Bounded Suboptimal Search in Linear Space: New Results

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

Bounded suboptimal search algorithms are usually faster than optimal ones, but they can still run out of memory on large problems. This paper makes three contributions. First, we show how solution length estimates, used by the current state-of-the-art linear-space bounded suboptimal search algorithm Iterative Deepening EES, can be used to improve unbounded-space suboptimal search. Second, we convert one of these improved algorithms into a linear-space variant called Iterative Deepening A^*_ϵ, resulting in a new state of the art in linear-space bounded suboptimal search. Third, we show how Recursive Best-First Search can be used to create additional linear-space variants that have more stable performance. Taken together, these results significantly expand our armamentarium of bounded suboptimal search algorithms.

Introduction

Verifying that a solution to a heuristic search problem is optimal requires expanding every node whose f value is less than the optimal solution cost. 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). Furthermore, algorithms like A* (Hart, Nilsson, and Raphael 1968) maintain an *open list*, containing nodes that have been generated but not yet expanded, and a *closed list*, containing all generated states, in order to prevent duplicated search effort. The memory requirements to store the open and closed lists also make it impractical for large problems.

Bounded suboptimal search algorithms trade solution cost for solving time in a principled way. Given a suboptimality bound w, they return a solution whose cost is $\leq w \cdot C^*$, where C^* is the optimal solution cost. Weighted A* (WA*, Pohl (1973)) is a best-first bounded suboptimal search that uses the evaluation function $f'(n) = g(n) + w \cdot h(n)$ and is often able to solve problems faster than A* and with less memory. While WA* uses an admissible heuristic, it is well-known that an inadmissible but accurate heuristic can be used effectively to guide search algorithms (Jabbari Arfaee, Zilles, and Holte 2011; Samadi, Felner, and Schaeffer 2008). EES is a state-of-the-art bounded suboptimal search algorithm that improves on a previous algorithm, A^*_ϵ (Pearl and Kim 1982), by incorporating an inadmissible heuristic as well as distances-to-go estimates (the number of edges separating a node from the goal) (Thayer and Ruml 2011). The first contribution of this paper is to demonstrate how using estimates of *solution length* (the depth of a node plus its distance-to-go), rather than just distance-to-go estimates, can improve A^*_ϵ and EES. Our results show that using solution length estimates result in state-of-the-art performance, especially for A^*_ϵ.

While bounded suboptimal search algorithms scale to problems that A* cannot solve, for large problems or tight suboptimality bounds, they can still overrun memory. Weighted IDA* (WIDA*), a bounded suboptimal variant of IDA*, requires only memory linear in the solution length. Iterative Deepening EES (IDEES, Hatem, Stern, and Ruml (2013)) is a recent state-of-the-art linear-space analog of EES that incorporates an inadmissible heuristic as well as solution length estimates and can solve problems faster than WIDA*. The second contribution of this paper is a linear-space variant of the improved A^*_ϵ called IDA^*_ϵ. This new algorithm achieves a new state-of-the-art in linear-space bounded suboptimal search, surpassing IDEES on some domains.

Unlike WA* and EES, which converge to greedy search as the suboptimality bound approaches infinity, WIDA* and IDEES behave like depth-first search, resulting in more costly solutions and in some cases, longer solving times than WA*. Recursive Best-First Search (RBFS, Korf (1993)) on the other hand, is a much better linear space analog of WA*. It uses the same best-first search order of WA* and converges to greedy search and returns cheaper solutions than WIDA* and IDEES. The third contribution of this paper is new linear space analogs for A^*_ϵ and EES based on RBFS: Recursive Best-First A^*_ϵ (RBA^*_ϵ) and Recursive Best-First EES (RBEES). In contrast to IDEES and IDA^*_ϵ, RBA^*_ϵ and RBEES use a best-first search order and are able to prioritize nodes with shorter estimates of solution length. Like their progenitors, they converge to greedy search as the suboptimality bound increases. Experimental results show that while the RBFS-based algorithms tend to be slower, they provide more stable performance and find cheaper solutions than the depth-first versions.

Taken together, the results presented in this paper significantly expand our armamentarium of bounded suboptimal

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

search algorithms in both the unbounded and linear-space settings.

Previous Work

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, as opposed to cheaper (lower cost) 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.

A^*_ϵ

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 focal 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 not qualify for entry into the focal list and 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 focal (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.

EES

Explicit Estimation Search (EES) is a recent state-of-the-art bounded suboptimal search algorithm that explicitly uses an inadmissible estimate of cost-to-go $\widehat{h}$ and an inadmissible distance-to-go estimate $\widehat{d}$. To incorporate multiple heuristics, EES needs to maintain three different orderings of the search frontier. The open list is ordered using an inadmissible estimate of solution cost $\widehat{f}(n) = g(n) + \widehat{h}(n)$. A focal list containing all nodes n for which $\widehat{f}(n) \leq w \cdot \widehat{f}(best_{\widehat{f}})$ is ordered using a potentially inadmissible estimate of distance-to-go $\widehat{d}(n)$. The node at the front of focal is denoted by $best_{\widehat{f}}$. EES usually expands the node that appears to have the shortest estimated distance to the goal, however, to preserve w-admissibility EES also maintains a *cleanup* list, containing all open nodes, ordered using an admissible estimate of solution cost $f(n) = g(n) + h(n)$. The pseudocode for the function used by EES to determine which node to select for expansion is given in Figure 1.

EES pursues nodes that appear to be near the goal by checking the focal list first. If the front of the focal list cannot guarantee w-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. If this node cannot guarantee w-admissibility then EES falls back to just expanding the node on the frontier with the lowest estimate of cost-to-go f.

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: Pseudocode for node selection in EES.

Iterative Deepening A*

Unfortunately WA*, A^*_ϵ and EES are all limited by the memory required to store the search frontier. For large problems, even these algorithms can exhaust the available memory. This motivates the development of linear-space search. Iterative-deepening A* (IDA*, Korf (1985)) is a heuristic search that requires memory only linear in the maximum depth of the search. This reduced memory complexity comes at the cost of repeated search effort. IDA* performs iterations of a bounded depth-first search where a path is pruned if $f(n)$ becomes greater than the threshold for the current iteration. After each unsuccessful iteration, the threshold is increased to the minimum f value among the nodes that were generated but not expanded during the previous iteration. This value is denoted by min_f. If f is monotonic, then IDA* is guaranteed to expand all nodes in best-first order; the same search order as A*, ignoring ties. However, if f is non-monotonic, then IDA* expands nodes in depth-first order (Korf 1993).

Simply weighting the heuristic in IDA* results in Weighted IDA* (WIDA*): a bounded suboptimal linear-space search algorithm using the same non-monotonic cost function $f'(n) = g(n) + w \cdot h(n)$ as WA*. As w increases, WIDA* prunes large portions of the search space and is often able to find w-admissible solutions quickly. However the remaining paths are searched in depth-first order, resulting in significantly longer solutions and in some cases, longer solving times. In contrast, WA* performs more like greedy search as the bound increases and finds cheaper solutions. Thus WIDA* is not an ideal linear-space analog for WA*.

Recursive Best-First Search

Recursive Best-First Search (RBFS) is a best-first search with linear space complexity. Unlike IDA*, RBFS expands nodes in best-first order even with a non-monotonic cost function. RBFS recursively expands nodes, ordering child nodes by cost and expanding the least cost children first. Each recursive call uses a local threshold that is the minimum cost of a node generated so far but not yet expanded. If all child costs exceed the local threshold, the search backtracks until it reaches a sub-tree with child nodes that have a cost less than or equal to the local threshold. Each time RBFS backtracks from a parent node, it *backs up* the cost of the best child that exceeded the local threshold. This allows the search to quickly pick up where it left off when revisiting

a sub-tree. WRBFS (Hansen and Zhou 2007) improves on RBFS by expanding nodes on the *stack frontier* (nodes connected to the current search path that are stored in memory) in best-first order rather than the *virtual frontier* (all nodes generated but not yet expanded, which may not be in memory and need to be regenerated). This speeds up search by tie breaking in favor of leaf nodes stored in memory, avoiding having to regenerate subtrees below the other nodes with the same f values.

Like IDA*, RBFS suffers from node re-generation overhead. In the worst case, IDA* and RBFS expand $O(N^2)$ nodes where N is the number of nodes expanded by A*, ignoring ties. While the node re-generation overhead of IDA* is easily characterized in terms of the heuristic branching factor, the overhead of RBFS depends on how widely separated promising nodes are located in the search tree. While RBFS has been shown to dominate WIDA* in terms of node expansions (Korf 1993), it is often slower because it has more overhead. However, RBFS finds cheaper solutions, matching the best-first search order and the cost of solutions returned by WA*, making it a better WA* analog.

Unfortunately, linear-space bounded suboptimal search algorithms such as WIDA* and RBFS are unable to directly use distance-to-go estimates to guide the search. WIDA* increases the upper bound at each iteration to explore deeper parts of the search tree. It is not possible to perform iterative deepening using just a distance-to-go estimate since this estimate will tend to decrease along a path. RBFS, on the other hand, uses a virtual frontier ordered by increasing f to perform a best-first search. One could imagine incorporating distance-to-go estimates by ordering the frontier by increasing $\widehat{d}$. Unfortunately, this results in poor performance because of excessive backtracking overhead.

Iterative Deepening EES

RBFS and WIDA* use only a single admissible heuristic for search guidance. Iterative Deepening EES (IDEES) is a recent linear-space bounded suboptimal search that is able to incorporate accurate, inadmissible heuristics and distance-to-go estimates. IDEES is based on the same iterative deepening depth-first search of WIDA*. Like WIDA*, it runs a sequence of limited depth-first search iterations, but instead of maintaining a single threshold of min_f, it maintains two thresholds: $t_{\widehat{l}}$ and $t_{\widehat{f}}$. By $\widehat{l}(n)$ we denote the length analogue of $\widehat{f}(n)$: the potentially inadmissible estimate of the total number of edges from the root to the nearest solution under n, $depth(n) + \widehat{d}(n)$. The threshold $t_{\widehat{f}}$ is a cost-based threshold, pruning nodes on the basis of an accurate, but potentially inadmissible heuristic. Thus, in an IDEES iteration, a node n is pruned (not expanded) if it either leads to a goal that is too far away or too expansive, i.e., $\widehat{l}(n) > t_{\widehat{l}}$ or $\widehat{f}(n) > t_{\widehat{f}}$.

The focal list of EES affords a Speedy search (greedy search on distance-to-go (Ruml and Do 2007)). However, because IDEES is a depth-first search, it can only approximate the search strategy of EES by using solution length estimates. This puts IDEES at a disadvantage when there is little error in distance-to-go estimates. EES can converge to Speedy search sooner than IDEES in this case. However, when the distance-to-go estimates have high error, using solution length will provide for a more robust search strategy. As we show below, using solution length estimates can actually improve existing algorithms that incorporate Speedy search like A^*_ϵ and EES.

Using Solution Length Estimates

A^*_ϵ can be viewed as an algorithm that performs a Speedy search with an adaptive upper bound on solution cost. Nodes that are estimated to be closer to a goal are expanded first. These nodes also tend to be the nodes with the highest estimate of solution cost f and are more likely to lead to inadmissible solutions, resulting in the thrashing behavior that is often observed in A^*_ϵ. EES explicitly tries to avoid inadmissible solutions by using $\widehat{f}$ to shape the focal list. However, if $\widehat{f}$ does not compensate enough for the error in f, then it is possible even for EES to be lead down unfruitful paths.

Intuitively, shorter paths will tend to be cheaper and more likely to lead to admissible solutions. By prioritizing nodes that are estimated to have shorter estimated overall solution length rather than least distance-to-go we can achieve faster search by avoiding many paths that are estimated to lead to inadmissible solutions.

We can view Speedy search as a special case of a weighted search using solution length. We define the solution length estimate of a node n as follows:

$$\widehat{l'}(n) = (1 - W_d) \cdot depth(n) + W_d \cdot \widehat{d}(n)$$

As the value for W_d increases, the search acts more like Speedy search, converging to Speedy search as W_d approaches 1. The evaluation function used to order the focal list in EES and A^*_ϵ is a special case of $\widehat{l'}$ with $W_d = 1$. As we will see in the experiments, we can improve the search behavior of A^*_ϵ and EES by having a non-zero weight on the depth component, thus informing these algorithms of the depth of potential solutions. Paths that are estimated to be long will fall to the back of the focal list and paths that are estimated to be shorter will be expanded first. In our experiments we use a slightly different, but equivalent, evaluation function where weight is only applied to the distance component $\widehat{d}(n)$ and takes on values higher than 1.

Experiments

To determine the effectiveness of using solution length estimates, we implemented A^*_ϵ and EES and compared them using solution length and distance-to-go estimates. 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). For simplicity, we used the same value used for the upper bound on solution cost W to set the value for W_d. All algorithms were written in Java using the optimizations suggested by Hatem, Burns, and Ruml (2013) and compiled with OpenJDK 1.6.0.24. All of our experiments were run on a machine with a CoreDuo 3.16 GHz processor and 8 GB of RAM running Linux.

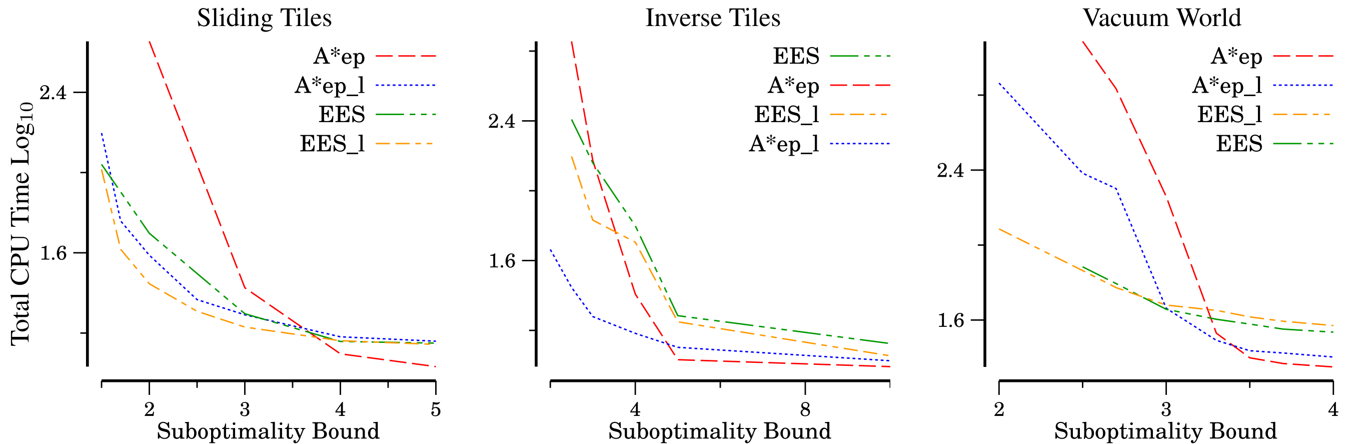

Figure 2: Total CPU time (in $\log_{10}$) for A^*_ϵ and EES using estimates of total length vs. estimates of distance-to-go.

15-Puzzle

We first consider a simple unit-cost domain: Korf's 100 15-puzzles (Korf 1985). We used the Manhattan distance heuristic for both heuristic and distance-to-go estimates. The first plot of Figure 2 summarizes the results for unit cost 15-puzzle. This plot shows the total CPU time in $\log_{10}$ for solving all 100 instances. Algorithms that use solution length estimates have suffix of *_l*. This plot shows that A^*_ϵ is much faster using solution length estimates at lower suboptimality bounds. A^*_ϵ cannot even solve all instances with suboptimality bounds less than 2 when using distance-to-go estimates because it exceeds the limit of 8 GB of available memory. EES using length estimates is the fastest algorithm overall and is faster than EES using distance-to-go estimates, however the improvement is small compared to the difference observed in A^*_ϵ. This may be explained by considering solution length to be a proxy for $\widehat{f}$. Since EES is already guided by $\widehat{f}$, the additional guidance from solution length is not as beneficial.

Next, we change the domain slightly by modifying the cost function. We set the cost to move a tile to the inverse of the number on the face of the tile. We use the inverse cost Manhattan distance for the heuristic and a unit cost Manhattan distance for the distance-to-go estimate. This simple change makes the Manhattan distance heuristic less effective in guiding the search, making the problems harder to solve. Moreover, distance-to-go estimates provide a significant advantage over using the Manhattan distance heuristic alone. The second plot in Figure 2 summarizes the results for inverse cost 15-puzzle. This plot echos the results seen in the plot for unit tiles: A^*_ϵ improves with solution length estimates for lower suboptimality bounds and EES is slightly improved.

Vacuum World

Next we compared these algorithms on the Vacuum World domain, the first state space introduced in Russell and Norvig (2010). In this domain a vacuum robot must clean up dirt distributed across a grid world. In the original description of this problem, each action in the world, 4-way movement and vacuuming, costs the same. In a more realistic setting, actions would 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 in the grid world. The edges of the tree are sorted longest to shortest. We take the sum of the product of each edge and the weight of the robot. The weight of the robot is the number of dirt piles cleaned so far. For the distance-to-go estimate we use a greedy traversal of all remaining dirt. Like inverse tiles, the distance-to-go estimates provide a significant advantage over using just the heuristic for search guidance. We used 150 randomly generated instances. Each instance has 10 piles of dirt and 40,000 possible locations on a 200x200 grid with 35% of the locations being blocked.

The third plot of Figure 2 summarizes the results for the heavy vacuum world. This plot shows the total CPU time in $\log_{10}$ for solving all 150 instances. In this plot we see again that using estimates of solution length improves the performance of A^*_ϵ and EES with the difference in A^*_ϵ being much greater. A^*_ϵ and EES with distance-to-go estimates cannot even solve all instances with a suboptimality bound less than 2.5 because they exceed the limit of 8 GB of available memory.

Taken together, these results suggest that using estimates of solution length to guide bounded suboptimal search algorithms like EES and A^*_ϵ is a simple and promising alternative to using just distance-to-go estimates. It is so effective that A^*_ϵ becomes a viable option for state-of-the-art bounded suboptimal search.

We now turn our attention to how solution length estimates provide an alternative that can be exploited in linear-space bounded suboptimal search.

Iterative Deepening A^*_ϵ

A^*_ϵ normally performs a Speedy search with an adaptive upper bound on solution cost. The upper bound ensures w-

```
IDA*_ε(init)
 1. min_f ← f(init); t_l̂ ← l̂(init)
 2. while min_f < ∞
 3.     min_f_next ← min_l_next ← ∞
 4.     if DFS(init) break
 5.     min_f ← min_f_next; t_l̂ ← min_l_next
 6. return solution

DFS(n)
 7. if n is a goal
 8.     solution ← n; return true
 9. else if (f(n) > w · min_f) or (l̂(n) > t_l̂)
10.     min_f_next ← min(min_f_next, f(n))
11.     min_l_next ← min(min_l_next, l̂(n))
12. else
13.     for child ∈ expand(n)
14.         if DFS(child), return true
15. return false
```

Figure 3: Pseudocode for IDA$^*_\epsilon$ using $\widehat{l}$.

admissibility while the Speedy search prioritizes nodes that appear to be closer to the goal. By replacing Speedy search with a weighted search on solution length, we can formulate a depth-first, linear-space approximation of A$^*_\epsilon$ using the same iterative deepening framework of IDA* (IDA$^*_\epsilon$). The open list of A$^*_\epsilon$ is replaced by iterative deepening on f and the focal list is replaced by iterative deepening on $\widehat{l}$. IDA$^*_\epsilon$ simulates the search strategy of A$^*_\epsilon$ by expanding nodes that are estimated to have shorter paths to the goal first and uses an upper bound on solution cost f to prune nodes that do not guarantee w-admissibility.

The pseudocode for IDA$^*_\epsilon$ is given by Figure 3. IDA$^*_\epsilon$ begins by initializing the minimum f and $\widehat{l}$ threshold to that of the initial state (line 1). Then it performs iterations of depth-first search, increasing the threshold on $\widehat{l}$ and tracking the minimum f of all pruned nodes (lines 3-5, 10, 11) for each iteration. A node is pruned (not expanded) if it cannot guarantee w-admissibility or if its estimate of solution length is greater than the current threshold (line 9). Since only nodes that guarantee w-admissibility are expanded, IDA$^*_\epsilon$ can terminate as soon as it expands a goal (lines 4, 7, 8, 14).

While the original A$^*_\epsilon$ uses distance-to-go estimates alone to order the focal list, IDA$^*_\epsilon$ uses estimates of solution length and is thus an approximation to the original. However, as our experiments have shown, A$^*_\epsilon$ performs better using estimates of solution length. Moreover, IDA$^*_\epsilon$ has less overhead per node expansion and enjoys many of the same optimizations of IDA* such as in place state modification (Burns et al. 2012) and does not require any complex data structures to synchronize the focal list (Thayer, Ruml, and Kreis 2009). The algorithm is similar to IDEES except that it doesn't use an additional inadmissible heuristic for pruning. IDEES improves on IDA$^*_\epsilon$ by incorporating an accurate but potentially inadmissible heuristic to prune additional nodes from the search.

```
RBFS(n, B)
 1. if n is a goal
 2.     solution ← n; return ∞
 3. C ← expand(n)
 4. if C is empty return ∞
 5. for each child n_i in C
 6.     if f(n) < F(n) then F(n_i) ← max(F(n), f(n_i))
 7.     else F(n_i) ← f(n_i)
 8. sort C in increasing order of F
 9. if |C| = 1 then F(C[2]) ← ∞
10. while (F(C[1]) ≤ B and F(C[1]) < ∞)
11.     F(C[1]) ← RBFS(C[1], min(B, F(C[2])))
12.     resort C
13. return F(C[1])
```

Figure 4: Pseudocode for RBFS.

Recursive Best-First Search

IDA$^*_\epsilon$ and IDEES are simpler to implement than their best-first progenitors. However, like WIDA*, they are both based on depth-first search and their performance advantage can degrade as the upper bound loosens. They also return higher cost solutions on average when compared to best-first algorithms like WA* and RBFS. RBFS follows the same best-first search order of WA*, even with a non-monotonic cost function, resulting in stable performance and cheaper solutions. This motivates the development of RBFS-based variants of these algorithms (RBA$^*_\epsilon$ and RBEES). We will begin by reviewing the original RBFS and then introduce the new algorithms.

The pseudocode for RBFS is shown in Figure 4. RBFS recursively expands nodes (line 3 and 11), ordering child nodes by cost (line 8) and expanding the least cost children first. Each recursive call uses a local threshold B that is the minimum cost of a node generated so far but not yet expanded. If all child costs exceed the local threshold, the search backtracks (lines 10 and 13).

Each time RBFS backtracks from a node, it *backs up* the cost of the best child that exceeded the local threshold (lines 11 and 13). The backed up value of a node is denoted by the function F which is initialized to f. If there is no second child, the evaluation $F(C[2])$ simply returns infinity.

RBFS stores all of the fringe nodes connected to the current search path, the stack frontier. The nodes on the stack frontier store backed up values for the portions of the search that have been generated so far, the virtual frontier. The virtual frontier is explored in best-first order by always expanding the best local node first while keeping track of the F value of the second best node which may be elsewhere on the virtual frontier. Because RBFS is best-first it can terminate when it expands a goal (lines 1 and 2).

Each time RBFS expands a node it sorts all the newly generated nodes according to F. While RBFS has been shown to dominate WIDA* in terms of node expansions (Korf 1993), the overhead from sorting often results in slower solving times. However, if node expansion is slow then RBFS can perform better than IDA*.

RBA$^*_\epsilon$ or RBEES ($init$)
1. $solution \leftarrow \infty$
2. $min_f \leftarrow f(init)$; $[t_{\widehat{f}} \leftarrow \widehat{f}(init)]$
3. while $min_f < \infty$ and $solution = \infty$
4. $\quad min_{f_{next}} [\leftarrow min_{\widehat{f}_{next}}] \leftarrow \infty$
5. $\quad$ if RBFS($init$, ∞) break
6. $\quad min_f \leftarrow min_{f_{next}}$; $[t_{\widehat{f}} \leftarrow min_{\widehat{f}_{next}}]$
7. return $solution$

RBFS(n, B)
8. if n is a goal
9. $\quad solution \leftarrow n$; return ∞
10. else if $(f(n) > w \cdot min_f)$ [or $\widehat{f}(n) > w \cdot min_{\widehat{f}}$]
11. $\quad min_{f_{next}} \leftarrow \min(min_{f_{next}}, f(n))$
12. $\quad [min_{\widehat{f}_{next}} \leftarrow \min(min_{\widehat{f}_{next}}, \widehat{f}(n))]$
13. $\quad$ return ∞
14. $C \leftarrow expand(n)$
15. if C is empty return ∞
16. for each child n_i in C
17. $\quad$ if $\widehat{l}(n) < L(n)$ then $L(n_i) \leftarrow max(L(n), \widehat{l}(n_i))$
18. $\quad$ else $L(n_i) \leftarrow \widehat{l}(n_i)$
19. sort C in increasing order of L
20. if only one child in C then $L(C[2]) \leftarrow \infty$
21. while $(L(C[1]) \leq B$ and $L(C[1]) < \infty)$
22. $\quad L(C[1]) \leftarrow$ RBFS($C[1]$, $min(B, L(C[2]))$)
23. $\quad$ resort C
24. return $L(C[1])$

Figure 5: Pseudocode for RBA$^*_\epsilon$ (excluding the bracketed portions) and RBEES (including the bracketed portions).

RBA$^*_\epsilon$ and RBEES

RBA$^*_\epsilon$ combines iterative deepening with RBFS. Like IDA$^*_\epsilon$, RBA$^*_\epsilon$ performs iterative deepening on solution cost estimates f and within each iteration it performs a bounded RBFS search, ordering nodes according to solution length estimates and pruning all nodes that exceed the upper bound. RBEES is similar, except that it incorporates an additional upper bound on solution cost using an accurate but inadmissible heuristic $\widehat{h}$. RBEES prunes a node n if it exceeds either upper bound.

The pseudocode for RBA$^*_\epsilon$ and RBEES is given by Figure 5, ignoring the code in square brackets for RBA$^*_\epsilon$. The function $RBFS$ is a straight forward adaptation of the original to use solution length estimates $\widehat{l}$ and L in place of f and F. Like IDA$^*_\epsilon$, the search begins by initializing the upper bound on solution cost to be the estimated solution cost of the initial state (line 2). The search proceeds by performing a series of RBFS searches (lines 3, 5, 8-24) expanding nodes in increasing order of solution length estimates $\widehat{l}$ (line 19) and pruning any node that exceeds the upper bound(s) (lines 10-13). The search can terminate as soon as it expands a goal and guarantee that the solution cost is w-admissible since it only expands nodes that do not exceed the upper bound $w \cdot min_f$ and are thus w-admissible (line 10).

Experiments

To determine the effectiveness of IDA$^*_\epsilon$, RBA$^*_\epsilon$ and RBEES, we compare them to WIDA*, WRBFS and IDEES on 3 different domains. We used the same path-based single step error correction to construct $\widehat{h}$, and the same machines and memory limits as in the previous experiments.

15-Puzzle

The first column in Figure 6 summarizes the results for unit tiles. The top plot shows the mean CPU time for solving all 100 instances with error bars that show a 95% confidence interval about the mean. In this plot we see that both RBA$^*_\epsilon$ and RBEES improve on the performance of WRBFS and perform about the same with bounds greater than 2. RBA$^*_\epsilon$ is not able to solve all instances with smaller bounds. IDEES is slightly faster and IDA$^*_\epsilon$ is the fastest algorithm overall. The bottom plot in the first column shows the mean solution costs for all 100 instances. As expected, the RBFS-based algorithms are slower but, like RBFS, they return better solutions than depth-first algorithms.

Next, we change the domain slightly by modifying the costs to be the sqrt of the number on the face of the tile. This provides a wide range of edge costs in a simple, well understood domain without making the problem much harder to solve optimally than the original. The original WIDA* is not able to solve these problems because of the wide range of edge costs. We use WIDA*$_{CR}$instead. The second column in Figure 6 summarizes the results for sqrt tiles. These plots echo the same results from unit tiles: the RBFS-based algorithms are slower overall but provide significantly cheaper solutions. Because of the wide range of f values, WRBFS suffers from excessive node re-expansion overhead and is not able to solve all instances at any of the suboptimality bounds. RBEES and RBA$^*_\epsilon$ are guided by d which has a much narrower distribution.

Next, we compared our algorithms using the inverse cost function. The third column in Figure 6 summarizes the results for inverse tiles. In this domain the Manhattan distance heuristic alone is less effective. The RBFS-based algorithms are only able to solve all instances at suboptimality bounds of 3 or larger. Again, WRBFS was not able to solve all instances at any of the suboptimality bounds. The depth-first search algorithms are much faster but again we see that for the cases where the RBFS-based algorithms could solve all instances, they return significantly cheaper solutions.

Heavy Pancakes

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

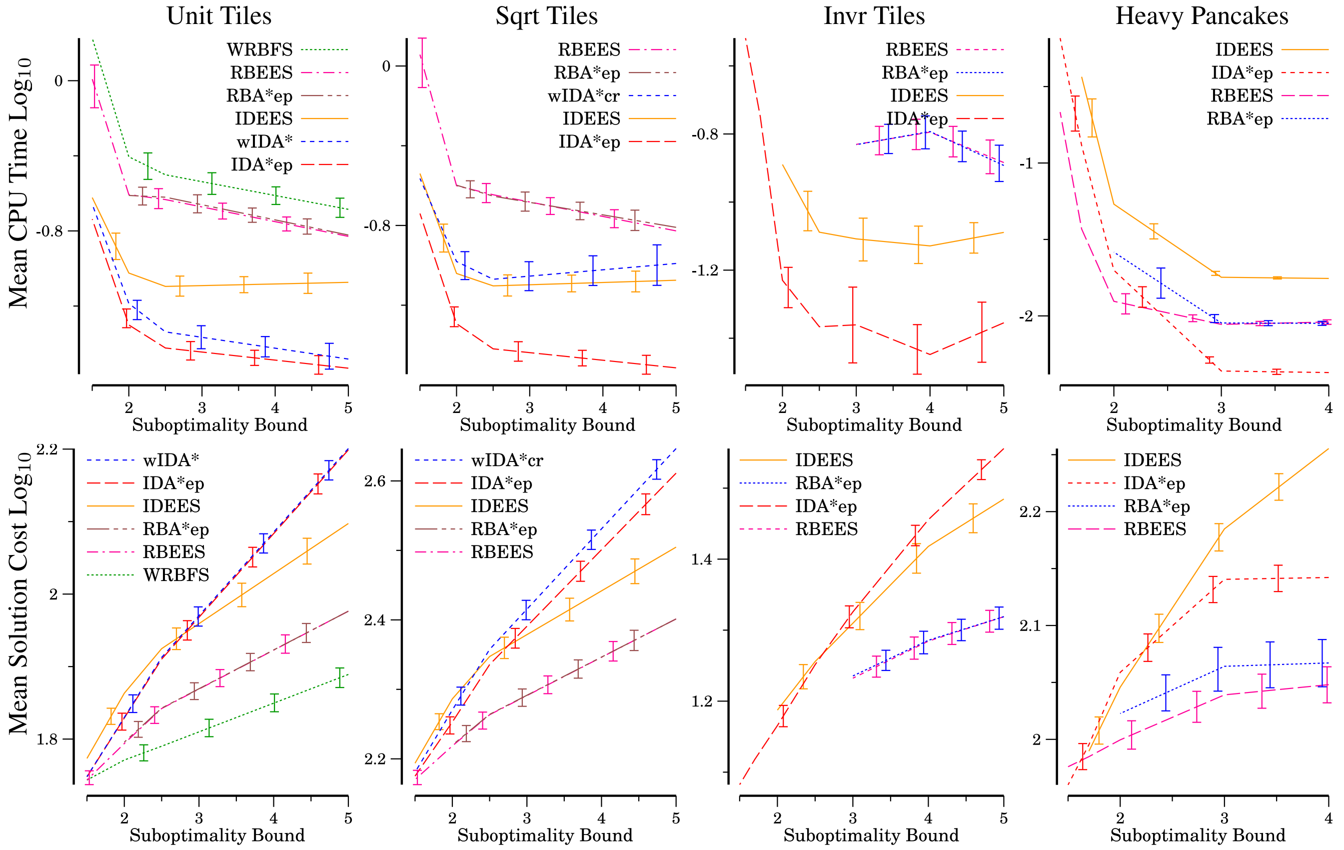

Figure 6: Mean CPU time and solution cost for the linear-space algorithms (in $\log_{10}$).

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 last column in Figure 6 summarizes the results for heavy pancakes. In these plots we see that RBEES is significantly faster than the depth-first algorithms at lower suboptimality bounds while IDA^*_ϵ is the fastest algorithm at higher bounds.

In summary, these results suggest that, like RBFS, RBA^*_ϵ and RBEES return significantly cheaper solutions than the depth-first-based algorithms, even with non-monotonic cost functions. While IDA^*_ϵ achieves new state-of-the-art results in the three sliding tiles domains, the new RBEES algorithm surpasses all other algorithms in the heavy pancakes domain at lower suboptimality bounds.

Conclusion

In this paper, we first demonstrated how using estimates of solution length can improve EES and transform A^*_ϵ into a highly competitive algorithm. We then showed how solution length estimates enable iterative deepening depth-first variant of A^*_ϵ and presented two new linear-space best-first bounded suboptimal search algorithms, RBA^*_ϵ and RBEES. Our experiments showed that IDA^*_ϵ achieves a new state of the art in the sliding-tiles domains while the RBFS-based algorithms provide significantly better solution cost in all domains as the bound loosens and also surpass previous work on the heavy pancake domain. Taken together, the results presented in this paper significantly expand our armamentarium of bounded suboptimal search algorithms in both the unbounded and linear-space settings.

Acknowledgments

We gratefully acknowledge support from the NSF (grants 0812141 and 1150068) and DARPA (grant N10AP20029).

References

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

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

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*.

Hatem, M.; Stern, R.; and Ruml, W. 2013. Bounded suboptimal heuristic search in linear space. In *Proceedings of SoCS-13*.

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

Korf, R. E. 1993. Linear-space best-first search. *Artificial Intelligence* 62(1):41–78.

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. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition.

Samadi, M.; Felner, A.; and Schaeffer, J. 2008. Learning from multiple heuristics. In *Proceedings of AAAI-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.; Ruml, W.; and Kreis, J. 2009. Using distance estimates in heuristic search: A re-evaluation. In *Proceedings of SoCS-09*.