widehat{d}$
10. $\quad$ if n is a goal
11. $\quad\quad solution \leftarrow n$; break
12. $\quad$ else
13. $\quad\quad closed \leftarrow closed \cup \{n\}$
14. $\quad\quad$ for $child \in$ expand(n)
15. $\quad\quad\quad$ if $f(child) > w \cdot t_f$ [or $\widehat{f}(child) > w \cdot t_{\widehat{f}}$]
16. $\quad\quad\quad\quad t_{f_{next}} \leftarrow \min(t_{f_{next}}, f(child))$
17. $\quad\quad\quad\quad$ [$t_{\widehat{f}_{next}} \leftarrow \min(t_{\widehat{f}_{next}}, \widehat{f}(child))$]
18. $\quad\quad\quad$ else if $child$ is not a duplicate
19. $\quad\quad\quad\quad open \leftarrow open \cup \{child\}$

Figure 2: Pseudo-code for SEES. Ignoring the code in square brackets gives pseudo-code for SA$^*_\epsilon$.

nodes in the focal list guarantee admissibility. In contrast, not all nodes that form the focal list for EES guarantee w-admissibility and EES can potentially expand nodes from either of three priority queues during search.

The pseudo code for SEES is given in Figure 2 including the code in square brackets. SEES is the same algorithm as SA$^*_\epsilon$ with the addition of an inadmissible node evaluation function for pruning (lines 1, 3, 5, 15 and 17). Compared to EES, the priority queue for the open list is replaced by iterative deepening on $\widehat{f}$ and the priority queue for the cleanup list is replaced by iterative deepening on f. This leaves just one priority queue, the focal list, ordered by $\widehat{d}(n)$. The proof for EES being w-admissible is identical to the proof for SA$^*_\epsilon$.

The iterative-deepening search, with thresholds t_f and $t_{\widehat{f}}$, results in a best-first search order on both f and $\widehat{f}$ when these functions are monotonic, and approximates a best-first search order otherwise. These thresholds play a similar role both to $best_f$ and $best_{\widehat{f}}$ from the original algorithms – restricting search to nodes that are estimated to guarantee w-admissibility.

SA$^*_\epsilon$ and SEES have far less overhead than their progenitors. Each requires only a single priority queue, sorted according to one evaluation function. Each node only needs to be stored in one priority queue, requiring less memory to store each node. Moreover, the focal list does not need to be synchronized.

Experiments

To determine the effectiveness of our simplified approach, we implemented SA$^*_\epsilon$ and SEES and compared them to A$^*_\epsilon$, EES and WA* on 4 different domains, including 3 of the simplest and most reproducible of the original 6 domains used by Thayer and Ruml [2011]. For all experiments, we used path-based single step error correction of the admissible heuristic h to construct a more accurate but potentially inadmissible heuristic $\widehat{h}$, as described by Thayer and Ruml [2011]. All algorithms were written in Java and compiled with OpenJDK 1.6.0.24. We implemented the optimizations recommended by Burns et al. [2012] and Hatem, Burns, and Ruml [2013] with the exception of bucket based priority queues and C++ templates. All experiments were run on a machine with a CoreDuo 3.16 GHz processor and 8 GB of RAM running Linux.

15-Puzzle

We evaluated the performance of SA$^*_\epsilon$ and SEES on a simple unit-cost domain by comparing them to A$^*_\epsilon$, EES and WA* on Korf's 100 15-puzzles [Korf, 1985b] using the Manhattan distance heuristic for both heuristic and distance-to-go estimates. The upper plot in the first column in Figure 3 shows mean node expansions at suboptimality bounds 1.5, 2, 3, 4 and 5. As the suboptimality bound increases, each algorithm requires fewer node expansions and all algorithms require roughly the same number of node expansions beyond a suboptimality bound of 3. However, when we examine solving times, the lower plot in the first column, we see that EES and A$^*_\epsilon$ are significantly slower overall while SEES performs as well as WA*. EES performs poorly compared to SEES and WA* because the overhead of maintaining multiple orderings of the search frontier outweighs the benefits of the search guidance provided by the inadmissible heuristic $\widehat{h}$. The node expansion rate for SEES is roughly 2.5 times faster than EES in this setting. SA$^*_\epsilon$ is faster than A$^*_\epsilon$ but the difference is not as dramatic.

Next, we change the domain slightly by modifying the cost function such that the cost to move a tile is the inverse of the number on the face of the tile. This provides a wide range of edge costs in a simple, well understood domain and previous work has shown that inverse costs make the problems harder to solve and differentiate algorithms more [Thayer and Ruml, 2011]. Moreover, distance-to-go estimates provide a significant advantage over using the Manhattan distance heuristic alone. In this domain WA* is not able to solve all instances with our timeout of 60 seconds at any suboptimality bound. The plots in the second column in Figure 3 show that while SA$^*_\epsilon$ expands roughly the same nodes as its progenitor, it solves problems faster as the bound loosens. SEES is the fastest algorithm at lower suboptimality bounds. All these algorithms are able to dramatically outperform WA* on this domain because they are able to incorporate distance-to-go estimates in search guidance.

These plots provide evidence that SEES is merely an approximation to EES, as it expands fewer nodes on average in this domain. We believe the difference in node expansions can be attributed to the non-monotonic upper bounds. While SEES uses fixed upper bounds that never decrease, EES sets this bound adaptively. Since the upper bounds never decrease for SEES, it may be able to find admissible paths to a goal sooner than EES. However, if best$_{\widehat{f}}$ significantly de-

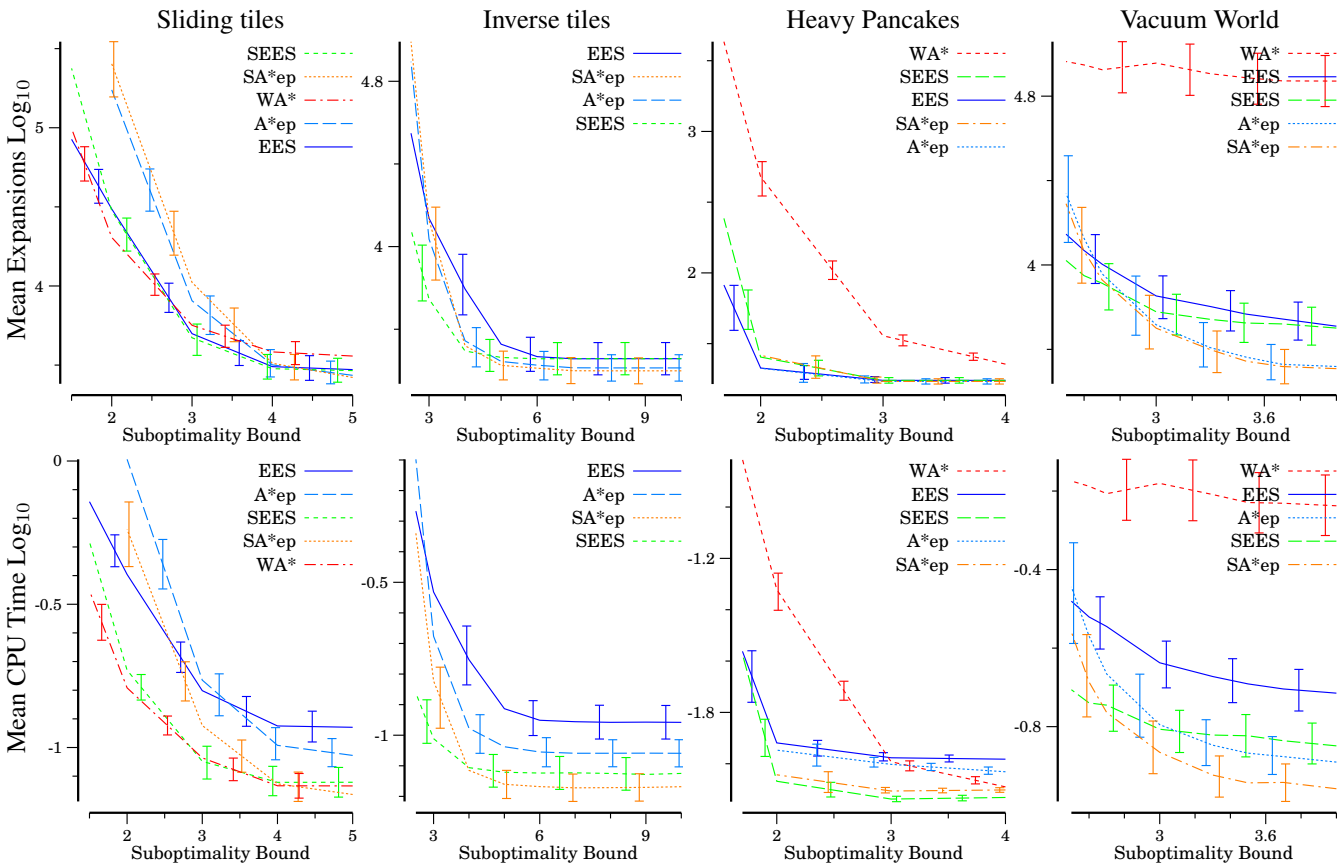

Figure 3: Comparison between SA^*_ϵ, SEES, EES and WA^* on sliding tiles, pancakes and vacuum world domains. Plots show mean node expansions and mean CPU time in seconds $\log_{10}$.

creases during search, then SEES might expand more nodes than EES and if it is roughly stable then SEES and EES should expand roughly the same nodes.

Pancake Puzzle

We also evaluated the performance of these algorithms on the heavy pancake puzzle using the gap heuristic. In this domain we must order a permutation of $\{1, ..., N\}$, where N is the number of pancakes, by reversing a contiguous prefix. The gap heuristic is a type of landmark heuristic that counts the number of non adjacent pancakes or "gaps" for each pancake in the stack [Helmert, 2010]. This heuristic has been shown to be very accurate, outperforming abstraction based heuristics. In heavy pancakes, the cost to flip a pancake is the value of its id. This produces a wide range of integer edge costs and distance-to-go estimates provide additional search guidance over using the heuristic alone. In our experiments we used 100 random instances with 14 pancakes. For the admissible heuristic, we adapted the gap heuristic and we used the basic unit cost form of the gap heuristic for the distance-to-go estimate. The plots in the third column show the number of node expansions and the mean solving time. WA^* is the worst performing algorithm and A^*_ϵ and SA^*_ϵ were only able to solve all instances within our memory limit of 8GB or our timeout of 500 seconds for suboptimality bounds greater than 2. In this domain SEES expands more nodes than EES. This is because at lower suboptimality bounds, SEES performs many iterations, resulting in many node re-expansions. However, SEES still solves problems as fast or faster than EES because it has significantly less overhead per expansion. SA^*_ϵ is faster than A^*_ϵ but again the difference is not as dramatic.

Vacuum World

The Vacuum World domain is the first state space introduced in Russell and Norvig [2003]. In this domain a vacuum robot must clean up dirt distributed across a grid world. In the version considered here, actions have different costs depending on how much dirt the robot is carrying. This variant is called the *Heavy* Vacuum World. In our experiments we used the same heuristic and distance estimates used by Thayer and Ruml [2011]. The admissible heuristic is computed by finding a minimum spanning tree for all remaining dirt piles, ignoring blocked locations. The edges of the tree are sorted by decreasing length and we sum the product of each edge and weight of the robot, starting with the current weight and increasing it for each edge. The distance-to-go estimate is a greedy traversal of all remaining dirt. The distance-to-go estimates provide a significant advantage over using just the heuristic alone. The fourth column of Figure 3 summarizes the results for the heavy vacuum world. As the suboptimality bound increases, all algorithms except WA^* require less

	IDA*	A*	SA*$_\epsilon$	SEES	A*$_\epsilon$	EES
CC	13	203	204	207	270	374
LoC	50	886	928	947	1,166	1,433

Table 1: McCabe's Cyclomatic Complexity (CC) and lines of code (LoC) metrics for our implementations.

time to solve all instances. WA* expands more nodes as the bound increases in some cases. However, the performance of the algorithms that incorporate distance-to-go estimates improve consistently and are significantly faster than WA*. In this domain SA*$_\epsilon$ performs better than SEES at higher suboptimality bounds. This is likely because the heuristic for this domain are misleading and SEES is also being guided by $\widehat{f}$. SA*$_\epsilon$ has the advantage in this domain of being guided only by the distance-to-go estimate.

Code Complexity

Although it is hard to quantify code complexity, we use two popular metrics to measure how much simpler SEES and SA*$_\epsilon$ are to implement. We counted lines of code and computed the Cyclomatic Complexity [McCabe, 1976] for our implementations of IDA*, A*, A*$_\epsilon$, EES, SEES, and SA*$_\epsilon$. Cyclomatic Complexity is a metric commonly used by software engineers to predict the number of defects that are likely to occur in a program. This metric indirectly measures the linearly independent paths through a program's source code. Table 1 shows the Cyclomatic Complexity of our implementations, along with the number of lines of Java code (excluding comments). From this table we see a wide range in complexity. IDA* is a variation on simple depth-first search and does not keep an open or closed list so it requires no data structures. A* is the next most complex algorithm. Our implementation uses a generalized binary heap for the open list and a hash-table with open addressing and quadratic collision resolution. SEES and SA*$_\epsilon$ are just slightly more complex than A*. They use the same binary heap and hash-table for the open and closed lists. EES is by far the most complex program. It uses the same binary heap and hash-table from our A* implementation but also requires a red-black tree to store the open list. SEES represents a 45% reduction in complexity over EES and SA*$_\epsilon$ represents a 24% reduction in complexity over A*$_\epsilon$.

Discussion

While we have focused on presenting very simple alternatives to A*$_\epsilon$ and EES, optimizations are possible that will further improve performance in certain situations.

Thayer and Ruml [2011] present an optimistic variant of EES, EES$_{opt}$, that borrows ideas from Optimistic Search [Thayer and Ruml, 2008], a bounded suboptimal search algorithm that ignores the admissible evaluation function until an incumbent solution is found. After an incumbent is found, a *cleanup* search phase uses the admissible heuristic to prove the incumbent is w-admissible. Like Optimistic Search, EES$_{opt}$ ignores the admissible heuristic, expanding just nodes in the focal list until an incumbent is found. Once an incumbent is found EES$_{opt}$ proceeds just like EES until $f(incumbent) \leq w \cdot f(best_f)$. We can use a similar technique to formulate an optimistic variant of SEES (SEES$_{opt}$). We do this by simply ignoring the f_t threshold until an incumbent solution is found. Once an incumbent is found SEES$_{opt}$ proceeds just like SEES until it can prove the incumbent is w-admissible or until it finds a new incumbent that is. This is advantageous when $\widehat{h}$ is very accurate.

SA*$_\epsilon$ and SEES use an iterative deepening search strategy inspired by IDA*. Like IDA*, SA*$_\epsilon$ and SEES expand a super-set of the nodes in the previous iteration. If the size of iterations grows geometrically, then the node regeneration overhead is bounded by a constant. The number of nodes expanded by IDA* can be $O(n^2)$ [Sarkar et al., 1991] in the worst case when the number of new nodes expanded in each iteration is constant. To alleviate this problem, Sarkar et al. introduce IDA*$_{\text{CR}}$ which tracks the distribution of f values of the pruned nodes during an iteration of search and uses it to find a good threshold for the next iteration. This is achieved by selecting the bound that will cause the desired number of pruned nodes to be expanded in the next iteration. We can apply a similar technique to SA*$_\epsilon$ and SEES. However, we did not find node re-expansion overhead to result in worse performance for SA*$_\epsilon$ and SEES when compared to their progenitors so we did not incorporate this technique.

We can eliminate all node re-expansion caused by iterative deepening by simply remembering search progress between iterations. Rather than deleting nodes that exceed the current thresholds, they are placed on a separate open list for the next iteration. The closed list persists across all iterations. This technique is reminiscent of Fringe search [Björnsson et al., 2005], an algorithm that has been shown to perform better than A* in some domains by replacing the priority queue with a iterative layered search. This optimization adds complexity to the implementation and we did not use it in any of our experiments.

Conclusion

In this paper, we presented simplified variants of A*$_\epsilon$ (SA*$_\epsilon$) and EES (SEES) that use a different implementation of the same fundamental ideas to significantly reduce search overhead and implementation complexity. In an empirical evaluation, we showed that SA*$_\epsilon$ and SEES, like the originals, outperform other bounded suboptimal search algorithms, such as WA*. We also confirmed that, while SA*$_\epsilon$ and SEES expand roughly the same number of nodes as the originals, they solve problems significantly faster, setting a new state of the art in our benchmark domains, and they are significantly easier to implement. This work provides a general approach that may be applicable to other complex multi-queue based search algorithms. We hope this work leads to faster, simplified and wider adoption of inadmissible heuristics and distance estimates in bounded suboptimal search.

Acknowledgments

We gratefully acknowledge support from the NSF (grants 0812141 and 1150068) and DARPA (grant N10AP20029). We also thank Jordan Thayer and Ethan Burns for helpful discussion and sharing code and experiments.

References

Björnsson, Y.; Enzenberger, M.; Holte, R. C.; and Schaeffer, J. 2005. Fringe search: beating A* at pathfinding on game maps. In *Proceedings of IEEE Symposium on Computational Intelligence and Games*, 125–132.

Burns, E.; Hatem, M.; Leighton, M. J.; and Ruml, W. 2012. Implementing fast heuristic search code. In *Proceedings of SoCS-12*.

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

Hatem, M.; Burns, E.; and Ruml, W. 2013. Faster problem solving in Java with heuristic search. *IBM developerWorks*.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *Proceedings of AAAI-08*.

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *Proceedings of SOCS-10*.

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

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

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

McCabe, T. 1976. A complexity measure. *IEEE Transactions on Software Engineering* SE-2(4):308–320.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-4(4):391–399.

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.

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

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*.

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

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

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

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of IJCAI-11*.

Thayer, J. T.; Dionne, A.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *Proceedings of ICAPS-11*.

Thayer, J. T.; Ruml, W.; and Kreis, J. 2009. Using distance estimates in heuristic search: A re-evaluation. In *Proceedings of SoCS-09*.

Wilt, C. M., and Ruml, W. 2012. When does Weighted A* fail? In *Proceedings of SoCS-12*.