ARA*: Formal Analysis

Maxim Likhachev, Geoff Gordon, Sebastian Thrun

July 2003

CMU-CS-03-148

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In real world problems, time for deliberation is often limited. Anytime algorithms are beneficial in these conditions as they usually find a first, possibly highly suboptimal, solution very fast and then continually work on improving the solution until allocated time expires. While anytime algorithms are popular, existing anytime search methods are unable to provide a measure of goodness of their results. In this paper we propose the ARA* algorithm. ARA* is an anytime heuristic search which tunes its performance bound based on available search time. It starts by finding a suboptimal solution quickly using a loose bound, then tightens the bound progressively as time allows. Given enough time it finds a provably optimal solution. In addition to the theoretical analysis we demonstrate the practical utility of ARA* with experiments on a simulated robot kinematic arm and dynamic path planning problem for an outdoor rover.

Keywords: search, anytime search, anytime heuristic search, weighted heuristics, anytime planning

1 Introduction

Optimal search is often infeasible for real world problems as we are given a limited amount of time for deliberation and the best solution given the time provided should be found [13]. In these conditions anytime algorithms [4, 8, 16] prove to be useful as they usually find a first, possibly highly suboptimal, solution very fast and then continually work on improving the solution until allocated time expires. Unfortunately, they can rarely provide bounds on the sub-optimality of their solutions. In fact, it is hard to do so as the cost of the optimal solution is often unknown until the optimal solution itself is found. Providing the sub-optimality bounds, on the other hand, is valuable as it allows one to judge the quality of the algorithm, to intelligently evaluate the quality of the past planning episodes and allocate time for future planning episodes accordingly, and finally to decide whether to continue or preempt search based on the current sub-optimality.

A* search with inflated heuristics (actual heuristic values are multiplied by an inflation factor $\epsilon > 1$) is sub-optimal but proves to be fast for many domains [2, 10, 15, 5, 1, 3] and also provides a bound on the sub-optimality, namely, the ϵ by which the heuristic is inflated [12]. To construct an anytime algorithm with sub-optimality bounds one could run a succession of these A* searches with decreasing inflation factors. This naive approach results in a series of solutions, each one with a sub-optimality factor equal to the corresponding inflation factor. The approach, however, wastes a lot of computation since each search iteration duplicates most of the efforts of the previous searches. One could also try to employ incremental heuristic searches (e.g., [9, 14]), but the sub-optimality bounds for each search iteration would no longer be guaranteed, not to mention that they only support admissible (underestimating) heuristics whereas inflated heuristics are usually inadmissible (overestimating).

To this end we propose the ARA* (Anytime Repairing A*) algorithm, which is an *efficient* anytime heuristic search that also runs A* with inflated heuristics in succession but reuses search efforts from previous executions in such a way that the sub-optimality bounds are still satisfied. As a result, a substantial speedup is achieved by not re-computing the state values that have been correctly computed in the previous iterations.

The only other anytime heuristic search known to us is described in [7]. It also first executes an A* with inflated heuristics and then continues to improve a solution. The only sub-optimality bound that it can guarantee, however, is the inflation factor of the first search, and thus any consequent search efforts do not result in the decrease of the sub-optimality bound. The search efforts of ARA*, in contrast, result in both the continuous decrease of the sub-optimality bound and a new solution that satisfies the bound.

In this paper we present both theoretical and empirical analysis of ARA*. For theoretical analysis we present several theorems that prove the sub-optimality bounds of the paths ARA* produces and the convergence of the sequence of these paths to an optimal path, and state formally a reason for the efficiency of ARA*. A full theoretical treatment of ARA* can be found in the appendix (a short version of this paper with no proofs can be found in [11].) The empirical analysis of ARA* is done on two different domains. An evaluation of ARA* on a simulated robot kinematic arm with six degrees of freedom shows up to 6-fold speedup over the succession of A* searches. We also demonstrate how ARA* enables us to successfully solve the problem of efficient path-planning for mobile robots that takes into account the dynamics of the robot.

2 The ARA* Algorithm

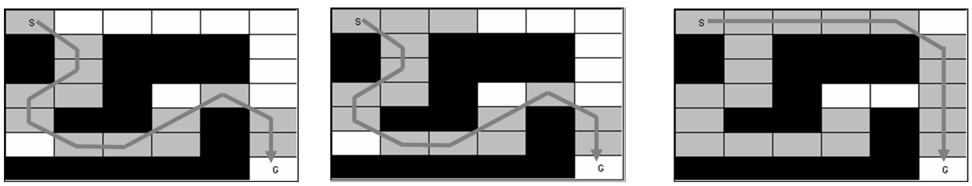
2.1 A* with Weighted Heuristic

Normally, A* takes as input a heuristic $h(s)$ which must be consistent. That is, $h(s) \leq c(s, s') + h(s')$ for any successor s' of s if $s \neq s_{goal}$ and $h(s) = 0$ if $s = s_{goal}$. Here $c(s, s')$ denotes the cost of an edge from s to s' and has to be positive. Consistency, in its turn, guarantees that the heuristic is admissible: $h(s)$ is never larger than the true cost of reaching the goal from s. Inflating the heuristic (that is, using $\epsilon * h(s)$ for $\epsilon > 1$) often results in much fewer state expansions and consequently faster searches. However, inflating the heuristic may also violate the admissibility property, and as a result, a solution is no longer guaranteed to be optimal. The pseudocode of A* with inflated heuristic is given in Figure 1 for easy comparison with our algorithm, ARA*, presented later.

A* maintains two functions from states to real numbers: $g(s)$ is the cost of the currently found path from the start node to s (it is assumed to be ∞ if no path to s has been found yet), and $f(s) = g(s) + \epsilon * h(s)$ is an estimate of the total distance from start to goal going through s. A* also maintains a priority queue, *OPEN*, of states which it plans to expand. The *OPEN* queue

01 $g(s_{start}) = 0$; $OPEN = \emptyset$;
02 insert s_{start} into $OPEN$ with $f(s_{start}) = \epsilon * h(s_{start})$;
03 while(s_{goal} is not expanded)
04   remove s with the smallest f-value from $OPEN$;
05   for each successor s' of s
06   if s' was not visited before then
07       $f(s') = g(s') = \infty$;
08   if $g(s') > g(s) + c(s, s')$
09       $g(s') = g(s) + c(s, s')$;
10       $f(s') = g(s') + \epsilon * h(s')$;
11   insert s' into $OPEN$ with $f(s')$;

Figure 1: A* with heuristic weighted by $\epsilon \geq 1$



$\epsilon = 2.5$ $\epsilon = 1.5$ $\epsilon = 1.0$ (optimal search)

Figure 2: A* search with weighted heuristic

is sorted by $f(s)$, so that A* always expands next the state which appears to be on the shortest path from start to goal. A* initializes the $OPEN$ list with the start state, s_{start} (line 02). Each time it expands a state s (lines 04-11), it removes s from $OPEN$. It then updates the g-values of all of s's neighbors; if it decreases $g(s')$, it inserts s' into $OPEN$. A* terminates as soon as the goal state is expanded.

Clearly, setting ϵ to 1 makes it a normal operation of A*, and the solution that it finds is guaranteed to be optimal. For $\epsilon > 1$ a solution can be sub-optimal, but the sub-optimality is bounded by a factor of ϵ: the length of the found solution is no larger than ϵ times the length of the optimal solution [12].

The example in Figure 2 shows the operation of A* algorithm with a heuristic inflated by $\epsilon = 2.5$, $\epsilon = 1.5$ and a normal heuristic ($\epsilon = 1$) on a simple grid world. In the example we use an eight-connected grid with black cells being obstacles. S denotes a start state, while G denotes a goal state. The cost of moving from one cell to its neighbor is one. The heuristic is the larger of the x and y distances from the cell to the goal. The cells which were expanded are shown in grey. (A* can stop search as soon as it is about to expand a goal state without actually expanding it. Thus, the goal state is not shown in grey.) Paths found by searches are shown with the grey polyline. A* searches with the inflated heuristics expand substantially fewer cells than A* with the normal heuristic, but their solution is sub-optimal.

2.2 ARA*: Reuse of Search Results

ARA* works by executing A* multiple times, starting with a large ϵ and decreasing ϵ prior to each execution until $\epsilon = 1$. As a result, after each search a solution is guaranteed to be within a factor ϵ of optimal. Running A* search from scratch every time we decrease ϵ, however, would be very expensive. We will now explain how ARA* reuses the results of the previous searches to save computation. The pseudocode of ARA* is given in Figures 3 and 5. We first explain the ImprovePath function (Figure 3) that recomputes a path for a given ϵ. In the next section we explain the Main function of ARA* (Figure 5) that repetitively calls the ImprovePath function with a series of decreasing ϵs.

Let us first introduce a notion of *local inconsistency* (we borrow this term from [9]). A state is called locally inconsistent every time its g-value is decreased (line 09, Figure 1) and until the next time the state is expanded. That is, suppose that state s is the best predecessor for some state s': that is, $g(s') = \min_{s'' \in pred(s')}(g(s'') + c(s'', s')) = g(s) + c(s, s')$. Then, if $g(s)$ decreases we get $g(s') > \min_{s'' \in pred(s')}(g(s'') + c(s'', s'))$. In other words, the decrease in $g(s)$ introduces a local inconsistency between the g-value of s and the g-values of its successors. Whenever s is expanded, on the other hand, the inconsistency of s is corrected by re-evaluating the g-values of the successors of s (line 08-09, Figure 1). This in turn makes the successors of s locally inconsistent. In this way the local inconsistency is propagated to the children of s via a series of expansions until they no longer rely on s, in which case none of their g-values are lowered, as a result none of them are inserted into *OPEN* list, and therefore the series of expansions that started with s stops. Given this definition of local inconsistency it is clear that *OPEN* list consists of exactly all locally inconsistent states as every time a g-value is lowered the state is inserted into *OPEN*, and every time a state is expanded it is removed from *OPEN* until the next time its g-value is lowered. Thus, *OPEN* list can be viewed as a set of states with which the propagation of local inconsistency should proceed.

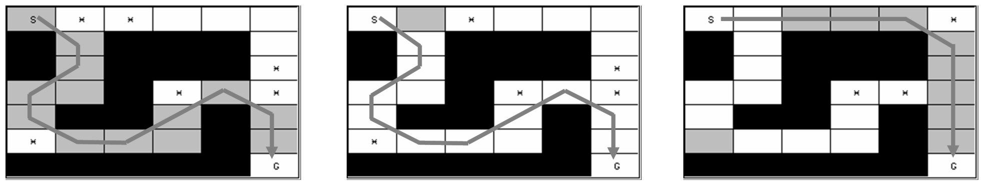
A* with a consistent heuristic is guaranteed not to expand any state more than once. Setting $\epsilon > 1$, however, may violate consistency, and as a result A* search may re-expand states multiple times. It turns out that if we restrict each state to be expanded no more than once, then the sub-optimality bound of ϵ of a solution still holds. To implement this restriction we check any state whose g-value is lowered and insert it into *OPEN* only if it has not been previously expanded (line 10, Figure 3). The set of expanded states is maintained in the *CLOSED* variable.

```
procedure fvalue(s)
01 return g(s) + ε * h(s);
procedure ImprovePath()
02 while(fvalue(s_goal) > min_{s∈OPEN}(fvalue(s)))
03   remove s with the smallest fvalue(s) from OPEN;
04   CLOSED←CLOSED ∪ {s};
05   for each successor s' of s
06     if s' was not visited before then
07       g(s') = ∞;
08     if g(s') > g(s) + c(s, s')
09       g(s') = g(s) + c(s, s');
10       if s' ∉ CLOSED
11         insert s' into OPEN with fvalue(s');
12       else
13         insert s' into INCONS;
```

Figure 3: ImprovePath function of ARA*

With this restriction we will expand each state at most once, but *OPEN* may no longer contain all the locally inconsistent states. In fact, it will only contain the locally inconsistent states that have not yet been expanded. It is important, however, to keep track of *all* the locally inconsistent states as they will be the starting points for inconsistency propagation in the future search iterations. We do this by maintaining the set *INCONS* of all the locally inconsistent states that are not in *OPEN* (lines 12-13, Figure 3). Thus, the union of *INCONS* and *OPEN* lists is then again exactly the set of all locally inconsistent states, and can be used as a starting point for inconsistency propagation before each new search iteration.

The only other difference between the ImprovePath function and A* is the termination condition. Since the ImprovePath function reuses search efforts from the previous executions s_{goal} may never become locally inconsistent and, thus may never be inserted into *OPEN*. As a result, the termination condition



initial search ($\epsilon = 2.5$) second search ($\epsilon = 1.5$) third search ($\epsilon = 1.0$)

Figure 4: ARA* search

of A* becomes invalid. A* search, however, can also stop as soon as $f(s_{goal})$ is equal to the minimal f-value among all the states on *OPEN* list. This condition is the condition that we use in the ImprovePath function (line 02, Figure 3). The new termination condition allows us also to avoid expanding s_{goal} as well as possibly some other states with the same f-value. (Note that ARA* no longer maintains f-values as variables since in between the calls to the ImprovePath function ϵ is changed, and it would be prohibitively expensive to update the f-values of all the states. Instead, fvalue(s) function is called to compute and return the f-values only for the states in *OPEN* and s_{goal}.)

2.3 ARA*: Iterative Execution of Searches

We now introduce the main function of ARA* (Figure 5) that performs a series of search iterations. It does initialization and then repetitively calls the ImprovePath function with a series of decreasing ϵs. The ImprovePath function is equivalent to a single call of A* with a heuristic weighted by ϵ except that the ImprovePath function restricts each state to at most one expansion and maintains all the inconsistent states in the union of *OPEN* and *INCONS* lists as we have just described. Before each call to the ImprovePath function a new *OPEN* list is constructed by moving into it the contents of the set *INCONS*. Since *OPEN* list has to be sorted by the current f-values of states it is also re-ordered (lines 08-09, Figure 5).

Thus, after each call to the ImprovePath function we get a solution that is sub-optimal by at most factor of ϵ. Within each execution of the ImprovePath function each state is expanded at most once, and we mainly save computations by not re-expanding the states which were locally consistent and whose g-values were already correct before a call to the ImprovePath function (Theorem 2 states this more precisely). For example, Figure 4 shows a series of

$g(s_{goal}) = \infty$; $g(s_{start}) = 0$;
$OPEN = CLOSED = INCONS = \emptyset$;
insert s_{start} into $OPEN$ with fvalue(s_{start});
ImprovePath();
publish current ϵ-suboptimal solution;
while $\epsilon > 1$
  decrease ϵ;
  Move states from $INCONS$ into $OPEN$;
  Update the priorities for all $s \in OPEN$ according to fvalue(s);
  $CLOSED = \emptyset$;
  ImprovePath();
  publish current ϵ-suboptimal solution;

Figure 5: Main function of ARA*

calls to the ImprovePath function. States that are locally inconsistent at the end of an iteration are shown with an asterisk. While the first call ($\epsilon = 2.5$) is identical to the A* call with the same ϵ (Figure 2), the second call to the ImprovePath function ($\epsilon = 1.5$) expands only 1 cell. This is in contrast to 15 cells expanded by A* search with the same ϵ. For both searches the sub-optimality factor decreases from 2.5 to 1.5. Finally, the third call to the ImprovePath function with ϵ set to 1 expands only 9 cells. The solution is now optimal, and the total number of expansion is 23. Only 2 cells are expanded more than once across all three calls to the ImprovePath function. Even a single optimal search from scratch expands not much fewer cells: 20 cells (Figure 2, $\epsilon = 1$).

2.4 Theoretical Properties of the Algorithm

In this section we present some of the theoretical properties of ARA*. For the proofs of these and other properties of the algorithm please refer to the appendix. In the theorems we use $g^*(s)$ to denote the cost of an optimal path from s_{start} to s. Let us also define a *greedy path* from s_{start} to s as a path that is computed by tracing it backward as follows: start at s, and at any state s_i pick a state $s_{i-1} = \arg\min_{s' \in pred(s_i)}(g(s') + c(s', s_i))$ until $s_{i-1} = s_{start}$. The first theorem then says that, for any state s with an f-value smaller than or equal to the minimum f-value in $OPEN$, we have computed a greedy path from s_{start} to s which is within a factor of ϵ of optimal. This theorem is equivalent to the combination of Theorems 9 and 11 in the appendix.

Theorem 1 *Whenever the ImprovePath function exits, for any state s with* $f(s) \leq \min_{s' \in \text{OPEN}}(f(s'))$*, we have* $g^*(s) \leq g(s) \leq \epsilon * g^*(s)$*, and the cost of a greedy path from* s_{start} *to s is no larger than* $g(s)$*.*

This theorem establishes the correctness of ARA*: each execution of the ImprovePath function terminates when $f(s_{goal})$ is no larger than the minimum f-value in *OPEN*, which means we have found a path from start to goal which is within a factor ϵ of optimal. Since before each iteration ϵ is decreased, ARA* gradually decreases the sub-optimality bound and finds new solutions to satisfy the bound.

The next theorem formalizes where the computational savings for ARA* search come from. According to it, unlike A* search with an inflated heuristic each search iteration in ARA* is guaranteed not to expand states more than once. Moreover, only states whose g-values can be lowered or which are locally inconsistent are expanded. Thus, if before a call to the ImprovePath function a g-value of a state has already been correctly computed by some previous search, then this state is guaranteed not to be expanded unless it is in the set of locally inconsistent states already and thus needs to update its neighbors (propagate local inconsistency). The following theorem is equivalent to the combination of Theorems 14 and 16 in the appendix.

Theorem 2 *Within each call to ImprovePath() a state is expanded at most once and only if it was locally inconsistent before the call to ImprovePath() or its g-value was lowered during the current execution of ImprovePath().*

3 Experimental Study

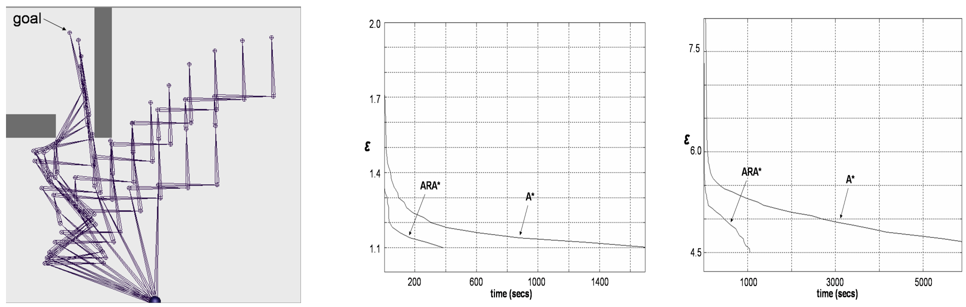
3.1 Robotic Arm

We first evaluate the performance of ARA* on a simulated 6 degree of freedom robotic arm (Figure 6). The base of the arm is fixed, and the task is to move its end-effector into the goal location. The initial configuration of the arm is the rightmost configuration. The grey rectangles are obstacles that the arm should go around. 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.) We discretitize the workspace into 50 by 50 cells and compute a distance from each cell to the cell containing the end-effector goal position taking into account that some cells are occupied by obstacles. This distance is our heuristic. In order for the heuristic not to overestimate true costs, joint angles are discretitized so as to never move the end-effector by more than one cell in a single action. The resulting state-space is over 3 billion states, and memory for states is allocated on demand.

In Figure 6, the leftmost figure shows the planned trajectory of the robot arm after the initial search of ARA* with $\epsilon = 3.0$. The time to plan this trajectory is about 0.04 secs. (By comparison, a search for an optimal trajectory is infeasible as it runs out of memory very quickly.) The plot in the middle shows that for a succession of A* searches it takes more than 4.5 times longer to reach $\epsilon = 1.1$ than for ARA*. In both cases ϵ is initially 3.0 and decreases in steps of 0.02 (2% sub-optimality). In the experiment for the middle plot all the actions have the same cost. In the experiment for the rightmost plot actions have non-uniform costs: changing a joint angle closer to the base is more expensive than changing a higher joint angle. As a result of the non-uniform costs our heuristic becomes less informative, and so search is much more expensive. In this experiment we decreased ϵ from 10 to 4.5 for ARA* (the succession of A* searches could only achieve $\epsilon = 4.65$). For ARA* it takes about 15 mins and 6 million state expansions to reach $\epsilon = 4.65$, while for the succession of A* searches it takes about 1.66 hours and over 40 million state expansions to reach the same ϵ (over 6-fold speedup by ARA*). Put another way, after 10 minutes ARA* reaches a bound of 4.95 while A* achieves only 5.45. While Figure 6 shows execution time, the comparison of state expanded (not shown) is almost identical. Finally, to evaluate the expense of the anytime property of ARA* we ran ARA* and an optimal A* search on an environment slightly simpler than the one in Figure 6 (for the optimal search to be feasible). Optimal A* search required about 5.8 mins (2,202,666 state expanded) to find an optimal solution, while ARA* required about 6.0 mins (2,207,178 state expanded) to decrease ϵ from 3.0 to 1.0 in steps of 0.2 and also guarantee the solution optimality (3% overhead).

3.2 Outdoor Robot Navigation

For us the motivation for this work was efficient path-planning for mobile robots in large outdoor environments, where optimal trajectories involve fast motion and sweeping turns at speed and as a result, it is particularly important to take advantage of the robot's momentum and find dynamic rather than static plans. We use a 4D state space: xy position, orientation, and velocity. High dimensionality combined with large environments results in very



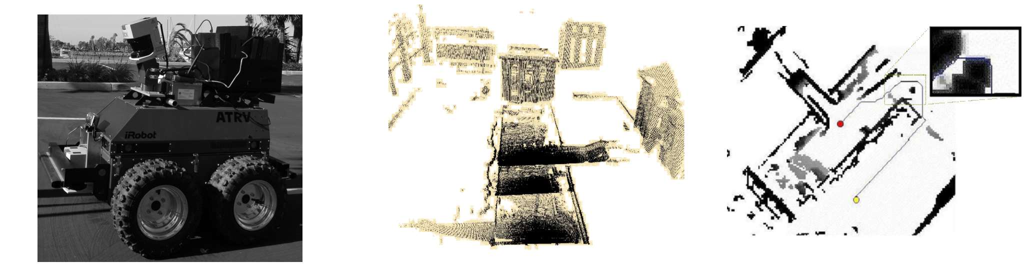
arm trajectory for $\epsilon = 3.0$ uniform costs non-uniform costs

Figure 6: robot arm experiment (the trajectory shown is downsampled by 6)

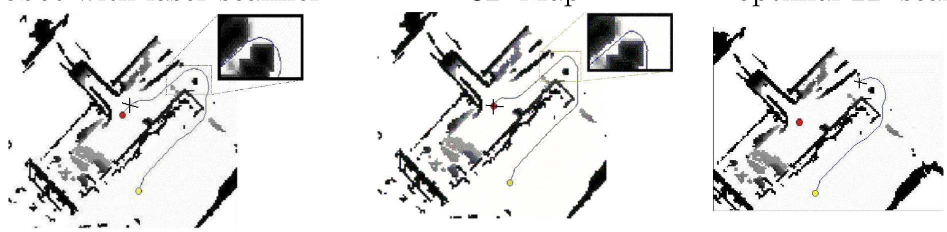
large state-spaces for the planner and makes it computationally infeasible for the robot to move and plan optimally every time it discovers new obstacles or modelling errors. To solve this problem we built a two-level planner: a 4D planner that uses ARA*, and a fast 2D (x, y) planner that uses A* search and whose results serve as the heuristics for the 4D planner. [1]

In Figure 7 we show the robot we used for navigation and a 3D laser scan [6] constructed by the robot of the environment we tested our system in. The scan is converted into a map of the environment (upper right figure). In black are shown what are believed to be obstacles by the robot. The size of the environment is 91.2 by 94.4 meters, and the map is discretitized into cells of 0.4 by 0.4 meters. Thus, the 2D state-space consists of 53808 states. The 4D state space has over 20 million states. The robot initial state is the upper circle, while its goal is the lower circle. To ensure safe operation we created a buffer zone with high costs around each obstacle. The squares in the upper-right corners of the figures show a magnified fragment of the map with grayscale proportional to cost. The 2D plan (upper right figure) makes sharp 45 degree turns when going around the obstacles, requiring the robot to come to complete stops. The optimal 4D plan results in a wider turn, and the velocity of the robot remains high throughout the whole trajectory. In the first plan computed by ARA* starting at $\epsilon = 2.5$ (lower middle figure)

[1]To interleave search with the execution of the best plan so far we perform 4D search backward. That is, the start of the search, s_{start}, is the actual goal state of the robot, while the goal of the search, s_{goal}, is the current state of the robot. Thus, s_{start} does not change as the robot moves and the search tree remains valid in between search iterations. Since heuristics estimate the distances to s_{goal} (the robot position) we have to recompute them during the reorder operation (line 09, Figure 5).



robot with laser scanner | 3D Map | optimal 2D search



optimal 4D search with A* after 25 secs | 4D search with ARA* after 0.6 secs ($\epsilon = 2.5$) | 4D search with ARA* after 25 secs ($\epsilon = 1.0$)

Figure 7: outdoor robot navigation experiment (cross shows the position of the robot)

the trajectory is much better than the 2D plan, but somewhat worse than the optimal 4D plan.

The time required for the optimal 4D planner was 11.196 secs, whereas the time for the 4D planner that runs ARA* to generate the shown plan was 556 msecs. As a result, the robot that runs ARA* can start executing a plan much earlier. Thus, the robot running optimal 4D planner is still near the beginning of its path to the goal after 25 seconds from the time it receives a goal location (the position of the robot is shown by cross). In contrast, in the same amount of time the robot running ARA* has advanced much further (lower right figure), and its plan by now has converged to optimal (ϵ was decreased to 1) and thus is no different from the one computed by the optimal 4D planner.

4 Conclusions

We have presented the first anytime heuristic search that works by continually decreasing a sub-optimality bound on its solution and finding new solutions that satisfy the bound on the way. It executes a series of searches with decreasing sub-optimality bound, where each search tries to reuse as much as possible of the results from the previous searches. The experiments show that ARA* is much more efficient than the best previous anytime search with provable performance bounds, namely a series of A* searches with decreasing ϵs, and can successfully be used for large robotic planning problems.

A ARA*: The Proofs

A.1 Pseudocode of ARA*

The pseudocode in Figure 8 is slightly different from the one presented in the main text. In particular, every state s now maintains an additional variable, $v(s)$, which is initially set to ∞, and then is reset to the g-value of s every time s expanded. This modification simplifies the proofs and makes the interpretation of local inconsistency clearer: a state s is called locally inconsistent iff $v(s) \neq g(s)$. Otherwise, the v-values are not used in the algorithm, and therefore it should be clear that the pseudocode in Figure 8 is algorithmically identical to the pseudocode of ARA* as presented in the main text of the paper.

Henceforth, all line numbers in the text of the proofs will refer to the pseudocode in Figure 8.

A.2 Notations

Some of the definitions given in this section are just repetitions of the ones in the main body of the paper, but we still present them here for an easier reference. A state s is called *locally inconsistent* iff $v(s) \neq g(s)$. Heuristics need to be consistent. That is, $h(s) \leq c(s, s') + h(s')$ for any successor s' of s if $s \neq s_{goal}$ and $h(s) = 0$ if $s = s_{goal}$. For any pair of states $s, s' \in succ(s)$ the cost between the two needs to be positive: $c(s, s') > 0$. $c^*(s, s')$ denotes the cost of a shortest path from s to s'. $g^*(s)$ denotes the cost of a shortest path from s_{start} to s. We restrict that $1 \leq \epsilon < \infty$.

Let us define $g_\epsilon(s) = \min_{s' \in pred(s)}(v(s') + \epsilon * c(s', s))$ if $s \neq s_{start}$ and $g_\epsilon(s) = 0$ otherwise (for $\epsilon = 1$ it can be shown that $g_\epsilon(s)$ is always equal to $g(s)$.) A state s is called ϵ *locally inconsistent* iff $v(s) > g_\epsilon(s)$. $f(s)$ is defined to be always equal to $g(s) + \epsilon * h(s)$. Let us also define a *greedy path* from s_{start} to s as a path that is computed by tracing it backward as follows: start at s, and at any state s_i pick a state $s_{i-1} = \arg\min_{s' \in pred(s_i)}(g(s') + c(s', s_i))$ until $s_{i-1} = s_{start}$.

Finally, any state s with undefined values (not visited) is assumed to have $v(s) = g(s) = \infty$. We also assume that $\min_{s \in OPEN}(\text{fvalue}(s)) = \infty$ if $OPEN = \emptyset$.

.

```
procedure fvalue(s)
return g(s) + ε * h(s);
procedure ImprovePath()
while(fvalue(s_goal) > min_{s∈OPEN}(fvalue(s)))
  remove s with the smallest fvalue(s) from OPEN;
  v(s) = g(s); CLOSED←CLOSED ∪ {s};
  for each successor s' of s
    if s' was not visited before then
      v(s') = g(s') = ∞;
    if g(s') > g(s) + c(s, s')
      g(s') = g(s) + c(s, s');
      if s' ∉ CLOSED
        insert s' into OPEN with fvalue(s');
      else
        insert s' into INCONS;
procedure Main()
g(s_goal) = v(s_goal) = ∞; v(s_start) = ∞;
g(s_start) = 0; OPEN = CLOSED = INCONS = ∅;
insert s_start into OPEN with fvalue(s_start);
ImprovePath();
publish current ε-suboptimal solution;
while ε > 1
  decrease ε;
  Move states from INCONS into OPEN;
  Update the priorities for all s ∈ OPEN according to fvalue(s);
  CLOSED = ∅;
  ImprovePath();
  publish current ε-suboptimal solution;
```

Figure 8: ARA*

A.3 Proofs

First, in the section A.3.1 we prove several lemmas and theorems about some of the more obvious properties of the main search loop (the body of the ImprovePath function). In the following section A.3.2 we prove several theorems that constitute the main idea behind ARA*. Finally, in the section A.3.3 we show how these theorems lead to the correctness of ARA*.

A.3.1 Properties of the main search loop

Most of the theorems in this section simply state the correctness of the program state variables such as heuristic values, g-values and $OPEN$, $INCONS$ and $CLOSED$ sets. The theorems also show that $g(s)$ is always an upper bound on the cost of a greedy path from s_{start} to s, and can never become smaller than the cost of a least-cost path from s_{start} to s, $g^*(s)$.

Lemma 3 *For any pair of states s and s', $\epsilon * h(s) \leq \epsilon * c^*(s, s') + \epsilon * h(s')$.*

Proof: According to [12] the consistency property is equivalent to the restriction that $h(s) \leq c^*(s, s') + h(s')$ for *any* pair of states s, s' and $h(s_{goal}) = 0$. The theorem then follows by multiplying the inequality with ϵ. ■

Lemma 4 *At any point of time for any state s, $v(s) \geq g(s)$.*

Proof: The theorem clearly holds before the ImprovePath function is called for the first time since at that point all the v-values are infinite. Afterwards, g-values can only decrease (line 09). For any state s, on the other hand, $v(s)$ only changes on line 04 when it is set to $g(s)$. Thus, it is always true that $v(s) \geq g(s)$. ■

Theorem 5 *At line 02, $g(s_{start}) = 0$ and for $\forall s \neq s_{start}$, $g(s) = \min_{s' \in pred(s)}(v(s') + c(s', s))$.*

Proof: The theorem holds after the initialization, when $g(s_{start}) = 0$ while the rest of g-values are infinite, and all the v-values are infinite. The only place where g- and v-values are changed afterwards is on lines 04 and 09. If $v(s)$ is changed in line 04, then it is decreased according to Lemma 4. Thus, it may only decrease the g-values of its successors. The test on line 08 checks this and updates the g-values if necessary. Since all costs are positive and never change, $g(s_{start})$ can never be changed: it will never pass the test on line 08, and thus is always 0. ■

Theorem 6 *At line 02, OPEN and INCONS are disjoint. Their union contains all and only locally inconsistent states. Of these states, INCONS contains exactly the ones which are also in CLOSED.*

Proof: The first time line 02 is executed $OPEN = \{s_{start}\}$ which is indeed locally inconsistent as $g(s_{start}) = 0 \neq \infty = v(s_{start})$. Also, $INCONS = CLOSED = \emptyset$, and all states besides s_{start} are locally consistent as they all have infinite v- and g-values.

During the following execution whenever we decrease $g(s)$ (line 09), and as a result make s locally inconsistent (Lemma 4), we insert it into either $OPEN$ or $INCONS$ depending on whether it is in $CLOSED$; whenever we remove s from $OPEN$ (line 03) we set $v(s) = g(s)$ (line 04) making the state locally consistent; whenever we move s from $INCONS$ to $OPEN$ (line 21), we remove s from $CLOSED$ (line 23). We never add s to $CLOSED$ while it is still in $OPEN$, and we never modify $v(s)$ or $g(s)$ elsewhere. ■

Corollary 7 *Before each call to the ImprovePath function OPEN contains all and only inconsistent states.*

Proof: Before each call to the ImprovePath function $CLOSED = INCONS = \emptyset$. Thus, from Theorem 6 it follows that $OPEN$ contains all and only locally inconsistent states. ■

Theorem 8 *Suppose s is selected for expansion on line 03. Then the next time line 02 is executed $v(s) = g(s)$, where $g(s)$ before and after the expansion of s is the same.*

Proof: Suppose s is selected for expansion. Then on line 04 $v(s) = g(s)$, and it is the only place where a v-value changes. We, thus, only need to show that $g(s)$ does not change. It could only change if $s \in succ(s)$ and $g(s) > v(s) + c(s, s)$. The second test, however, implies that $c(s, s) < 0$ since we have just set $v(s) = g(s)$. This contradicts to our restriction that costs are positive. ■

Theorem 9 *At line 02, for any state s, the cost of a greedy path from s_{start} to s is no larger than $g(s)$, and $v(s) \geq g(s) \geq g^*(s)$.*

Proof: $v(s) \geq g(s)$ holds according to Lemma 4. We thus only need to show that the cost of a greedy path from s_{start} to s is no larger than $g(s)$, and $g(s) \geq g^*(s)$. The statement follows if $g(s) = \infty$. We thus assume a finite g-value.

Consider a greedy path from s_{start} to s: $s_0 = s_{start}, s_1, ..., s_k = s$. Then from the definition of such path for any $i > 0$, $g(s_i) = v(s_{i-1}) + c(s_{i-1}, s_i) \geq g(s_{i-1}) + c(s_{i-1}, s_i)$ from Theorem 5 and Lemma 4. For $i = 0$, $g(s_i) = g(s_{start}) = 0$. Thus, $g(s) = g(s_k) \geq g(s_{k-1}) + c(s_{k-1}, s_k) \geq g(s_{k-2}) + c(s_{k-2}, s_{k-1}) + c(s_{k-1}, s_k) \geq ... \geq \sum_{j=1..k} c(s_{j-1}, s_j)$. That is, $g(s)$ is at least as large as the cost of the greedy path from s_{start} to s. Since the cost can not be smaller than the cost of a least-cost path we also have $g(s) \geq g^*(s)$.

A.3.2 Main theorems

We now prove two theorems which constitute our main results about ARA*. These theorems guarantee that ARA* is ϵ sub-optimal: when it finishes its processing for a given ϵ, it has identified a set of states for which its cost estimates $g(s)$ are no more than a factor of ϵ greater than the optimal costs $g^*(s)$. In section A.3.3 we will then prove corollaries which show that given such cost estimates the greedy paths that ARA* finds to these states are sub-optimal by at most ϵ.

If we set our initial ϵ to 1, the ImprovePath function in ARA* is essentially equivalent to the A* algorithm. The only difference is that ARA* assumes that our heuristic is consistent, while A* is defined for any admissible heuristic.[2] For intuition, here is a very brief summary of how our proofs below would apply to A*: we would start by showing that the *OPEN* list always contains all locally inconsistent states. (These states are arranged in a priority queue ordered by their f values.) We say a state s is ahead of the *OPEN* list if $f(s) \leq f(u)$ for all $u \in OPEN$. We then prove by induction that states which are ahead of *OPEN* have already been assigned their correct optimal path length. The induction works because, when we expand the state at the head of the *OPEN* queue, its optimal path depends only on states which are already ahead of the *OPEN* list.

The proofs for ARA* are somewhat more complicated than for A* because the heuristic is inflated and therefore may be inadmissible, and the *OPEN* list may also not contain all locally inconsistent states. (Some of these states may be in *INCONS* because they have already been expanded.) Therefore, we will examine the set Q instead:

$$Q = \{u \mid v(u) > g_\epsilon(u) \wedge v(u) > \epsilon * g^*(u)\} \tag{1}$$

[2]It is actually possible to use ARA* with an inconsistent heuristic, but doing so is beyond the scope of this report.

This set contains all ϵ locally inconsistent states except those whose g-values are already within a factor of ϵ of their true costs.

The set Q takes the place of the *OPEN* list in the next theorem. In particular, Theorem 10 says that all states which are ahead of Q have their g-values within a factor of ϵ of optimal. Theorem 11 builds on this result by showing that *OPEN* is always a superset of Q, and therefore the states which are ahead of *OPEN* are also ahead of Q. (Theorem 10 is actually stronger than required for the proof of Theorem 11, but we prove the strong version because it might be useful for optimizations in the future.)

Theorem 10 *At line 02, let Q be defined according to the definition 1. Then for any state s with ($f(s) \leq f(u)\ \forall u \in Q$), it holds that $g(s) \leq \epsilon * g^*(s)$.*

Proof: We prove by contradiction. Suppose there exists an s such that $f(s) \leq f(u)\ \forall u \in Q$, but $g(s) > \epsilon * g^*(s)$. The latter implies that $g^*(s) < \infty$. We also assume that $s \neq s_{start}$ since otherwise $g(s) = 0 = \epsilon * g^*(s)$ from Theorem 5.

Consider a least-cost path from s_{start} to s, $\pi(s_0 = s_{start}, ..., s_k = s)$. The cost of this path is $g^*(s)$. Such path must exist since $g^*(s) < \infty$. Our assumption that $g(s) > \epsilon * g^*(s)$ means that there exists at least one $s_i \in \pi(s_0, ..., s_{k-1})$ whose $v(s_i) > \epsilon * g^*(s_i)$. Otherwise,

$$\begin{aligned}
g(s) = g(s_k) = \min_{s' \in pred(s)} (v(s') + c(s', s_k)) &\leq \\
v(s_{k-1}) + c(s_{k-1}, s_k) &\leq \\
\epsilon * g^*(s_{k-1}) + c(s_{k-1}, s_k) &\leq \\
\epsilon * (g^*(s_{k-1}) + c(s_{k-1}, s_k)) = \epsilon * g^*(s_k) &= \epsilon * g^*(s)
\end{aligned}$$

Let us now consider $s_i \in \pi(s_0, ..., s_{k-1})$ with the smallest index $i \geq 0$ (that is, the closest to s_{start}) such that $v(s_i) > \epsilon * g^*(s_i)$. We will now show that $s_i \in Q$. If $i = 0$ then $g_\epsilon(s_i) = g_\epsilon(s_{start}) = 0$ according to the definition of the g_ϵ-values. Thus: $v(s_i) > \epsilon * g^*(s_i) = 0 = g_\epsilon(s_i)$, and $s_i \in Q$. If $i > 0$ then

$$\begin{aligned}
v(s_i) > \epsilon * g^*(s_i) &= \\
\epsilon * (g^*(s_{i-1}) + c(s_{i-1}, s_i)) &\geq \\
v(s_{i-1}) + \epsilon * c(s_{i-1}, s_i) &
\end{aligned}$$

since we picked s_i to be the closest state to s_{start} with $v(s_i) > \epsilon * g^*(s_i)$. Thus,

$$\begin{aligned} v(s_i) > v(s_{i-1}) + \epsilon * c(s_{i-1}, s_i) &\geq \\ \min_{s' \in pred(s_i)} (v(s') + \epsilon * c(s', s_i)) &= g_\epsilon(s_i) \end{aligned}$$

As such, it must again be the case that $s_i \in Q$.

We will now also show that $g(s_i) \leq \epsilon * g^*(s_i)$. It is clearly so when $i = 0$ according to Theorem 5. For $i > 0$,

$$\begin{aligned} g(s_i) = \min_{s' \in pred(s_i)} (v(s') + c(s', s_i)) &\leq \\ v(s_{i-1}) + c(s_{i-1}, s_i) &\leq \\ \epsilon * g^*(s_{i-1}) + c(s_{i-1}, s_i) &\leq \\ \epsilon * g^*(s_i) & \end{aligned}$$

We will now show that $f(s) > f(s_i)$, and finally arrive at a contradiction. According to our assumption

$$\begin{aligned} g(s) > \epsilon * g^*(s) &= \\ \epsilon * (c^*(s_0, s_i) + c^*(s_i, s_k)) &= \\ \epsilon * g^*(s_i) + \epsilon * c^*(s_i, s_k) &\geq \\ g(s_i) + \epsilon * c^*(s_i, s) & \end{aligned}$$

Adding $\epsilon * h(s)$ on both sides and using Lemma 3:

$$\begin{aligned} f(s) = g(s) + \epsilon * h(s) &> \\ g(s_i) + \epsilon * c^*(s_i, s) + \epsilon * h(s) &\geq \\ g(s_i) + \epsilon * h(s_i) = f(s_i) & \end{aligned}$$

The inequality $f(s) > f(s_i)$ implies, however, that $s_i \notin Q$ since $f(s) \leq f(u)$ $\forall u \in Q$. But this contradicts what we have proved earlier. ■

Theorem 11 *At line 02, for any state s with ($f(s) \leq f(u)$ $\forall u \in$ OPEN), it holds that $g(s) \leq \epsilon * g^*(s)$.*

Proof: Let Q be defined according to the definition 1. Now consider any state s with $v(s) > g_\epsilon(s)$. It is then also true that $v(s) > g(s)$ since

$g_\epsilon(s) = \min_{s' \in pred(s)}(v(s') + \epsilon * c(s', s)) \geq \min_{s' \in pred(s)}(v(s') + c(s', s)) = g(s)$ for any state $s \neq s_{start}$ and $g_\epsilon(s_{start}) = g(s_{start}) = 0$. Thus, s is also locally inconsistent. Hence, for any state $u \in Q$ it holds that u is locally inconsistent.

According to Corollary 7 every time the ImprovePath function is called *OPEN* contains all locally inconsistent states. Therefore $Q \subseteq OPEN$, because as we have just shown any state $u \in Q$ is also locally inconsistent. Thus, if any state s has $f(s) \leq f(u)\ \forall u \in OPEN$, it is also true that $f(s) \leq f(u)\ \forall u \in Q$, and $g(s) \leq \epsilon * g^*(s)$ from Theorem 10. Thus, within each call to the ImprovePath function, the first time line 02 is executed the theorem holds.

Also, because before each call to ImprovePath(), $CLOSED = \emptyset$, the following statement, denoted by (*), holds every time line 02 is executed for the first time within each call to the ImprovePath function: for any state $s \in CLOSED$ $v(s) \leq \epsilon * g^*(s)$.

We will now show by induction that the theorem continues to hold for the consecutive executions of the line 02 *within* each call to the ImprovePath function. Suppose the theorem and the statement (*) held during all the previous executions of line 02, and they still hold when a state s is selected for expansion on line 03. We need to show that the theorem holds the next time line 02 is executed.

We first prove that the statement (*) still holds during the next execution of line 02. Since the v-value of only s is being changed and only s is being added to *CLOSED*, we only need to show that $v(s) \leq \epsilon * g^*(s)$ during the next execution of line 02 (that is, after the expansion of s). Since when s is selected for expansion on line 03 $f(s) = \min_{u \in OPEN}(f(u))$, we have $f(s) \leq f(u)\ \forall u \in OPEN$. According to the assumptions of our induction then $g(s) \leq \epsilon * g^*(s)$. From Theorem 8 it then also follows that the next time line 02 is executed $v(s) \leq \epsilon * g^*(s)$, and hence the statement (*) still holds.

We now prove that after s is expanded the theorem itself also holds. We prove it by showing that Q continues to be a subset of *OPEN* the next time line 02 is executed. According to Theorem 6 *OPEN* set contains all locally inconsistent states that are not in *CLOSED*. Since, as we have just proved, the statement (*) holds the next time line 02 is executed, all states s in *CLOSED* set have $v(s) \leq \epsilon * g^*(s)$. Thus, any state s that is locally inconsistent and has $v(s) > \epsilon * g^*(s)$ is guaranteed to be in *OPEN*. Now consider any state $u \in Q$. As we have shown earlier any state u is locally inconsistent, and $v(u) > \epsilon * g^*(u)$ according to the definition of Q. Thus, $u \in OPEN$. This shows that $Q \subseteq OPEN$. Consequently, if any state s has

$f(s) \leq f(u)\ \forall u \in OPEN$, it is also true that $f(s) \leq f(u)\ \forall u \in Q$, and $g(s) \leq \epsilon * g^*(s)$ from Theorem 10. This proves that the theorem holds during the next execution of line 02, and proves the whole theorem by induction. ∎

A.3.3 Correctness of ARA*

The corollaries in the this section show how the theorems in previous section lead quite trivially to the correctness of ARA*.

Corollary 12 *Each time the ImprovePath function exits the following holds for any state s with* $f(s) \leq \min_{s' \in OPEN}(f(s'))$*: the cost of a greedy path from* s_{start} *to s is no larger than* $\epsilon * g^*(s)$*.*

Proof: According to Theorem 11 the condition $f(s) \leq \min_{s' \in OPEN}(f(s'))$ implies that $g(s) \leq \epsilon * g^*(s)$. The proof then follows by direct application of Theorem 9. ∎

Corollary 13 *Each time the ImprovePath function exits the following holds: the cost of a greedy path from* s_{start} *to* s_{goal} *is no larger than* $\epsilon * g^*(s_{goal})$*.*

Proof: According to the termination condition of the ImprovePath function, upon its exit $f(s_{goal}) \leq \min_{s' \in OPEN}(f(s'))$. The proof then follows from Corollary 12. ∎

A.3.4 Efficiency of ARA*

Several theorems in this section provide some theoretical guarantees about the efficiency of ARA*.

Theorem 14 *Within each call to ImprovePath() no state is expanded more than once.*

Proof: Suppose a state s is selected for expansion for the first time within a particular execution of the ImprovePath function. Then, it is removed from *OPEN* set on line 03 and inserted into *CLOSED* set on line 04. It can then never be inserted into *OPEN* set again unless the ImprovePath function exits since any state that is about to be inserted into *OPEN* set is checked against

CLOSED set membership on line 10. Because only the states from *OPEN* set are selected for expansion, s can therefore never be expanded second time within the same execution of the ImprovePath function. ■

Theorem 15 *Within each call to ImprovePath() a state s is expanded only if $v(s)$ can be lowered during its expansion.*

Proof: Only the states from *OPEN* can be selected for expansion. Any such state is locally inconsistent according to Theorem 6. Moreover, for any such state s it holds that $v(s) > g(s)$ from Lemma 4. From Theorem 8 it then follows that $v(s)$ is set to $g(s)$ during the expansion of v and thus is lowered. ■

Theorem 16 *Within each call to ImprovePath() a state s is expanded only if it was already locally inconsistent before the call to ImprovePath() or its g-value was lowered during the current execution of ImprovePath().*

Proof: According to Theorem 6 any state s that is selected for expansion on line 03 is locally inconsistent. If a state s was already locally inconsistent before the call to ImprovePath() then the theorem is immediately satisfied. If a state s was *not* locally inconsistent before the call to the ImprovePath function, then its g- and v- values were equal. Since $v(s)$ can only be changed during the expansion of s, it must have been the case that $g(s)$ was changed, and the only way for it to change is to decrease on line 09. Thus, the g-value of s was lowered during the current execution of ImprovePath(). ■

References

[1] A. Bagchi and P. K. Srimani. Weighted heuristic search in networks. *Journal of Algorithms*, 6:550–576, 1985.

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

[3] P. P. Chakrabarti, S. Ghosh, and S. C. DeSarkar. Admissibility of AO* when heuristics overestimate. *Artificial Intelligence*, 34:97–113, 1988.

[4] T. L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1988.

[5] S. Edelkamp. Planning with pattern databases. In *Proc. of the European Conference on Planning (ECP)*, 2001.

[6] D. Haehnel. Personal communication, 2003.

[7] E. Hansen, S. Zilberstein, and V. Danilchenko. Anytime heuristic search: First results. Tech. Rep. CMPSCI 97-50, University of Massachusetts, 1997.

[8] E. J. Horvitz. Problem-solving design: Reasoning about computational value, trade-offs, and reasources. In *Proc. of the Second Annual NASA Research Forum*, 1987.

[9] S. Koenig and M. Likhachev. Incremental A*. In *Advances in Neural Information Processing Systems (NIPS) 14*. Cambridge, MA: MIT Press, 2002.

[10] R. E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[11] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. in submission, 2003.

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

[13] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[14] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[15] R. Zhou and E. A. Hansen. Multiple sequence alignment using A*. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2002. Student abstract.

[16] S. Zilberstein and S. Russell. Approximate reasoning using anytime algorithms. In *Imprecise and Approximate Computation*. Kluwer Academic Publishers, 1995.