Informed Backtracking Beam Search

Christopher Wilt
Department of Computer Science
University of New Hampshire
wilt@cs.unh.edu

Abstract

Beam searches are a very effective algorithm for solving problems that prove intractable to complete algorithms like weighted A* and greedy search. Unfortunately, the inadmissible pruning that is the hallmark of a beam search makes the algorithm incomplete. Existing complete extensions to beam search expand pruned nodes systematically according to generation order. Selecting pruned nodes to expand based upon generation order often causes the expansion of the most promising nodes to be delayed. This can cause the complete backtracking beam searches to expend unnecessary effort exploring less promising parts of the search space leading to delayed solutions, lower quality solutions, or both. In this paper, we show that considering heuristic information when deciding where to backtrack can significantly improve the performance of backtracking beam searches. Although the beam searches using informed backtracking outperform the other kinds of restarting beam searches, none of the backtracking beam search algorithms are competitive with weighted A* and other best-first searches.

Introduction

Beam searches are a very effective algorithm for solving problems that prove intractable to complete algorithms like weighted A* (Pohl 1970) and greedy search (Doran and Michie 1966). In order to make the search more tractable, beam searches only consider the most promising nodes, and eliminate the less promising nodes from contention.

The inadmissible pruning done by beam searches make the algorithm incomplete. One option that can ameliorate this problem is to never permanently inadmissably prune a node, but instead to simply delay expansion of that node. Two existing extensions to beam search are beam search using limited discrepancy backtracking (BULB) (Furcy and Koenig 2005a) and beam-stack search (Zhou and Hansen 2005). These algorithms do bookkeeping to track where nodes were pruned, and if the search ever runs out of nodes to expand without finding a solution, the algorithms expand the pruned nodes. These existing extensions to beam search expand pruned nodes systematically according to expansion order.

Selecting pruned nodes to expand based upon expansion order often causes the expansion of the most promising nodes to be delayed. Considering heuristic information when deciding where to backtrack can significantly improve the performance of restarting beam searches. Despite this improvement, the beam searches with heuristically informed backtracking are not as effective as weighted A* and other best-first search algorithms.

Previous Work

Previous attempts to ameliorate the problems associated with inadmissible pruning in beam search can be classified into two broad categories. The first approach, called complete anytime beam (CAB) search was introduced by Zhang (Zhang 1998). This algorithm calls a beam search, and if the beam search fails to find a solution, it tries again with a larger beam. Eventually this algorithm will find a solution because at some point the beam will become so large that nothing important is pruned. Although this approach is simple and reasonably effective in some domains like grid path planning, it has the serious drawback that the amount of space required to search to a given depth increases as time passes. Another drawback associated with this approach is that no information from the first searches is used in the subsequent searches, resulting in wasted computational effort.

An alternative approach is to expand pruned nodes without restarting the search. This is the approach taken in BULB (Furcy and Koenig 2005a) and beam-stack search (Zhou and Hansen 2005). Both beam-stack search and BULB reconsider pruned nodes without restarting the search, but they do so in a very systematic manner that does not consider heuristic information. When selecting pruned nodes to expand, BULB begins with the first nodes that were pruned. Beam-stack search uses the opposite approach, expanding nodes that were most recently pruned. Both algorithms approaches allow a complete exploration of the search space, but this completeness comes at a cost. When deciding where to backtrack, BULB and beam-stack search use expansion order to decide which pruned nodes to expand, which means that the most promising pruned nodes might have to wait a significant amount of time before finally being expanded. We propose an alternative extension to beam search that incorporates the heuristic evaluation of a node, $h(n)$, when deciding which pruned nodes to expand. We evaluated this new algorithm by comparing it to existing complete beam searches on two domains that are particularly difficult for incomplete beam searches, grid path planning and dynamic robot path planning. Our empirical

results show that beam searches that backtrack to the best nodes according to $h(n)$ outperform all other kinds of backtracking beam search, but that these algorithms fall short of the performance of weighted A* and other alternative greedy search algorithms.

Beam Search with Informed Backtracking

BULB and beam-stack search opt to begin backtracking at the top or bottom of the search tree because those nodes can easily be regenerated and expanded. This allows the pruned nodes to be deleted since they can be regenerated later. The simplest way to incorporate a more informed backtracking into a beam search is to never permanently prune a node, but instead to only temporarily delay the expansion of the less promising nodes. The pruned nodes are all put in a collection until the beam search terminates without a solution, at which point the most promising nodes are removed from the collection until a new beam is full (contains as many nodes as the beam width), and the search is resumed using the new beam. Depending on how often the search backtracks, the collection should be either an unordered list, which incurs a linear cost when backtracking but has constant time insertions, or a some kind of sorted data structure with logarithmic inserts and logarithmic removals. Which approach is preferable depends on the ratio of backtracks to nodes, which will vary across domains.

Although the algorithm is straightforward, it does have a single very important parameter. In order to select the most promising nodes, there must be a comparator that is capable of identifying the most promising nodes. There are a number of logical choices for defining the most promising nodes. The simplest sort predicates compare the nodes on the heuristic evaluation of the cost to the cheapest goal, $h(n)$ or the estimated cost of the cheapest solution going through that node, $f(n)$. These algorithms are called informed backtracking on $h(n)$ and $f(n)$ respectively. One advantage of backtracking on $f(n)$ is that when the search returns a solution, the lowest $f(n)$ from all of the pruned nodes as well as the nodes remaining in the beam provides a lower bound on the optimal cost solution. Other algorithms can also provide this bound, but finding the pruned node with the smallest $f(n)$ is not easy for the other algorithms, since the pruned nodes are sorted according to some other criteria.

Another option is to sort the nodes according to the quality of the node relative to the quality of the nodes that were actually selected for expansion. Figure 2 shows a picture detailing the concept of indecision. For each depth layer, the $f(n)$ value is noted for the most promising node at that level. For all other nodes, their quality is $f(n) - f(n)_{best_expanded}$. When using this function to evaluate nodes, the algorithm is called informed backtracking on indecision min. A similar metric is used to compare the $f(n)$ value of each node to the lowest quality node at its level that won expansion. The evaluation function in that case is $f(n)$ - $f(n)_{worst_expanded}$. This algorithm is called informed backtracking on indecision max.

Another way to sort pruned nodes is to compare their quality relative to other nodes with the same estimated number of steps to the goal, $d(n)$. With this evaluation crite-

```
INFORMED BACKTRACKING BEAM SEARCH(beam_width)
1  current_layer = {initial}
2  considered = {initial}
   // considered is a collection of generated nodes
3  next_layer = ∅
4  extra_nodes = ∅
5  solution = NIL
6  while (solution not found and
   (current_layer ≠ ∅ or extra_nodes ≠ ∅))
7       solution = expand_layer
8  Return solution

EXPAND_LAYER
1  if current_layer == ∅
2       fill current_layer with nodes from extra_nodes
3  For all x ∈ current_layer
4       expand_node(x)
5  current_layer = next_layer
6  next_layer = ∅

EXPAND_NODE(node)
1   children = expand (node)
2   for each child in children
3        if there is space in next_layer and child ∉ considered
4             insert child into next_layer
5             Add child to considered
6        elseif next_layer contains beam_width nodes and
         child ∉ considered
7             Add child to considered
8             if child is better than worst item in next_layer
9                  removed = worst item from next_layer
10                 Add removed to extra_nodes
11                 Remove removed from next_layer
12                 Add child to next_layer
13            else
14                 Add child to extra_nodes
15       elseif there is space in next_layer and child ∈ considered
16            incumbent = child in considered
17            if child better than incumbent
18                 replace incumbent with child in hash table
19                 remove incumbent from next_layer
20                 Add child to next_layer
21       elseif there is no space in next_layer and child ∈ considered
22            incumbent = child in considered
23            if child better than incumbent and
              incumbent ∈ next_layer
24                 replace incumbent with child in hash table
25                 remove incumbent from next_layer
26                 Add child to next_layer
27            elseif child is better than worst item in next_layer
28                 replace incumbent with child in hash table
29                 removed = worst item from next_layer
30                 Add removed to extra_nodes
31                 Remove removed from next_layer
32            elseif child is worse than worst item in next_layer
33                 replace incumbent with child in considered
34                 replace incumbent with child in extra_nodes
```

Figure 1: Informed Backtracking Beam Search Algorithm

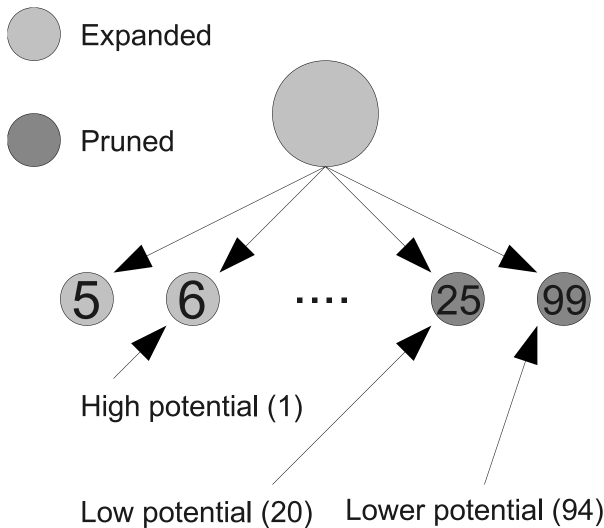

Figure 2: Indecision

ria, for each $d(n)$ value, the algorithm tracks the best $f(n)$ found on any node with that $d(n)$ value. When nodes are pruned, their quality is defined to be the difference between the node's $f(n)$ value and the best $f(n)$ value found with the same $d(n)$ value, or $f(n) - f(n)_{best_with_same_d}$.

Another option is to dynamically select the most effective sort predicate. Selecting the sort predicate can be viewed as a K-arm bandit problem where each sort predicate is an arm, and the reward provided by selecting each action is how deep the search is able to go with nodes selected according to that sort predicate. Unfortunately, this method is of little value since the most effective policy is to always sort the nodes on their heuristic value.

The algorithm bears a striking resemblance to MSC-kWA* (Furcy and Koenig 2005b), and for certain parameter settings, the algorithms are identical. If the beam search uses f to order the rejected nodes, it is the same as MSC-kWA* where the weight is 1.0 and the size of the commit list is the same as K. While MSC-kWA* modifies expansion order by changing the number of nodes that are expanded at one time and changing the size of the pool of nodes to expand, informed backtracking modifies expansion order by modifying how the pruned nodes are to be sorted in ways that go beyond changing the weight in f' as MSC-kWA* does.

There is significant reason to believe that beam searches using informed backtracking will outperform both varieties of backtracking beam search as well as complete anytime beam search. When selecting which nodes to backtrack to, BULB and beam-stack search both select nodes based upon search expansion order, making no consideration for heuristic evaluation which beam search using informed backtracking does. The heuristic often contains valuable information about a node's potential, helping to differentiate between promising nodes and nodes that probably are not worth further consideration. BULB and beam-stack search disregard this information, and it is no surprise that beam searches using informed backtracking are able to find high quality solutions in less time by using the heuristic when choosing pruned nodes to expand. Another advantage of beam searches with informed backtracking is that they do not redo any work, as is the case with complete anytime beam search.

Informed Backtracking Results

Two domains that proved particularly problematic for beam searches are grid path planning and dynamic robot path planning (Wilt, Thayer, and Ruml 2010). In both of those domains, beam searches using sufficiently small beams often fail to find any solution because they prune all nodes that lead to a goal. In domains that lack dead ends like sliding tile puzzles and pancake puzzles, beam searches that do no backtracking at all are extremely effective. Since a backtracking and a non-backtracking beam search perform exactly the same if the ordinary non-backtracking beam search finds a solution, we shall only consider domains where the non-backtracking beam searches often fail to find solutions.

Algorithms tested were run on Dell Optiplex 960, Core2 duo E8500 3.16 GHz, 8 GB RAM machines running 64-bit Linux. The algorithms were implemented in Objective Caml and compiled to native binaries. Beam searches were all run with beam widths of 50000, 10000, 5000, 1000, 500, 100, 50, 10, 5, and 3. Algorithms were allowed to run for 300 seconds. Algorithms that did not find a solution in the required time period are considered to have failed to find a solution.

The grids considered in grid path planning were 2000 cells across and 1200 cells tall. The obstacles were randomly placed. In each instance, approximately 35% of the cells were blocked, so there are about 840,000 states. Movement was allowed in the four cardinal directions. In unit cost grid path planning, each move costs 1. In life cost grid path planning, the cost of the move is the same as the y coordinate. Consequently, grid path planning with life costs the shortest path is rarely the same as the same as the best path. In both variants, the heuristic calculated the distance of the optimal path assuming no obstacles. In the grid path planning domains, many states have the same f estimate, so we break ties in favor of nodes with low h.

The dynamic robot path planning domain is almost identical to that used by Likhachev, Gordon, and Thrun (2003) (we used different constants to discretize the space). With our discretization constants, there are approximately 2^{11} states in the state space. In this domain, the goal is to drive a robot from the start pose (velocity and location) to the goal pose by manipulating the heading and velocity. Constraints are placed upon what moves are allowed based upon physics. For example, there are limits on the acceleration and the robot's turn radius depends on how fast it is moving. For a heuristic, we remove these constraints and assume the robot can move at its maximum velocity along the shortest path between its current location and the goal. We randomly placed lines throughout the space to serve as obstacles. The dynamic robot path planning domain poses special challenges to beam searches because the space not only has nodes that do not have any children, but it also has nodes beneath which there are children, but zero goals. If the beam search fills the entire beam with these nodes that do not lead

to a goal, the algorithm terminates without returning a solution.

Before comparing informed backtracking to BULB, beam-stack search, or complete anytime beam search, we must first compare the different informed backtracking predicates against one another to find what is the most effective way to evaluate pruned nodes. As can be seen in Figure 3 repopulating the beam with nodes with the lowest $h(n)$ value is the clear choice for finding solutions quickly. In the plots in this figure, as well as all subsequent plots shown, the X axis is the log base 10 of the raw CPU time. The Y axis plots the normalized solution quality. Finding the best solution of all algorithms tested earns a score of 1. Failing to find a solution earns a score of 0. Finding a solution that is not as good as the best known solution earns a score proportional to how suboptimal the solution in question is. Anytime algorithms were terminated after finding their first solution. For each algorithm/parameter pair, the results from all instances were averaged to place each point on the graph. The most effective algorithms are colored to make them more distinct.

In dynamic robot path planning and grid path planning with unit cost, selecting pruned nodes to expand based on $h(n)$ is always the best choice. In grid path planning with life costs, if higher quality solutions are desired the pruned nodes should be selected in order of their $f(n)$. The more exotic ordering predicates based upon indecision did not perform anywhere near as well as simply expanding nodes in $h(n)$ or $f(n)$ order, as the best these node evaluation criteria were able to achieve is parity. In the average case, the best choice for finding a solution quickly is unquestionably $h(n)$.

Backtracking Results

Figure 4 shows the results of the most effective informed backtracking beam searches along with existing alternatives. We normalize the quality of a solution as between 0 and 1. We perform this normalization by dividing the cost of the best solution found for the problem by the cost of the solution returned by an algorithm. Failing to find a solution earns a score of 0 and finding the best solution over all algorithms earns a score of 1. We calculate this number for each instance, and then take the average across all instances with a selected beam width to produce one point on the graph. For a particular algorithm, we plot a point for all beam widths considered and connect each point to the two beam widths that are adjacent to it. Although all of the algorithms are anytime algorithms (or easy to convert into anytime algorithms), we terminated all algorithms after they returned their first solution. Some algorithms failed to find a solution within the 300 second time limit, so they were charged with the entire 300 seconds they consumed, and the solution was assessed to be of 0 quality.

As we recall from the previous section, in the dynamic robot path planning domain and in grid path planning with unit cost, the most effective way to rank pruned nodes was the $h(n)$ function. For the grid path planning with life costs, the evaluation function was either $h(n)$ if a solution is desired as quickly as possible or $f(n)$ if we are looking for a high quality solution.

In the dynamic robot path planning domain, shown in the left pane of figure 4 we can see that none of the other complete beam search variants beats backtracking on $h(n)$ in terms of both finding solutions quickly and finding high quality solutions, as long as the appropriate beam width is selected.

In grid path planning with life costs, shown in the center pane of figure 4, backtracking on $h(n)$ is more effective than any existing complete alternative, as long as an appropriate beam width is selected. None of the backtracking beam searches can compete with backtracking on $f(n)$ when evaluating in terms of solution quality.

In grid path planning with unit cost, shown in the right pane of figure 4, we can see that restarting on h is more effective at finding solutions quickly, but there is a tiny region where complete anytime beam search and beam-stack search provide a better time-solution quality trade off.

Comparison with Alternative Algorithms

Although we have just seen that informed backtracking beam searches outperform BULB, beam-stack search and complete anytime beam search, one very important question that has yet to be considered is how the informed backtracking beam searches compare to other algorithms like greedy search, weighted A*, and MSC-kWA*.

When comparing against MSC-kWA* a major issue is selecting parameters. We ran the algorithm with a variety of weights and sizes of k and the commit list size and the single most important parameter proved to be the weight, dominating the effects of the other parameters. For this reason, we allow the weight to vary in the plot for MSC-kWA*, and kept the other parameters constant. We observed that the performance of MSC-kWA* was fairly constant no matter what the size of the commit list was and the k used, but using a commit list size of 10 and a k of 3 produced as good a solution-time trade-off as any other k/commit list size pair. For this reason, we kept the size of the commit list constant at 10 and kept k constant at 3 and allowed the weight to vary.

In the left pane of Figure 5 we can see that weighted A* is the clear choice in the dynamic robot path planning domain, finding high quality solutions faster than any alternative as long as the weight is not too low. In the grid path planning domains with both life and unit costs weighted A* is still a very effective algorithm, but its dominance is less complete in these domains. We can see in grid path planning with life costs that with high weights, MSC-kWA* provides better solutions given the same amount of time. In grid path planning with unit costs, seen in the right pane of Figure 5, there is a small region where MSC-kWA* provides better solutions in the same amount of time and a different region where informed backtracking beam searches using $h(n)$ provide the best solutions in the same amount of time. Although these small regions exist, weighted A* is more robust, consistently providing a very strong, if not the best, time-solution quality trade-off.

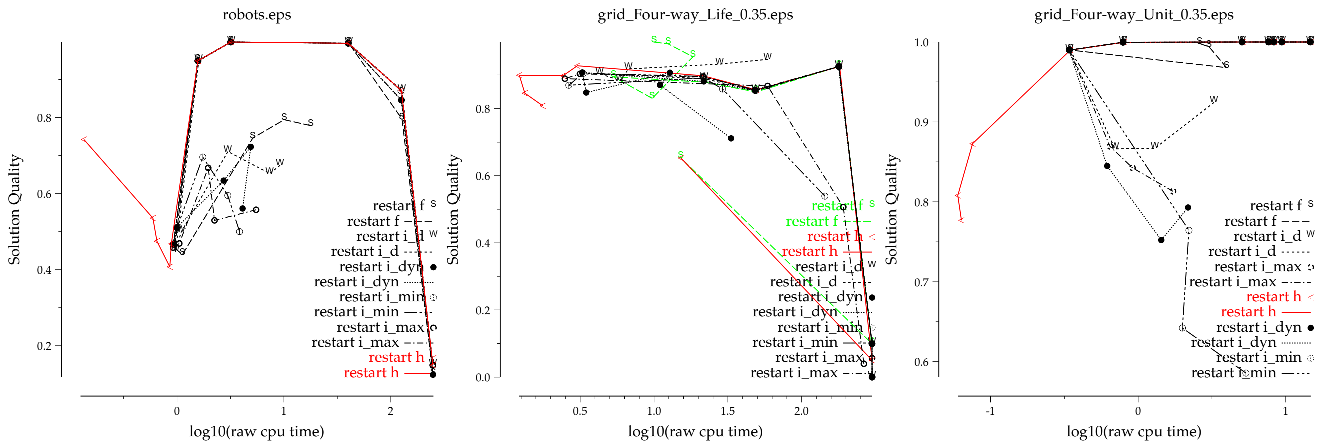

Figure 3: Informed Backtracking with various evaluation functions

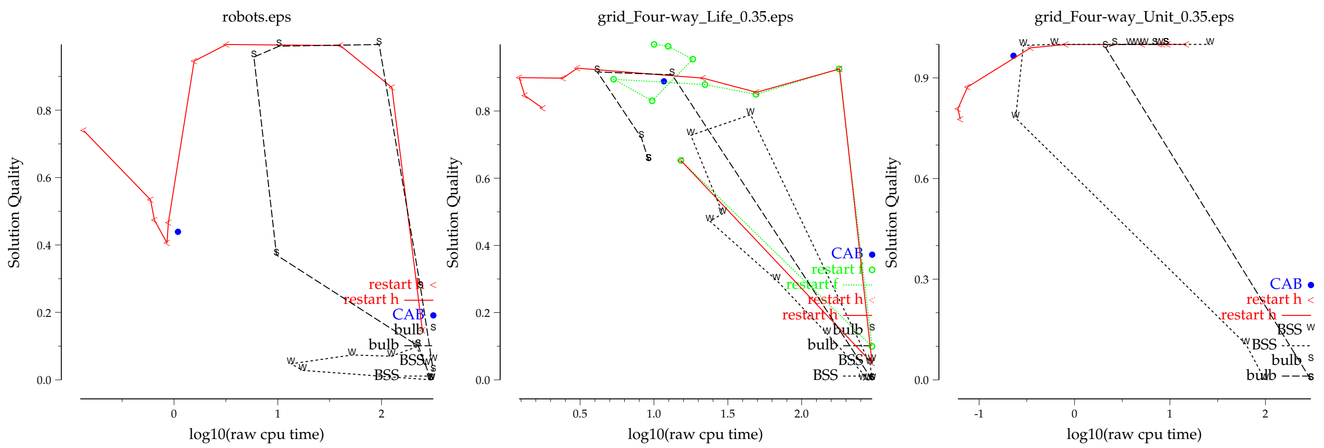

Figure 4: Informed Backtracking with other complete beam searches

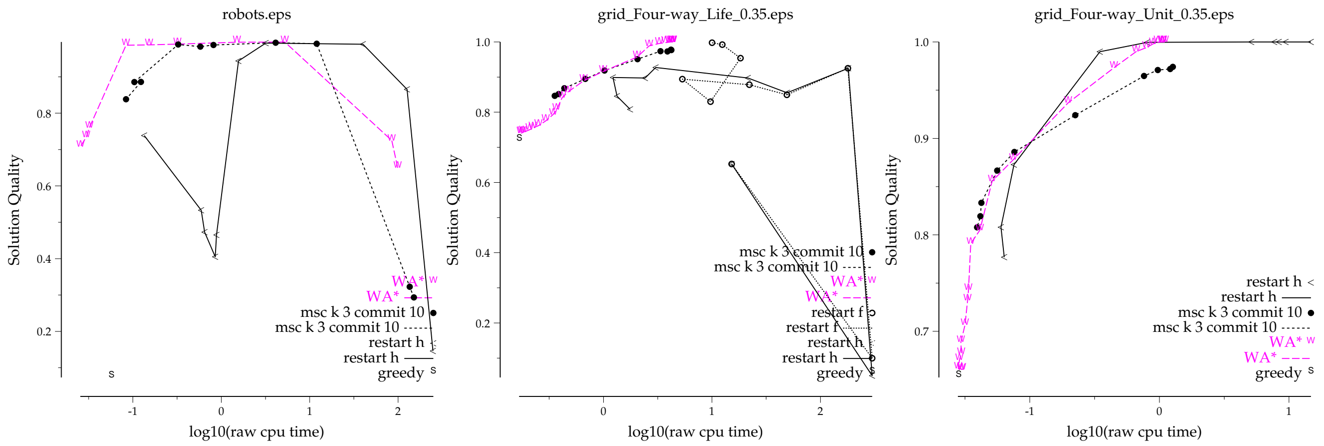

Figure 5: Informed Backtracking with other search algorithms

Discussion

One very important question is why backtracking on $h(n)$ is so effective. Intuitively, it is clearly better to use heuristic information when deciding where to backtrack to, which explains why informed backtracking works so much better than BULB or beam-stack search. It is also no surprise that beam search using informed backtracking outperforms complete anytime beam search, since beam search using informed backtracking never re-expands nodes, as complete anytime beam search does.

What is less clear is why $h(n)$ is so much better than $f(n)$ and the indecision-based evaluation functions. If the beam search repopulates the beam using only the heuristic evaluation of the pruned nodes, the beam search is behaving similarly to a greedy search, and in these three domains greedy search proves quite effective at finding solutions quickly, always finding a solution in less time than any kind of informed backtracking beam search. Backtracking on $h(n)$ forces the backtracking phase of the algorithm to be extremely greedy, which seems to help it find a solution quicker than the more deliberative evaluation functions that all use various derivatives of $f(n)$.

Future Work

The informed backtracking algorithm, as proposed, can consume an space proportional to the size of the state space. The algorithm keeps all nodes ever generated, and never deletes any nodes, no matter how unpromising the node may appear. One major advantage of BULB and beam-stack search is that they operate in space linearly proportional to the maximum depth to which the search progresses. The informed backtracking beam search algorithm could be modified to only require storing the nodes that were actually expanded, which would incur the same cost that BULB and beam-stack search pay which is generating the same node from the same parent multiple times.

This can be accomplished by tracking the quality of the best child not expanded for all nodes that are expanded. If the parent nodes are kept sorted according to the quality of their most promising pruned child, finding the most promising pruned node to expand is simply a question of regenerating the children of the parent node with the most promising pruned child and finding the child that was pruned. In the event that children nodes have equal quality, the nodes should be sorted lexicographically to impose a full ordering which will allow the algorithm to uniquely identify which child node to keep. This will allow the algorithm to reduce its memory consumption by a factor of the branching factor of the domain in consideration, but comes at the cost of requiring nodes to be re-expanded.

Conclusion

Incorporating heuristic information when deciding which pruned nodes to expand is clearly a good decision, as it outperforms other varieties of backtracking beam search in terms of both time to solution and quality of solutions found. Although informed backtracking beam searches are the most successful kind of backtracking beam search, the algorithm falls significantly short of the performance of alternatives like weighted A*.

Although beam search with informed backtracking does not work as well as weighted A* on problems like dynamic robot path planning or grid path planning, beam searches with informed backtracking maintain one of the core advantages of beam searches, which is that beam searches scale very gracefully. Weighted A* derivatives work poorly on the 7x7 sliding tile puzzle, failing to find solutions a significant amount of the time. Both basic beam searches and restarting beam searches are able to solve these large problems that are out of reach for weighted A* type searches.

References

Doran, J. E., and Michie, D. 1966. Experiments with the graph traverser program. In *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 235–259.

Furcy, D., and Koenig, S. 2005a. Limited discrepancy beam search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 125–131.

Furcy, D., and Koenig, S. 2005b. Scaling up WA* with commitment and diversity. In *International Joint Conference on Artificial Intelligence*.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proceedings of the Seventeenth Annual Conference on Neural Information Porcessing Systems (NIPS-03)*.

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

Wilt, C.; Thayer, J.; and Ruml, W. 2010. A comparison of greedy search algorithms. In *Symposium on Combinatorial Search*.

Zhang, W. 1998. Complete anytime beam search. In *Proceedings of AAAI-98*, 425–430.

Zhou, R., and Hansen, E. A. 2005. Beam-stack search: Integrating backtracking with beam search. In *Proceedings of ICAPS-05*.