The Joy of Forgetting: Faster Anytime Search via Restarting

Silvia Richter
IIIS, Griffith University, Australia
and NICTA
silvia.richter@nicta.com.au

Jordan T. Thayer and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
{jtd7, ruml} at cs.unh.edu

Abstract

Anytime search algorithms solve optimisation problems by quickly finding a usually suboptimal solution and then finding improved solutions when given additional time. To deliver a solution quickly, they are typically greedy with respect to the heuristic cost-to-go estimate h. In this paper, we first show that this low-h bias can cause poor performance if the heuristic is inaccurate. Building on this observation, we then present a new anytime approach that restarts the search from the initial state every time a new solution is found. We demonstrate the utility of our method via experiments in PDDL planning as well as other domains. We show that it is particularly useful for hard optimisation problems like planning where heuristics may be quite inaccurate and inadmissible, and where the greedy solution makes early mistakes.

Introduction

Heuristic search is a widely used framework for solving optimisation problems. In particular, given an admissible heuristic and sufficient resources, the A* algorithm (Hart *et al.* 1968) can be used to find an optimal solution with maximum efficiency (Dechter and Pearl 1988). However, in large problems we may not want to spend, or be able to afford, the resources necessary for calculating an optimal solution. A popular approach in this circumstance is complete anytime search, in which a (typically suboptimal) solution is found quickly, followed over time by a sequence of progressively better solutions, until the search is terminated or solution optimality has been proven.

A* can be modified to trade solution quality for speed by weighting the heuristic by a factor $w > 1$ (Pohl 1970). The resulting algorithm, called *weighted A** or WA* for short, searches more greedily the larger w is. Assuming an admissible heuristic, the solution it finds differs from the optimal by no more than the factor w. This ability to balance speed against quality while providing bounds on the suboptimality of solutions makes WA* an attractive basis for anytime algorithms (Hansen *et al.* 1997; Likhachev *et al.* 2004; Hansen and Zhou 2007; Likhachev *et al.* 2008). For example, WA* can be run first with a very high weight, resulting in a greedy best-first search that tries to find a solution as quickly as possible. Then the search can be continued past the first solution to find better ones. By reducing the weight over time, the search can be made progressively less greedy and more focused on quality.

Complete anytime algorithms that are not based on WA* have also been proposed (Zhang 1998; Zhou and Hansen 2005; Aine *et al.* 2007). In these cases, an underlying global search algorithm is usually made more greedy and local by initially restricting the set of states that can be expanded, thus giving a depth-first emphasis to the search which leads to finding a first solution quickly. Whenever a solution is found, the restrictions are relaxed, allowing for better solutions to be found.

Our starting point is to note that the greediness employed by anytime algorithms to quickly find a first solution can also create problems for them. If the heuristic goal distance estimates are inaccurate, for example, the search can be led into an area of the search space that contains no goal or only poor goals. In this case, existing anytime algorithms typically rely on *pruning* to efficiently exhaust the useless area and move to a more promising area of the search space. However, an inadmissible heuristic cannot be used safely for pruning. Furthermore, anytime algorithms typically *continue* their search after finding the first solution, rather than starting a new search. This seems plausible as it avoids duplicate effort. However, a key observation that we will discuss in detail below is that continued search may perform badly in problems where the first solution found is far from optimal and finding a significantly better solution requires searching a different part of the search space. Indeed, *restarting* the search in such cases can be beneficial, as it allows changing bad decisions that were made near the start state more quickly. This in turn means the quality of the best-known solution may improve faster with restarts than if we continued to explore the area of the search space around the previous solution.

Building on this observation that forgetfulness can be bliss, we propose an anytime algorithm that incorporates helpful restarts while retaining most of the knowledge discovered in previous search phases. Our approach differs substantially from existing anytime approaches, as previous work on anytime algorithms has concentrated explicitly on *avoiding* duplicate work (Likhachev *et al.* 2004), and traditional anytime algorithms retain all information between search phases. We show that the opposite can be useful, namely discarding information which may be biased due to

greediness. To our knowledge, this is the first time that the idea of restarts, popular in depth-first search and stochastic local search, is used in best-first search. We show that our method significantly outperforms existing anytime algorithms in PDDL planning, while it is competitive in other standard benchmark domains of heuristic search.

Previous Approaches

We first present some existing approaches and discuss the circumstances under which their effectiveness may suffer.

A* and WA* Several anytime algorithms are based on Weighted A* (WA*), which is in turn based on A*. The A* algorithm uses two data structures: Open and Closed. In the beginning, Open contains only the start state and Closed is empty. Iteratively one of the states from Open is selected, moved from Open to Closed, and expanded, and its successor states are inserted into Open. The state expanded at each step is one which minimises the function $f(s) = g(s) + h(s)$, where $g(s)$ is the cost of the best path currently known from the start state to state s, and $h(s)$ is the heuristic value of s, i. e., an estimate of the cost of the best path from s to a goal state.

A* is often used with consistent heuristics, i. e., ones that satisfy $h(s) \leq c(s, s') + h(s')$ for all s and s', where $c(s, s')$ is the cost of reaching s' from s. With consistent heuristics, A* never needs to expand a state more than once because it is guaranteed that when a state is expanded, it has been reached via a cheapest possible path. Consequently, when a goal state is selected for expansion, an optimal solution is found. When using *inconsistent* heuristics in A*, however, it is possible that we find a cheaper path to a state after it has been expanded. To preserve optimality, we have to re-expand it to propagate the cheaper costs to its successors.

WA* is a variant of A* using the selection function $f' = g + w \cdot h$, with $w > 1$. Even if the heuristic function h is *admissible*, i. e., never overestimates the true goal distance, the weighting by w introduces inadmissibility. Inadmissibility in turn implies inconsistency, which means that a) we may have to re-expand states and b), when we select a goal state for expansion we are not guaranteed to have found an optimal solution anymore. However, the solution is guaranteed to be w*-admissible*, i. e. the ratio between its cost and the cost of an optimal solution is bounded by w.

Anytime Algorithms Based on WA* Anytime algorithms based on WA* run WA* but do not stop upon finding a solution. Rather, they report the solution and continue the search, possibly adjusting their data structures and parameters. A straightforward adjustment, for example, is to use the cost of the best known solution for pruning: after we have found a solution of cost c, we do not need to consider any states that are guaranteed to lead to solutions of cost c or more. This is the approach taken by Hansen and Zhou in their Anytime WA* algorithm (2007). Updating the cost bound is the only adjustment in Anytime WA*; in particular the weight w is never decreased. (While the authors discuss this option, they did not find it to improve results on their benchmarks.)

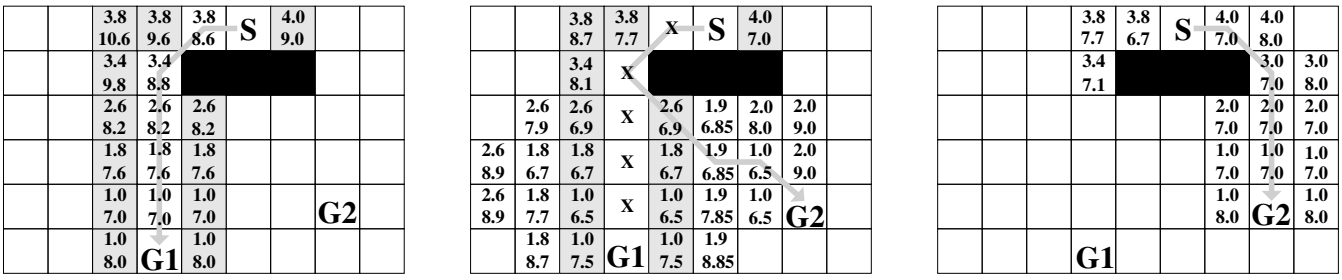
The Anytime Repairing A* (ARA*) algorithm by Likhachev et al. (2004) assumes an admissible heuristic and reduces the weight w each time it proves w-admissibility of the incumbent solution. This implicitly prunes the search space as no state is ever expanded whose f'-value is larger than that of the incumbent solution. When reducing w, ARA* updates the f'-values of all states in Open according to the new weight. Furthermore, ARA* avoids re-expanding states within search phases (where we use "phase" to denote the part of search between two weight changes). Whenever a shorter path to a state is discovered, and that state has already been expanded in the current search phase, the state is not re-expanded immediately. Instead it is suspended in a separate list, which is inserted into Open only at the beginning of the next phase. The rationale behind this is that even without re-expanding states the next solution found is guaranteed to be within the current sub-optimality bound (Likhachev *et al.* 2004). Of course, it may be that this solution is worse than it would have been had we re-expanded.

Other Anytime Algorithms Recent anytime algorithms that are not based on WA* include the A*-based Anytime Window A* algorithm (Aine *et al.* 2007) and beam-stack search (Zhou and Hansen 2005) which is based on breadth-first search. In both cases, the underlying global search algorithm is made more greedy by initially restricting the set of states that can be expanded. In Window A*, this is done by using a "window" that slides downwards in a depth-first fashion: whenever a state with a larger g-value (level) than previously is expanded, the window slides down to that level of the search space. From then on, only states in that level and the h levels above can be expanded, where h is the height of the window. Initially h is zero and it is increased by 1 every time a new solution is found.

In beam-stack search, only the b most promising nodes in each level of the search space are expanded, where the beam width b is a user-supplied parameter. The algorithm remembers which nodes have not yet been expanded and returns to them later. However, beam-stack search explores the entire search space below the chosen states before it backtracks on its decision. This may be inefficient if the heuristic is (locally) quite inaccurate and a wrong set of successor states is chosen, e. g. states from which no near-optimal solution can be reached. Likewise, Anytime Window A* can also suffer from its strong depth-first focus if the heuristic estimates are inaccurate and vary significantly locally.

The Effect of Low-h Bias

Anytime WA* and ARA* keep the Open list between search phases (possibly re-ordering it). After finding a goal state s_g, Open will usually contain many states that are close to s_g in the search space because the ancestors of s_g have been expanded; furthermore, those states are likely to have low heuristic values because of their proximity to s_g. Hence, even if we update Open with new weights, we are likely to expand most of the states around s_g before considering states that are close to the start state. This can be critical: in many optimisation problems decisions in the first half of a solution can greatly influence the quality of the solution.



(a) initial search, $w = 2$ (b) continued search, $w = 1.5$ (c) restarted search, $w = 1.5$

Figure 1: The effect of low-h bias. For all grid states generated by the search, h-values are shown above f'-values. (a) Initial WA* search finds a solution of cost 6. (b) Continued search expands many states around the previous Open list (grey cells), finding another sub-optimal solution of cost 6. (c) Restarted search quickly finds the optimal solution of cost 5.

Consider the small gridworld example in Fig. 1. The task is to reach a goal state ($G1$ or $G2$) from the start state S, where the agent can move with cost 1 to each of the 8 neighbours of a cell if they are not blocked. The heuristic is an inaccurate estimate of the straight-line goal distance of a cell. In particular, the heuristic underestimates distances in the left half of the grid. We start with weight 2 in Fig. 1a. Because the heuristic values to the left of S happen to be lower than to the right of S, the search expands the states to the left and finds goal $G1$ with cost 6. The grey cells are generated, but not expanded in this search phase, i. e., they are in Open. In Fig. 1b, the search continues with a reduced weight of 1.5. A solution with cost 5 consists in turning right from S and going to $G2$. However, the search will first expand all states in Open that have an f'-value smaller than 7. After expanding a substantial number of states, the second solution it finds is a path which starts off left of S and takes the long way around the obstacle to $G2$, again with cost 6. If we instead *restart* with an empty Open list after the first solution (Fig. 1c), far fewer states are expanded. The critical state to the right of S is expanded quickly and the optimal path is found. Note that in this example, it is in particular the high local variability of the heuristic values that leads the greedy search astray and makes restarts useful. We will see an example of this low-h-bias effect in practice in the discussion of our planning results.

Restarting WA* (RWA*)

To overcome low-h bias, we propose Restarting WA*, or RWA* for short. It runs iterated WA* with decreasing weight, always re-expanding states when it encounters a cheaper path. RWA* differs from ARA* and Anytime WA* by not keeping the Open list between phases. Whenever a better solution is found, the search empties Open and restarts from the initial state. It does, however, re-use search effort in the following way: besides Open and Closed, we keep a third data structure "Seen". When the new search phase begins, the states from the old Closed list are moved to Seen. When RWA* generates a state in the new search, there are three possibilities: Case 1: The state has never been generated before (it is neither in Open nor Closed nor Seen). In this case, RWA* behaves like WA*, calculates the heuristic value of the state and inserts it into Open. Case 2: The state has been encountered in previous search phases but not in the current phase (it is in Seen). In this case, RWA* can look up the heuristic value of the state rather than having to calculate it. In addition, RWA* checks whether it has found a cheaper path to the state or whether the previously found path is better, and keeps the better one. The state is removed from Seen and put into Open. Case 3: The state has already been encountered in this search phase (it is in Open or Closed). In this case, RWA* again behaves like WA*, re-inserting the state into Open only if it has found a shorter path to the state. Complete pseudo-code is shown in Fig. 2. Note that while this is a conceptually simple presentation of RWA*, it can be implemented in a more memory-efficient way by using a boolean value "seen" in each state of the Open/Closed list rather than keeping the seen states in a separate list.

In short, previous search effort is re-used by not calculating the heuristic value of a state more than once, and by making use of the best path to a state found so far. Compared to ARA* and Anytime A*, our method may have to re-expand many states that were already expanded in previous phases. However, in cases where the calculation of the heuristic accounts for the largest part of computation time (e. g., in our planning experiments it is 80%), the duplicated expansions do not have much impact on runtime. Thus, RWA* re-uses most of the previous search effort, but its restarts allow for more flexibility in discovering different solutions.

Restarts in Other Search Paradigms

Restarts are a well-known and successful technique in combinatorial search and optimisation, e. g. in the areas of propositional satisfiability and constraint-satisfaction problems. Together with randomisation, they are used in systematic search (Gomes *et al.* 1998) as well as local search (Selman *et al.* 1992) to escape from barren areas of the search space. A restart is typically executed if no solution has been found after a certain number of steps. In these approaches restarting would be useless without randomisation, as the algorithm would behave exactly the same each time. By con-

```
RWA*(w_0, φ)
 1  bound ← ∞, w ← w_0, Seen ← ∅
 2  while not interrupted and not failed
 3  do Closed ← ∅, Open ← {startstate}
 4     while not interrupted and not Open empty
 5     do remove s with minimum f'(s) from Open
 6        for s' ∈ SUCCESSORS(s)
 7        do if heuristic admissible and f(s') ≥ bound
 8                or g(s') ≥ bound
 9              then continue
10           if IS_GOAL(s')
11             then break
12           cur_g ← g(s) + TRANSITION_COST(s, s')
13           switch
14             case s' ∉ (Open ∪ Seen ∪ Closed) :
15                  g(s') ← cur_g, pred(s') ← s
16                  calculate h(s') and f'(s')
17                  insert s' into Open
18             case s' ∈ Seen :
19                  if cur_g < g(s')
20                    then g(s') ← cur_g, pred(s') ← s
21                  move s' from Seen to Open
22             case cur_g < g(s') :
23                  g(s') ← cur_g,  pred(s') ← s
24                  if s' ∈ Open
25                    then update s' in Open
26                    else  move s' fr. Closed to Open
27     if new solution found
28       then report incumbent solution s'
29            bound = g(s')
30            w ← max(1, w × φ)
31            Seen ← Seen ∪ Open ∪ Closed
32       else return failure
```

Figure 2: Pseudo-code for the RWA* algorithm

trast, our algorithm is deterministic, and its restarts serve the purpose of revisiting nodes near the start of the search tree to promote exploration when the node evaluation function has changed.

This is perhaps most closely related to the motivation behind limited-discrepancy search (LDS) (Harvey and Ginsberg 1995; Furcy and Koenig 2005). LDS attempts to overcome the tendency of depth-first search to visit solutions that differ only in the last few steps. It backtracks whenever the current solution path exceeds a given limit on the number of discrepancies from the greedy path, restarting from the root each time this limit is increased. Like random restarts and the backtracking in limited-discrepancy search, our restarts mitigate the effect of bad heuristic recommendations early in the search. Our use of restarts differs from limited-discrepancy backtracking in that we restart upon finding a solution, rather than upon reaching a preset limit on the number of discrepancies from the greedy path. As opposed to random restarts, we restart upon success (when finding a new solution) rather than upon failure (when no solution could be found).

Lastly, incomplete methods can be made complete by restarting with increasingly relaxed heuristic restrictions (Zhang 1998). By contrast, our restarts occur within a complete algorithm.

Our use of restarts also raises an interesting connection between anytime search in deterministic and stochastic domains. An analogy can be made between RWA* and real-time dynamic programming (RTDP), a popular anytime algorithm for solving Markov decision processes (Barto *et al.* 1995). Both algorithms follow a greedy strategy relative to the current state values until they find a goal, at which point they return to the initial state for another trial. In RTDP, Bellman back-ups are performed to increase the accuracy of heuristic values, while in RWA*, the weight is decreased for a similar effect. The procedures do have significant differences: RTDP makes irrevocable moves in the style of a local search, while RWA* keeps an open list. RTDP requires a labeling procedure to detect optimality (Bonet and Geffner 2003), while RWA* can terminate after a trial with weight 1 (when being used with an admissible heuristic; otherwise it can be extended easily to prove optimality by exhausting those states in Open that have a lower g-value than the incumbent solution). It would be an interesting topic for future work to investigate whether importing ideas from RTDP, such as backing up heuristic values, could benefit the anytime profile of RWA*.

To our knowledge, restarts have not been used in a complete best-first search before. Our use of restarts in an A*-type algorithm is remarkable because at first glance, it would seem unnecessary: A* keeps a queue of all generated but yet unexpanded nodes, and is thus not thought to suffer from premature commitment. Our insight is that weighted A* effectively makes such commitments due to its low-h bias.

Empirical Evaluation

We compare the performance of RWA* with existing complete anytime approaches: Anytime A* (Hansen and Zhou 2007), ARA* (Likhachev *et al.* 2004), Anytime Window A* (Aine *et al.* 2007), and beam-stack search (Zhou and Hansen 2005). For beam-stack search, we implemented the regular version rather than the memory-limited divide-and-conquer variant. All parameters of the competitor algorithms were carefully tuned for best performance. The algorithms based on WA* (RWA*, Anytime A* and ARA*) all share the same code base, since they differ only in few details; and they perform best for the same starting weights. For beam-stack search, we experimented with beam width values between 5 and 10,000 and plot the best of those runs. We also conducted experiments with iteratively broadening beam widths, but did not obtain better results than with the best fixed value. Beam-stack search expects an initial upper bound on the solution cost. After consulting its authors, we chose a safe but large number as bound, as it is not clear how to come up with a reasonable upper bound for our experiments up-front.

In addition we compare against an alternative version of Anytime A*. For this algorithm, which originally does not decrease its weight between search phases, we experienced improved performance if we do decrease the weight. Thus, we also report results for a variant dubbed "Anytime A* WS", (WS for weight schedule), where the weight is decreased as in RWA* or ARA*.

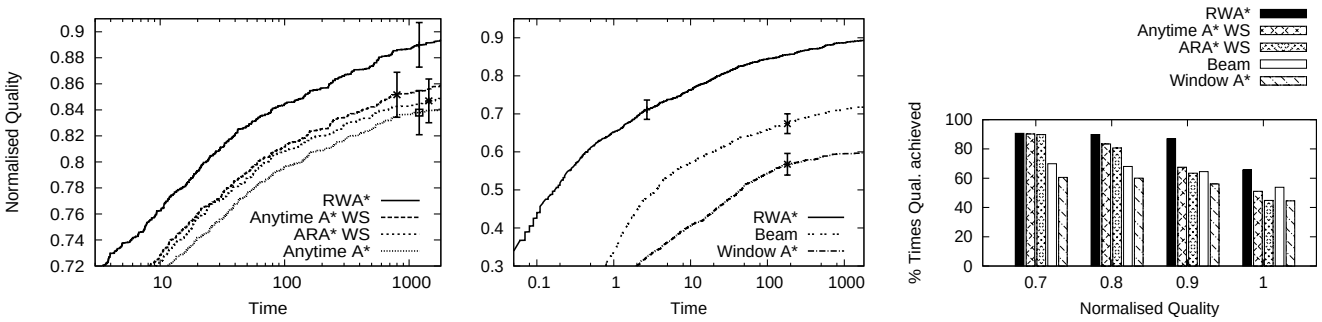


Figure 3: Anytime performance in PDDL planning.

Classical PDDL Planning

Planning is a notoriously hard problem where optimal solutions can be prohibitively expensive to obtain even under very optimistic assumptions (Helmert and Röger 2008). To efficiently find a (typically suboptimal) solution, inadmissible heuristics are employed. In our experiment, we use the popular FF heuristic and all classical planning tasks from the International Planning Competitions between 1998 and 2006.

Experimental Setup We implemented each of the search algorithms within the Fast Downward planning framework (Helmert 2006) used by the winners of the satisficing track in the International Planning Competitions in 2004 and 2008. This framework originally uses greedy best-first search, which we replaced with the respective search algorithms. All other settings of Fast Downward were held fixed to isolate the effect of changing the search mechanism.

Fast Downward has two options built in that can enhance the search for a solution. Since our goal is to achieve best possible performance for planning, we make use of these search enhancements. The first is *delayed heuristic evaluation*, which means that states are stored in Open with their parent's heuristic value rather than their own. This technique is helpful if the computation of the heuristic is very expensive compared to the other components of the search (like expansions of states). If many more states are generated than expanded, delayed evaluation leads to a great reduction in the number of times we need to calculate heuristic estimates, if at a loss in accuracy.

The second search enhancement in Fast Downward is the use of *preferred operators* (Helmert 2006). Preferred operators denote which successors of a state are deemed to lie on a promising path to the goal. They can be used as a second source of search information, complementing the heuristic. In particular, when run with the preferred-operator option, Fast Downward uses two Open lists, one with all operators and one with only preferred operators, and selects the next state from the two lists in an alternating fashion. For the A*-based algorithms, the preferred-operator mechanism can be used in a straightforward way and improves results notably; it was thus incorporated. Beam-stack search is the one algorithm that does not use preferred operators as it is not obvious how to incorporate them.

For the WA*-based algorithms, we experimented with several starting weights and corresponding weight sequences, yet found that the relative performance did not change much. The weight sequence which gave overall best results and is used in the results below is 5, 3, 2, 1.5, 1. ARA* was adapted to use a weight schedule like RWA* and Anytime A* WS, since the original mechanism for decreasing weights in ARA* depends on the heuristic being admissible (ARA* originally reduces its weight by a very small amount whenever it proves that the incumbent solution is w-admissible for the current weight w). For beam-stack search, a beam width of 500 resulted in best anytime performance.

The time and memory limits were 30 minutes and 3 GB respectively for each task, running on a 3 GHz Intel Xeon CPU. In order to compare on meaningful benchmark tasks only, we select a subset of them as follows: Firstly, we exclude tasks that none of the algorithms could solve. Secondly, since we are interested in showing the improvement of solutions over time, we exclude tasks where all algorithms found only one solution (and hence no improvement occurred). Thirdly, we exclude "trivial" tasks where all algorithms find the same best solution within less than one second. This leaves 1096 of the original 1612 tasks.

Results Fig. 3 summarises the results. We plot normalised quality scores following the definition used in the most recent planning competition (Do *et al.* 2008). For each point in time, the average normalised solution quality for algorithm A is computed as follows: for each task, A accrues the normalised quality score Q^*/Q, where Q is the cost of A's current solution and Q^* is cost of the overall best solution found by any of the algorithms. Hence, the score ranges between 0 (task not solved yet) and 1 (A has found the best solution quality Q^*). These scores are then averaged over all tasks, and error bars in the plots give 95% confidence intervals on the mean. It is intentional that unsolved tasks negatively influence the score of an algorithm. The alternative would be to compare only on the tasks solved by all algorithms. However, that would give an advantage to methods that are less greedy, i. e., those that solve fewer problems but find solutions of better quality. Hence that would not reward good anytime performance (consider that breadth-first search would win in that scenario).

As the left and centre panels of Fig. 3 show, RWA* performs best, outperforming the other algorithms by a substan-

	1st solution				2nd solution				3rd (final) solution			
	len	*c.i.*	*exp*	*t*	*len*	*c.i.*	*exp*	*t*	*len*	*c.i.*	*exp*	*t*
RWA*	165	-	1142	0.07	163	153	2370	0.08	**125**	**1**	16919	0.84
ARA*	165	-	1142	0.06	163	153	1220	0.07	161	145	5743	0.28
Anytime A* WS	165	-	1142	0.07	163	153	1231	0.08	161	145	5629	13.25
Anytime A*	165	-	1142	0.07	163	153	1255	0.07	161	145	206480	0.28

Table 1: Solution sequences for the largest task in the gripper domain. The plan length is denoted by *len*. The first step in which a solution deviates from the previous is denoted by *c.i.* (change index); *exp* is the number of states expanded until the solution is found, and *t* the runtime in seconds.

tial margin. In the left panel, we compare against the other WA*-based algorithms. The weight-decreasing variant of Anytime A* (Anytime A* WS) is on par with (or slightly better than) ARA*, whereas the original Anytime A* algorithm performs worse than ARA*. It is notable that while all four WA*-based algorithms are very closely related, differing only in a few lines of code, the simple addition of the restarts leads to such a significant improvement in performance.

The centre panel shows the results for Window A* and beam-stack search. Beam-stack search and Window A* perform significantly worse than the WA*-based algorithms. This is mainly due to solving fewer problems, as a significant determiner in gross performance is the number of problems solved by a certain time. By the time cut-off, all WA*-based algorithms solve (essentially the same) 84% of the tasks, while beam-stack search and Window A* solve only 78% and 63%, respectively. For 3% of the tasks it holds that they are solved by beam-stack search (Window A*) but not by the WA* algorithms. The opposite is true for 9% (24%) of the tasks.

The right panel of Fig. 3 shows for the best version of each algorithm how often it achieved (at least) a certain solution quality. As can be seen, RWA* achieves high-quality solutions significantly more often than the other algorithms. The best quality is achieved by RWA* in 65% of the tasks, respectively, while all others achieve it in $\leq 51\%$ of the tasks. The average number of solutions found was 3 for the weight-decreasing WA* methods and beam-stack search, 5 for non-decreasing Anytime A* and 1 for Window A*.

When examining the distribution of RWA*'s performance over the different domains, we found that RWA* outperforms the other WA* algorithms in 40% of the domains, while being on par in the remaining 60%. In no domain did RWA* perform notably worse than any of the other WA* methods. Beam-stack search and Window A* show different strengths and weaknesses compared to the WA*-based algorithms. In the Optical Telegraph domain beam-stack search performs well whereas all other algorithms perform badly. Beam-stack search and Window A* perform worst on large problems, e. g. in the Schedule and Logistics 1998 domains.

Summarising the results, we find that the WA*-based methods perform significantly better than Window A* and beam-stack search. We believe that this is due to the *global* WA*-based algorithms being better able to deal with inadmissible and locally highly varying heuristic values, whereas Window A* and beam-stack search commit early to certain areas of the search space. Exhausting a state-space area is particularly difficult here as inadmissible heuristic values cannot be used for pruning. Furthermore, beam-stack search cannot make use of the planning-specific search enhancement mentioned before, preferred operators.

We also performed experiments without preferred operators and delayed evaluation for all algorithms. There, the performance of all algorithms becomes drastically worse (e. g., the WA*-based methods leave twice as many problems unsolved and in addition show worse anytime performance on the solved problems). RWA* still performs better than all other algorithms, though by a much smaller margin than with the search enhancements.

We furthermore conducted experiments with other planning heuristics (not shown), including the recent landmark heuristic (Richter *et al.* 2008) and the context-enhanced additive heuristic (Helmert and Geffner 2008). Compared to the experiments described above, which use the FF heuristic, the performance of all algorithms was worse with these other heuristics. However, the *relative* performance of the algorithms was similar, with RWA* outperforming the other methods in all cases, though by smaller margins.

A Detailed Example We argue that RWA* shows such good results because restarts encourage changes in the beginning of a plan rather than the end. The largest task in the gripper domain provides an illustration. Gripper tasks consist in transporting balls (here 42) from one room to another with the help of a two-armed robot. The initial WA* search phase finds a plan in which all balls are transported separately, whereas in the optimal plan the robot always carries two balls at a time. Window A* does not solve this problem, while beam-stack search finds only one solution (the optimal) after 2.5 seconds. The WA* algorithms find three improving solutions, with the first solution found after less than one second, see Table 1. All these plans are found very quickly, in less than 15 seconds. However, the last plan found by RWA* is optimal, whereas the other WA* algorithms do not find any improved solutions during the remaining 29 minutes.

The noteworthy aspect in this example is the *indices of change* for the plans, i. e. the steps in which subsequent plans first differ from their predecessors. For ARA* and the two Anytime A* variants these change indices are fairly high ($\geq$ 145), denoting the fact that their subsequent plans only differ in the last 20 actions from the first plan found. For RWA*, the third solution has a change index of 1, i. e. it differs in

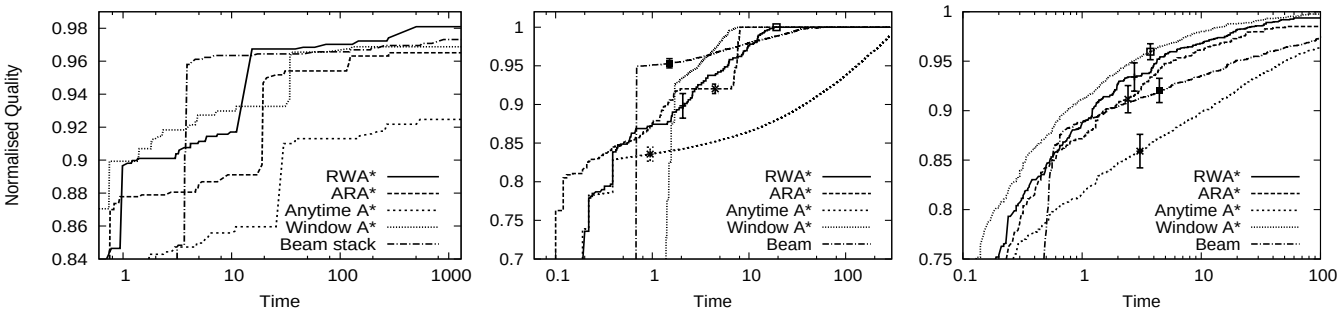


Figure 4: Anytime performance for robotic arm motion planning (left), grid pathfinding (centre), and the 15-puzzle (right).

the first action from the previous plans. This shows that the big jump in solution quality for RWA* indeed results from further exploration near the start state, whereas the other algorithms unsuccessfully spend their effort in deeper areas of the search space.

Other Experiments

We also ran experiments in three other benchmark domains: the robotic arm domain, the gridworld pathfinding domain and the sliding-tile puzzle. While these domains do not have the properties that we consider good for restarts (inadmissible or locally strongly varying heuristics), we chose them because of their previous use as anytime benchmarks (robotic arm), or their standing as traditional benchmarks in the search literature (gridworld and sliding-tile puzzle).

Robot Motion Planning The second domain we use for comparison is that of a simulated 20-degree-of-freedom robotic arm, as used by Likhachev et al. (2004). The base of the arm is fixed, and the task is to move its end-effector to the goal while navigating around obstacles. An action is defined as a change of the global angle at a particular joint. The workspace is discretised into 50×50 cells, and the heuristic of a state is the distance from the current location of the robot's end-effector to the goal, taking obstacles into account. The size of the state space is over 10^{26}, and in most instances it is infeasible to find an optimal solution.

As before, we eliminate trivial instances, leaving 17 of 22 tasks kindly made available to us by Maxim Likhachev. For ARA*, we use the settings suggested by Likhachev *et al.*, starting with a weight of 3 and decreasing it by 0.02 each time; the other WA*-based algorithms also start with weight 3. Note that because the heuristic is admissible, ARA* can prove suboptimality bounds and make informed decisions on when to reduce its weight (namely, whenever a new bound is proven). For RWA*, on the other hand, a decrease in weight incurs a substantial overhead each time due to the restart. This is why larger and less frequent weight decreases work better for it. We use a similar weight decay for RWA* as in the planning experiment (a factor of 0.84), reducing the weight whenever an improved solution is found.

The results are shown in the left panel of Fig. 4. RWA* outperforms ARA* and Anytime A* for all timeouts above 1 second and shows overall very good anytime performance, including reaching the best final quality. Window A* does well in this domain and shows best results in the early stages of search. Anytime A* performs notably worse than ARA* here, though we found that its weight-decreasing variant (not shown) achieves similar performance as ARA*. Beam-stack search, here with a beam width of 100, takes longer than most of the other methods to achieve high performance, but then comes up steeply and achieves almost the same final quality as RWA*. (Error bars are not shown here due to the high variance on these hand-designed instances.)

Gridworld Gridworld tasks consist in finding a path from the top left to the bottom right in a 2000×1200 cell grid, using 4-way movement. We randomly generated 20 tasks with a probability of 0.35 of each cell being blocked. We used a unit cost function and a Manhattan distance heuristic which ignores obstacles. The starting weight was 2 for the WA*-based algorithms, the weight schedule of RWA* being 2, 1.5, 1.25, 1.125, while ARA* uses small weight decrements of 0.2. Beam-stack search used a beam width of 50.

Results are shown in the centre panel of Fig. 4. The tasks are solved optimally within 60 seconds by all algorithms except non-decreasing Anytime A*. RWA* and ARA* show the best anytime performance and perform very similarly. ARA* performs better than the others in the first split second, beam-stack search between seconds 0.8–3. However, beam-stack search and Window A* take longer than RWA* and ARA* to achieve good quality. Anytime A* did not perform well, though we found again that its weight-decreasing variant (not shown) performs similarly to RWA* and ARA*. With smaller beam widths, the score of beam-stack search rises faster in the beginning but it takes longer to achieve perfect quality; with larger beam widths the reverse holds. With beam width 100, for example, beam-stack search performs similarly to Window A* in this domain.

Sliding-Tile Puzzle In the sliding-tile puzzle domain, we tested on the 100 15-puzzle instances from Korf (1985) using the Manhattan distance heuristic. The starting weight for the WA* algorithms was 3, the weight sequence for RWA* being 3, 2, 1.5, 1.25, 1 while ARA* uses small decrements of 0.2. The beam width in beam-stack search was 500.

The results are shown in the right panel of Fig. 4. While these tasks are harder to solve optimally, there are comparatively few different solutions, leading to similar performance of all WA*-based algorithms that decrease weight. Window A* performs best, with RWA* second and ARA* third. Beam-stack search performs less well, showing the same behaviour relative to its beam width as in the gridworld, and non-decreasing Anytime A* again performs worst.

Summary While in these domains RWA* does not dominate its competitors as notably as in the PDDL planning experiment, we note that its performance is continually very good. Compared to its most similar competitors, the other WA* algorithms, RWA* is always better (robotic arm) or on par (gridworld, sliding-tile puzzle). This may suggest that even in cases where the restarts are not helpful, they do little harm in terms of computation time. The methods that are not based on WA* (Window A* and beam-stack search) perform well in some domains but badly in others. By contrast, RWA* shows robustly good performance over all domains.

Discussion

As we saw in the example tasks from gridworld and the gripper domain, restarting anytime search overcomes the low-h bias that tends to expand nodes near a known goal. The desire to revisit nodes near the start state stems from the fact that a heuristic evaluation function is often less informed near the root of the search tree than near a goal and that early mistakes can be important to correct (Harvey and Ginsberg 1995; Furcy and Koenig 2005). For example, in problems with multiple goal states the heuristic may misjudge the relative distance of the goals and lead a greedy search into a suboptimal part of the search space. Similarly, domains with inaccurate and locally strongly varying heuristics can lead to early mistakes and thus benefit from restarts.

Conclusion

We have demonstrated an important dysfunction in conventional approaches to anytime search: by trying to re-use previous search effort from a greedy search, traditional anytime methods suffer from low-h bias and tend to improve the incumbent solution starting from the end of the solution path. As we showed in this paper, this can be a poor strategy when the heuristic makes early mistakes. The counter-intuitive approach of clearing the Open list, as in the Restarting Weighted A* algorithm, can lead to much better performance in such domains while doing little harm in others.

Acknowledgements

We thank Malte Helmert, Charles Gretton, and Patrik Haslum for their very helpful input, and Maxim Likhachev for making his code available.

NICTA is funded by the Australian Government, as represented by the Department of Broadband, Communications and the Digital Economy, and the Australian Research Council, through the ICT Centre of Excellence program. We also gratefully acknowledge support from NSF grant IIS-0812141 and the DARPA CSSG program.

References

Sandip Aine, P. P. Chakrabarti, and Rajeev Kumar. AWA* – A window constrained anytime heuristic search algorithm. In *Proc. IJCAI 2007*, pages 2250–2255, 2007.

Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *AIJ*, 72(1):81–138, 1995.

Blai Bonet and Hector Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. ICAPS 2003*, 2003.

Rina Dechter and Judea Pearl. The optimality of A*. In Laveen Kanal and Vipin Kumar, editors, *Search in Artificial Intelligence*, pages 166–199. Springer-Verlag, 1988.

Minh Binh Do, Malte Helmert, and Ioannis Refanidis. IPC-6, deterministic part. Web site, `http://ipc.informatik.uni-freiburg.de/`, 2008.

David Furcy and Sven Koenig. Limited discrepancy beam search. In *Proc. IJCAI 2005*, pages 125–131, 2005.

Carla P. Gomes, Bart Selman, and Henry A. Kautz. Boosting combinatorial search through randomization. In *Proc. AAAI-98*, pages 431–437, 1998.

Eric A. Hansen and Rong Zhou. Anytime heuristic search. *JAIR*, 28:267–297, 2007.

Eric A. Hansen, Shlomo Zilberstein, and Victor A. Danilchenko. Anytime heuristic search: First results. CMPSCI 97-50, University of Massachusetts, Amherst, September 1997.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proc. IJCAI-95*, pages 607–613, 1995.

Malte Helmert and Héctor Geffner. Unifying the causal graph and additive heuristics. In *Proc. ICAPS 2008*, pages 140–147, 2008.

Malte Helmert and Gabriele Röger. How good is almost perfect? In *Proc. AAAI 2008*, pages 944–949, 2008.

Malte Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

Richard E. Korf. Depth-first iterative-deepending: An optimal admissible tree search. *AIJ*, 27(1):97–109, 1985.

Maxim Likhachev, Geoffrey J. Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proc. NIPS-03*, 2004.

Maxim Likhachev, Dave Ferguson, Geoffrey J. Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *AIJ*, 172(14):1613–1643, 2008.

Ira Pohl. Heuristic search viewed as path finding in a graph. *AIJ*, 1:193–204, 1970.

Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proc. AAAI 2008*, pages 975–982, 2008.

Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. AAAI-92*, pages 440–446, 1992.

Weixiong Zhang. Complete anytime beam search. In *Proc. AAAI-98*, pages 425–430, 1998.

Rong Zhou and Eric A. Hansen. Beam-stack search: Integrating backtracking with beam search. In *Proc. ICAPS 2005*, pages 90–98, 2005.