Anytime Dynamic A*: The Proofs

Maxim Likhachev, Dave Ferguson, Geoff Gordon,
Anthony Stentz and Sebastian Thrun [a]

April 2005
CMU-RI-TR-05-12

[a]S. Thrun is affiliated with Stanford University,
all other authors are affiliated with Carnegie Mellon University.

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper presents a thorough theoretical analysis of a recently developed anytime incremental planning algorithm called Anytime Dynamic A* (AD*). The analysis includes proofs for the correctness of the algorithm and as well as the proofs for several properties of the algorithm related to its efficiency.

Keywords: search, planning, anytime planning, incremental planning, re-planning, anytime incremental planning

1 Introduction

This paper presents a theoretical analysis of the anytime incremental planning algorithm, Anytime Dynamic A* (AD*), that is described in [3]. Mainly, the analysis includes proofs for the correctness of the algorithm. It also, however, proves several theorems regarding the efficiency of the algorithm. For the intuitive explanation of the algorithm as well as its empirical evaluation please refer to [3].

The analysis is organized as follows. First, in section 2 we describe some of the notations as well as the assumptions the algorithm makes about its input. Next, in section 3 we give the pseudocode of AD*. In this section we also explain some of the differences between the pseudocode of AD* as presented here and the pseudocode of AD* as presented in [3]. Finally, in section 4 we present the proofs of all the theorems about AD*. The section is split into few subsections with the hope of simplifying the task of reading the proofs. The most important and interesting theorems about the algorithm are presented in section 4.2. The actual statements about the correctness (ϵ sub-optimality) and the efficiency of the algorithm are presented in sections 4.3 and 4.4 respectively.

2 Notation

In the following we assume AD* operates on a finite size graph. The set of states is denoted by S. $succ(s)$ denotes the set of successors of state $s \in S$ and $pred(s)$ denotes the set of predecessors of state s.

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'. For $s = s'$ we define $c^*(s, s') = 0$. $g^*(s)$ denotes the cost of a shortest path from s_{start} to s. The task of AD* is to maintain a path from s_{start} state to s_{goal} state that has a cost of at most $\epsilon * c^*(s, s_{\text{goal}})$. AD* may dynamically change ϵ to trade off the quality of solution with the computational expense of computing it. We restrict the range of ϵ values to $1 \leq \epsilon < \infty$.

In this paper AD* searches forward, from s_{start} towards s_{goal}. Consequently, the provided heuristic values need to be forward consistent: $h(s) \leq c(s, s') + h(s')$ for any $s, s' \in succ(s)$ and $h(s) = 0$ for $s = s_{\text{goal}}$.

AD* maintains two estimates of start distances (the cost of a shortest path from s_{start} to the state) for a state $s \in S$: $g(s)$ and $v(s)$. We call a state

s *inconsistent* iff $v(s) \neq g(s)$, *overconsistent* iff $v(s) > g(s)$, *underconsistent* iff $v(s) < g(s)$ and *consistent* iff $v(s) = g(s)$. AD* also maintains backpointers, $bp(s)$, that point to the predecessor state of s via which a currently found path from s_{start} to s goes.

To make the following proofs easier to read we assume that the min operation on an empty set returns ∞, arg min operation on the set consisting of only infinite values returns **null** and any state s with undefined values (in other words, a state that has not been visited yet) has $v(s) = g(s) = \infty$ and $bp(s) =$ **null**. Additionally, in the computation $g(s) = v(bp(s)) + c(bp(s), s)$ it is assumed that if $bp(s) =$ **null** then $g(s)$ is set to ∞.

Finally, throughout the proofs we sometimes refer to a path from s_{start} to s defined by backpointers. This path is defined as a path that is computed by tracing it backward as follows: start at s, and at any state s_i pick a state $s_{i-1} = bp(s_i)$ until $s_{i-1} = s_{\text{start}}$.

3 Pseudocode of AD*

The pseudocode somewhat differs from the one given in [3] as here we have incorporated some of the important optimizations that were described in [1] and were applicable to AD*. The direction of search is changed. So, AD* as presented here searches forward, from s_{start} towards s_{goal}. We have also changed the name of the variable $g(s)$ (where s is a state in the input graph) onto $v(s)$ and the name of the variable $rhs(s)$ onto $g(s)$. This makes the variable $g(s)$ more consistent with the commonly accepted g-values used by A* search [4]. For example, during the first search of AD* these g-values are exact equivalents of the g-values that A* maintains. Algorithmically, the pseudocode of AD* as presented here is the same as the pseudocode of AD* as presented in [3] except for line 2 in figure 1 where a $>$ sign was changed into a $\geq$ sign. [1]

[1]The use of $>$ sign (as in [3]) still results in a correct algorithm if one only cares about the solution and its ϵ sub-optimality. It may, however, violate ϵ sub-optimality of the values of some other states, namely, the states that do not belong to the found solution but should still be ϵ sub-optimal as one of the theorems about AD* claims.

```
1  procedure key(s)
2  if (v(s) ≥ g(s))
3    return [g(s) + ϵ * h(s); g(s)];
4  else
5    return [v(s) + h(s); v(s)];

6  procedure UpdateSetMembership(s)
7  if (v(s) ≠ g(s))
8    if (s ∉ CLOSED) insert/update s in OPEN with key(s);
9    else if (s ∉ INCONS) insert s into INCONS;
10 else
11   if (s ∈ OPEN) remove s from OPEN;
12   else if (s ∈ INCONS) remove s from INCONS;

13 procedure ComputePath()
14 while(key(s_goal) > min_{s∈OPEN}(key(s)) OR v(s_goal) < g(s_goal))
15   remove s with the smallest key(s) from OPEN;
16   if (v(s) > g(s))
17     v(s) = g(s); CLOSED←CLOSED ∪ {s};
18     for each successor s' of s
19       if s' was not visited before then
20         v(s') = g(s') = ∞; bp(s') = null;
21       if g(s') > g(s) + c(s, s')
22         bp(s') = s;
23         g(s') = g(bp(s')) + c(bp(s'), s'); UpdateSetMembership(s');
24   else //propagating underconsistency
25     v(s) = ∞; UpdateSetMembership(s);
26     for each successor s' of s
27       if s' was not visited before then
28         v(s') = g(s') = ∞; bp(s') = null;
29       if bp(s') = s
30         bp(s') = arg min_{s''∈Pred(s')} v(s'') + c(s'', s');
31         g(s') = v(bp(s')) + c(bp(s'), s'); UpdateSetMembership(s');
```

Figure 1: ComputePath function in AD*

4 Proofs

The proofs are split into several subsections. The section 4.1 mostly proves that ComputePath function in figure 1 correctly maintains its variables. The section 4.2 is more interesting and proves the key properties of the algorithm. The section 4.3 uses these key properties to quite trivially derive the ϵ sub-optimality property of the algorithm. Finally, section 4.4 proves several properties regarding the efficiency of the algorithm.

```
1  procedure Main()
2  g(s_goal) = v(s_goal) = ∞; v(s_start) = ∞; bp(s_goal) = bp(s_start) = null;
3  g(s_start) = 0; OPEN = CLOSED = INCONS = ∅; ε = ε_0;
4  insert s_start into OPEN with key(s_start);
5  forever
6    ComputePath();
7    publish ε-suboptimal solution;
8    if ε = 1
9      wait for changes in edge costs;
10   for all directed edges (u, v) with changed edge costs
11     update the edge cost c(u, v);
12     if v ≠ s_start
13       if v was not visited before then
14         v(v) = g(v) = ∞; bp(v) = null;
15       bp(v) = arg min_{s''∈Pred(v)} v(s'') + c(s'', v);
16       g(v) = v(bp(v)) + c(bp(v), v); UpdateSetMembership(v);
17   if significant edge cost changes were observed
18     increase ε or re-plan from scratch (i.e., re-execute Main function);
19   else if ε > 1
20     decrease ε;
21   Move states from INCONS into OPEN;
22   Update the priorities for all s ∈ OPEN according to key(s);
23   CLOSED = ∅;
```

Figure 2: Main function in AD*

4.1 Low-level Properties

Most of the theorems in this section are merely stating the correctness of the program state variables such as v-, g-values and bp-values, and $OPEN$, $CLOSED$ and $INCONS$ sets.

Lemma 1 *For any pair of states s and s', $\epsilon * h(s) \leq \epsilon * c^*(s, s') + \epsilon * h(s')$.*

Proof: The consistency property required of heuristics is equivalent to the restriction that $h(s) \leq c^*(s, s') + h(s')$ for *any* pair of states s, s' and $h(s_{\text{goal}}) = 0$ ([4]). The theorem then follows by multiplying the inequality with ϵ. ■

Theorem 2 *At line 14 in ComputePath function, all v- and g-values are non-negative, $bp(s_{\text{start}}) =$ **null**, $g(s_{\text{start}}) = 0$ and for $\forall s \neq s_{\text{start}}$, $bp(s) = \arg\min_{s' \in pred(s)}(v(s') + c(s', s))$, $g(s) = v(bp(s)) + c(bp(s), s)$.*

Proof: The theorem holds the first time line 14 in ComputePath function is executed due to the initialization performed by function Main: the g- and v-values of all states except for s_{start} are infinite, and $v(s_{start}) = \infty$ and $g(s_{start}) = 0$. Also, the bp-values of all states are equal to **null**. In other words, for every state $s \neq s_{\text{start}}$, $bp(s) =$ **null** and $v(s) = g(s) = \infty$, and for $s = s_{\text{start}}$, $bp(s) =$ **null**, $g(s) = 0$ and $v(s) = \infty$. Thus, the theorem holds when line 14 is executed for the first time.

We will now prove that theorem continues to hold during the next execution of line 14 and thus holds by induction. Before line 14 is executed next two mutually exclusive possibilities exist. One possibility is that the body of the while loop in ComputePath function was executed. Another possibility is that the code in between two calls to ComputePath function in function Main was executed (lines 7-23).

Let us first consider the former possibility, namely another execution of the body of the while loop in ComputePath function. The only places where g-, v- and bp-values are changed and therefore the theorem might be affected are lines 17, 22, 23, 25, 30, and 31. The initialization of newly visited states on lines 20 and 28 does not change the values of states according to our convention of assuming that all states that have not been visited so far have infinite v- and g-values and their bp-values are equal to **null**. Let us now examine each of the lines that do change the values of states.

If $v(s)$ is set to $g(s)$ on line 17, then it is decreased since s is being expanded as overconsistent (i.e., $v(s) > g(s)$ before the expansion according to the test on line 16). Thus, it may only decrease the g-values of its successors. The test on line 21 checks this for each successor of s and updates the bp- and g-values if necessary. Since all costs are positive $bp(s_{\text{start}})$ and $g(s_{\text{start}})$ can never be changed: it will never pass the test on line 21, and thus is always 0. Since $v(s)$ is set to $g(s)$ it still remains non-negative. Consequently, when the g-values of the successors of s are re-calculated their g-values also remain non-negative.

If $v(s)$ is set to ∞ on line 25, then it either stays the same or increases since it is the "else" case of the test on line 16 (i.e., $v(s) \leq g(s)$ before the expansion of s). (As we will show in later theorems the inequality will always be strict as no consistent state is ever expanded.) Thus, it may only affect the g- and $bp-$ values of the successors of s if their bp-value is equal to s. The test on line 29 checks this for each successor of s and re-computes the bp- and g-values if necessary. Since $bp(s_{\text{start}}) =$ **null** the test will never pass for s_{start} and therefore $bp(s_{\text{start}})$ and $g(s_{\text{start}})$ can never be changed. Since

$v(s)$ is set to ∞ it remains non-negative. Consequently, when the g-values of the successors of s are re-calculated their g-values also remain non-negative. The theorem thus holds during the next execution of line 14 if the body of the ComputePath while loop was executed in between.

Suppose now that before the next execution of line 14 the code that resides in between lines 7 and 23 in function Main is executed. During this execution costs may change on line 11. The *bp*- and g-values, however, are updated correctly on the following lines 15 and 16. No other code during this execution changes any of the state values that appear in the theorem (the initialization code on line 14 does not change state values according to our assumption that all states that we haven't seen before are assumed to have infinite v- and g-values and *bp*-values equal to **null**.) Hence, independently of the execution path the theorem holds the next time line 14 is executed, and thus it holds always by induction. ■

Theorem 3 *At line 14 in ComputePath function, OPEN and CLOSED are disjoint, OPEN contains only inconsistent states, the union of OPEN and CLOSED contains all inconsistent states (and possibly others) and INCONS contains all and only inconsistent states that are also in CLOSED.*

Proof: We will prove the theorem by induction. Consider the first execution of line 14 in ComputePath function. Due to the initialization performed by Main function, at this point $CLOSED = INCONS = \emptyset$ and $OPEN$ contains only s_{start}. Because after the initialization for every state $s \neq s_{\text{start}}$, $v(s) = g(s) = \infty$, and for $s = s_{\text{start}}$, $v(s) = \infty \neq 0 = g(s)$, $OPEN$ contains all and only inconsistent states and the theorem holds.

We now need to show that the theorem continues to hold the next time line 14 in ComputePath function is executed given that the theorem held during all the previous executions of this line. Before line 14 is executed next the executed code could either be the body of the while loop in ComputePath function or the code in between two calls to ComputePath function in function Main (lines 7-23).

Let us first consider another execution of the body of the while loop in ComputePath function. In particular, let us examine all the lines of the body of the while loop in ComputePath where we change v- or g-values of states or their set membership during the following execution of ComputePath. On line 15 we remove a state s from $OPEN$. This state is expanded as either overconsistent or underconsistent.

If state s is overconsistent, then on line 17 we insert it into *CLOSED* but since we set $v(s) = g(s)$ on the same line, the state is consistent. The state is thus correctly not a member of *OPEN* and can not be a member of *INCONS* since at the last execution of line 14 it was a member of *OPEN*, and *OPEN* and *INCONS* were disjoint according to the statement of the theorem. The state s thus does not violate the theorem after line 17 is executed.

If state s is underconsistent, then on line 25 we set $v(s) = \infty$ but also call UpdateSetMembership function on the same line. On all the other lines of ComputePath where we modify either v- or g-values of states except for state initialization (lines 20 and 28) we also call UpdateSetMembership function. The state initialization code does not change the values of states according to our convention of assuming that all states that have not been visited so far have infinite v- and g-values. We therefore only need to show that UpdateSetMembership function operates consistently with the theorem.

In UpdateSetMembership function if a state s is inconsistent and is not in *CLOSED* it is inserted into *OPEN*, otherwise it is inserted into *INCONS* (unless it is already there). Since *OPEN* and *CLOSED* were disjoint before we called UpdateSetMemberhip, this procedure ensures that an inconsistent state s does appear in either *OPEN* or *CLOSED* but not both and does appear in *INCONS* if it also belongs to *CLOSED*. If a state s is consistent and belongs to *OPEN* then it is correctly removed from it. It does not belong to *CLOSED* since these sets were disjoint before UpdateSetMembership was called. Consequently, s also does not belong to *INCONS* because it is a subset of *CLOSED*. If a state s is consistent and does not belong to *OPEN*, then it may potentially belong to *INCONS*. We check this and remove s from *INCONS* if it is there on line 12.

Suppose now that before the next execution of line 14 in ComputePath the code that resides in between lines 7 and 23 in function Main is executed. During this execution state values may change on line 16 (the initialization code on line 14 does not change state values according to our assumption that all states that we haven't seen before are assumed to have infinite v- and g-values). On the same line, however, we again correctly update the set membership by calling UpdateSetMembership function on the same line. On line 21 we move *INCONS* into *OPEN*. Hence, *INCONS* becomes empty and *OPEN* contains all and only inconsistent states. We then set *CLOSED* to empty set on line 23. This makes the sets to be consistent with the statement of the theorem. ■

Theorem 4 *Suppose an overconsistent state s is selected for expansion on line 15 in ComputePath. Then the next time line 14 in ComputePath is executed $v(s) = g(s)$, where $g(s)$ before and after the expansion of s is the same.*

Proof: Suppose an overconsistent state s (i.e., $v(s) > g(s)$) is selected for expansion. Then on line 17 in ComputePath $v(s) = g(s)$, and it is the only place where a v-value changes while expanding an overconsistent state. We, thus, only need to show that $g(s)$ does not change. It could only change if $s \in succ(s)$ and $g(s) > g(s) + c(s, s)$ (test on line 21 in ComputePath) which is impossible since all costs are positive. ■

Theorem 5 *The following properties hold for any two states $s, s' \in S$ and the definition of the procedure **key** as given in figure 1:*

(a) if $c^*(s', s_{\text{goal}}) < \infty$, $v(s') \geq g(s')$, $v(s) > g(s)$ and $g(s') > g(s) + \epsilon * c^*(s, s')$, then key$(s')$ > key(s);

(b) if $c^*(s', s_{\text{goal}}) < \infty$, $v(s') \geq g(s')$, $v(s) < g(s)$ and $g(s') \geq v(s) + c^*(s, s')$, then key$(s')$ > key(s);

Proof: Consider first property (a): we are given two arbitrary states s' and s such that $c^*(s', s_{\text{goal}}) < \infty$, $v(s') \geq g(s')$, $v(s) > g(s)$ and $g(s') > g(s) + \epsilon * c^*(s, s')$, and we need to show that these conditions imply key(s') > key(s). Given the definition of key function in figure 1 we need to show that $[g(s') + \epsilon * h(s'); g(s')] > [g(s) + \epsilon * h(s); g(s)]$. We examine the inequality $g(s') > g(s) + \epsilon * c^*(s, s')$ and add $\epsilon * h(s')$, which is finite since $c^*(s', s_{\text{goal}})$ is finite and heuristics are consistent. We thus have $g(s') + \epsilon * h(s') > g(s) + \epsilon * c^*(s, s') + \epsilon * h(s')$ and from Lemma 1 we obtain $g(s') + \epsilon * h(s') > g(s) + \epsilon * h(s)$ that guarantees that the desired inequality holds.

We now prove property (b): we are given two arbitrary states s' and s such that $c^*(s', s_{\text{goal}}) < \infty$, $v(s') \geq g(s')$, $v(s) < g(s)$ and $g(s') \geq v(s) + c^*(s, s')$, and we need to show that these conditions imply key(s') > key(s). Given the definition of key function in figure 1 we need to show that $[g(s') + \epsilon * h(s'); g(s')] > [v(s) + h(s); v(s)]$. Since $v(s) < g(s)$, $v(s)$ is finite. Consider now the inequality $g(s') \geq v(s) + c^*(s, s')$. Because $v(s) < \infty$ and costs are positive we can conclude that $g(s') > v(s)$. We now add $\epsilon * h(s')$ to both sides of the inequality and use the consistency of heuristics as follows $g(s') + \epsilon * h(s') \geq v(s) + c^*(s, s') + \epsilon * h(s') \geq v(s) + c^*(s, s') + h(s') \geq v(s) + h(s)$.

Hence, we have $g(s') + \epsilon * h(s') \geq v(s) + h(s)$ and $g(s') > v(s)$. These inequalities guarantee that $[g(s') + \epsilon * h(s'); g(s')] > [v(s) + h(s); v(s)]$. ■

4.2 Main Theorems

We now prove two main theorems about AD*. These theorems guarantee ϵ sub-optimality of each search iteration (whatever the current value of ϵ is): every time ComputePath returns it has identified a set of states s for which a path from s_{start} to s as defined by backpointers is guaranteed to be of cost no larger than $g(s)$ which in turn is no more than a factor of ϵ greater than the optimal cost $g^*(s)$.

Single-shot optimal search algorithms such as Dijkstra's and A* search can often be proven based on the property that every time a state s with the smallest priority among all candidates for expansion is selected for expansion, all the states that lie on an optimal path from s_{start} to s have already been expanded and have correct values while all the states that have not been yet expanded have values that are bounded below by their optimal values. As a result, when expanding a state s the predecessor of s that lies on an optimal path from s_{start} to s can easily be identified as the state $s' \in pred(s)$ that minimizes the sum of the value of s' and the cost $c(s', s)$. Consequently, the value of s during its expansion can be set to the sum of the value of s' and the cost $c(s', s)$.

In case of AD* things get a bit more complicated because ComputePath function needs only to compute an ϵ sub-optimal solution and may also encounter states whose values are actually smaller than their respective optimal values since costs may decrease in between search iterations. To deal with this we introduce a set Q:

$$Q = \{u \mid v(u) < g(u) \vee (v(u) > g(u) \wedge v(u) > \epsilon * g^*(u))\} \qquad (1)$$

The set Q contains all underconsistent states and all those overconsistent states whose v-values are larger than a factor of ϵ of their optimal values, g^*-values. Given such a set, we can now formulate a property that is equivalent to the property that we have described above for single-shot optimal search algorithms: if we select any overconsistent or consistent state s whose priority is smaller than or equal to the smallest priority of states in Q, then s has a g-value that is at most a factor of ϵ larger than $g^*(s)$ and the path defined

by backpointers from s_{start} to s has a cost no larger than $g(s)$. This is put formally in Theorem 6.

Theorem 7 builds on this result by showing that *OPEN* is always a superset of Q, and therefore any overconsistent or consistent state s whose priority is smaller than or equal to the smallest priority of states in *OPEN* has a g-value that is at most a factor of ϵ larger than $g^*(s)$ and the path defined by backpointers from s_{start} to s has a cost no larger than $g(s)$. This property can be used to explain the operation of AD* quite simply. When selecting for expansion a state s as a state with the smallest priority among all the states in *OPEN*, AD* guarantees to handle s correctly: if s is overconsistent, then setting $v(s) = g(s)$ makes $v(s)$ at most ϵ sub-optimal and thus removes s from set Q, while if s is underconsistent, then setting $v(s) = \infty$ forces s to become overconsistent and the next time it is selected for expansion it will be overconsistent.

Theorem 6 *At line 14 in ComputePath, let Q be defined according to the definition 1. Then for any state s with $(c^*(s, s_{\text{goal}}) < \infty \wedge v(s) \geq g(s) \wedge key(s) \leq key(u)\ \forall u \in Q)$, it holds that (i) $g(s) \leq \epsilon * g^*(s)$, (ii) the cost of the path from s_{start} to s defined by backpointers is no larger than $g(s)$.*

Proof: (i) We first prove statement (i). We prove by contradiction. Suppose there exists an s such that $(c^*(s, s_{\text{goal}}) < \infty \wedge v(s) \geq g(s) \wedge key(s) \leq key(u)\ \forall u \in Q)$, but $g(s) > \epsilon * g^*(s)$. The latter implies that $g^*(s) < \infty$. We also assume that $s \neq s_{\text{start}}$ since otherwise $g(s) = 0 = \epsilon * g^*(s)$ from Theorem 2.

Consider a least-cost path from s_{start} to s, $\pi(s_0 = s_{\text{start}}, ..., s_k = s)$. The cost of this path is $g^*(s)$. Such path must exist since $g^*(s) < \infty$. We will now show that such path must contain a state s' that is overconsistent and whose v-value overestimates $g^*(s')$ by more than ϵ. As a result, such state is a member of Q. We will then show that its key, on the other hand, must be strictly smaller than the key of s. This, therefore, becomes a contradiction since s according to the theorem assumptions has a key smaller than or equal to the key of any state in Q.

Our assumption that $g(s) > \epsilon * g^*(s)$ means that there exists at least one $s_i \in \pi(s_0, ..., s_{k-1})$, namely s_{k-1}, whose $v(s_i) > \epsilon * g^*(s_i)$. Otherwise,

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

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

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 first show that $\epsilon * g^*(s_i) \geq g(s_i)$. It is clearly so when $i = 0$ according to Theorem 2 which says that $g(s_i) = g(s_{\text{start}}) = 0$. For $i > 0$ we use the fact that $v(s_{i-1}) \leq \epsilon * g^*(s_{i-1})$ from the way s_i was chosen,

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

We thus have $v(s_i) > \epsilon * g^*(s_i) \geq g(s_i)$, which also implies that $s_i \in Q$.

We now show that $\text{key}(s) > \text{key}(s_i)$, and therefore arrive at a contradiction. According to our assumption

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

Hence, we have $g(s) > g(s_i) + \epsilon * c^*(s_i, s)$, $v(s_i) > \epsilon * g^*(s_i) \geq g(s_i)$, $v(s) \geq g(s)$ and $c^*(s, s_{\text{goal}}) < \infty$ from theorem assumptions. Thus, from Theorem 5, property (a), it follows that $\text{key}(s) > \text{key}(s_i)$. This inequality, however, implies that $s_i \notin Q$ since $\text{key}(s) \leq \text{key}(u)$ $\forall u \in Q$. But this contradicts to what we have proven earlier.

(ii) Let us now prove statement (ii). We assume that $g(s) < \infty$ for otherwise the statement holds trivially. Suppose we start following the back-pointers starting at s. We need to show that we will reach s_{start} at the cumulative cost of the transitions less than or equal to $g(s)$ (we assume that if we encounter a state with *bp*-value equal to **null** before s_{start} is reached then the cumulative cost is infinite).

We first show that we are guaranteed not to encounter an underconsistent state or a state with *bp*-value equal to **null** before s_{start} is reached. Once we have this property proven, we will be able to show that the cost of the path

is bounded above by $g(s)$ simply from the fact that at each backtracking step in the path the g-value can only be larger than or equal to the sum of the g-value of the state the backpointer points to and the cost of the transition. Consequently, the g-value can never underestimate the cost of the remaining part of the path. The property that we are guaranteed not to encounter an underconsistent state or a state with *bp*-value equal to **null** before s_{start} is reached is based on the fact that any such state will have a key strictly smaller than the key of s or have an infinite g-value. The first case is impossible because key(s) is smaller than or equal to the key of any state in Q and this set already contains all underconsistent states. The second case can also be shown to be impossible quite trivially.

We thus first prove by contradiction the property that we are guaranteed not to encounter an underconsistent state or a state with *bp*-value equal to **null** before s_{start} is reached while following backpointers from s to s_{start}. Suppose the sequence of backpointer transitions leads us through the states $\{s_0 = s, s_1, \ldots, s_i\}$ where s_i is the first state that is either underconsistent or has $bp(s_i) =$ **null** (or both). It could not have been state s since $v(s) \geq g(s)$ from the assumptions of the theorem and $g(s) < \infty$ implies $bp(s) \neq$ **null** according to Theorem 2 (except when $s = s_{\text{start}}$ in which case the theorem holds trivially). We now show that s_i can not be underconsistent. Since all the states before s_i are *not* underconsistent and have defined backpointer values we have $g(s) = v(s_1)+c(s_1, s) \geq g(s_1)+c(s_1, s) = v(s_2)+c(s_2, s_1)+c(s_1, s) \geq \ldots \geq v(s_i) + \sum_{k=1..i} c(s_k, s_{k-1}) \geq v(s_i) + c^*(s_i, s)$. If s_i was underconsistent, then we would have had $c^*(s, s_{\text{goal}}) < \infty$, $v(s) \geq g(s)$, $v(s_i) < g(s_i)$ and $g(s) \geq v(s_i)+c^*(s_i, s)$, and this, according to Theorem 5 property (b), would imply that $key(s) > key(s_i)$ which means that $s_i \notin Q$ and therefore can not be underconsistent according to the definition of Q. We will now show that $bp(s_i)$ can not be equal to **null** either. Since s_i is not underconsistent $v(s_i) \geq g(s_i)$. From our assumption that $g(s) < \infty$ and the fact that $g(s) \geq v(s_i) + c^*(s_i, s)$ it then follows that $g(s_i)$ is finite. As a result, from Theorem 2 $bp(s_i) \neq$ **null** unless $s_i = s_{\text{start}}$. Hence, as we backtrack from s to s_{start} the path defined by backpointers we are guaranteed to have states that are not underconsistent and whose *bp*-values are not equal to **null** except for s_{start}.

We are now ready to show that the cost of the path from s_{start} to s defined by backpointers is no larger than $g(s)$. Let us denote such path as: $s_0 = s_{\text{start}}, s_1, ..., s_k = s$. Since all states on this path are either consistent or overconsistent and their *bp*-values are defined (except for s_{start}), for any i,

$k \geq i > 0$, we have $g(s_i) = v(s_{i-1}) + c(s_{i-1}, s_i) \geq g(s_{i-1}) + c(s_{i-1}, s_i)$ from Theorem 2. For $i = 0$, $g(s_i) = g(s_{\text{start}}) = 0$ from the same theorem. 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 path from s_{start} to s as defined by backpointers. ■

Theorem 7 *At line 14 in ComputePath, for any state s with ($c^*(s, s_{\text{goal}}) < \infty \wedge v(s) \geq g(s) \wedge key(s) \leq key(u)\ \forall u \in OPEN$), it holds that (i) $g(s) \leq \epsilon * g^*(s)$, (ii) the cost of the path from s_{start} to s defined by backpointers is no larger than $g(s)$.*

Proof: Let Q be defined according to the definition 1. To prove the theorem we will show that Q is a subset of *OPEN* and then appeal to Theorem 6. We will show that Q is a subset of *OPEN* by induction. We will first show that it holds initially because *OPEN* contains all inconsistent states initially and set Q is a subset of those. Afterwards, we will show that any state $s \in CLOSED$ always remains either consistent or overconsistent but with $v(s) \leq \epsilon * g^*(s)$. Given that the union of *OPEN* and *CLOSED* contains all inconsistent states, it is then clear that *OPEN* contains at least all those (and possibly other) inconsistent states that are in Q.

We now prove the theorem. From the definition of set Q it is clear that for any state $u \in Q$ it holds that u is inconsistent (that is, $v(u) \neq g(u)$). According to Theorem 3 and the fact that right before ComputePath is called *CLOSED* is always empty (lines 3 and 23 in function Main) when the ComputePath function is called *OPEN* contains all inconsistent states. Therefore $Q \subseteq OPEN$, because as we have just said any state $u \in Q$ is also inconsistent. Thus, if any state s has $\text{key}(s) \leq \text{key}(u)\ \forall u \in OPEN$, it is also true that $\text{key}(s) \leq \text{key}(u)\ \forall u \in Q$. From the direct application of Theorem 6 it then follows that the first time line 14 in ComputePath is executed the theorem holds.

Also, because during the first execution of line 14 $CLOSED = \emptyset$ (lines 3 and 23 in function Main), the following statement, denoted by (*), trivially holds when line 14 is executed for the first time within any particular call to ComputePath function: for any state $v \in CLOSED$ it holds that $g(v) \leq v(v) \leq \epsilon * g^*(v)$ and $g(v) < v(s') + c^*(s', v)\ \forall s' \in \{s'' \mid v(s'') < g(s'')\}$. We will later prove that this statement always holds and thus all states $v \in CLOSED$ are either consistent or overconsistent but ϵ sub-optimal (i.e., $v(v) \leq \epsilon * g^*(v)$).

We will now show by induction that the theorem continues to hold for the subsequent executions of line 14 within the current call to ComputePath. Suppose the theorem and the statement (*) held during all the previous executions of line 14, and they still hold when a state s is selected for expansion on line 15. We need to show that the theorem holds the next time line 14 in ComputePath is executed.

We first prove that the statement (*) still holds during the next execution of line 14. Suppose first we select an overconsistent state s to be expanded. Because it is added to *CLOSED* immediately afterwards, we need to show that it does not violate statement (*). Since when s is selected for expansion on line 15 $\text{key}(s) = \min_{u \in OPEN}(\text{key}(u))$, we have $\text{key}(s) \leq \text{key}(u)\ \forall u \in OPEN$. According to the assumptions of our induction then $g(s) \leq \epsilon * g^*(s)$. From Theorem 4 it then also follows that the next time line 14 is executed $g(s) = v(s) \leq \epsilon * g^*(s)$. To show that $g(s) < v(s') + c^*(s', s)\, \forall s' \in \{s'' \mid v(s'') < g(s'')\}$ after s is expanded we show that this is true right before s is expanded and therefore since v-values of all states except for s do not change during the expansion of s and $g(s)$ does not change either (Theorem 4) it still holds afterwards. To show that the inequality held before the expansion of s we note that according to our assumptions *CLOSED* contained no underconsistent states and they were all therefore in *OPEN* (Theorem 3); from the way s was selected from *OPEN* it then followed that $\text{key}(s) \leq \text{key}(s')\, \forall s' \in \{s'' \mid v(s'') < g(s'')\}$; finally, the fact that s was overconsistent ($v(s) > g(s)$) implies that $g(s) < v(s') + c^*(s', s)\, \forall s' \in \{s'' \mid v(s'') < g(s'')\}$ because otherwise $c^*(s, s_{\text{goal}}) < \infty, v(s) > g(s)$, $v(s') < g(s')$ and $g(s) \geq v(s') + c^*(s', s)$ would imply $\text{key}(s) > \text{key}(s')$ according to the Theorem 5, property(b). As for the rest of the states the statement (*) follows from the following observations: only v-value of s was changed and s is not underconsistent after its expansion (it is in fact consistent according to Theorem 4); since $g(s)$ decreased during the expansion of s the g-values of its successors could only decrease implying that they could not have violated the statement (*); and finally no other changes to either v- or g-values were done and no operations except for insertion of s were done on *CLOSED*.

Suppose now an underconsistent state s is selected for expansion. Because it is not added to *CLOSED*, we only need to show that statement (*) remains to hold true for all the states that were in *CLOSED* prior to the expansion of s. Since only v-value of s has been changed, none of the v-values of states in *CLOSED* are changed. We will now show that none of their g-values could have changed either. Since prior to the expansion of s, s was

underconsistent and statement (*) held by our induction assumptions, it was true that for any state $v \in CLOSED$, $g(v) < v(s) + c^*(s, v)$. This means that $bp(v) \neq s$ (Theorem 2) and therefore the test on line 29 will not pass and $g(v)$ will not change during the expansion of s. Finally, we will now show that the newly introduced underconsistent states could not have violated the statement (*) either. The v-values of states that were underconsistent before s was expanded were not changed (v-value of only s was changed and s could not remain underconsistent as its v-value was set to ∞). Suppose some state s' became underconsistent as a result of expanding s. We need to show that after the expansion of s, for any state $v \in CLOSED$ it holds that $g(v) < v(s') + c^*(s', v)$. Since s' became underconsistent as a result of expanding s it must be the case that before the expansion of s $v(s') \geq g(s')$ and $bp(s') = s$ (in order for $g(s')$ to change). Consequently before the expansion of s, $v(s') \geq g(s') = v(s) + c(s, s')$. Since before the expansion of s statement (*) held, for any state $v \in CLOSED$ $g(v) < v(s) + c^*(s, v)$. We thus had $g(v) < v(s) + c^*(s, v) \leq v(s) + c(s, s') + c^*(s', v) \leq v(s') + c^*(s', v)$. This continues to hold after the expansion of s since neither $v(s')$ nor $g(v)$ changes during the expansion of s as we have just shown. Hence the statement (*) continues to hold the next time line 14 in ComputePath is executed.

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 14 is executed. According to Theorem 3 *OPEN* set contains all inconsistent states that are not in *CLOSED*. Since, as we have just proved, the statement (*) holds the next time line 14 is executed, all states s in *CLOSED* set have $g(s) \leq v(s) \leq \epsilon * g^*(s)$. Thus, any state s that is inconsistent and has either $g(s) > v(s)$ or $v(s) > \epsilon * g^*(s)$ (or both) is guaranteed to be in *OPEN*. Now consider any state $u \in Q$. As we have shown earlier such state u is inconsistent, and either $g(u) > v(u)$ or $v(u) > \epsilon * g^*(u)$ (or both) according to the definition of Q. Thus, $u \in OPEN$. This shows that $Q \subseteq OPEN$. Consequently, if any state s has $c^*(s, s_{\text{goal}}) < \infty \wedge v(s) \geq g(s) \wedge \text{key}(s) \leq \text{key}(u)\ \forall u \in OPEN$, it is also true that $c^*(s, s_{\text{goal}}) < \infty \wedge v(s) \geq g(s) \wedge \text{key}(s) \leq \text{key}(u)\ \forall u \in Q$, and the statement of the theorem holds from Theorem 6. This proves that the theorem holds during the next execution of line 14 in ComputePath, and proves the whole theorem by induction. ■

4.3 Correctness

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

Corollary 8 *Every time ComputePath function exits the following holds for any state s with* $(c^*(s, s_{\text{goal}}) < \infty \wedge v(s) \geq g(s) \wedge key(s) \leq \min_{s' \in OPEN}(key(s')))$*: the cost of the path from* s_{start} *to s defined by backpointers is no larger than* $\epsilon * g^*(s)$*.*

Proof: The corollary follows directly from Theorem 7 after we combine the statements (i) and (ii) of the theorem together. ■

Corollary 9 *Every time ComputePath function exits the following holds: the cost of the path from* s_{start} *to* s_{goal} *defined by backpointers is no larger than* $\epsilon * g^*(s_{\text{goal}})$*.*

Proof: According to the termination condition of the ComputePath function, upon its exit $(v(s_{\text{goal}}) \geq g(s_{\text{goal}}) \wedge \text{key}(s_{\text{goal}}) \leq \min_{s' \in OPEN}(\text{key}(s')))$. The proof then follows from Corollary 8 noting that $c^*(s_{\text{goal}}, s_{\text{goal}}) = 0$. ■

4.4 Efficiency

Several theorems in this section provide some theoretical guarantees about the efficiency of AD*. We do not prove it here, but it can also be shown (see [2]) that when AD* calls the ComputePath function to (re-)compute an optimal solution (in other words, with $\epsilon = 1$), no state whose v-value is already equal to its g^*-value is expanded.

Theorem 10 *Within any particular execution of ComputePath function once a state is expanded as overconsistent it can never be expanded again (independently of it being overconsistent or underconsistent).*

Proof: Suppose a state s is selected for expansion as overconsistent for the first time during the execution of the ComputePath function. Then, it is removed from *OPEN* set on line 15 and inserted into *CLOSED* set on line 17. It can then never be inserted into *OPEN* set again unless the ComputePath function exits since any state that is about to be inserted into *OPEN* set is

checked against membership in *CLOSED* on line 8. Because only the states from *OPEN* set are selected for expansion, s can therefore never be expanded second time. ■

Theorem 11 *No state is expanded more than once as underconsistent within any particular execution of ComputePath function.*

Proof: Once a state is expanded as underconsistent its v-value is set to ∞. As a result, unless the state is expanded as overconsistent this state can never become underconsistent again. This is so because for a state to be underconsistent it needs to have its v-value strictly less than its g-value, which implies that the v-value needs to be finite. The only way for a v-value to change its value onto a finite value, on the other hand, is during the expansion of the state as an overconsistent state. However, if the state is expanded as overconsistent then according to Theorem 10 the state is never expanded again. Thus, a state can be expanded at most once as underconsistent. ■

Corollary 12 *No state is expanded more than twice within any particular execution of ComputePath function.*

Proof: According to theorems 10 and 11 each state can be expanded at most once as underconsistent and at most once as overconsistent. Since there are no other ways to expand states, this leads to the desired result: each state is expanded at most twice. ■

Theorem 13 *A state s is expanded by ComputePath only if either it was inconsistent right before ComputePath was called or its v-value was altered by ComputePath at some point during its current execution.*

Proof: Let us pick a state s such that right before a call to ComputePath function it was consistent and during the execution of ComputePath its v-value has never been altered. Then it means that $v_{afterComputePath}(s) = v_{beforeComputePath}(s) = g_{beforeComputePath}(s)$. Since only states from *OPEN* are selected for expansion and *OPEN* contains only inconsistent states, then in order for s to have been selected for expansion, it should have had $v(s) \neq g(s)$. Because the v-value of s remains the same throughout the ComputePath function execution, it has to be the case that the g-value of s has changed since the beginning of ComputePath. If s is expanded as overconsistent then

$v(s)$ is changed by setting it to $g(s)$, whereas if s is expanded as underconsistent then $v(s)$ is increased by setting it to ∞ (it could not have been equal to ∞ before since it was underconsistent, i.e., $v(s) < g(s) \leq \infty$). Both cases contradict to our assumption that $v(s)$ remained the same throughout the execution of ComputePath. ■

References

[1] S. Koenig and M. Likhachev. Incremental A*. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS) 14*. Cambridge, MA: MIT Press, 2002.

[2] M. Likhachev. *Search-based Planning for Large Dynamic Environments*. PhD thesis, Carnegie Mellon University, 2005. In Preparation.

[3] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.

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