Fast and Loose in Bounded Suboptimal 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

Ephrat Bitton
Dept. of Industrial Eng. and Operations Res.
University of California
Berkeley, CA 94720 USA
ebitton at berkeley.edu

Abstract

Applications often demand we tackle problems that are too large to solve optimally. In this paper, our aim is to solve shortest-path problems as quickly as possible while guaranteeing that solution costs are bounded within a specified factor of optimal. We explore two approaches. First, we extend the approach taken by weighted A^*, in which all expanded nodes are guaranteed to remain within the bound. We prove that a looser bound than weighted A^*'s can be used and show how an arbitrary inadmissible heuristic can be employed. As an example, we show how temporal difference learning can learn a heuristic on-line. Second, we show how an aggressive search that expands nodes potentially outside the bound can be modified to ensure bounded solution quality. We test these methods on grid-world path-finding and temporal planning benchmarks, showing that these methods can surpass weighted A^*'s performance.

Introduction

Tasks as important and diverse as planning and multiple sequence alignment can be represented as shortest-path problems. If sufficient computation is available, optimal solutions to such problems can be found using A^* search with an admissible heuristic (Hart, Nilsson, & Raphael 1968). However, in many practical scenarios, one is willing to accept a suboptimal solution in return for reduced computation time. In this paper, we consider the setting in which one wants the fastest search possible while guaranteeing that the suboptimality of the resulting solution is bounded to within a given factor of the optimal solution's cost.

The best previously-proposed algorithm for this problem is weighted A^* (Pohl 1970), in which the traditional node evaluation function f is modified to place additional weight on the heuristic evaluation function h, as in $f'(n) = g(n)+w\cdot h(n)$. By penalizing nodes with large h values, the search becomes greedier. The solution returned by weighted A^* is within a factor of w of optimal, a condition we will term w-admissibility. Weighted A^* is beautifully simple and often performs well, but other algorithms have been proposed. One is dynamically weighted A^* (Pohl 1973), which requires an estimate of the depth of the solution and then decreases w from its original value at the root of the tree to 1 at the estimated goal depth. This maintains w-admissibility. Another algorithm is A^*_ϵ (Pearl & Kim 1982), which requires both the traditional estimate of cost-to-go h but also an estimate of the search effort or distance-to-go d. Of all the nodes in the open list whose f value is within a factor of w of the minimum f value of any node in open, A^*_ϵ expands that node whose d value is minimum. A^*_ϵ is w-admissible. These two newer algorithms have not displaced weighted A^*, which remains widely used, and in the experiments reported below we will see that they do not find solutions as quickly.

In this paper, we propose two new techniques for w-admissible heuristic search. First, we extend the approach taken by weighted A^*, in which all expanded nodes are guaranteed to remain within the bound. We prove that a looser bound than weighted A^*'s can be used and show how an arbitrary inadmissible heuristic can be employed to guide the search. As an example, we show how temporal difference learning can learn a heuristic on-line. Second, we show how an aggressive search that expands nodes potentially outside the bound can be modified to ensure bounded solution quality. We investigate the behavior of these approaches empirically and find that the first consistently outperforms weighted A^* on grid-world pathfinding problems and that both are highly competitive on temporal planning benchmarks.

Maintaining Bounded Sub-optimality

Our first approach will be a strict one, following in the tradition of weighted A^*. Search will be guided by an inadmissible heuristic. We will achieve the desired suboptimality bound by ensuring that no expanded node could violate it. We do this by restricting how large the inadmissible heuristic values can be. We wish to allow as loose a bound as possible, so that the inadmissible heuristic has freedom to guide the search, while still maintaining w-admissibility. The following theorem is analogous to those commonly given for weighted A^* (Pearl 1984), just slightly broader. We will assume that h is admissible. The optimal cost of a path from the root to node n will be notated g^* and *opt* will represent an optimal solution. Our inadmissible heuristic is $\widehat{h}$ and $\widehat{f}(n) = g(n) + \widehat{h}(n)$. We assume that $\widehat{h}$ is 0 at goals. The clamped node evaluation function $\widetilde{f}$ is formed as $\widetilde{f}(n) = \min(\widehat{f}(n), w \cdot f(n))$. Figure 1 provides an outline of such an algorithm.

1. *open* $\leftarrow \{$*initial*$\}$
2. if *open* is empty, return failure
3. $n \leftarrow$ remove node from *open* with lowest $\widetilde{f}(n)$
4. if n is a goal, return it
5. add n to *closed*
6. for each of n's children c
7. if c is duplicated in *open* then
8. keep in *open* the copy with lower $g(c)$
9. else if c is duplicated in *closed* then
10. if $f(c) \leq f(duplicate)$ then
11. add c to *open*
12. else add c to *open*
13. go to step 3

Figure 1: Bounded sub-optimality search using a clamped heuristic. $\widetilde{f}(n)$ is defined in the text.

Theorem 1 *A best-first search guided by $\widetilde{f}$ will return a solution s with $g(s) \leq w \cdot g^*(\text{opt})$.*

Proof: Consider the optimal path to *opt*. If all nodes on this path have been expanded, we have the optimal solution and the theorem holds trivially. Otherwise, let n be the deepest node on *open* that lies along the optimal path to *opt*. Such a node must exist because an optimal path to *opt* exists by definition, the root is on it, and if a parent node is on it, one of the children must be, and all children are inserted into *open*. $\widehat{h}(s) = 0$ because s is a goal, so $g(s) = \widehat{f}(s) = \widetilde{f}(s)$ by the definitions of $\widehat{f}$ and $\widetilde{f}$. $\widetilde{f}(s) \leq \widetilde{f}(n)$ because s was selected for expansion before n. $\widetilde{f}(n) \leq w \cdot f(n)$ by the definition of $\widetilde{f}$. So we have $g(s) \leq w \cdot f(n)$. Furthermore, $w \cdot f(n) \leq w \cdot f(\textit{opt}) = w \cdot g^*(\textit{opt})$ by the admissibility of h. □

Note that $\widetilde{f}(n)$ can be significantly greater than weighted A^*'s $f'(n)$ because $w(g(n) + h(n)) \geq g(n) + w \cdot h(n)$. They are only the same when $g(n) = 0$, which occurs only at the root. In practice, this difference may be enough to allow for an improved search order if we have additional useful information to bring to bear in the form of $\widehat{f}(n)$.

An Aggressive Approach

The above approach in which the heuristic function is clamped such that we can guarantee the admissibility bound is very strict. No node is expanded which could not lead to a w-admissible goal. Alternatively, one could expand nodes more aggressively, exploring nodes which might lead to a w-inadmissible solution, and then enforce the sub-optimality bound after finding a goal. For example, in their discussion of the Anytime weighted A^* algorithm, Hansen & Zhou (2007) show how the lowest f value of any node in the open list is a lower bound on the optimal solution value. Although they do not mention it, one could use this technique to perform anytime search using an inadmissible heuristic until a desired sub-optimality bound is reached.

Taking this idea one step further, one could search according to any inadmissible heuristic and then, once a solution has been identified, switch to exploring the open list in order of f value. This two-phase search would first try to find a solution, then try to raise the lower bound to prove w-admissibility. Because a higher weight in weighted A^* can result in faster searches, and because the quality of solutions returned by weighted A^* are often better than the weight suggests, such a strategy could result in faster total search time than weighted A^* itself. The risk in such a technique is that the solution found during the first phase might not be w-admissible, causing the algorithm to behave like A^*, expanding all nodes with f values less than the optimal solution. However, it is easy to guard against this worst case: if there is a node whose inadmissible value is less than that of the incumbent solution, then that node is selected for expansion. Figure 2 is a sketch of such an algorithm. $\widehat{f}$ can be any arbitrarily inadmissible heuristic, though in the experiments below, it is set to be $\widehat{f}(n) = g(n) + ((b-1) \cdot 2 + 1) \cdot h(n)$, which is essentially $\widehat{f}(n) = g(n) + w \cdot h(n)$ where w is twice as inadmissible as *b*, the bound.

1. *open* $\leftarrow \{initial\}$
2. *flist* $\leftarrow \{initial\}$
3. *incumbent* $\leftarrow \infty$
4. if $\widehat{f}$(*incumbent*) $< \widehat{f}$(first on *flist*) then
5. $n \leftarrow$ remove node from *open* with lowest $\widehat{f}(n)$ and remove n from *flist*
6. else $n \leftarrow$ remove node from *flist* with lowest $f(n)$ and remove n from *open*
7. add n to *closed*
8. if g(*incumbent*)$/f$(first on *flist*) $\leq b$ then
9. return *incumbent*
10. if n is a goal then
11. $incumbent \leftarrow n$
12. else for each child c of n
13. if c is duplicated in *open* then
14. if c is better than the duplicate then
15. replace the duplicate in *open* and *flist*
16. else if c is duplicated in *closed* then
17. if c is better than the duplicate then
18. add c to *open* and *flist*
19. else add c to *open* and *flist*
20. go to step 4

Figure 2: Aggressive Search with cleanup

Learning an Inadmissible Heuristic

Both the clamped heuristic and clean-up-based approaches can use an arbitrarily inadmissible heuristic. While inadmissible heuristics are often formed by weighting an admissible heuristic, we are free to consider any scheme. One idea that has been mentioned in passing by several authors, including Michie & Ross (1969) and Nilsson (1998), but never (to our knowledge) actually pursued, is to learn a heuristic function during search using the method of temporal differences.

If h were perfect, then the f value of a parent node would

be the same as the lowest f among its children. However, admissible heuristics usually underestimate the cost to the goal and f rises in value as the search progresses. The rise in value from a parent to its best child is a measurement of the error in the heuristic. As nodes are expanded during search, one can calculate the average one step error, e_h, in h. If one then assumes like Pearl & Kim (1982) that a function d is available to estimate search steps to the goal, one can estimate a corrected value as $\widehat{h}(n) = h(n) + e_h d(n)$. This estimate can occur, for example, just after step 6 in Figure 1, using the f values of the children.

In addition to calculating e_h as the average error in h at all expansions, one can use any of a variety of methods for calculating the offset for the corrected heuristic. One approach is to use the immediate e_h to correct for the error in the heuristic, but its behavior is erratic since it has no history to temper the current measurement. We can calculate the e_h at each node as a weighted average of the error observed before the parent and the error observed at the parent. This forces e_h at a node to remain constant and manages to include a portion of the previous experience. Unfortunately as the search progresses it misses out on an increasing number of samples. Any number of additional error models exist, some far more complicated than those above. Added complexity may produce a more accurate correction, but it comes at the cost of additional overhead. Exploring the nature of this trade off and exploiting it are a subject of future work.

A problem with on-line estimate of e_h is that if that estimate is changing over time, as it is when modeled by a global average, then so are the $\widehat{h}$ and $\widehat{f}$ of every node. The overhead of re-sorting the open list after every expansion seems prohibitive. Two possible approximations are 1) to re-sort the open list only occasionally, perhaps at a geometrically growing interval and 2) to not re-sort at all and have each node keep forever the $\widehat{f}$ value computed when it was generated. In the experiments reported below we used the second technique and a global average for e_h, which seems to outperform all other combinations. It should be noted that not resorting the open list has no affect on the w-admissibility of the solution.

Empirical Evaluation

Although the two approaches we have presented for bounded sub-optimality search are w-admissible and have the ability to use $\widetilde{f}$ values greater than the f' values used by weighted A^* running within the same bounds, we have no guarantee that they will find solutions faster. To gain a sense of their behavior, we implemented several algorithms and tested them on two challenging benchmark search problems: grid-world pathfinding and temporal planning. All algorithms where implemented in Objective Caml and compiled to native binaries on 64-bit Intel Linux systems. We provide a textbook implementation of weighted A^* (noted as wA*), clamped heuristic search using on-line estimation of e_h (clamped adaptive), and aggressive search using $\widehat{f}(n) = g(n) + (1 + 2(w-1))h(n)$ with a clean-up phase (aggressive w/ cleanup). We also tested A^*_ϵ and dynamically weighted A^*, but their performance was very poor and we do not include them in our results here.

Grid-world Planning

We considered several classes of simple path planning problems on a 2000 by 1200 grid, using either 4-way or 8-way movement, three different probabilities of blocked cells, and two different cost functions. The start state was in the lower left corner and the goal state was in the lower right corner. In addition to the standard unit cost function, under which moves have the same cost everywhere, we tested a graduated cost function in which moves along the upper row are free and the cost goes up by one for each lower row. We call this cost function 'life' because it shares with everyday living the property that a short direct solution that can be found quickly (shallow in the search tree) is relatively expensive while a least-cost solution plan involves many annoying economizing steps. In 8-way movement worlds, diagonal movement costs $\sqrt{2}$ times as much as movement in any of the cardinal directions. Under both cost functions, simple analytical lower bounds (ignoring obstacles) are available for the cost $g(n)$ and distance $d(n)$ (in search steps) to the cheapest goal. The obstacle density introduces error to the heuristics and challenge to the problems.

Figure 3 shows the algorithms' performances on the hardest problems we considered in each of the four classes of worlds (35% blocked cells in the four-way worlds, 45% in the eight-way worlds). The x axis represents the sub-optimality bound used, with 1 being optimal and 3 being three times the optimal solution cost. Samples were taken at 3, 2.5, 2, 1.75, 1.5, 1.3, 1.2, 1.15, 1.1, 1.05, 1.01, 1.001, 1.0005, and 1. The y axis is the number of nodes generated, normalized by the number of nodes generated by an optimal A^* search and averaged over 20 random worlds. Error bars indicate 95% confidence intervals around the mean, although they are typically so tight as to be invisible.

In the unit cost panels of the figure, clamped adaptive search performs better than weighted A^* when the sub-optimality bound is relatively tight (less than 1.25). As the bound loosens, the adaptive search searches according to the corrected evaluation function $\widehat{f}$. This value, while responsive to the properties of the search space, is not directly influenced by the sub-optimality bound and the search does not become as aggressively greedy as weighted A^*. As we approach the high end of weights, clamped adaptive has trouble competing with weighted A^*, though the weight at which weighted A^* begins outperforming clamped adaptive increases with problem difficulty, where difficulty can be measured by the reduction in node generations that the algorithms can achieve as the sub-optimality bound loosens. The shallowest decrease is for four-way problems with the 'life' cost model.

We also tested aggressive search with clean-up. You can see from the figure that the search performs as if it were weighted A^* running with a higher weight; in the panels this is seen by the line resembling that of weighted A^* but shifted left. Although similar to anytime weighted A^*, the performance of the two algorithms differs substantially. Our aggressive search with cleanup completed all of the plan-

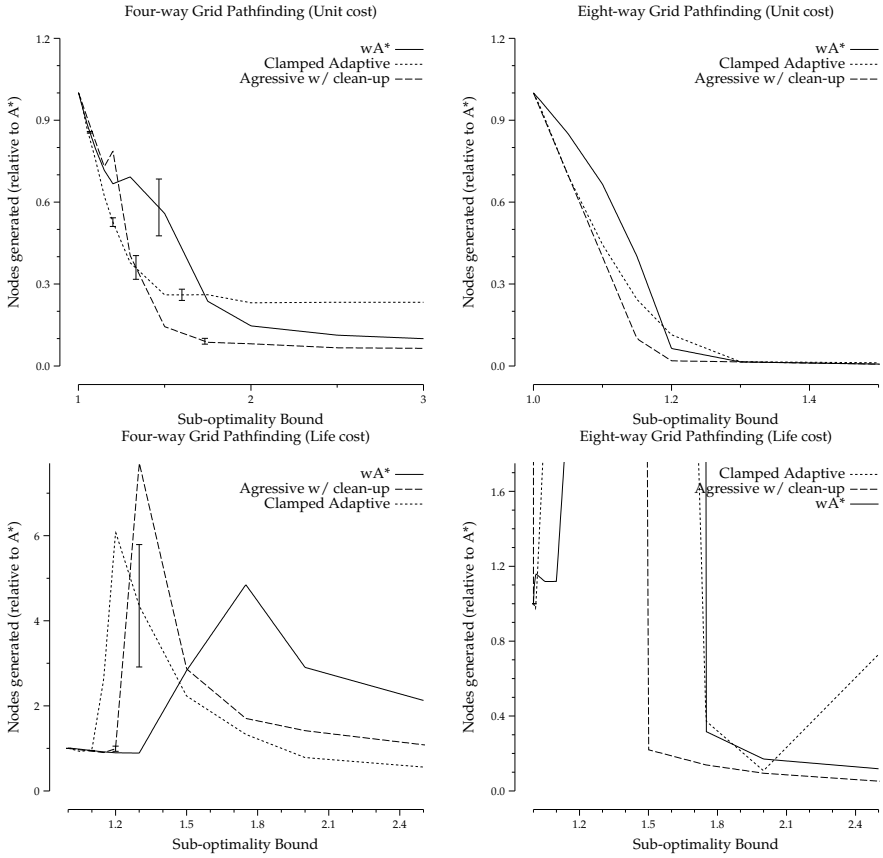

Figure 3: Performance on grid-world path-finding problems.

ning problems within the allotted time, 4 minutes, while our implementation of anytime weighted A^* had trouble solving instances where the suboptimality bound was not extremely low or very high. We attribute the increase in performance to the explicit raising of the lower bound.

Plots using CPU time instead of nodes were also generated, to account for the fact that on-line learning consumes some additional overhead at each node expansion. This overhead was typically only 10% in our not-particularly-optimized implementation when measured in grid-world, a domain with very fast node expansion. The results were very similar, and averaging across instances introduces some degree of noise due to running the experiments across multiple machines, so we present the cleaner plots here.

Temporal Planning

There has been increasing interest over the last ten years in applying heuristic search algorithms to AI planning problems (Bonet & Geffner 2001; Zhou & Hansen 2006). It is also a domain in which optimal solutions can be extremely expensive to obtain (Helmert & 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 the goal state S_G to reach the initial state S_I (Bonet & Geffner 2001). To guide the search, we compute $h(n)$ using the admissible H^2 heuristic of the TP4 planner (Haslum & Geffner 2001). This heuristic estimates the shortest makespan within which each single predicate or pair of predicates can be reached from the initial state S_I. This is computed once via dynamic programming before starting the search, taking into account the pairwise mutual exclusion relations between actions in the planning problem. 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 & Nebel 2001) that approximates the closest shortest solution in terms of makespan from a given search node. The number of regression steps in this plan is then used as the distance estimate to the cheapest solution.

Figure 4 shows results on the hardest benchmark problem from each domain that A^* could solve within four minutes. Again, the x axis represents the sub-optimality bound, where 1 is optimal and 3 is three times the optimal cost. Samples were taken at the same points as in grid-world path-finding. The y axis is the number of generated nodes relative to an optimal A^* search. A values of -1, as for clamped adaptive on high bounds in the zenotravel domain (upper left), means

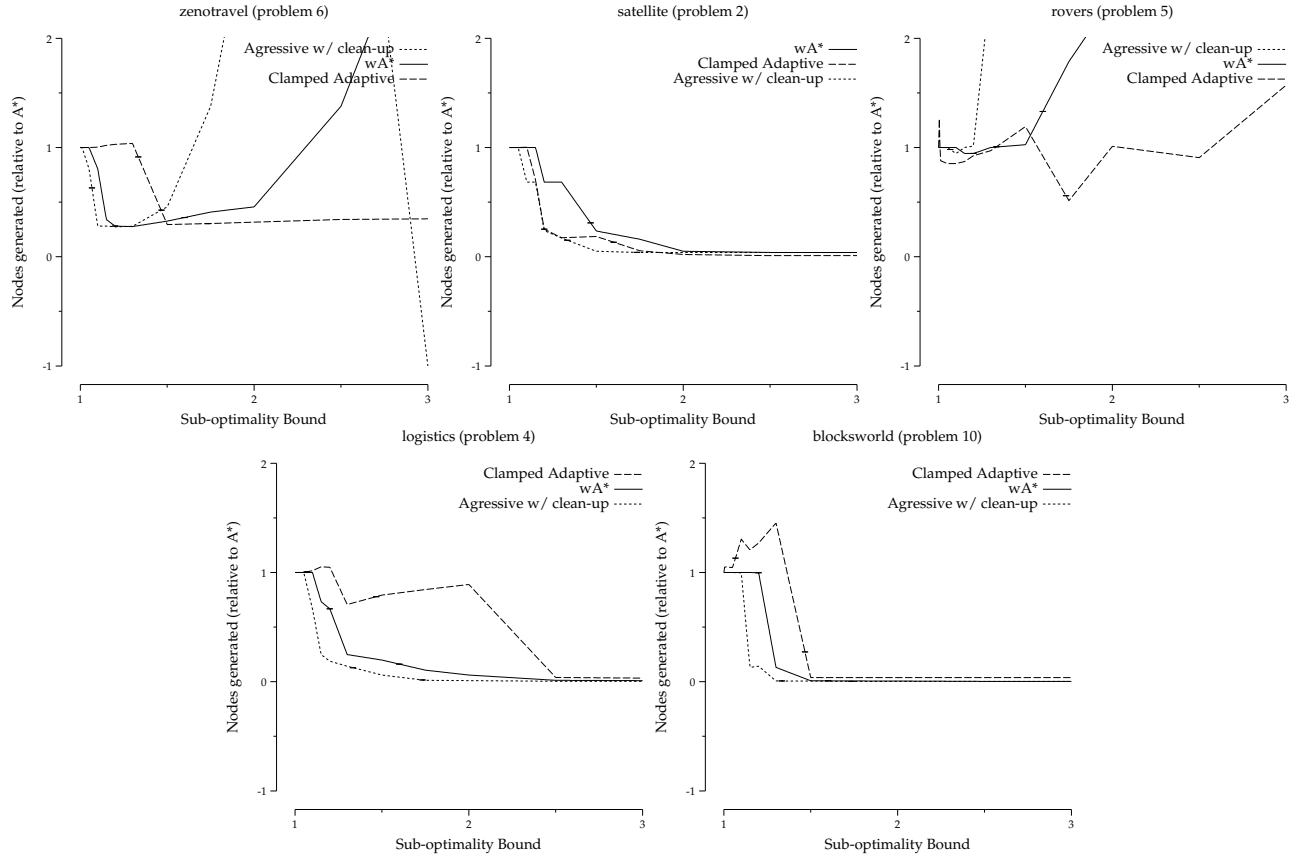

Figure 4: Performance on difficult temporal planning problems.

	wA*			CA	
Domain	dnf	worse	similar	worse	dnf
blocksworld			2	5	3
logistics			2	3	
rovers		1	4		
satellite		1	3		
zenotravel	1	4	1	1	

Table 1: Qualitative performance of weighted A^* (wA*) versus clamped adaptive (CA) on temporal planning problems. dnf: did not finish; worse: performed worse.

Domain	w A* worse	similar	AwC worse
blocksworld	4	5	1
logistics	4	1	0
rovers	2	1	2
satellite	2	1	1
zenotravel		2	5

Table 2: Qualitative performance of weighted A^* (wA*) versus aggressive search with clean-up (AwC).

that the algorithm in question did not produce a plan within four minutes.

Overall, the aggressive search strategy performed well in this domain, and significantly better than in grid-world. Often, the aggressive search performed just as weighted A^* would have done with a higher weight. This can be seen as a copy of the weighted A^* curve shifted to the left, which while present in all of the panels is probably easiest to see in zenotravel and logistics. Despite its widespread use in the planning community, weighted A^* seems quite sensitive to the choice of weight, often exhibiting a U-shaped performance curve of improvement and then disintegration (eg, zenotravel, rovers).

The performance of the clamped adaptive search was mixed, sometimes providing enormous speedups (zenotravel, satellite, rovers) and sometimes performing very poorly (logistics, blocksworld). To give a broader sense of its performance, Table 1 provides a qualitative overview. Each row represents a different planning domain and the columns represent the relative performance of weighted A^* (wA*) versus clamped adaptive search (CA). Each cell shows the number of problem instances in that domain where the corresponding algorithm performed better. This overview suggests that the two algorithms are roughly comparable. Looking at the individual problems, it appeared that clamped adaptive search improved relative to weighted A^* as the problems became more difficult, but we have not yet quantified this impression.

Table 2 provides a similar qualitative comparison for aggressive search with clean-up. It seems to have very com-

plementary strengths to clamped adaptive search.

Discussion

Applications often demand we tackle problems that are too large to solve optimally. When abandoning optimality, there are two basic strategies: bound the time taken by the search or bound the quality of the resulting solution. Anytime algorithms and real-time search are the main approaches taken to the bounded-time problem. For bounded sub-optimality, weighted A^* has reigned for decades as the superior technique. We have presented two new approaches for using an inadmissible heuristic function in search and shown that they can guarantee a desired sub-optimality bound. One approach takes a strict perspective and restricts the node evaluation function such that w-admissibility can be proved. The other is more optimistic, using an inadmissible heuristic to find a solution and then performing additional expansions as necessary if the bound has not been achieved. We introduced a new inadmissible heuristic based on on-line temporal difference learning that attempts to estimate the average error in h. We also showed that duplicate dropping, a technique pioneered by ARA* (Likhachev, Gordon, & Thrun 2004), can find broader use in heuristic search.

We tested these techniques on two challenging benchmark domains, grid-world path-finding and temporal planning. In grid-world, the clamped adaptive technique edged out weighted A^*, the first technique to do so. Its advantage seemed to increase as problems get more difficult. In temporal planning, the results were more complex. Clamped adaptive seemed comparable to weighted A^*, although perhaps more robust on hard problems. Aggressive search with clean-up also performed well here, as the domain contains few, if any, duplicates.

Conclusions

We addressed the problem of heuristic search with bounded sub-optimality, introducing two approaches for using inadmissible heuristics while maintaining a quality guarantee. We showed that it is feasible to improve a heuristic function on-line during search. While our empirical evaluation did not produce a clear winner, both new methods demonstrated advantages over weighted A^*. Clamped Adaptive performs well when searching for goals which are nearly optimal in complicated search spaces while an aggressive search with cleanup can effective shift the performance of weighted A^* towards one, offering faster searches at the same w-admissibility. This is especially noticeable when bounds are nearly optimal.

In addition to exploring the performance of these methods on additional domains, it may be fruitful to test additional inadmissible heuristics. For example, it should be possible to learn on-line how to combine multiple heuristics, including pessimistic ones (Sadikov & Bratko 2006). The approaches we have defined are general and can encompass a wide variety of heuristic evaluation functions.

Acknowledgements

Many thanks to Minh Do for the planner implementation and to Rong Zhou for helpful discussions

References

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

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.

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. *Journal of Artificial Intelligence Research* 14:253–302.

Likhachev, M.; Gordon, G.; and Thrun, S. 2004. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proceedings of NIPS 16*.

Michie, D., and Ross, R. 1969. Experiments with the adaptive graph traverser. In *Machine Intelligence 5*, 301–318.

Nilsson, N. J. 1998. *Artificial Intelligence: A New Synthesis*. San Francisco, CA: Morgan Kaufmann.

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.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

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

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.

Sadikov, A., and Bratko, I. 2006. Pessimistic heuristics beat optimistic ones in real-time search. In *Proceedings of ECAI-06*, 148–152.

Zhou, R., and Hansen, E. 2006. Breadth-first heuristic search. *Artificial Intelligence* 170(4–5):385–408.