Bounded Suboptimal Heuristic Search in Linear Space

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

Roni Stern
School of Engineering and Applied Sciences
Harvard University
Cambridge, Massachusetts 02138 USA
roni.stern at *gmail.com*

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

Abstract

It is commonly appreciated that solving search problems optimally can overrun time and memory constraints. Bounded suboptimal search algorithms trade increased solution cost for reduced solving time and memory consumption. However, even suboptimal search can overrun memory on large problems. The conventional approach to this problem is to combine a weighted admissible heuristic with an optimal linear space algorithm, resulting in algorithms such as Weighted IDA* (wIDA*). However, wIDA* does not exploit distance-to-go estimates or inadmissible heuristics, which have recently been shown to be helpful for suboptimal search. In this paper, we present a linear space analogue of Explicit Estimation Search (EES), a recent algorithm specifically designed for bounded suboptimal search. We call our method Iterative Deepening EES (IDEES). In an empirical evaluation, we show that IDEES dramatically outperforms wIDA* on domains with non-uniform edge costs and can scale to problems that are out of reach for the original EES.

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. 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. A* maintains 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. Unfortunately, the memory requirements of A* also make it impractical for large problems.

These concerns have motivated the development of bounded suboptimal search algorithms. Bounded suboptimal search algorithms trade solution quality for solving time. A bounded suboptimal search algorithm is given a user-defined suboptimality bound w and is guaranteed to return a solution of cost $C \leq w \cdot C^*$. A range of bounded suboptimal search algorithms have been proposed, many of which are based on the well known Weighted A* (wA*) algorithm (Pohl 1973). wA* is a best-first search using the $f'(n) = g(n) + w \cdot h(n)$ evaluation function. However, it still maintains open and closed lists. Ideally, bounded suboptimal searches generate fewer nodes than optimal searches and therefore run faster and use less memory. However, for large problems or tight suboptimality bounds, they can still run out of memory.

Linear-space search algorithms are designed for difficult problems where the open and closed lists would exhaust memory. A linear-space search algorithm only requires memory that is linear in the depth of the search. Examples of linear space algorithms include IDA* (Korf 1985), RBFS (Korf 1993) and their bounded suboptimal equivalents Weighted IDA* and Weighted RBFS (Korf 1993), denoted wIDA* and wRBFS, respectively.

All the algorithms mentioned above only exploit an admissible heuristic. There has been much previous work over the past decade demonstrating 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). Inadmissible heuristics can even be learned online (Thayer, Dionne, and Ruml 2011) and used to speed up bounded suboptimal search (Thayer and Ruml 2008; Thayer, Dionne, and Ruml 2011). One contribution of this paper is to demonstrate how inadmissible heuristics can be used to speed up a linear space search while preserving bounded suboptimality.

Furthermore, recent work has shown that wA* can perform very poorly in domains where edge costs 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 $d(n)$. An efficient bounded suboptimal search algorithm called EES has been developed that uses $d(n)$, $h(n)$ and an online learned inadmissible heuristic $\widehat{h}(n)$ (Thayer and Ruml 2011). Unfortunately, like A*, EES must store all unique nodes that are generated during search.

In this paper we present Iterative Deepening Explicit Estimation Search (IDEES); a linear space bounded suboptimal search algorithm that uses both admissible and inadmissible

heuristic estimates of cost-to-go as well as distance-to-go. Experimental results on several benchmark domains show a speed up of several orders of magnitude over wIDA* and in some cases IDEES is the only algorithm capable of solving problems within a reasonable time bound. IDEES shows that the notions behind EES, of using more than the traditional $h(n)$ to guide the search, can be implemented beyond openlist-based search and without requiring complex data structures.

In the remainder of this paper we describe Weighted IDA* (Korf 1993) in detail and prove its w-admissibility. Then we describe Explicit Estimation Search (Thayer and Ruml 2011), which forms the foundation for IDEES; our linear space bounded suboptimal search algorithm. Then we describe IDEES and prove its correctness. Finally, we show experimental results illustrating the strengths and weaknesses of IDEES.

Weighted IDA*

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 in the previous iteration. This value is denoted by min_f .

Simply weighting the heuristic in IDA* results in Weighted IDA* (wIDA*): a bounded suboptimal linear space search algorithm. wIDA* is a linear space analogue of wA*, which, like wA*, is also guaranteed to be w-admissible. To the best of our knowledge, the proof for wIDA* being w-admissible has never been published and we provide it below.

Theorem 1 *The solution returned by wIDA* is w-admissible.*

Proof: Let C be the cost of the goal returned by wIDA* and assume by 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 $C \leq t_i$. On the other hand, the goal was not expanded in the previous iteration, where the threshold was lower than t_i. Hence, at least one node p that is on the optimal path to the goal has $f'(p) \geq t_i$. Therefore:

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

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

IDA*$_{CR}$

Each iteration of IDA* and wIDA* expands a super set of the nodes in the previous iteration. If the number of nodes expanded in each iteration grows geometrically, then the number of nodes expanded by IDA* is $O(n)$, where n is the number of nodes that A* would expand (Sarkar et al. 1991). In domains with non-uniform edge costs, there can be many unique f values and the standard minimum-out-of-threshold schedule of IDA* may lead to only a few new nodes being expanded in each iteration. The number of nodes expanded by IDA* can be $O(n^2)$ in the worst case when the number of new nodes expanded in each iteration is constant (Sarkar et al. 1991).

To alleviate this problem, Sarkar et al. introduce IDA*$_{CR}$. IDA*$_{CR}$ tracks the distribution of f values during an iteration of search and uses it to find a good threshold for the next iteration. This is achieved by selecting the threshold that will cause the desired number of pruned nodes to be expanded in the next iteration. If the successors of these pruned nodes are not expanded in the next iteration, then this scheme is able to accurately double the number of nodes between iterations. If the successors do fall within the threshold on the next iteration, then more nodes may be expanded than desired.

Since the threshold is increased liberally, the search cannot immediately halt when a goal is expanded. To verify optimality IDA*$_{CR}$ is required to check that all other nodes in the current iteration do not lead to a solution with cost $C' < C$, where C is the cost of the incumbent solution. This can be done by performing branch-and-bound on the remaining nodes in the final iteration, pruning nodes with $f \geq C$, and updating C if a goal is found with lower cost.

wIDA*$_{CR}$

Using a weighted evaluation function for wIDA* can result in many distinct f values, resulting in too few extra expansions at every threshold. To handle this, we propose combining wIDA* with the threshold setting mechanism of IDA*$_{CR}$. Instead of keeping track of the distribution of f values of the pruned nodes, as in IDA*$_{CR}$, we keep track of the distribution of f' values. This distribution is used to set the threshold of the next iteration to cause the desired number of pruned nodes to be expanded. We will see below that wIDA*$_{CR}$ is superior to regular wIDA* for domains with non-uniform edge costs.

The stopping condition for wIDA*$_{CR}$ is also different than that of IDA*$_{CR}$. To verify w-admissibility in wIDA*$_{CR}$, it is necessary to check that all other nodes in the current iteration do not lead to a solution of cost C', where $C > w \cdot C'$. This can be done by performing branch-and-bound on the remaining nodes in the final iteration, pruning nodes with $w \cdot f \geq C$, and updating C if a goal is found with lower cost. We also use this pruning rule and branch-and-bound approach in our IDEES algorithm, which will be explained later in this paper.

Explicit Estimation Search

The motivating logic behind wIDA* is similar to that of wA*: searching with an evaluation function $f' = g + w \cdot h$ is expected to lead to finding a solution faster as w is increased. Recent work, however, has shown that this is not always the case (Wilt and Ruml 2012). In particular, in domains with non-uniform edge cost, increasing w does not

necessarily cause the search to find a solution faster and in fact can have no correlation with the speed of the search. In non-uniform edge cost domains it is useful to distinguish between the distance-to-go of a node n — the length of the shortest path from n to a goal, and the cost-to-go of a node n — the cost of the lowest-cost path from n to a goal. Several efficient bounded suboptimal search algorithms have been proposed (Thayer, Ruml, and Kreis 2009; Thayer and Ruml 2011) that exploit two heuristic estimates: an estimate of cost-to-go $h(n)$ and an estimate of distance-to-go $d(n)$.

In addition to $d(n)$ and $h(n)$, the state-of-the-art bounded suboptimal search algorithm Explicit Estimation Search (EES, Thayer and Ruml, 2011) also uses information learned during the search to guide the search. EES constructs and uses unbiased but possibly inadmissible estimates of the distance-to-go and cost-to-go, denoted by $\widehat{d}(n)$ and $\widehat{h}(n)$, respectively. Similarly, $\widehat{f}(n) = g(n) + \widehat{h}(n)$ is the inadmissible counterpart of $f(n)$. These online learned estimates are generated using a domain independent method that attempts to generalize the error of the heuristic that is observed in a single step (Thayer, Dionne, and Ruml 2011). EES uses these inadmissible estimates and maintains three relevant data structures to track the following nodes in the open list (OPEN):

$$
\begin{aligned}
best_f &= \operatorname*{argmin}_{n \in OPEN} f(n) \\
best_{\widehat{f}} &= \operatorname*{argmin}_{n \in OPEN} \widehat{f}(n) \\
best_{\widehat{d}} &= \operatorname*{argmin}_{n \in OPEN \wedge \widehat{f}(n) \leq w \cdot \widehat{f}(best_{\widehat{f}})} \widehat{d}(n)
\end{aligned}
$$

$best_f$ provides a lower bound on the cost of an optimal solution, $best_{\widehat{f}}$ provides the estimated optimal solution cost using the corrected cost-to-go estimate and $best_{\widehat{d}}$ is the node that is estimated to be closest to the goal among all nodes that are estimated to lead to goals within the suboptimality bound w. EES uses the following rule to chose which node to expand:

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** $best_f$

In this way, EES attempts to pursue $\widehat{d}$, the node leading to the nearest estimated goal that is estimated to be within the suboptimality bound. While shown to be very effective in a wide range of domains, EES still stores every node it generates in memory.

Iterative Deepening EES

In this section we propose a linear space algorithm that uses the same admissible and inadmissible estimates used by EES, and follows the same intuition of pursuing the nearest goal estimated to be within the bound. We call this algorithm Iterative Deepening EES (IDEES). IDEES takes advantage of both distance-to-go and cost-to-go heuristic estimates ($h(n)$ and $d(n)$, respectively), and their online corrected counterparts ($\widehat{h}(n)$ and $\widehat{d}(n)$, respectively).

```
IDEES(init)
1. inc_f ← ∞
2. t_f̂ ← h(init), t_l̂ ← d(init)
3. min_f ← ∞
4. while inc_f > w · min_f
5.    min_f_next ← ∞
6.    if DFS(init) break
7.    min_f ← min_f_next
8.    t_f̂ ← NEXT-THRESH(data_f̂)
9.    t_l̂ ← NEXT-THRESH(data_l̂)
10. return incumbent

DFS(n)
11. if n is a goal
12.   if f(n) < inc_f
13.      incumbent ← n
14.   return inc_f ≤ w · min_f
15. else if inc_f = ∞ AND (f̂(n) > w · t_f̂ OR l̂(n) > t_l̂)
16.   prune n
17.   min_f_next ← min(min_f_next, f(n))
18. else if inc_f < ∞ AND inc_f ≤ w · f(n)
19.   prune n
20. else
21.   for child ∈ expand(n)
22.      OBSERVE(data_f̂, f̂(child))
23.      OBSERVE(data_l̂, l̂(child))
24.      if DFS(child)), return true
25. return false
```

Figure 1: Pseudo-code for IDEES.

IDEES, like wIDA*, 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 distance analogue of $\widehat{f}$: the potentially inadmissible estimate of the total distance from the root to the nearest solution under n. The first threshold, $t_{\widehat{l}}$, is a distance-based threshold, pruning nodes that are estimated to be on a path to a goal that is more than $t_{\widehat{l}}$ steps from the start state. This is estimated by pruning nodes with $\widehat{l}(n) = depth(n) + \hat{d}(n) > t_{\widehat{l}}$, where $depth(n)$ is the distance between n and the start state. The second threshold, $t_{\widehat{f}}$, is a cost-based threshold, pruning nodes on the basis of an inadmissible heuristic. This is estimated by pruning nodes with $\widehat{f}(n) = g(n) + \widehat{h}(n) > t_{\widehat{f}}$. Thus, in an IDEES iteration, a node n is pruned (i.e, not expanded) if either $\widehat{l}(n) > t_{\widehat{l}}$ or $\widehat{f}(n) > t_{\widehat{f}}$.

Figure 1 lists the pseudo code of IDEES. IDEES maintains three values: the cost based threshold $t_{\widehat{f}}$, the distance based threshold $t_{\widehat{l}}$ and the cost of the best solution seen so far (the incumbent solution), denoted by inc_f and initialized to ∞ (line 1 in the pseudo code from Figure 1). In every iter-

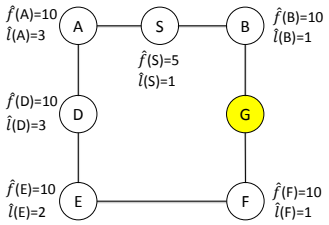

Figure 2: An example in which IDEES behaves well. IDEES would expand only S,B and G while wIDA* would expand all the nodes except B.

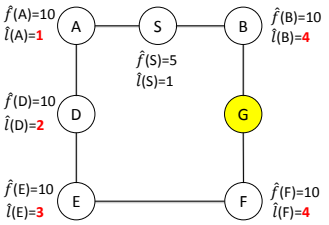

Figure 3: A pathological example of IDEES behavior. IDEES would expand all the nodes, requiring four iterations while wIDA* would require only a single iteration. This occurs when the heuristic estimates are very misleading.

ation of IDEES, a limited depth-first search is executed (line 6, and described in lines 11–25). During an IDEES iteration, nodes with either $\widehat{l}(n) > t_{\widehat{l}}$ or $\widehat{f}(n) > w \cdot t_{\widehat{f}}$ are pruned (lines 15-16). The $\widehat{f}$ of every pruned node is recorded in $data_{\widehat{f}}$, which is a histogram of observed $\widehat{f}$ values and the number of pruned nodes with that value (line 22). This histogram is used to set the $\widehat{f}$ threshold for the next iteration, as is done in IDA*$_{\text{CR}}$(line 8). Similarly, the $\widehat{l}$ of every pruned node is stored in $data_{\widehat{l}}$ (line 23) and is used for setting the $\widehat{l}$ threshold (line 9).

If a goal is found, the value of inc_f is updated (lines 11-14). Once a goal is found, the purpose of the iteration becomes to prove that the goal is w-admissible (or to find a goal that is). Therefore, nodes are pruned or expanded only according to their $w \cdot f$ value. Nodes with $w \cdot f \geq inc_f$ are pruned (line 18-19), while nodes with $w \cdot f < inc_f$ are expanded, regardless of their $\widehat{l}$ and $\widehat{f}$ (line 15).

To determine when it can halt, IDEES maintains in every iteration the minimal f value seen for all pruned nodes (this is stored in $min_{f_{next}}$, see line 17). When IDEES finishes a complete iteration, the value of $min_{f_{next}}$ is stored in min_f, and serves as a lower bound on the optimal solution (line 7). Thus, if $inc_f \leq w \cdot min_f$ IDEES can halt with a w-admissible solution (line 4). Note that even during the iteration, IDEES can halt immediately when an incumbent solution is found for which it holds that $inc_f \leq w \cdot min_f$ (lines 14 and 24).

Figure 2 shows an example in which IDEES behaves well. Assume that S is the start node, G is the goal node and the node ordering used by all algorithms is to evaluate A before B. Each node is labeled with its $\widehat{f}$ value and $\widehat{l}$ value. Running wIDA* will result in expanding all the nodes: S, A, D, E, F, G. By contrast, running IDEES will only result in expanding nodes S, B and G, as the distance threshold $t_{\widehat{l}}$ would be initially set to 1, pruning node A.

Figure 3 shows a pathological case for IDEES. As in Figure 2, S is the start node and G is the goal node and the node ordering used by all search algorithms expands A before B. However, in Figure 3 the $\widehat{l}$ estimates are misleading, causing IDEES to have 4 iterations while wIDA*will only have a single iteration where it expands all the nodes. Note that in general IDEES uses CR to avoid such redundant iterations. However, in this case CR is not able to guess the fitting threshold for $\widehat{l}$ to expand enough nodes to be efficient.

Theoretical Analysis

Next we prove formally that IDEES returns solutions that are w-admissible.

Weighted IDA* simply stops when a goal node is expanded and a bounded suboptimal solution is guaranteed. As with wIDA*$_{\text{CR}}$, stopping IDEES when the first goal node has been expanded may not guarantee w-admissibility. This is especially true in IDEES, since nodes may be pruned according to either $t_{\widehat{l}}$ or $t_{\widehat{f}}$. To overcome this, we must continue the search until the cost of the incumbent solution is within a factor w of the cost of the optimal solution. This is done by keeping track of the minimal f value of all the pruned nodes (min_f), and only halting when the incumbent solution is smaller than $w \cdot min_f$ (line 4 in the pseudo-code shown in Figure 1).

Theorem 2 *IDEES is guaranteed to return a solution that is w-admissible.*

Proof: At every iteration IDEES keeps track of min_f of all nodes that have been pruned and terminates only when it has found a goal node n such that $f(n) \leq w \cdot min_f$ (lines 4, 6, 14 and 17). Therefore, when IDEES terminates and returns the incumbent, it is w-admissible. □

Setting the Cost and Distance Thresholds

A key component that affects the performance of IDEES is how the $t_{\widehat{f}}$ and $t_{\widehat{l}}$ thresholds are updated. In our experiments and in the description above, we have used the same approach taken by IDA*$_{\text{CR}}$ by maintaining a distribution of observed $\widehat{f}$ and $\widehat{l}$ values and choosing the thresholds such that the number of nodes that will be expanded in the next iteration and were pruned in the previous iteration is doubled. However, other options to set the thresholds exist. For example, one can set the $t_{\widehat{f}}$ threshold to the minimum $\widehat{f}(n)$ of all pruned nodes for each iteration, and similarly set $t_{\widehat{l}}$ to the minimum $\widehat{l}(n)$ of all pruned nodes for each iteration. More sophisticated threshold setting mechanisms might also be considered (Burns and Ruml 2012).

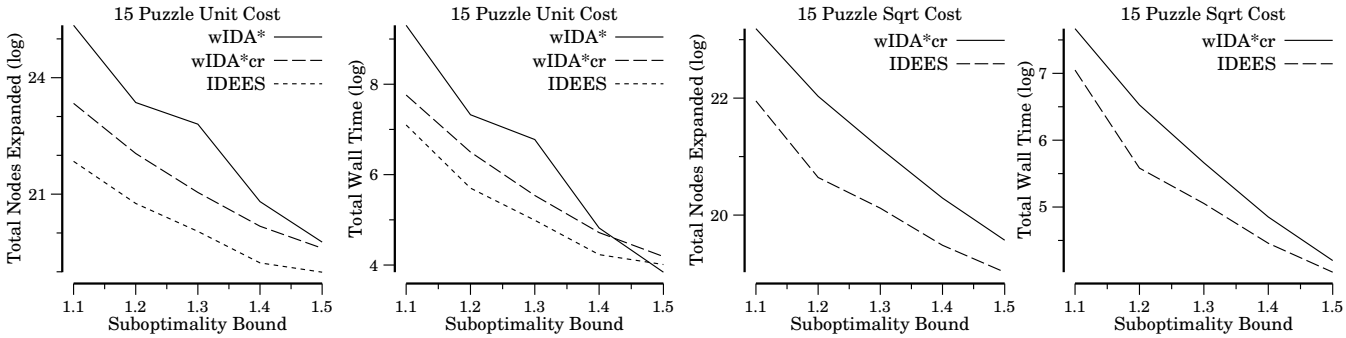

Figure 4: Comparison between IDEES, wIDA* and wIDA*$_{CR}$ on Korf's 100 15 puzzles. Plots show the total time in seconds and total number of nodes expanded for solving all 100 instances.

	Weight	Time	Expanded
wIDA*$_{CR}$	1.2	186,061	32,077,127,188
	1.3	37,158	6,357,882,067
	1.4	7,602	1,297,420,767
	1.5	1,531	259,623,405
IDEES	1.2	5,676	355,001,273
	1.3	419	29,998,895
	1.4	217	16,148,375
	1.5	127	9,308,167

Table 1: Performance summary on Heavy Pancakes. Table shows the total time in seconds and the total number of nodes expanded for solving all 100 instances.

Experimental Results

For our experiments we implemented IDEES and wIDA* and compared them on two permutation puzzle domains with a variety of edge cost functions. We use many of the optimizations recommended by Burns et al. (2012) and Hatem et al. (in press) . For the algorithms that incorporate the CR threshold setting technique we set the number of buckets to 100. All of our experiments were run on a machine with a dual-core 3.16 GHz processor and 8 GB of RAM running Linux. We implemented our algorithms in Java and compiled with OpenJDK 1.6.0.24.

Unit Costs

To see if IDEES can be competitive on a standard unit cost benchmark we first compared IDEES to wIDA* and wIDA*$_{CR}$ on the 15 puzzle with unit costs and the Manhattan distance heuristic. In these experiments we used suboptimality bounds ranging from 1.1 to 1.5. It has been shown that Weighted A* performs better than EES on this domain even though EES expands fewer nodes. This is because of the significant overhead associated with managing the multiple priority queues that EES uses. IDEES does not have this overhead and therefore has a much higher expansion rate.

The first two plots in Figure 4 show the total number of nodes expanded and the total wall time for solving all instances of Korf's 100 15 puzzles. In all cases IDEES expands many fewer nodes and solves all instances in less time than IDA*$_{CR}$. IDEES is able to prune parts of the search space that have higher distance-to-go, finding w-admissible solutions faster than wIDA*$_{CR}$. For most suboptimality bounds, wIDA*$_{CR}$ is faster than plain wIDA*. For certain suboptimality bounds IDA* has extremely fast expansion rates and is able to solve problems faster.

Real Costs

We can modify the edge cost function of the 15 puzzle to have real values by taking the square root of the tile as the cost for moving it. For example, exchanging the blank tile with the 3 tile would cost $\sqrt{3}$. IDA* performs poorly on domains where edge costs do not fall within a narrow range of integers — this was the original motivation for the threshold setting technique of IDA*$_{CR}$ (Sarkar et al. 1991). Our implementation of IDEES uses this same threshold setting technique for setting $t_{\widehat{f}}$ and $t_{\widehat{l}}$. The next two plots in Figure 4 show the total number of nodes expanded and the total wall time for solving all instances of Korf's 100 15 puzzles with the sqrt cost function. In all cases IDEES expands fewer nodes and solves all instances in less time.

We also evaluated the performance of these algorithms on the heavy pancake puzzle using the gap heuristic. In this domain each pancake, in a single stack of pancakes, begins in an initial position and has a desired goal position. The objective is to *flip* portions of stack until all pancakes are in their goal positions. The gap heuristic is a type of landmark heuristic that counts the number of non adjacent pancakes or "gaps" above each pancake in the stack. The gap heuristic has been shown to be very accurate; outperforming abstraction based heuristics.

In heavy pancakes, each pancake is given an id $\{1...N\}$ where N is the number of pancakes and the cost to flip a pancake is the value of its id. This produces a wide range of integer edge costs. In our experiments we used 100 random instances of the heavy pancake puzzle with N set to 14. wIDA*$_{CR}$required over 2 days to solve all instances with a suboptimality bound of 1.2 and over 10 hours to solve all instances with a suboptimality bound of 1.3. IDEES is over 32 times faster, solving all instances with a suboptimality bound of 1.2 in just 1.5 hours, and is significantly faster with other bounds; expanding many fewer nodes than wIDA*. Ta-

ble 1 shows the total number of nodes expanded and the total wall time for solving all instances.

We found IDEES to perform poorly compared to wIDA* on large unit cost pancake puzzles using the gap heuristic and an inadmissible heuristic learned using single step error. In this setting the heuristic is very accurate and the inadmissible heuristic overestimates. IDA* is able to find solutions quickly, being guided by the accurate heuristic while IDEES eagerly prunes many possible paths to the goal because it is guided by an inadmissible heuristic that makes these paths look unpromising. IDEES is most advantageous when $d(n)$ provides additional information beyond $h(n)$.

Discussion

Like IDA*, IDEES suffers on search spaces that form highly connected graphs. Because it uses depth-first search, it cannot detect duplicate search states except perhaps those that form cycles in the current search path. This problem, which is intrinsic to IDA*, motivates the use of a transposition table or closed list. IDEES combined with a closed list is easier to implement than EES, because it does not require complex datastructures. Furthermore, IDEES may even perform faster than EES, as the overhead per node is smaller. This is an exciting direction for future work.

However, even with checking for duplicates via a closed list, the search can still perform poorly if there are many paths to each node in the search space. If the search first expands a node via a suboptimal path, it will have to re-expand it again later when it finds the same state by a better path. Because of this, iterative deepening techniques like IDA* or IDEES are not suitable for every domain.

In our implementation of IDEES, we prune nodes according to $t_{\widehat{f}}$ and $t_{\widehat{l}}$ thresholds. At each iteration we update these thresholds using the CR technique. We tried a few variations for updating the thresholds, including taking the minimum value observed during an iteration and only updating the $t_{\widehat{l}}$ threshold when min_f does not change. We found that IDEES performed best when using the CR technique and updating both thresholds at every iteration regardless of changes to min_f. When IDEES finds an incumbent, it must visit all nodes n such that $w \cdot f(n) < f(incumbent)$ in order to prove the incumbent is w-admissible. When IDEES finds an incumbent, one potential strategy is to prune all nodes n such that $w * f(n) \geq f(incumbent)$. However, in our implementation we still prune such nodes according to the $t_{\widehat{f}}$ and $t_{\widehat{l}}$ thresholds as it is possible for IDEES to find an incumbent with better f.

Recursive Best First Search (RBFS) is another linear space search algorithm (Korf 1993). One can construct another bounded suboptimal search by weighting the heuristic in RBFS (wRBFS). We would expect wRBFS to perform significantly worse than IDEES because it does not take advantage of distance-to-go estimates or accurate, inadmissible heuristics. Investigating this empirically and seeing how the ideas of EES and IDEES might be combined with RBFS is an interesting direction for future work.

Conclusion

EES is an effective bounded suboptimal search that is capable of exploiting inadmissible heuristics and distance-to-go estimates to find solutions faster than other bounded suboptimal search algorithms such as wA*. However, EES stores every unique node that it generates, its implementation is non-trivial, and its node expansions can be expensive compared to other algorithms such as wIDA* and even wA*. In this paper we presented IDEES, a linear space analogue to EES. IDEES is simple to implement and because it does not maintain any priority queues, its expansion rate is similar to wIDA*. The results of our experiments show that IDEES can expand fewer nodes and find w-admissible solutions faster than wIDA*. It also demonstrates that the ideas of EES extend to linear space heuristic search algorithms.

Acknowledgments

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

References

Burns, E., and Ruml, W. 2012. Iterative-deepening search with on-line tree size prediction. In *Learning and Intelligent OptimizatioN Conference (LION)*, 1–15.

Burns, E.; Hatem, M.; Leighton, M. J.; and Ruml, W. 2012. Implementing fast heuristic search code. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*.

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. in press. Faster problem solving in Java with heuristic search. *IBM developerWorks*.

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.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computation issues in heuristic problem solving. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 12–17.

Samadi, M.; Felner, A.; and Schaeffer, J. 2008. Learning from multiple heuristics. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*.

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 International Conference on Automated Planning and Scheduling (ICAPS)*.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Thayer, J. T.; Dionne, A.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*.

Thayer, J. T.; Ruml, W.; and Kreis, J. 2009. Using distance estimates in heuristic search: A re-evaluation. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*.

Wilt, C. M., and Ruml, W. 2012. When Does Weighted A* Fail? In *Proceedings of the Symposium on Combinatorial Search (SoCS)*.