

# A New Principle for Incremental Heuristic Search: Theoretical Results

**Sven Koenig**

Computer Science Department  
University of Southern California  
Los Angeles, CA 90089-0781  
skoening@usc.edu

**Maxim Likhachev**

Computer Science Department  
Carnegie Mellon University  
Pittsburgh, PA 15213-3891  
maxim+@cs.cmu.edu

## Abstract

Planning is often not a one-shot task because either the world or the agent's knowledge of the world changes. In this paper, we introduce a new principle that can be used to solve a series of similar search tasks faster with heuristic search methods than running individual searches in isolation, by updating the heuristics over time to make them more informed and thus future searches more focused. This principle is simple and easy to integrate into heuristic search methods, and it is easy to prove the correctness of the resulting heuristic search methods.

## Introduction

Planning is seldomly a one-shot task because either the world or the agent's knowledge of the world changes (Kott, Saks, & Mercer 1999), in which case the current plan might no longer apply or become very suboptimal. Examples include low-level trajectory planning and high-level evacuation planning in crisis situations. It is therefore not surprising that planning researchers have studied re-planning and plan re-use extensively, although most of this research was conducted in the 1970s. The question is how to solve a series of similar planning tasks faster than by solving the individual planning tasks in isolation, for example, by using experience with previous planning tasks to solve the current planning task. Recent developments have focused on re-planning with the heuristic search method A\* (= incremental heuristic search) (Koenig, Furcy, & Bauer 2002), which has two advantages: First, incremental versions of A\* are interesting for planning researchers because many state-of-the-art planners are now based on heuristic search (Bonet & Geffner 2001; Hoffmann 2000). Second, incremental versions of A\* are able to make guarantees about the solution quality after every re-planning episode, even after many re-planning episodes, whereas the solution quality of many other re-planning and plan re-use methods degrades quickly. The existing incremental versions of A\*, namely LPA\* (Koenig & Likhachev 2002b), D\* Lite (Koenig & Likhachev 2002a) and D\* (Stentz 1995), are all based on the same principle and have several disadvantages, namely

---

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

that it is difficult to prove them correct and that there are situations where they are slower than standard A\* and thus should not be used. In this paper, we introduce a new principle that can also be used to solve a series of similar search tasks faster with heuristic search methods than by running individual searches in isolation, but the idea behind our principle is to update the heuristics over time to make them more informed and thus future searches more focused. This principle is simple and easy to integrate into heuristic search methods, and it is easy to prove the correctness of the resulting heuristic search methods. We then show how our principle can be incorporated into A\*, resulting in Adaptive A\* that expands no more states than standard A\* and thus cannot be slower than standard A\* (except for a small number of bookkeeping actions). This paper is focused on theoretical results and complements our earlier paper on experimental results (Koenig & Likhachev 2005).

## New Principle for Incremental Search

Our new principle for incremental heuristic search is simple but powerful. Let  $gd[s]$  denote the goal distance of state  $s$  (= the cost of a cost-minimal path from state  $s$  to the goal state) and  $f^* = gd[s_{start}]$  denote the cost of the cost-minimal path found after an A\* search. Let  $s$  denote any state that was expanded during the A\* search,  $h[s]$  denote the consistent heuristic of this state  $s$  (= an estimate of its goal distance) used during the A\* search, and  $g[s]$  and  $f[s] = g[s] + h[s]$  denote its g-value and f-value, respectively, after the A\* search. Then,  $g[s]$  is the cost of a cost-minimal path from the start state to state  $s$  since state  $s$  was expanded during the A\* search. The costs of cost-minimal paths satisfy the following relationship

$$\begin{aligned} f^* &\leq g[s] + gd[s] \\ f^* - g[s] &\leq gd[s]. \end{aligned}$$

Thus,  $f^* - g[s]$  is an admissible estimate of the goal distance of state  $s$ . It can thus be used as new admissible heuristic of state  $s$ . We will show later that the new heuristics are not only admissible but also consistent. Furthermore, since state  $s$  was expanded during the A\* search, it holds that

$$\begin{aligned} f[s] &\leq f^* \\ g[s] + h[s] &\leq f^* \\ h[s] &\leq f^* - g[s]. \end{aligned}$$

Thus, the new heuristic  $f^* - g[s]$  of state  $s$  is no smaller than the old heuristic  $h[s]$  of state  $s$ . Consequently, the new heuristics dominate the old heuristics, and thus any A\* search with the new heuristics cannot expand more states than an otherwise identical A\* search with the old heuristics.

### Adaptive A\*

We now show how to incorporate these updates of the heuristics into A\*, resulting in Adaptive A\*. Adaptive A\* repeatedly finds cost-minimal paths for graph search problems with the same goal vertex on given graphs whose edge costs can increase between searches. It uses A\* searches (Nilsson 1971) to find the cost-minimal paths and updates the heuristics over time to make them more informed and thus future A\* searches more focused.

### Search Tasks and Notation

We use the following notation to describe search tasks:  $S$  denotes the finite set of states.  $A(s)$  denotes the finite set of actions that can be executed in state  $s \in S$ .  $c[s, a] > 0$  denotes the cost of executing action  $a \in A(s)$  in state  $s \in S$ , and  $succ(s, a) \in S$  denotes the resulting successor state.  $gd[s]$  denotes the goal distance of state  $s \in S$ , that is, the cost of a cost-minimal path from state  $s$  to the goal state  $s_{goal}$ . The start state  $s_{start}$  can change between searches, and the action costs can increase (but not decrease) between searches. The objective of each search is to find a cost-minimal path from the current start state to the goal state according to the current action costs. The user supplies initial estimates  $H(s)$  of the goal distances of the states  $s$  that must be consistent but can be the completely uninformed zero heuristics if more informed heuristics are not available.

### Eager Adaptive A\*

The eager version of Adaptive A\* uses our principle for incremental heuristic search in a straight-forward way by performing an A\* search and then updating the heuristics of all states  $s$  that were expanded during the A\* search (= states that are in the CLOSED list after the A\* search) by assigning  $h[s] := f^* - g[s]$ . It does not update the heuristics of the states that were generated during the A\* search but remained unexpanded (= states that are in the OPEN list after the A\* search) since their new heuristics cannot be larger than their old heuristics (but can be smaller). Future A\* searches cannot expand more states with the new heuristics than otherwise identical A\* searches with the old heuristics but will often expand many fewer states and thus run much faster, which justifies the small amount of runtime required for updating the heuristics of all expanded states after each A\* search. In that sense, Adaptive A\* is guaranteed to improve its runtime more and more over time and outperforms a standard version of A\* that does not modify the user-supplied heuristics.

### Illustration of Eager Adaptive A\*

We use a simple path-planning problem on a four-connected gridworld to demonstrate the advantage of the eager version

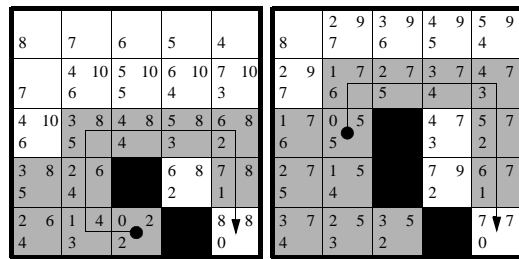


Figure 1: A\*

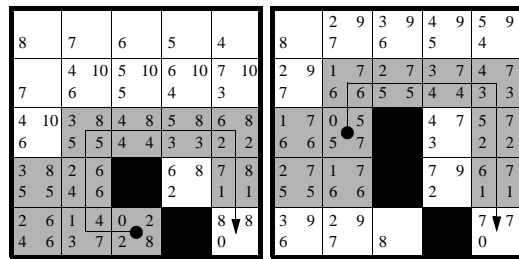


Figure 2: Eager Adaptive A\*

of Adaptive A\* over the standard version of A\*. An agent has to move from a given start cell to a given goal cell by repeatedly moving from its current cell to an unblocked (= white) adjacent cell in one of the four compass directions. Every move has unit cost for the agent. We use the Manhattan distances as user-supplied consistent heuristics. Figures 1 and 2 show the first path-planning problem on their left sides and a second path-planning problem on their right sides. The black circles are the start cells. The arrows show the planned paths from the start cells to the goal cell, which is in the south-east corner. Some action costs increase (but none decreases) from the first path-planning problem to the second one since one additional cell becomes blocked. Figure 1 shows the resulting searches of the standard version of A\*, and Figure 2 shows the resulting searches of the eager version of Adaptive A\*. Both search methods break ties between cells with the same f-values in favor of cells with larger g-values and remaining ties in the following order, from highest to lowest priority: east, south, west and north. All cells have their heuristic in the lower left corner. Generated cells also have their g-value in the upper left corner and their f-value in the upper right corner. Expanded cells are shown in grey. For the eager version of Adaptive A\*, expanded cells have their updated heuristic in the lower right corner. Note that the standard version of A\* re-expands the three leftmost cells in the bottom row because the heuristics are misleading and form a local minimum. The eager version of Adaptive A\* avoids these re-expansions since it updates the heuristics.

### Lazy Adaptive A\*

The eager version of Adaptive A\* first performs an A\* search and then updates the heuristics of the states that were expanded during the search. It has to update the heuris-

```

1  procedure InitializeState(s)
2  if search[s] ≠ counter AND search[s] ≠ 0
3    if g[s] + h[s] < pathcost[search[s]]
4      h[s] := pathcost[search[s]] - g[s];
5      g[s] := ∞;
6  search[s] := counter;
7  procedure ComputePath()
8  while g[sgoal] > mins' ∈ OPEN(g[s'] + h[s'])
9    delete the state s with the smallest priority g[s] + h[s] from OPEN;
10   for each a ∈ A(s)
11     InitializeState(succ(s, a));
12     if g[succ(s, a)] > g[s] + c[s, succ(s, a)]
13       g[succ(s, a)] := g[s] + c[s, succ(s, a)]
14       tree[succ(s, a)] := s;
15       if succ(s, a) is in OPEN then delete it from OPEN;
16       insert succ(s, a) into OPEN with priority g[succ(s, a)] + h[succ(s, a)];
17  procedure Main()
18  counter := 0;
19  for every state s ∈ S
20    h[s] := H(s);
21    search[s] := 0;
22    g[s] := ∞;
23  forever
24    counter := counter + 1;
25    InitializeState(sstart);
26    InitializeState(sgoal);
27    g[sstart] := 0;
28    OPEN := ∅;
29    insert sstart into OPEN with priority g[sstart] + h[sstart];
30    ComputePath();
31    pathcost[counter] := g[sgoal];
32    identify the path using the tree pointers and use it;
33    set sstart to the current start state (if changed);
34    update the increased edge costs (if any);

```

Figure 3: Lazy Adaptive A\*

tics after the search (rather than: during the search) because it needs to know the cost of the cost-minimal path found, which is only known at the end of the search. A disadvantage of updating the heuristics of all states that were expanded during the search is that one potentially updates the heuristics of states that are not needed in future searches. The lazy version of Adaptive A\* therefore remembers some information when a state  $s$  is expanded during the search (such as its g-value  $g[s]$ ) and some information when the search is over (such as the cost  $f^*$  of the cost-minimal path found), and then uses that information to compute the new heuristic  $h[s] := f^* - g[s]$  of state  $s$  when it is needed during future searches.

Figure 3 contains the pseudo code of the lazy version of Adaptive A\*. `ComputePath()` implements an A\* search to determine a cost-minimal path from the start state of the goal state. `counter` equals  $x$  during the  $x$ th invocation of `ComputePath()`, that is, the  $x$ th search. `pathcost[x]` is the cost  $f^*$  of the cost-minimal path found during the  $x$ th search. The value of `search[s]` is the number of the latest search during which state  $s$  was generated. Whenever a state  $s$  is about to be generated for the first time during some search on Line 11 and thus its heuristic is needed for the first time during this search, then `InitializeState(s)` checks whether new information is available to update the heuristic of the state. This is the case if the state was

generated during a previous search ( $search[s] \neq 0$ ) and then expanded during the same search ( $g[s] + h[s] < pathcost[search[s]]$ ) but not yet generated during the current search ( $search[s] \neq counter$ ). (An expanded state  $s$  with  $g[s] + h[s] = pathcost[search[s]]$  can be ignored since an update of its heuristic would leave the heuristic unchanged.) If so, then `InitializeState(s)` calculates the new heuristic of the state on Line 4 by subtracting the g-value of the state at the end of the search when the state was expanded ( $g[s]$ ) from the cost of the cost-minimal path found during the same search (`pathcost[search[s]]`). The behavior of the A\* searches for the lazy and eager versions of Adaptive A\* is always identical since they use the same heuristics. The only difference is when the heuristics are calculated.

### Properties of Adaptive A\*

We now prove two important properties of Adaptive A\*, making use of the following known properties of A\* searches with consistent heuristics (Pearl 1985): First, they expand every state at most once. Second, the g-values of every expanded state and the goal state are equal to the cost of a cost-minimal path from the start state to the respective state. Thus, one knows a cost-minimal path from the start state to all these states. Third, the f-values of the series of expanded states over time are monotonically nondecreasing. Thus,  $f[s] \leq f^*$  for all states  $s$  that were expanded during the A\* search (= states that are in the CLOSED list after the A\* search) and  $f^* \leq f[s]$  for all states  $s$  that were generated during the A\* search but remained unexpanded (= states that are in the OPEN list after the A\* search). The proofs refer to the eager version of Adaptive A\*.

**Theorem 1** *The heuristics of the same state are monotonically nondecreasing over time and thus indeed become more informed over time.*

*Proof:* Assume that the heuristic of state  $s$  is updated. Then, state  $s$  was expanded and it thus holds that  $f[s] \leq f^*$ . Thus,  $h[s] = f[s] - g[s] \leq f^* - g[s]$  and the update cannot decrease the heuristic of state  $s$  since it changes the heuristic from  $h[s]$  to  $f^* - g[s]$ . ■

**Theorem 2** *The heuristics indeed remain consistent and thus also admissible.*

*Proof:* We prove this property by induction on the number of A\* searches performed by Adaptive A\*. The initial heuristics are provided by the user and consistent. It thus holds that  $h[s_{goal}] = 0$ . This continues to hold since the goal state is not expanded and its heuristic thus not updated. It also holds that  $h[s] \leq h[succ(s, a)] + c[s, a]$  for all non-goal states  $s$  and actions  $a$  that can be executed in them. Assume that some action costs increase. Let  $c$  denote the action costs before all increases and  $c'$  denote the action costs after all increases. Then,  $h[s] \leq h[succ(s, a)] + c[s, a] \leq h[succ(s, a)] + c'[s, a]$  and the heuristics thus remain consistent. Now assume that the heuristics are updated. Let  $h$  denote the heuristics before all updates and  $h'$  denote the heuristics after all updates. We distinguish three cases:

- First, both  $s$  and  $succ(s, a)$  were expanded, which implies that  $h'[s] = f^* - g[s]$  and  $h'[succ(s, a)] = f^* -$

$g[succ(s, a)]$ . Also,  $g[succ(s, a)] \leq g[s] + c[s, a]$  since the A\* search discovers a path from the current state via state  $s$  to state  $succ(s, a)$  of cost  $g[s] + c[s, a]$  during the expansion of state  $s$ . Thus,  $h'[s] = f^* - g[s] \leq f^* - g[succ(s, a)] + c[s, a] = h'[succ(s, a)] + c[s, a]$ .

- Second,  $s$  was expanded but  $succ(s, a)$  was not, which implies that  $h'[s] = f^* - g[s]$  and  $h'[succ(s, a)] = h[succ(s, a)]$ . Also,  $g[succ(s, a)] \leq g[s] + c[s, a]$  for the same reason as in the first case, and  $f^* \leq f[succ(s, a)]$  since state  $succ(s, a)$  was generated but not expanded. Thus,  $h'[s] = f^* - g[s] \leq f[succ(s, a)] - g[s] = g[succ(s, a)] + h[succ(s, a)] - g[s] = g[succ(s, a)] + h'[succ(s, a)] - g[s] \leq g[succ(s, a)] + h'[succ(s, a)] - g[succ(s, a)] + c[s, a] = h'[succ(s, a)] + c[s, a]$ .
- Third,  $s$  was not expanded, which implies that  $h'[s] = h[s]$ . Also,  $h[succ(s, a)] \leq h[succ(s, a)]$  since the heuristics of the same state are monotonically nondecreasing over time. Thus,  $h'[s] = h[s] \leq h[succ(s, a)] + c[s, a] \leq h'[succ(s, a)] + c[s, a]$ .

Thus,  $h'[s] \leq h'[succ(s, a)] + c[s, a]$  in all three cases and the heuristics thus remain consistent. ■

These properties trivially imply the correctness of Adaptive A\*. The proof of the correctness of LPA\* in (Koenig, Likhachev, & Furcy 2004), on the other hand, is 18 pages long.

## Experimental Results

We performed experiments in randomly generated four-connected mazes of size  $201 \times 201$  that were solvable. We generated their random corridor structure with a depth-first search and then removed 750 walls. Adaptive A\* expanded more than 20 percent fewer cells than the standard version of A\* but needed about 10 percent more time per cell expansion. Overall, Adaptive A\* ran more than 10 percent faster than the standard version of A\*. Additional results are reported in (Koenig & Likhachev 2005). We have been able to achieve even larger improvements in different kinds of domains. Note, however, that all of these results are implementation, compiler, and computer dependent.

## Conclusions

In this paper, we introduced a new principle for incremental heuristic search, namely to update the heuristics over time to make them more informed and thus future searches more focused. This principle is simple and easy to integrate into heuristic search methods, and it is easy to prove the correctness of the resulting heuristic search methods. We demonstrated that our new principle for incremental heuristic search can be incorporated into the heuristic search method A\*, resulting in Adaptive A\*. In future work, we intend to explore additional applications of our principle. For example, a previous incremental heuristic search method (LPA\*) has already been applied to symbolic re-planning problems, resulting in the SHERPA planning system (Koenig, Furcy, & Bauer 2002). We intend to use Adaptive A\* in a similar way to solve symbolic re-planning problems.

## Acknowledgments

This research has been partly supported by an NSF award to Sven Koenig under contract IIS-0350584. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

## References

- Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.
- Hoffmann, J. 2000. FF: The fast-forward planning systems. *Artificial Intelligence Magazine* 22(3):57–62.
- Koenig, S., and Likhachev, M. 2002a. D\* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, 476–483.
- Koenig, S., and Likhachev, M. 2002b. Incremental A\*. In Dietterich, T.; Becker, S.; and Ghahramani, Z., eds., *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.
- Koenig, S., and Likhachev, M. 2005. Adaptive A\* [poster abstract]. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, 1311–1312.
- Koenig, S.; Furcy, D.; and Bauer, C. 2002. Heuristic search-based replanning. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling*, 294–301.
- Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong Planning A\*. *Artificial Intelligence* 155(1-2):93–146.
- Kott, A.; Saks, V.; and Mercer, A. 1999. A new technique enables dynamic replanning and rescheduling of aeromedical evacuation. *Artificial Intelligence Magazine* 20(1):43–53.
- Nilsson, N. 1971. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.
- Pearl, J. 1985. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.
- Stentz, A. 1995. The focussed D\* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1652–1659.