ANA*: Anytime Nonparametric A*

Jur van den Berg[1] and **Rajat Shah**[2] and **Arthur Huang**[2] and **Ken Goldberg**[2]
[1]University of North Carolina at Chapel Hill. E-mail: berg@cs.unc.edu.
[2]University of California, Berkeley. E-mail: {rajatm.shah, arthurhuang, goldberg}@berkeley.edu.

Abstract

Anytime variants of Dijkstra's and A* shortest path algorithms quickly produce a suboptimal solution and then improve it over time. For example, ARA* introduces a weighting value (ε) to rapidly find an initial suboptimal path and then reduces ε to improve path quality over time. In ARA*, ε is based on a linear trajectory with ad-hoc parameters chosen by each user. We propose a new Anytime A* algorithm, Anytime Nonparametric A* (ANA*), that does not require ad-hoc parameters, and adaptively reduces ε to expand the most promising node per iteration, adapting the greediness of the search as path quality improves. We prove that each node expanded by ANA* provides an upper bound on the suboptimality of the current-best solution. We evaluate the performance of ANA* with experiments in the domains of robot motion planning, gridworld planning, and multiple sequence alignment. The results suggest that ANA* is as efficient as ARA* and in most cases: (1) ANA* finds an initial solution faster, (2) ANA* spends less time between solution improvements, (3) ANA* decreases the suboptimality bound of the current-best solution more gradually, and (4) ANA* finds the optimal solution faster. ANA* is freely available from Maxim Likhachev's Search-based Planning Library (SBPL).

1 Introduction

The A* algorithm (Hart, Nilsson, and Raphael 1968) is widely used to compute minimum-cost paths in graphs in applications ranging from map navigation software to robot path planning to AI for games. Given an admissible heuristic, A* is guaranteed to find an optimal solution (Dechter and Pearl 1985). In time-critical applications such as robotics, rather than waiting for the optimal solution, *anytime* algorithms quickly produce an initial, suboptimal solution and then improve it over time. Most existing anytime A* algorithms are based on *Weighted A**, which inflates the heuristic node values by a factor of $\varepsilon \geq 1$ to trade off running time versus solution quality (Pohl 1970). Weighted A* repeatedly expands the "open" state s that has a minimal value of:

$$f(s) = g(s) + \varepsilon \cdot h(s),$$

where $g(s)$ is the current-best cost to move from the start state to s, and $h(s)$ is the heuristic function, an estimate of the cost to move from s to a goal state. The higher ε, the greedier the search and the sooner a solution is found. If the heuristic is admissible (i.e. a lower-bound on the true distance to the goal), the suboptimality of the solution is bounded by ε. That is, the solution is guaranteed to be no costlier than ε times the cost of the optimal path. These observations can be used in anytime algorithms, for instance as in ARA* (Likhachev, Gordon, and Thrun 2004), which initially runs Weighted A* with a large value of ε to quickly find an initial solution, and continues the search with progressively smaller values of ε to improve the solution and reduce its suboptimality bound. To our knowledge, all existing anytime A* algorithms require users to set parameters. ARA*, for instance, has two parameters: the initial value of ε and the amount by which ε is decreased in each iteration. Setting these parameters requires trial-and-error and domain expertise (Aine, Chakrabarti, and Kumar 2007).

This motivated us to develop an anytime A* algorithm that does not require parameters. Instead of minimizing $f(s)$, Anytime Nonparametric A* (ANA*) expands the open state s with a *maximal* value of

$$e(s) = \frac{G - g(s)}{h(s)},$$

where G is the cost of the current-best solution, initially an arbitrarily large value. The value of $e(s)$ is equal to the maximal value of ε such that $f(s) \leq G$. Hence, continually expanding the node s with maximal $e(s)$ corresponds to the greediest possible search to improve the current solution that in effect automatically adapts the value of ε as the algorithm progresses and path quality improves. In addition, we will prove that the maximal value of $e(s)$ provides an upper bound on the suboptimality of the current best solution, which is hence gradually reduced while ANA* searches for an improved solution. ARA*, in contrast, lets the value of ε follow a "linear" trajectory, resulting in highly unpredictable search times between the fixed decrements of ε.

In addition to eliminating ad-hoc parameters, results of experiments in representative search domains suggest that ANA* has superior "anytime characteristics" compared to ARA*. That is: (1) ANA* usually finds an initial solution faster, (2) ANA* usually spends less time between solution improvements, (3) ANA* more gradually decreases the suboptimality bound of the current-best solution, and (4) ANA* usually finds the optimal solution faster.

Our implementation of ANA* is freely available from Maxim Likhachev's Search-based Planning Library (SBPL) at `http://www.cs.cmu.edu/~maxim/software.html`.

2 Preliminaries and Previous Work

2.1 Dijkstra's, A*, and Weighted A*

Dijkstra's algorithm, introduced in 1959 (Dijkstra 1959), finds a minimum-cost path between a start state s_{start} and a goal state s_{goal} (or a set of goal states) in a graph with non-negative edge costs. For each state s in the graph, it maintains a value $g(s)$, which is the minimum cost proven so far to reach s from s_{start}. Initially, all $g(s)$ are ∞, except for the start state, whose g-value is initialized at 0. The algorithm maintains an *OPEN* queue containing all *locally inconsistent* states, i.e. states s that may have successors s' for which $g(s') > g(s) + c(s,s')$, where $c(s,s')$ is the cost of traversing the edge between s and s'. Initially, *OPEN* only contains the start state s_{start}. Continually, Dijkstra's algorithm extracts the state s from *OPEN* with minimal $g(s)$ value, and *expands* it. That is, it updates the g-values of the successors of s, and puts them on the *OPEN* queue if their g-value was decreased. This continues until state s_{goal} is extracted from *OPEN*, or until *OPEN* is empty, in which case a solution does not exist. Dijkstra's algorithm is optimal in terms of big-O notation (Cormen, Leiserson, and Rivest 1990), and runs in $O(n \log n + k)$ time, where n and k are the number of states and edges, respectively.

The *A* algorithm*, introduced in 1968, is a generalization of Dijkstra's algorithm that improves its running time in practice if a *heuristic* is available, by focusing the search towards the goal (Hart, Nilsson, and Raphael 1968). The difference from Dijkstra's is that A* expands the state s in *OPEN* with a minimal value of $g(s)+h(s)$, where $h(s)$ is the heuristic that estimates the cost of moving from s to s_{goal}. Let $c^*(s,s')$ denote the cost of the optimal path between s and s'. If the heuristic is *admissible*, i.e. if $h(s) \leq c^*(s, s_{\text{goal}})$ for all s, A* is guaranteed to find the optimal solution in optimal running time (Dechter and Pearl 1985). If the heuristic is also *consistent*, i.e. if $h(s) \leq c^*(s,s') + h(s')$ for all s and s', it can be proven that no state is expanded more than once by the A* algorithm. Note that A* is equivalent to Dijkstra's when $h(s) = 0$.

*Weighted A** extends A* by allowing to trade-off running time and solution quality (Pohl 1970). It is similar to A*, except that it inflates the heuristic by a value $\varepsilon \geq 1$ and expands the state s in *OPEN* with minimal $f(s) = g(s) + \varepsilon \cdot h(s)$. The higher ε, the greedier the search, and the sooner a solution is typically found. The *suboptimality* of solutions found by Weighted A* is bounded by ε, i.e. the solution is guaranteed to be no costlier than ε times the cost of the optimal solution. Weighted A* may expand states more than once (as the inflated heuristic $\varepsilon \cdot h(s)$ is typically not consistent). However, if $h(s)$ itself is consistent, it can be proven that restricting states to being expanded no more than once does not invalidate the ε-suboptimality bound (Likhachev, Gordon, and Thrun 2004).

2.2 Anytime A* Algorithms Based on Weighted A*

Anytime Heuristic Search (AHS) (Hansen and Zhou 2007) is an anytime version of Weighted A*. It finds an initial solution for a given value of ε, and continues the search after an initial solution is found (with the same ε). Each time the goal state is extracted from *OPEN*, an improved solution is found. Eventually, AHS will find the optimal solution. Throughout, AHS expands the state in *OPEN* with minimal $f(s) = g(s) + \varepsilon \cdot h(s)$, where ε is a parameter of the algorithm. The suboptimality of intermediate solutions can be bounded by $G / \min_{s \in OPEN}\{g(s) + h(s)\}$, as G, the cost of the current-best solution, is an upper bound of the cost of the optimal solution, and $\min_{s \in OPEN}\{g(s) + h(s)\}$ is a lower bound of the cost of the optimal solution.

*Anytime Repairing A** (ARA*) (Likhachev, Gordon, and Thrun 2004) is also based on Weighted A*. It finds an initial solution for a given initial value of ε, and continues the search with progressively smaller values of ε to improve the solution and reduce its suboptimality bound. The value of ε is decreased by a fixed amount each time an improved solution is found or the current-best solution is proven to be ε-suboptimal. The $f(s)$-values of the states $s \in OPEN$ are then updated to account for the new value of ε. The initial value of ε and the amount by which it is decreased in each iteration are parameters of the algorithm. The algorithm we present in this paper was motivated by ARA*. We will discuss their relation in detail in Section 4.

*Restarting Weighted A** (RWA*) (Richter, Thayer, and Ruml 2010) is similar to ARA*, but it restarts the search each time ε is decreased. That is, each search is started with only the start state on the *OPEN* queue. It reuses the effort of previous searches by putting the states explored in previous iterations on a *SEEN* list. Each time the search encounters a seen state, it is put back on the *OPEN* queue regardless of whether its g-value was decreased. Restarting has proven to be effective in situations where the quality of the heuristic varies substantially across the search space. As with ARA*, the initial value of ε and the amount by which it is decreased in each iteration are parameters of the algorithm.

2.3 Other Anytime A* Algorithms

*Anytime Window A** (AWA*) (Aine, Chakrabarti, and Kumar 2007) is based on unweighted A*, but only expands states within an active window that slides along with the "deepest" state expanded so far. Only states with a g-value inside the window (i.e. the g-value is larger than the largest g-value among the states expanded so far minus the window size) are put on the *OPEN* queue. The other states are put on an auxiliary *SUSPEND* list. Iteratively, the window size is increased to broaden the search, which will eventually become equivalent to A*. A problem of this algorithm is that if it "misses" the goal state in its initial search (i.e. the goal state is outside the window), it exhausts the entire search space before the initial search terminates, which may take prohibitively long.

Beam-Stack search (BSS) (Zhou and Hansen 2005) and *ITSA** (Furcy 2006) are based on beam search, which is an

IMPROVESOLUTION()

```
1: while OPEN ≠ ∅ do
2:    s ← arg max_{s∈OPEN}{e(s)}
3:    OPEN ← OPEN \ {s}
4:    if e(s) < E then
5:       E ← e(s)
6:    if ISGOAL(s) then
7:       G ← g(s)
8:       return
9:    for each successor s' of s do
10:      if g(s) + c(s,s') < g(s') then
11:         g(s') ← g(s) + c(s,s')
12:         pred(s') ← s
13:         if g(s') + h(s') < G then
14:            Insert or update s' in OPEN with key e(s')
```

ANA*()

```
1: G ← ∞; E ← ∞; OPEN ← ∅; g(s_start) ← 0
2: Insert s_start into OPEN with key e(s_start)
3: while OPEN ≠ ∅ do
4:    IMPROVESOLUTION()
5:    Report current E-suboptimal solution
6:    Update keys e(s) in OPEN and prune states if g(s) +
      h(s) ≥ G
```

Figure 1: The Anytime Nonparametric A* (ANA*) algorithm. In practice the values for ∞ are set to large numerical values.

augmented form of breadth-first search in which only the b most promising states of each layer are expanded. Beam search itself is incomplete, i.e. it may not find a solution even if one exists, but BSS and ITSA* make beam search complete by adding backtracking mechanisms. Iteratively increasing the beam width b provides the anytime characteristic. These algorithms have the same problem as Anytime Window A*, as the initial search is essentially a depth-first search that may "miss" the goal state.

3 Anytime Nonparametric A*

In this section, we present our anytime nonparametric algorithm ANA* and discuss its properties.

3.1 Algorithm

Our algorithm ANA* is given in Fig. 1. Throughout the algorithm, a global variable G is maintained storing the cost of the current-best solution. Initially, $G = \infty$, as no solution has yet been found.

IMPROVESOLUTION implements a version of A* that is adapted such that it expands the state $s \in OPEN$ with the *maximal* value of

$$e(s) = \frac{G - g(s)}{h(s)} \tag{1}$$

(line 2). Each time a state s is expanded, it is checked whether the g-value of the successors s' of s can be decreased (line 10). If so, $g(s')$ is updated (line 11) and the predecessor of s' is set to s (line 12) such that the solution can be reconstructed once one is found. Subsequently, s' is inserted into the *OPEN* queue with key $e(s')$, or if it was already on the *OPEN* queue, its key $e(s')$ is updated (line 14). States for which $g(s) + h(s) \geq G$, or equivalently $e(s) \leq 1$, are not put on the *OPEN* queue, though, as such states will never contribute to improving the current-best solution (line 13). As a result, when a goal state is extracted from *OPEN*, it is guaranteed that a solution has been found with lower cost than the current-best solution, so G is updated (line 7) and IMPROVESOLUTION terminates (line 8).

ANA* is the "main" function that iteratively calls IMPROVESOLUTION to improve the current solution. It starts by initalizing the g-value of the start state s_{start} to zero and putting it on *OPEN* (line 2). In the first iteration, $G = \infty$, as no solution has yet been found, in which case IMPROVESOLUTION expands the state in *OPEN* with the smallest $h(s)$-value, and in case of ties the one with the smallest $g(s)$-value. This follows naturally from Equation (1) if one thinks of G as a very large but finite number. This is equivalent to executing Weighted A* minimizing $f(s)$ with $\varepsilon = \infty$, so the search for an initial solution is maximally greedy.

Each time IMPROVESOLUTION terminates, either an improved solution has been found, or the *OPEN* queue has run empty, in which case the current-best solution is optimal (or no solution exists if none was found yet). After an improved solution has been found by IMPROVESOLUTION, the solution may be reported (line 5 of ANA*) and the keys of the states in *OPEN* are updated to account for the new value of G (line 6). States s for which $g(s) + h(s) \geq G$ are pruned from *OPEN*, as they will never contribute to an improved solution (line 6). Subsequently, the *OPEN* queue is reordered given the updated keys, and IMPROVESOLUTION is called again. This repeats until *OPEN* is empty, in which case the optimal solution has been found. Note that successive executions of IMPROVESOLUTION reuse the search effort of previous iterations.

3.2 Suboptimality Bound

Our algorithm ANA* provides a suboptimality bound of the current-best solution that gradually decreases as the algorithm progresses. Each time a state s is selected for expansion in line 2 of IMPROVESOLUTION, its $e(s)$-value bounds the suboptimality of the current-best solution. We prove the theorem below. We denote by G^* the cost of the optimal solution, so G/G^* is the true suboptimality of the current-best solution (recall that G is the cost of the current solution). Further, we denote by $g^*(s) = c^*(s_{\text{start}}, s)$ the cost of the optimal path between the start state s_{start} and s.

Lemma: If the optimal solution has not yet been found, there must be a state $s \in OPEN$ that is part of the optimal solution and whose g-value is optimal, i.e. $g(s) = g^*(s)$.

Proof: Initially, $s_{\text{start}} \in OPEN$ is part of the optimal solution and $g(s_{\text{start}}) = g^*(s_{\text{start}}) = 0$. At each iteration a state s from *OPEN* is expanded. s is either part of the optimal solution and $g(s)$ is optimal, or not. In the latter case, the state with the above property remains in *OPEN* (its g-value is optimal and cannot be decreased). In the former case, s must have a successor s' that is part of the optimal solution. The suc-

cessor's updated g-value is optimal since edge (s,s') is part of the optimal solution and $g(s') = g(s) + c(s,s')$. This continues until the goal state is dequeued with optimal g-value when the optimal path has been found. □

Theorem: *Each time a state s is selected for expansion in line 2 of* IMPROVESOLUTION*, its $e(s)$-value bounds the suboptimality of the current solution:*

$$\max_{s \in OPEN} \{e(s)\} \geq \frac{G}{G^*}.$$

Proof: We assume that the heuristic is admissible. If the current solution is optimal, the theorem trivially holds, as *OPEN* only contains states with an e-value greater than 1. If the optimal solution has not yet been found, there must be a state $s' \in OPEN$ that is part of the optimal solution and whose g-value is optimal, i.e. $g(s') = g^*(s')$ (see Lemma). The minimal cost to move from s' to the goal is $G^* - g^*(s')$, since s' is part of the optimal solution. As the heuristic is admissible, $h(s') \leq G^* - g^*(s')$. Therefore:

$$e(s') = \frac{G - g^*(s')}{h(s')} \geq \frac{G - g^*(s')}{G^* - g^*(s')} \geq \frac{G}{G^*},$$

where the last inequality follows as $G > G^* \geq g^*(s') \geq 0$. So, $\max_{s \in OPEN}\{e(s)\} \geq e(s') \geq G/G^*$. □

In the algorithm of Fig. 1, we keep track of the suboptimality bound of the current-best solution in the variable E. Initially $E = \infty$, as no solution has been found yet. Each time a state s is encountered in line 2 of IMPROVESOLUTION with $e(s) < E$, we update the value of E (line 5). Hence, the algorithm gradually decreases the suboptimality bound of the current solution while it is searching for an improved solution.

4 Comparison with ARA*

Selecting the state $s \in OPEN$ with a maximal value of $e(s)$ for expansion as we do in ANA* can intuitively be understood as selecting the state that is most promising for improving the current-best solution, as $e(s)$ is the ratio of the "budget" that is left to improve the current solution $(G - g(s))$ and the estimate of the cost between the state and the goal $h(s)$. This, however, is not our motivation for choosing the ordering criterion $e(s)$; in fact, it is derived by careful analysis of the existing anytime algorithm ARA* (Likhachev, Gordon, and Thrun 2004). We discuss this connection in detail in this section.

For completeness, a simplified version of the ARA* algorithm is given in Fig. 2.[1] Like Weighted A*, ARA* expands the state $s \in OPEN$ with a minimal value of

$$f(s) = g(s) + \varepsilon \cdot h(s).$$

[1] A feature of ARA* that is not included in our description is that it blocks a state to be opened more than once within the same iteration of IMPROVESOLUTION. Instead of reinserting these states on the *OPEN* queue, they are stored in an auxiliary list called *INCONS*, whose states will be put back in *OPEN* only in the next iteration. If the heuristics are consistent, this optimization does not invalidate the suboptimality bound of the solution that is found. This optimization is not relevant for the discussion, and is left out for simplicity.

IMPROVESOLUTION()
1: **while** $OPEN \neq \emptyset$ **and** $\min_{s \in OPEN}\{f(s)\} \leq G$ **do**
2: $s \leftarrow \arg\min_{s \in OPEN}\{f(s)\}$
3: $OPEN \leftarrow OPEN \setminus \{s\}$
4: **if** ISGOAL(s) **then**
5: $G \leftarrow g(s)$
6: **return**
7: **for each** successor s' of s **do**
8: **if** $g(s) + c(s,s') < g(s')$ **then**
9: $g(s') \leftarrow g(s) + c(s,s')$
10: $\text{pred}(s') \leftarrow s$
11: **if** $g(s') + h(s') < G$ **then**
12: Insert or update s' in *OPEN* with key $f(s')$

ARA*($\varepsilon_0, \Delta\varepsilon$)
1: $G \leftarrow \infty$; $\varepsilon \leftarrow \varepsilon_0$; $OPEN \leftarrow \emptyset$; $g(s_{\text{start}}) \leftarrow 0$
2: Insert s_{start} into *OPEN* with key $f(s_{\text{start}})$
3: **while** $OPEN \neq \emptyset$ **do**
4: IMPROVESOLUTION()
5: Publish current ε-suboptimal solution.
6: $\varepsilon \leftarrow \varepsilon - \Delta\varepsilon$
7: Update keys $f(s)$ in *OPEN* and prune states if $g(s) + h(s) \geq G$

Figure 2: A simplified version of the ARA* algorithm. ARA* is the "main" function.

ARA* is similar in structure to ANA*, and iteratively calls its version of IMPROVESOLUTION, initially with $\varepsilon = \varepsilon_0$, and after each iteration, ε is decreased by a fixed amount $\Delta\varepsilon$ (line 6 of ARA*). IMPROVESOLUTION terminates either when an improved solution is found (line 6), which is then guaranteed to be ε-suboptimal, or when $\min_{s \in OPEN}\{f(s)\} > G$ (line 1), in which case the current-best solution is proven to be ε-suboptimal.

The initial value ε_0 of ε and the amount $\Delta\varepsilon$ by which it is decreased after each iteration are parameters of the ARA* algorithm. Setting these parameters is non-trivial. A first property of a good anytime algorithm is that it finds an initial solution as soon as possible, such that a solution can be given even if little time is available. Ideally, therefore, $\varepsilon_0 = \infty$, as the higher ε, the greedier the search and the sooner a solution is found. However, setting $\varepsilon_0 = \infty$ is not possible in ARA*, as ε is later decreased with finite steps (line 6 of ARA*). For that reason, ε is initialized with a finite value ε_0 in ARA*.

A second desirable property is to reduce the time spent between improvements of the solution, such that when the current-best solution is requested by the host, the least amount of time has been spent in vain. The amount $\Delta\varepsilon$ by which ε is decreased should therefore be as small as possible (this is also argued by the authors of ARA* in (Likhachev, Gordon, and Thrun 2004)). However, if ε is decreased by too little, it is possible that the subsequent iteration of IMPROVESOLUTION does not expand a single state: recall that IMPROVESOLUTION terminates when $\min_{s \in OPEN}\{f(s)\} > G$. If ε is hardly decreased in the next iteration, it might still be the case that $\min_{s \in OPEN}\{f(s)\} > G$. So, what is the maximal value of ε for which at least one state can be expanded?

That is when

$$\varepsilon = \max_{s \in OPEN} \{e(s)\}, \quad (2)$$

which follows from the fact that $f(s) \leq G \iff \varepsilon \leq e(s)$. The one state that can then be expanded is indeed the state $s \in OPEN$ with a maximal value of $e(s)$. This is precisely the state that ANA* expands.

As an alternative to ANA*, one could imagine an adapted version of ARA* that uses Equation (2) to decrease ε by the least possible amount after each iteration of IMPROVESOLUTION. This would also allow initializing ε at ∞ for the first iteration. However, such an algorithm would very often have to update the $f(s)$-keys of the states $s \in OPEN$ (and reorder the *OPEN* queue) to account for the new value of ε. This takes $O(n)$ time, if n is the number of states in *OPEN*. Also, ARA* is not maximally greedy to find an improved solution: after the new value of ε has been determined, it remains fixed during the subsequent iteration of IMPROVESOLUTION. However, new states s for which $f(s) < G$ may be put on the *OPEN* queue and expanded during that iteration of IMPROVESOLUTION. If $f(s) < G$, state s would also have been expanded if ε were increased again (up to $e(s)$). A higher ε corresponds to a greedier search, so instead one could always maximize ε such that there is at least one state $s \in OPEN$ for which $f(s) \leq G$. This is equivalent to what ANA* does, by continually expanding the state $s \in OPEN$ with a maximal value of $e(s)$.

In summary, ANA* improves on ARA* in five ways: (1) ANA* does not require parameters to be set; (2) ANA* is maximally greedy to find an initial solution; (3) ANA* is maximally greedy to improve the current-best solution; (4) ANA* gradually decreases the suboptimality bound of the current-best solution; and (5) ANA* only needs to update the keys of the states in the *OPEN* queue when an improved solution is found.

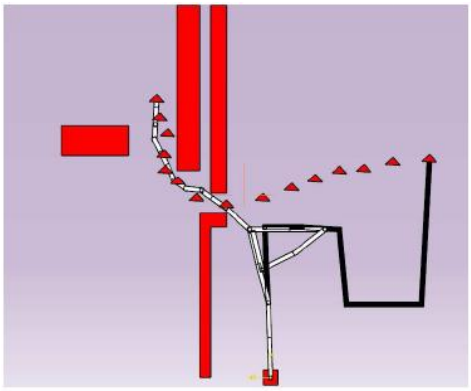
5 Experimental Results

In preliminary testing (see Section 7) we found that ARA* had the strongest performance compared to other anytime A* algorithms, so we focused our experiments on the comparison between ANA* and ARA*.

We implemented ANA* with Maxim Likhachev's SBPL library, which is publicly available on his website and contains a framework with his own ARA* planner and multiple benchmark domains (Likhachev, Gordon, and Thrun 2004). We tested ANA* and ARA* on Likhachev's robotic arm trajectory planning problem. As this problem has high branching factor and few duplicate states and may not be representative (Thayer and Ruml 2010), we also perform experiments with the Gridworld and multiple sequence alignment problems, which have search domains of bounded depth and relatively small branching factor.

All experiments were implemented in C++ and executed on a 32-bit Windows, Intel Pentium Dual Core machine with 1.8 GHz processors and 3GB of RAM.

The quality of an anytime algorithm is measured by its performance with respect to several metrics: (1) the speed with which the algorithm finds an initial solution, (2) the speed and (3) frequency with which the algorithm decreases the suboptimality bound of the current-best solution, (4) the speed with which the algorithm converges to the optimal solution, and (5) the frequency with which each algorithm improves the current-best solution. In our experiments we explore the relative performance of ARA* and ANA* with respect to these five metrics.



(a) 20DOF Arm Navigating Around Obstacles

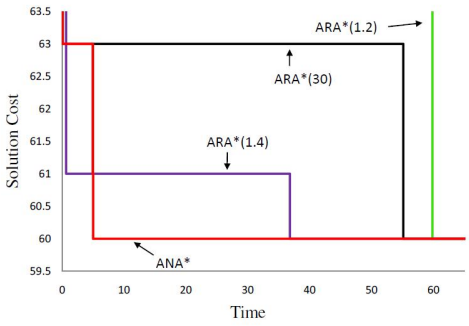
Figure 3: Robotic Arm Trajectory: The red rectangles are obstacles; initially, the robot end effector is at the right in black. The final configuration reaches through the opening and is illustrated in white; triangles indicate the end effector trajectory as computed by ANA*.

5.1 Robot Arm Experiment

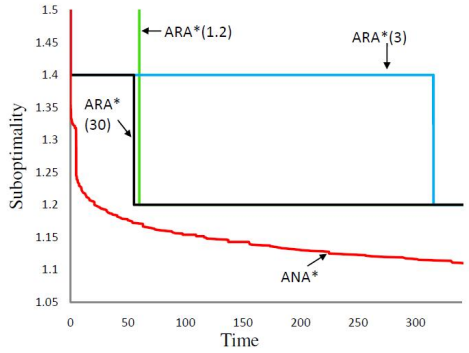
We simulate both a 6-degree-of-freedom arm and the 20-degree-of-freedom (DOF) arm shown in Fig. 3(a). The arms have a fixed base. The objective is to move the end-effector from its initial location to the pre-determined goal location while avoiding obstacles (indicated by red rectangles). An action is defined as a change of a global angle of any particular joint (i.e., the next joint further along the arm rotates in the opposite direction to maintain the global angle of the remaining joints). The cost of each action for the arm can either be non-uniform, i.e. changing a joint angle closer to the base is more expensive than changing a joint angle closer to the end-effector, or all actions can have the same cost.

The environment is discretized into a 50x50 2D grid. The heuristic is calculated as the shortest distance from the current location of the end-effector to the goal location that avoids obstacles. To avoid having the heuristic overestimate true costs, joint angles are discretized so as to never move the end-effector by more than one cell on the 50x50 grid in a single action. Memory for each state is allocated on demand, resulting in an expansive planning space of over 3 billion states for a 6 DOF robot arm and over 10^{26} states for the 20 DOF robot arm.

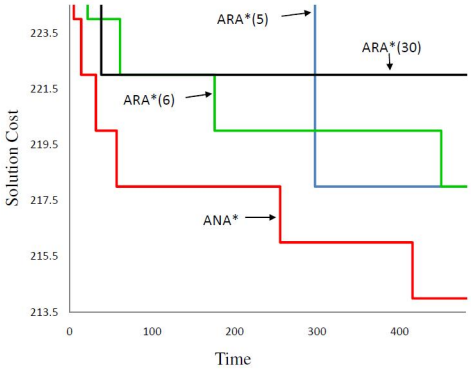
For each planning domain, we executed ARA* with different values of parameters ε_0 and $\Delta\varepsilon$. We found that ARA* run times and performance were not linearly correlated with these parameters, suggesting that finding good values is non-trivial. In the interest of space, we focus on the effect of parameter ε_0 and fix the ARA* parameter $\Delta\varepsilon$ at the value of 0.2 as recommended by (Likhachev, Gordon, and Thrun



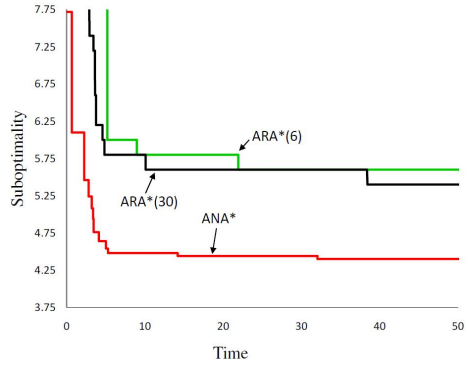
(a) 6DOF, Uniform



(b) 6DOF, Uniform, Suboptimality

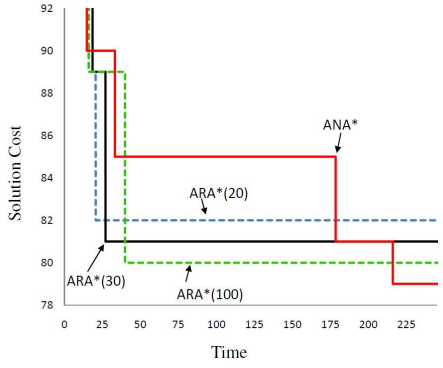


(c) 6DOF, Non-uniform

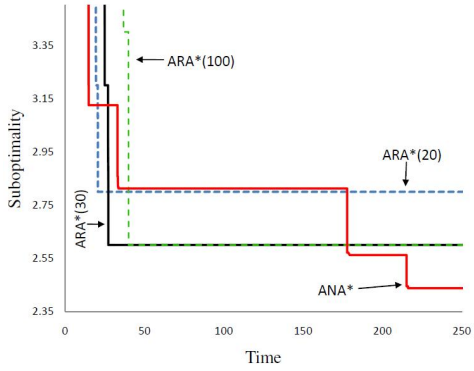


(d) 6DOF, Non-uniform, Suboptimality

Figure 4: Experimental results of the robot motion planning problem; illustrating performance over time for 6 DOF robotic arm



(a) 20DOF, Uniform



(b) 20DOF, Uniform, Suboptimality

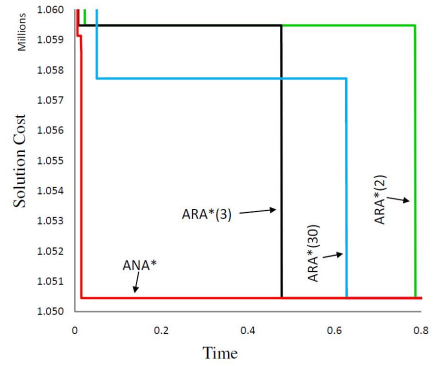
Figure 5: Experimental results of the robot motion planning problem; illustrating performance over time for 20 DOF robotic arm

2004).

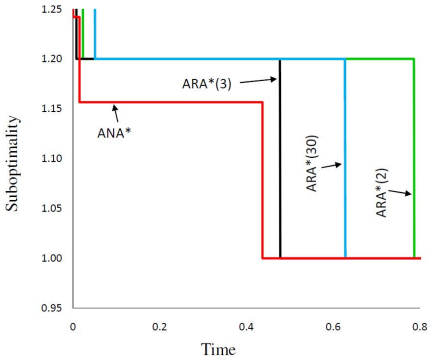
Figs. 4(a) and Fig. 4(b) illustrate the cost and suboptimality, respectively, of the current-best solution over time for ARA* with various parameter values and for ANA* in the 6-DOF arm domain with uniform cost. We consider the suboptimality and current-best cost to be infinite until an initial solution is found. The vertical lines in the graphs signify the first solution found for each algorithm, as the suboptimality and best-cost values drop from infinity to that value. In this experiment, the optimal solution was found by ANA* before ARA*. For ANA* this required 5.0 seconds, whereas ARA* with $\varepsilon_0 = 1.4$ required 36.8 seconds. ANA* finds an initial solution of suboptimality 2.9 in 0.016 seconds, and its rapid convergence to ε=1 and consistent decrease in suboptimality are illustrated by the graph and represent the anytime nature of ANA*.

Figs. 4(c) illustrates the the solution cost of the current-best solution over time for both algorithms in the 6-DOF arm domain with *non-uniform* cost. ANA* finds an initial solution of cost 386 in 0.078 seconds, while ARA* with the best ε_0 takes 0.090 seconds to find an initial solution with the same cost. Over time, ANA* consistently maintains a solution of lower cost in comparison to ARA*.

Fig. 5(b) illustrates the suboptimality of the current-best solution over time for each planner in the *20-DOF* arm do-

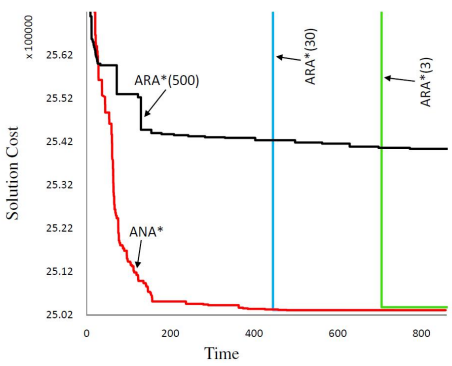


(a) Solution Cost

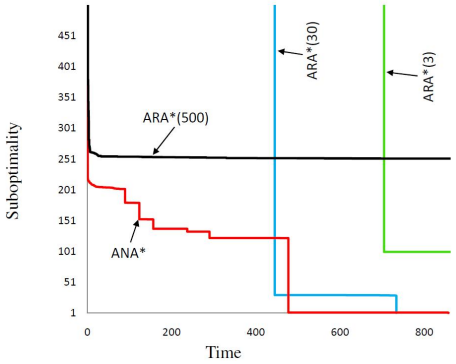


(b) Suboptimality

Figure 6: Experimental results of the 100x1200 grid-world path-planning experiment with obstacles and uniform cost.



(a) Without obstacles



(b) Without obstacles, Suboptimality

Figure 7: Experimental results of the 5000x5000 grid-world path-planning experiments with random cost.

main where each action has *uniform* cost. Again, ANA* finds an initial solution (14.8 seconds) faster than ARA*. While ARA* achieves a better solution for a period of time, ANA* finds a series of solutions lower than those achieved by ARA* for any ε_0 value within 250 seconds. Due to the expansiveness of the search space in this experiment, neither algorithm can prove optimality of the best solution it finds before it runs out of memory. The graph shows that ANA* has the most number of steps in its decrease to a lower sub-optimality. This indicates a larger number of improvements to the current-best solution cost and illustrates the maximal greediness of ANA* to improve on the current-best solution.

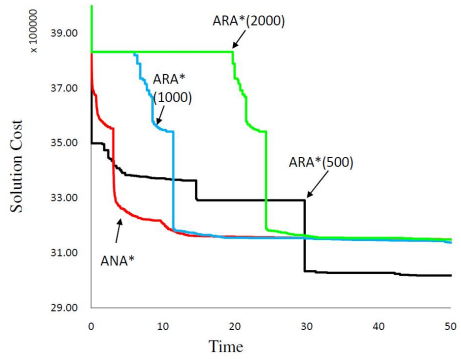
5.2 Gridworld Planning

We consider two planar Gridworld path-planning problems with different sizes and number of obstacles from the SBPL library mentioned above. In both problems, we set the start state as the cell at the top left corner, and the goal state at the bottom right cell (Thayer and Ruml 2008). The first grid-world problem is a 100x1200 8-connected grid with obstacles, with unit uniform cost to move between adjacent obstacle-free cells. The second grid-world problem is a 5000x5000 4-connected grid in which each transition between adjacent cells is assigned a random cost between 1 and 1,000. We considered two cases for this environment, one with obstacles and one without.

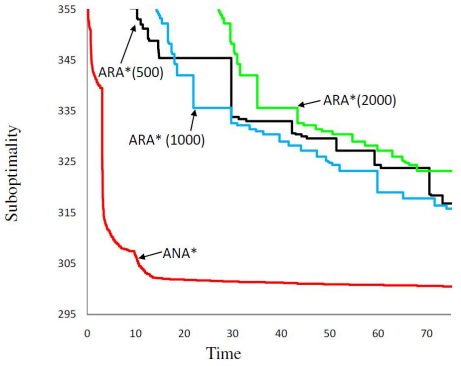
Fig. 6 shows the solution cost results for the 100x1200 gridworld experiment, with fixed transition cost and obstacles in the environment. The disparity in the ARA* results illustrates a non-linear relationship between ε_0 and performance of ARA*, suggesting that setting this value is non-trivial, as a higher or lower ε_0 value does not necessarily guarantee better or worse performance. ARA* was tested with $\varepsilon_0 \in [2,500]$. ANA* finds an initial solution first (in 4 ms), and reaches an optimal solution in the smallest amount of time ($E = 1$ in 15 ms), whereas the next best ARA* takes 478 ms to obtain the same solution.

Figs. 7(a) and 8(a) show results without and with obstacles in the 5000x5000 domain with random transition cost. ARA* was tested with $\varepsilon_0 \in [3,500]$ and $\varepsilon_0 \in [3,2000]$, respectively. The results suggest that the performance of ANA* is superior in all cases of the domain without obstacles.

In the domain with obstacles, ANA* finds an initial solution first (in 50ms), but after 50s, ARA* with $\varepsilon_0 = 500$ has found a solution of lower cost than ANA*. ANA* required an additional 193s to find this solution. This is due to the fact that ANA* improves its current-best solution so often (with small improvements) that the overhead of updating the keys in the *OPEN* queue reduces the effective running time of ANA* in this case. Note, however, that the values for ε_0 produce very different results for ARA* in these domains: in Fig. 7(a), $\varepsilon_0 = 30$ is the best of those tested, while in Fig. 8(a) $\varepsilon_0 = 500$ is best of those tested. ANA* does not

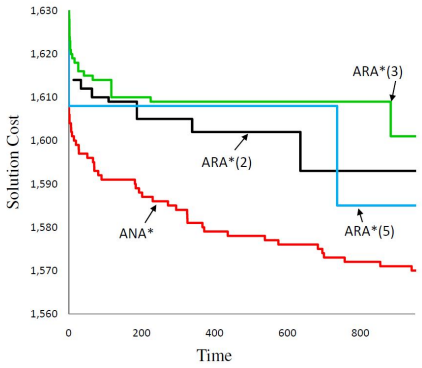


(a) With obstacles

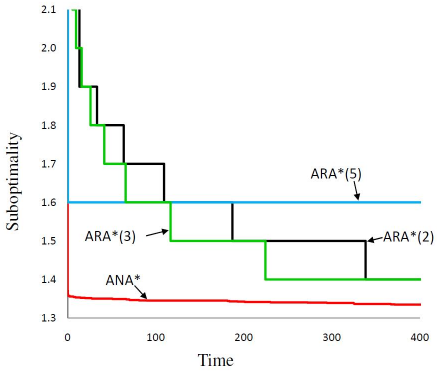


(b) With obstacles, Suboptimality

Figure 8: Experimental results of the 5000x5000 grid-world path-planning experiments with random cost.



(a) Solution cost v. Time



(b) Suboptimlaity v. Time

Figure 9: Results for the multiple sequence alignment problem with five protein sequences.

require setting of parameters.

5.3 Multiple Sequence Alignment

The multiple sequence alignment problem is central for computational molecular biology and can be formalized as a shortest-path problem in an n-dimensional lattice, where n is the number of sequences to be aligned (Yoshizumi, Miura, and Ishida 2000). The state representation is the number of characters consumed so far from each string; a goal is reached when all characters are consumed (Ruml and Do 2007). Moves that consume from only some of the strings represent the insertion of a gap character into the others. We computed alignments of five sequences at a time, using the standard sum-of-pairs cost function in which a gap costs 2, a substitution (mismatched non-gap characters) costs 1, and costs are computed by summing all the pairwise alignments. Each problem consists of five dissimilar protein sequences obtained from (Ikeda and Imai 1994), and (Kobayashi and Imai 1998). The heuristic function *h(n)* was based on optimal pairwise alignments that were precomputed by dynamic programming.

Fig 9(a) compares the performance of ANA* to that of ARA*. ANA* found an initial solution (after 11ms) before ARA* in all cases. ANA* took an average of 18s to improve the current-best solution, while the best run of ARA* averaged 200s, resulting in a better area score for ANA*. Also, ANA* found a new solution 52 times compared to 16 for the best case of ARA*. ARA* was tested with $\varepsilon_0 \in [2, 100]$ (the ideal value of ε_0 for ARA* was small in this case).

6 Further Discussion

In section 5, ANA* was compared to ARA*, with the claim that ARA* possessed the strongest anytime characteristics out of all of the other anytime algorithms we tested. In this section we delve into a discussion of the preliminary tests, and explain our reasoning behind choosing ARA* as the algorithm selected for comparison. These tests were executed on the same machine mentioned earlier, a 32-bit Windows, Intel Pentium Dual Core machine with 1.8 GHz processors and 3GB of RAM.

In the preliminary experiments, each algorithm is run in two Gridworld path-planning domains. The first is a 1000 x 1000 grid with 4-way movement, where each edge has a random cost set before execution, and obstacles are absent. The second is a 10000 x 10000 grid with 4-way movement such that each edge has equal cost and obstacles are present. For each algorithm that required setting of parameters, values for ε_0 and $\Delta\varepsilon$ were selected that gave the best anytime characteristic for that algorithm.

For the first domain, of random edge-cost in the absence of obstacles, Fig 10(a) displays the comparative performance of each algorithm. In the experiment, ANA* finds the optimal solution in 7.8 seconds, ARA* requires 9 sec-

onds, while AHS takes 9.5 seconds, and RWA* finds it in 44.5 seconds. The suboptimality bound for ANA* decreases 162 times before reaching the optimal cost. RWA*, on the other hand, updates the suboptimality bound 11 times, but takes the longest time to reach the optimal solution. ANA* also finds the initial solution the fastest, returning a solution at 19 milliseconds. The randomized edge-cost environment hindered RWA*, and restarting in such an environment proved detrimental, since it forced the algorithm to repeatedly explore the same states over multiple iterations.

In the second domain, of uniform cost, ANA* reaches the optimal solution in 227 milliseconds, while ARA* takes 273 milliseconds, RWA* takes 396 milliseconds, and finally AHS takes 520 milliseconds. The suboptimality bound of ANA* is decreased 6 times before reaching the final answer, while ARA* updates the suboptimality value three times and RWA* decreases it twice. Though AHS decreases the bound 7 times, it takes longer to reach the optimal solution compare ANA*. The speed at which the initial solution is found is relatively similar for all algorithms in this experiment, with ANA* taking 110 milliseconds; second only to RWA* at 107 milliseconds. Fig 10(c) shows the comparison of the area under the cost vs. time curve for each algorithm.

From these experiments, we observed that ARA* outperformed the other anytime algorithms based on Weighted A*, making it a strong candidate to use for the final benchmark tests. Also, the conceptual basis for our algorithm was derived from ARA*. AHS and RWA* in contrast, exhibit inconsistent anytime behavior. In some cases, they excel in one criteria, such as the frequency of improvement on the current-best solution, for a certain experiment, and sometimes fairing poorly in the same area, given a different environment. ANA* and ARA* proved to have a more consistent performance with respect to our metrics of comparison.

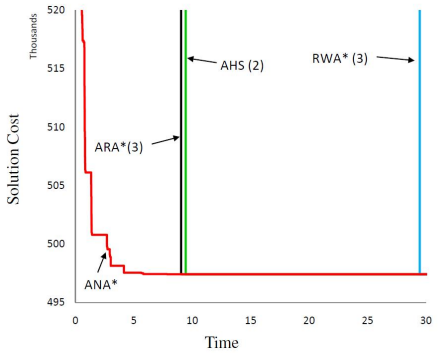
7 Conclusion

We present ANA*, a new anytime A* algorithm that requires no parameters to be set by a user and is easy to implement. Both qualitative analysis and quantitative experimental results suggest that ANA* has superior anytime characteristics compared to existing anytime A* algorithms. ANA* introduces a novel order-criterion to choose the state to expand, which was derived through analyzing the existing anytime algorithm ARA* (Likhachev, Gordon, and Thrun 2004).

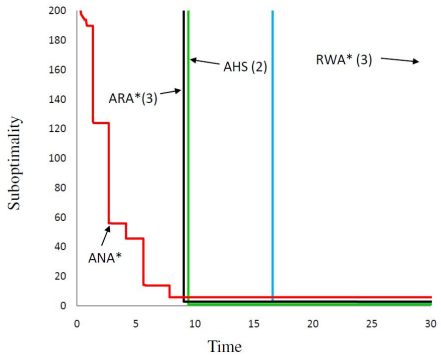
Subjects of ongoing research include graph search in *dynamic domains*, where the cost of transitions between states changes over time and solutions need to be updated quickly. In (Likhachev et al. 2008), an anytime algorithm for dynamic domains was presented that is based on ARA*. We are currently exploring how ANA* can be adapted for application in dynamic domains, such that it offers similar advantages: ANA* is as easy to implement as ARA*, has comparable and in many cases superior performance, and frees users from the burden of setting parameters.

8 Acknowledgements

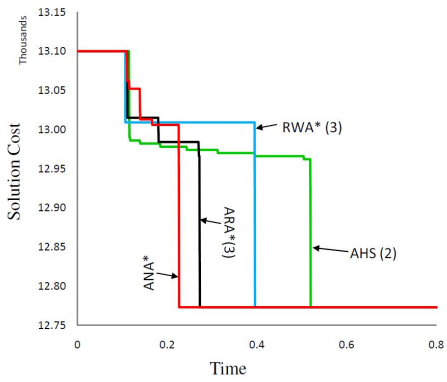
We thank Maxim Likhachev for making his implementation of ARA* available to us, as well as publishing the code



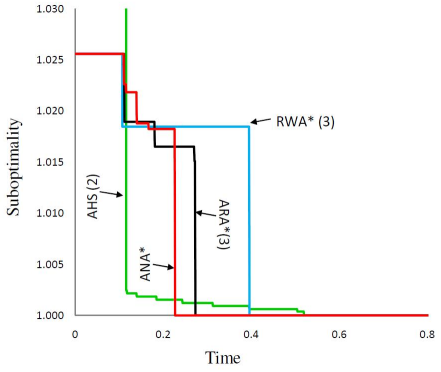
(a) Random cost grid without obstacles



(b) Random cost grid without obstacles, Suboptimality



(c) Uniform cost grid with obstacles



(d) Uniform cost grid with obstacles, Suboptimality

Figure 10: Experimental results of preliminary testing: Comparing ANA* with ARA*, RWA* and AHS search algorithms in the Gridworld environment

for ANA*. Our implementation of ANA* is freely available in his Search-based Planning Library (SBPL) at `http://www.cs.cmu.edu/~maxim/software.html`. We also acknowledge the support of NIH Research Award 1R01EB-006435-01A1, and the Center for Information Technology Research in the Interest of Society (CITRIS), UC Berkeley.

References

Aine, S.; Chakrabarti, P.; and Kumar, R. 2007. AWA* – a window constrained anytime heuristic search algorithm. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2250–2255.

Cormen, T.; Leiserson, C.; and Rivest, R. 1990. *Introduction to Algorithms*. MIT press.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery* 32(3):505–536.

Dijkstra, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Furcy, D. 2006. ITSA*: iterative tunneling search with A*. In *Proc. AAAI Workshop on Heuristic Search, Memory-Based Heuristics and their Applications*, 21–26.

Hansen, E., and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research* 28:267–297.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics* 4(2):100–107.

Ikeda, T., and Imai, H. 1994. Fast A* algorithms for multiple sequence alignment. In *Proc. Workshop on Genome Informatics*, 90–99.

Kobayashi, H., and Imai, H. 1998. Improvement of the A* algorithm for multiple sequence alignment. In *Genome Informatics Series*, 120–130.

Likhachev, M.; Ferguson, D.; Gordon, G.; Stentz, A.; and Thrun, S. 2008. Anytime search in dynamic graphs. *Artificial Intelligence* 172(14):1613–1643.

Likhachev, M.; Gordon, G.; and Thrun, S. 2004. ARA*: anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems* 16.

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

Richter, S.; Thayer, J.; and Ruml, W. 2010. The joy of forgetting: faster anytime search via restarting. In *Proc. Int. Conf. on Automated Planning and Scheduling*, 137–144.

Ruml, W., and Do, M. 2007. Best-first utility-guided search. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2378–2384.

Thayer, J., and Ruml, W. 2008. Faster than weighted A*: an optimistic approach to bounded suboptimal search. In *Proc. Int. Conf. on Automated Planning and Scheduling*, 355–362.

Thayer, J., and Ruml, W. 2010. Anytime heuristic search: frameworks and algorithms. In *Proc. Symp. on Combinatorial Search*.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with partial expansion for large branching factor problems. In *Proc. Nat. Conf. on Artificial Intelligence*, 923–929.

Zhou, R., and Hansen, E. 2005. Beam-stack search: integrating backtracking with beam search. In *Proc. Int. Conf. on Automated Planning and Scheduling*, 90–98.