Using Distance Estimates in Heuristic Search

Jordan T. Thayer and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
jtd7, ruml at cs.unh.edu

Abstract

This paper explores the use of an oft-ignored information source in heuristic search: a search-distance-to-go estimate. Operators frequently have different costs and cost-to-go is not the same as search-distance-to-go. We evaluate two previous proposals: dynamically weighted A* and A$^*_\epsilon$. We present a revision to dynamically weighted A* that improves its performance substantially in domains where the search does not progress uniformly towards solutions, and particularly in certain temporal planning problems. We show how to incorporate distance estimates into weighted A* and improve its performance in several domains. Both approaches lead to dramatic performance increases in popular benchmark domains.

Introduction

Heuristic search is used to solve a wide variety of problems. When sufficient resources are available, optimal solutions can be found using A* search with an admissible heuristic (Hart, Nilsson, and Raphael 1968). In practical settings one is often willing to accept suboptimal solutions in order to reduce the computation required. In this paper, we consider the setting in which one wants the fastest possible search where the solution is to be within a bounded factor of the optimal solution.

The purpose of bounded suboptimal search is not really to find a solution whose cost is within a given bound of the optimal, but rather, given a desired quality bound, to produce an acceptable solution as quickly as possible. These solutions should be of high quality but not at the expense of speed. To do this, bounded suboptimal algorithms make a controlled transition between performing like A* when the bound is tight and greedy search when the bound is lax.

Sometimes all actions have the same cost and becoming greedy with respect to the cost of a solution is equivalent to becoming greedy with respect to the length of a solution. Searches that consider both the estimated cost-to-go, given by a heuristic evaluation function h, as well as the search-distance-to-go, given by a distance estimation function d, can perform better in domains where h and d differ. In these domains a search that is greedy with respect to solution cost may take longer than a search that focuses on finding the

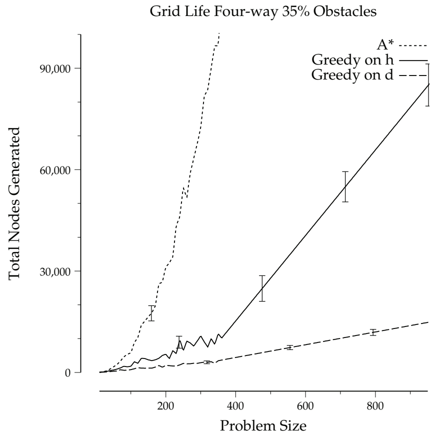

Figure 1: Greedy search on d outperforms search on h.

nearest solution. Figure 1 demonstrates the extreme case, where no bound is placed on the quality of the returned solution in a grid pathfinding problem. We see that not only do we find solutions faster when greedily searching on d than when we do on h, but that the performance gap between these two approaches increases with problem size. Domains where solution cost and length differ are common, including popular benchmarks such as temporal planning and many variations of pathfinding.

And yet many algorithms fail to take advantage of the information provided by d. Although there were several proposals in the 70s and 80s, A$^*_\epsilon$ (Pearl and Kim 1982) and dynamically weighted A* (Pohl 1973) among them, exploitation of distance estimates has fallen out of favor. This is caused in part by the poor performance of these algorithms. We explain their poor performance and show how to correct it. Additionally, we provide a technique for incorporating distance-to-go information in weighted A* and show that searches which incorporate d perform well on important problems, allowing us to both solve problems optimally faster than A* and avoid poor performance at high weights in temporal planning.

A$^*_\epsilon$

A$^*_\epsilon$ expands nodes that are as close to a solution as possible while still guaranteeing w-admissibility. It maintains two ordered lists, the first is identical to that used by A*, where nodes are ordered according to the cost function $f_{A^*}(n) = g(n) + h(n)$, where g is the cost of travelling to a node from the root of the search. At the front of the open list is the node with minimum f_{A^*}, f_{min}. In order to select nodes that are close to a goal, A^*_ϵ maintains a list of nodes sorted on d, called the focal list. The node at the front of focal is the node with minimum d, d_{min}. To find a solution, it is expanded, and its children are placed into open until d_{min} is a solution.

To ensure that the solution returned by A^*_ϵ is w-admissible, A^*_ϵ only places those nodes that have f_{A^*} values that are within a factor w of f_{min} onto the focal list. This means that $f_{A^*}(d_{min}) \leq w \cdot f_{A^*}(f_{min})$, given by the construction of the focal list. If we are using an admissible heuristic h, we know that $f_{A^*}(f_min) \leq f_{A^*}(opt)$ where opt is a solution with optimal cost. Putting these together, we can infer that $f_{A^*}(d_{min}) \leq w \cdot f_{A^*}(opt)$. This property holds for every d_{min}. Eventually it will be a solution if one exists, and its quality will be bounded.

The performance of A^*_ϵ is mediocre. It works well for loose bounds, where the algorithm searches greedily on d, but fails to find solutions otherwise as shown by the first panel of Figure 2. When using an admissible h function, the f values of nodes cannot decrease, and typically increase, as one descends from the root. Along a path to a goal, d tends to decrease. Thus, nodes with low d will often have relatively high f_{A^*} values and d_{min} is often the node with the highest f_{A^*} in focal. Children of d_{min} are not likely to be included in focal. This creates a phenomenon where most nodes on focal are expanded in sequence with none of their children making it on to focal until the node with minimum f and highest d is expanded and focal is refilled. During each of these cycles, little progress is made toward the goal. One way to avoid needing to flush the focal list in order to expand f_{min} is to just explicitly expand it every so often, here every tenth extension. Panel 1 of Figure 2 shows that this scheduled algorithm performs much better, confirming that it is indeed f_{min} stagnation leading to the poor performance of A^*_ϵ.

Dynamically Weighted A*

Dynamically weighted A* searches nodes in a best-first order as determined by a node evaluation function f_{dwA*}. It applies a weighting factor w to the heuristic evaluation function h to encourage the search to behave greedily, decreasing the size of this factor as the search progresses. More concretely, D_n is the depth of a node, D_{goal} is the depth of solutions, and $f_{dwA^*} = g(n) + w \cdot max(0, ((1 - \frac{D_n}{D_{goal}})) \cdot h(n)$.

This reduces the weight applied to the cost-to-go estimate as the search progresses, rewarding progress. The depth of goals isn't always known a priori, and in these situations, D_{goal} can be approximated by the search-distance-to-go heuristic d at the root. The max term accounts for problems where nodes can be deeper than the projected depth of a goal.

Dynamically weighted A* performs poorly in domains where the search proceeds unevenly towards goals, as it does in grid pathfinding. The third panel of Figure 3 shows the poor performance of the algorithm. The line labeled dwA* is dynamically weighted A*, and revised dwA* is our improved version of the algorithm that makes no assumption on the relationship between the depth of a node and its distance to a goal. Decreasing the weight by which the heuristic evaluation function is multiplied as depth increases encourages progress in any direction. This is fine in when each step away from the goal, regardless of the action, takes the search equally closer to a goal but is bad in general.

Improving Dynamically Weighted A*

We now show how d can be used to remove this dangerous assumption. Let $\epsilon = w - 1$ where w is the desired suboptimality bound. Changing the node evaluation function of dynamically weighted A* to $f'_{dwA^*} = g(n) + (1 + \frac{d(n)}{d(root)} \cdot \epsilon) \cdot h(n)$ captures the idea that not all nodes are progressing towards a goal evenly. Instead of being rewarded for distance away from the root, nodes are rewarded for their proximity to a goal. The third panel of Figure 3 shows that this approach can dramatically improve the performance of dynamically weighted A*. In particular, notice that for a suboptimality bound of 1, that is when finding optimal solutions, revised dynamically weighted A* can outperform A* by a factor of 2 due to the differences in tie breaking.

Revised dynamically weighted A* must take special care of nodes that appear to be further away from a goal node than the root does, lest it loose its bounded suboptimality guarantee. When $d(n) > d(root)$, then $\frac{d(n)}{d(root)} > 1$ and $(1 + \frac{d(n)}{d(root)} \cdot \epsilon) > w$, and this threatens bounded suboptimality. By restricting the values f'_{dwA^*} can assume, suboptimality guarantees can be preserved. In the results presented in this paper, we use the node evaluation function $f(n) = min(f'_{dwA^*}(n), w \cdot f_{A^*}(n))$.

Breaking Ties Using Distance Estimates

Weighted A* is an elegant solution to bounded suboptimal search where the traditional node evaluation function of A*, $f_{A^*} = g(n) + h(n)$ is changed to increase the cost to go by a factor w, as in $f_{wA^*} = g(n) + w \cdot h(n)$. This emphasis on the heuristic evaluation function causes weighted A* to behave more like a greedy search on h. While this is an effective strategy where heuristics are accurate and h and d are identical, Figure 1 shows that better performance can be obtained by focusing solely on d when bounds are loose.

One easy and effective way to incorporate this information into the search is simply to use d to break ties. There are several conceivable tie breaking rules, and it may not immediately be obvious that breaking ties on d is the best possible approach. We examined breaking ties randomly and breaking ties in favor of high g. Panel 2 of Figure 2 shows the results in grid pathfinding. Tie breaking on d is clearly the best technique. The differences in tie-breaking behavior only manifest for tight bounds, where $1 \leq w \leq 2$. Beyond this weight there are relatively few ties that remain to be broken. Tie breaking will only have a large impact if there are many ties among nodes with a cost equal to that of the returned solution. There are a surprisingly large number of ties in grid worlds.

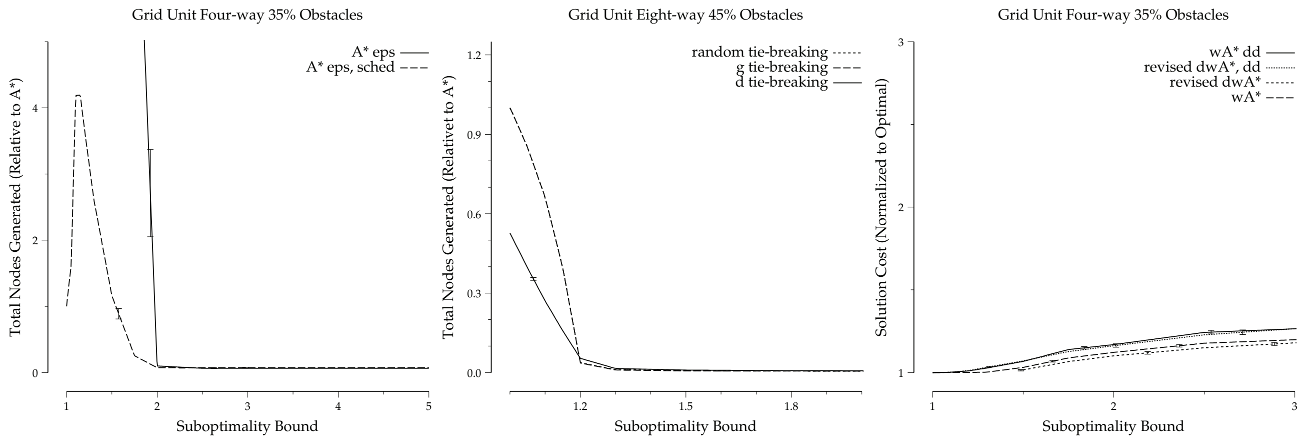

Figure 2: Performance of searches using d.

Extended Evaluation

We have conducted a comprehensive empirical analysis of how techniques for including d perform in a variety of benchmark domains. Here we reproduce the most exciting results. All algorithms were implemented in Objective Caml, compiled to 64-bit native code executables, and run on a collection of Intel Linux systems. We sampled all the algorithms at the following suboptimality bounds: 1, 1.005, 1.001, 1.01, 1.05, 1.1, 1.15, 1.2, 1.3, 1.5, 1.75, 2, 2.5 and 3.

Grid Pathfinding

Following Thayer and Ruml (2008) we tested on many grid pathfinding problems. We show 95% confidence intervals averaged over 100 instances for game boards and 20 instances otherwise. We considered two variants of algorithms in this domain, one in which duplicate nodes are ignored, and one in which they aren't. Panel three of Figure 2 shows the impact on solution quality of ignoring duplicate states, nearly none. Weighted A* may drop duplicates while maintaining its bounded suboptimality guarantee, but dynamically weighted A* may not. In practice, even without the guarantee, dynamically weighted A* never returns a solution outside of the desired bound.

Uniform Distribution We consider simple path planning problems on a 2000 by 1200 grid. We allow for eight-way movement where diagonal moves cost $\sqrt{2}$ times as much as cardinal directions. Simple analytical lower bounds are available for the cost h and search-distance d to the cheapest goal. In the results reported here, we use an admissible d function, although this is not a requirement for bounded suboptimality. Our results are shown in Figure 3. A^*_ϵ and revised dynamically weighted A* without duplicate dropping are omitted as they fail to solve a majority of the instances.

Lines We consider problems using lines for obstacles. These lines were of lengths ranging between 5% and 25% of the board's diagonal, and 75 such lines are scattered on the board. Results are presented in the center panel of Figure 3. Duplicate dropping is still needed to perform reasonably on problems of this size, so A^*_ϵ and revised dynamically weighted A* are omitted.

Game Boards Following Bulitko et al. (2007), we tested on several pathfinding problems from a popular real-time strategy game, again allowing for eight-way movement. Start and goal locations were selected at random. Instances were then further grouped by the length of their optimal solution. Figure 3 shows results for the most challenging instances we ran against which have an optimal solution length somewhere between 160 and 180 steps. The size of these problems limits the impact of duplicate dropping.

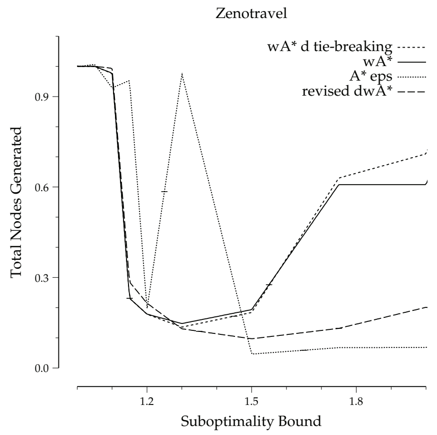

Figure 4: Using d avoids catastrophe in temporal planning.

The results clearly show that incorporating search distance to go information dramatically improves the performance of algorithms, allowing us to find optimal solutions three times faster than less informed algorithms. Tie-breaking on d is often enough to see this performance increase, although revised dynamically weighted A* also finds solutions faster than A*.

Temporal Planning

Planning is a domain in which optimal solutions can be extremely expensive to obtain (Helmert and Röger 2007). We tested our algorithms on 31 temporal planning problems from five benchmark domains taken from the 1998 and 2002 International Planning Competitions where the objective function is to minimize the plan duration (makespan).

To find the plan, we used the temporal regression planning framework in which the planner searches backwards from

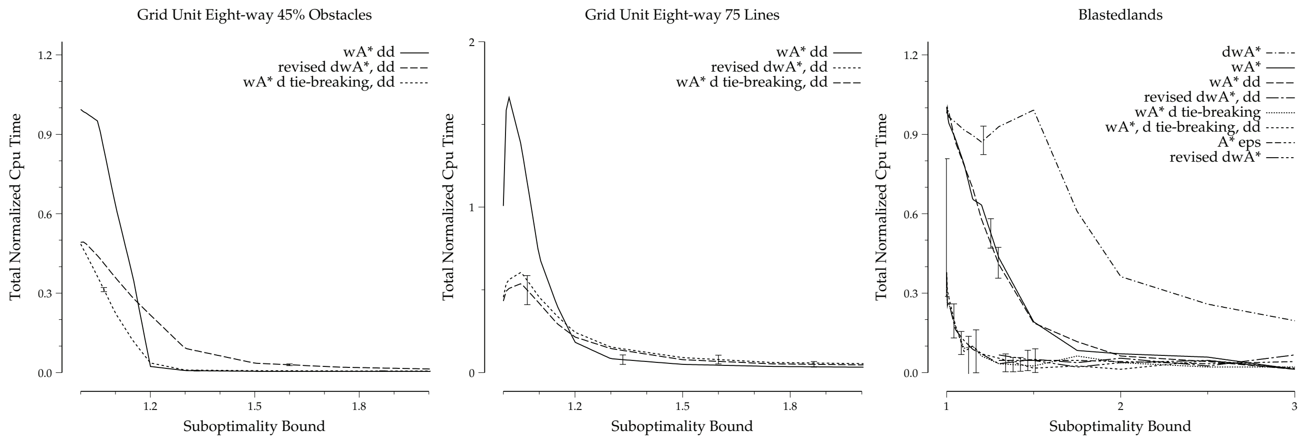

Figure 3: More information helps when solving pathfinding problems.

the goal state (Bonet and Geffner 2001). To guide the search, we compute $h(n)$ using the admissible H^2 heuristic of the TP4 planner (Haslum and Geffner 2001). In order to compute a search-distance-to-go function d, we also computed the expected number of steps to reach the shortest makespan solution. This value was estimated by first extracting a relaxed plan (Hoffmann and Nebel 2001) that approximates the closest shortest solution in terms of makespan. Results are presented in terms of nodes generated for clarity.

We found that the patterns of algorithm performance over the 31 benchmark instances fell into roughly three categories. In some problems including d information has no discernible impact on search. Other times, algorithms behave erratically showing no definite trend across weights. Not infrequently we see the behavior shown in Figure 4, where reliance, not just tie breaking, on d avoids the tendency of searches to perform worse as weights increase in this domain.

Conclusions

We have examined two previous approaches and two novel ways of incorporating distance estimates into heuristic search. A^*_ϵ occasionally has good results, but is unpredictable. Dynamically weighted A* is inappropriate for problems where distance from the root and nearness to a solution are only loosely related, which is to say most domains. These two short comings have lead many to mistakenly assume that examining distance to go estimates will be fruitless. We have clearly shown that this information is useful for improving the performance of heuristic search. Often simply breaking ties on d gives a dramatic performance increase, both for optimal solutions and bounded suboptimal solutions. Weighted A* with tie-breaking on d is only effective when there are a number of ties to be broken. This is obviously the case in grid-world pathfinding, given the domain and the performance of the algorithm in it. Whenever there are many different real valued edge costs, tie-breaking on d isn't going to help because there will not be very many ties. Fortunately, in these domains tie breaking cannot damage the algorithms performance either, as there are few chances for tie-breaking to make a mistake. Stronger reliance on d, as in revised dynamically weighted A* and A^*_ϵ can prevent catastrophic failures in temporal planning. This shows that d is both immediately useful and worthy of further study.

Acknowledgments

We gratefully acknowledge support from NSF grant IIS-0812141 and from the DARPA CSSG program. Many thanks to Minh B. Do for the temporal planner used in these experiments. Thanks to Jeff Kreis for his work in visualizing the errors and performance of A^*_ϵ.

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Bulitko, V.; Sturtevant, N.; Lu, J.; and Yau, T. 2007. Graph abstraction in real-time heuristic search. *JAIR* 30:51–100.

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.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proceedings of ECP-01*.

Helmert, M., and Röger, G. 2007. How good is almost perfect? In *Proceedings of the ICAPS-2007 Workshop on Heuristics for Domain-independent Planning: Progress, Ideas, Limitations, Challenges*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

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.

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