Path-Adaptive A* for Incremental Heuristic Search in Unknown Terrain

Carlos Hernández
Departamento de Ingeniería Informática
Universidad Católica de la Ssma. Concepción
Concepción, Chile
chernan@usc.edu

Pedro Meseguer
Institut d’Investigació en Intel·ligència Artificial, CSIC
Campus UAB, 08193 Bellaterra, Spain
pedro@iiia.csic.es

Xiaoxun Sun  Sven Koenig
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781, USA
{xiaoxuns,skoenig}@usc.edu

Abstract
Adaptive A* is an incremental version of A* that updates the h-values of the previous A* search to make them more informed and thus future A* searches more focused. In this paper, we show how the A* searches performed by Adaptive A* can reuse part of the path of the previous search and terminate before they expand a goal state, resulting in Path-Adaptive A*. We demonstrate experimentally that Path-Adaptive A* expands fewer states per search and runs faster than Adaptive A* when solving path-planning problems in initially unknown terrain.

Introduction
Consider agents that have to navigate from given start coordinates to given goal coordinates in initially unknown terrain and can sense obstacles only around themselves, such as computer-controlled characters in real-time computer games (Bulitko and Lee 2006) and robots (Stentz 1994). Planning with the freespace assumption is a popular approach to solve such tasks (Koenig, Tovey, and Smirnov 2003): The agent repeatedly finds and then follows a cost-minimal unblocked path from its current coordinates to the goal coordinates, taking into account the obstacles that it has sensed already but assuming that no additional obstacles are present. It repeats the process when it senses obstacles on its path while it follows the path. Thus, the agent has to search repeatedly. Incremental search can be used to speed up these similar searches. Adaptive A*, for example, is an incremental version of A* that updates the h-values of the previous A* search to make them more informed and thus future A* searches more focused (Koenig and Likhachev 2005). Our key observation is that part of the current cost-minimal path remains cost-minimal when the agent senses obstacles on the path, namely the obstacle-free suffix of the path. We show how the next A* search can reuse this part of the path and terminate before it expands a goal state, resulting in Path-Adaptive A*. We demonstrate experimentally that Path-Adaptive A* expands fewer states per search and runs faster than Adaptive A* when solving navigation problems in initially unknown terrain.

Notation
S is the finite set of states (vertices). A ⊂ S × S is the finite set of actions (edges). Executing action a = (s, s’) moves from state s ∈ S to state s’ ∈ S with cost c(s, s’) > 0. Succ(s) := {s’ ∈ S | (s, s’) ∈ A} is the set of successor states of state s ∈ S. Assume that we are given a cost-minimal path Path(s0, sn) = (s0, . . . , sn) from state s0 ∈ S to state sn ∈ S. Then, nextstate(s0) = s ∈ S is the state after state s on the path. (nextstate(sn) = null.) backstate(s1) = s0 is the state before state s on the path. (backstate(s) = null.) We also use the standard terminology and notation from A* (Pearl 1985). In addition, d(s, s’) denotes the cost of a cost-minimal path from state s ∈ S to state s’ ∈ S. H(s, s’) denotes the Manhattan distances. sstart denotes the initial cell of the agent (and is updated as the agent moves) and sgoal denotes its goal cell.

Adaptive A*
Adaptive A* (AA*) is an incremental version of A*, based on a principle first described in (Holte et al. 1996), that solves a series of searches in the same state space with
Path-Adaptive A*

Path-Adaptive A* (Path-AA*) applies AA* to solve navigation problems in initially unknown terrain using planning with the freespace assumption. The agent repeatedly finds minimal paths. AA* performs standard A* searches but updates the h-values after an A* search to make them more informed and thus future A* searches more focused. As a result, AA* would update the h-values of all other expanded states they expand and thus how fast they are. Our objective is to make them expand a state on the reusable path as quickly as possible. The first A* search of Path-AA* breaks ties in favor of larger g-values, which is known to be a good tie-breaking strategy for A*. The following A* searches break ties in favor of states s whose estimated distance \(ed(s)\) to the reusable path is smallest, as given by the user-provided H-values. Path-AA* initially sets \(p := r\) and \(p' := nextstate(p)\). It then computes the estimated distance \(ed(s) := \min(H(s, p), H(s, p'))\) and, if \(H(s, p) > H(s, p')\), advances \(p\) and \(p'\) by assigning \(p := p'\) and \(p' := nextstate(p)\), see Figure 2.

Figure 3 shows an example that illustrates the advantage of breaking ties using our optimization over breaking ties in favor of larger g-values. The agent started in cell C1 and then the A* search from state \(s'_{\text{start}}\) to state \(x\) and the part of the reusable path from state \(x\) to state \(s_{\text{goal}}\), see Figure 1. It then updates the h-values of all states \(s\) expanded during the A* search by assigning \(h(s) := f(x) - g(s)\).

Path-AA* is correct: Consider a state \(s\) that is on a cost-minimal path from \(s_{\text{start}}\) to \(s_{\text{goal}}\) found during a previous A* search. That A* search expanded state \(s\) and AA* thus updated its h-value to \(h(s) = f(s_{\text{goal}}) - g(s) = d(s_{\text{start}}, s_{\text{goal}}) - d(s_{\text{start}}, s) = d(s, s_{\text{goal}})\). Both states \(x\) and \(x'\) are on a cost-minimal path found during a previous A* search. Now assume that the current A* search from \(s'_{\text{start}}\) to \(s_{\text{goal}}\) is about to expand state \(x\) with f-value \(f(x)\). It holds that \(f(x) = g(x) + h(x) = g(x) + d(x, s_{\text{goal}}) = g(x) + d(x, x') + d(x', s_{\text{goal}}) = g(x') + h(x') = f(x')\). The sequence of f-values of the states expanded by an A* search with h-values that satisfy the triangle inequality is monotonically non-decreasing (Pearl 1985). Thus, the A* search can expand state \(x'\) next with f-value \(f(x') = f(x)\). By induction, it can expand all states on the reusable path from \(x\) to \(s_{\text{goal}}\) in sequence and would terminate when it is about to expand state \(s_{\text{goal}}\) with f-value \(f(s_{\text{goal}}) = f(x)\). AA* would then update the h-values of the expanded states \(s\) on the reusable path from \(x\) to \(s_{\text{goal}}\) by assigning \(h(s) := d(x, s_{\text{goal}})\) since these states are again on a cost-minimal path. This update would not change their h-values and thus could be omitted.

Optimization of Tie Breaking

Formally, we define the reusable path as follows: Assume that the agent follows Path\((s_{\text{start}}, s_{\text{goal}})\) and is at \(s'_{\text{start}}\) when it discovers that the cost of at least one action on the path increased while it follows the path, then it repeats the process. Our key observation is that part of its path remains a cost-minimal path, namely the suffix of the path without action cost changes. The A* searches of Path-AA* reuse this part of the path and thus terminate before they expand a goal state.

When the next A* search is about to expand a state \(x\) on the reusable path Path\((r, s_{\text{goal}})\) it terminates. Path-AA* updates the cost-minimal path by concatenating the path found by the A* search from state \(s'_{\text{start}}\) to state \(x\) and the part of the reusable path from state \(x\) to state \(s_{\text{goal}}\), see Figure 1. It then updates the h-values of all states \(s\) expanded during the A* search by assigning \(h(s) := f(x) - g(s)\).
moved to cell C2, the goal state is in cell C6 and the reusable path is shown as a dashed arrow. Every cell generated by the A* search has its g-value in the upper-left corner, h-value in the lower-left corner and f-value in the upper-right corner. Expanded cells are shaded. The path found by the A* search is shown as a solid arrow. When ties are broken using our optimization, every cell generated by the A* search has its ed-value in the lower right corner. p is set to cell C4 and never advanced. The A* search expands two cells fewer when breaking ties using our optimization.

**Pseudocode of Path-Adaptive A**

Figure 4 shows the pseudocode of Path-AA* without the optimization of tie breaking. This version of Path-AA* extends the lazy version of AA* (Sun, Koenig, and Yeo 2008), that updates the h-value of a state only when it is needed during a future A* search (Koenig and Likhachev 2006a). We use this version of Path-AA* since it ran faster in our experiments than the version of Path-AA* that extends the eager version of AA*, that updates the h-value of a state after the A* search that expanded the state (as described before). Procedure InitializeState updates the h-value of a state, and procedure ComputePath performs an A* search from $s_{start}$ to $s_{goal}$. We modified the pseudocode of AA* as follows: ComputePath now terminates when it expands $s_{goal}$ for a state $x$ on the reusable path $Path(r, x)$ (line 25). In the latter case, it first calls procedure CleanPath to remove $Path(r, x)$ from the cost-minimal path to yield $Path(x, s_{goal})$ (line 27) and then procedure MakePath to obtain the cost-minimal path $Path(s_{start}, x)$ found by the A* search (line 28). The new cost-minimal path $Path(s_{start}, s_{goal})$ is then the concatenation of $Path(s_{start}, x)$ and $Path(x, s_{goal})$. In procedure Main, the agent repeatedly moves from $s_{start}$ to $nextstate(s_{start})$ along the cost-minimal path (line 53). If the costs of one or more actions on the cost-minimal path increase, then the cost-minimal path is shortened (lines 55-57) and the procedure repeats.

**Experimental Evaluation**

We performed experiments in empty gridworlds in which we blocked randomly chosen cells (see Figure 5 left), acyclic mazes whose corridor structure was generated by depth-first search with random tie-breaking (see Figure 5 center), cyclic mazes that resulted from acyclic mazes in which we unblocked randomly chosen blocked cells and two maps adapted from World of Warcraft with 10,280 and 309,600 unblocked cells, respectively (see Figure 5 right) (Koenig and Sun 2008; Sun, Koenig, and Yeo 2008). We chose the start and goal cells randomly and averaged over 2,000 gridworlds of each kind.

Table 1 shows our results for AA*, Path-AA* (which breaks ties in favor of larger g-values), Path-AA*-opt (which breaks ties using our optimization) and D*Lite (Koenig and Likhachev 2002) on a LINUX PC with a Pentium Core Duo CPU. All runtimes are reported in milliseconds. Path-AA* was faster than AA* in every case (with respect to total search time per test case, search time per search episode, cell expansions per test case and cell expansions per search episode), and Path-AA*-opt was faster than Path-AA* (with respect to the same criteria). In quite a few cases, Path-AA* and Path-AA*-opt were even faster than D* Lite (Koenig and Likhachev 2002), an alternative state-of-the-art algorithm.
In this paper, we showed how the A* searches performed by Adaptive A* can reuse part of the path of the previous search and terminate before they expand a goal state, resulting in Path-Adaptive A*. We also developed a good strategy for breaking ties among states with the same f-value. Finally, we demonstrated experimentally that Path-Adaptive A* expands fewer states per search and runs faster than Adaptive A* when solving path-planning problems in initially unknown terrain. It is future work to combine Path-Adaptive A* with recent optimization techniques for Adaptive A* (Sun et al. 2009) to speed it up even further and to apply the ideas behind Path-Adaptive A* to real-time search algorithms, such as RTAA* (Koenig and Likhachev 2006b) and LSS-LRTA* (Koenig and Sun 2008) and LRTA*LS(k, d) (Hernandez and Meseguer 2008), to speed up real-time search.

**References**


---

**Table 1: Experimental results.**

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
<th>(f)</th>
<th>(g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>170.0</td>
<td>33.0</td>
<td>0.008</td>
<td>695.8</td>
<td>34.2</td>
<td>5110.6</td>
<td>198.2</td>
</tr>
<tr>
<td>170.0</td>
<td>33.0</td>
<td>0.009</td>
<td>1360.0</td>
<td>41.2</td>
<td>15520.0</td>
<td>131.1</td>
</tr>
<tr>
<td>174.0</td>
<td>34.0</td>
<td>0.007</td>
<td>1003.0</td>
<td>29.5</td>
<td>15510.0</td>
<td>8.2</td>
</tr>
<tr>
<td>175.0</td>
<td>34.0</td>
<td>0.031</td>
<td>4648.9</td>
<td>136.7</td>
<td>15510.0</td>
<td>12.3</td>
</tr>
<tr>
<td>179.0</td>
<td>34.0</td>
<td>0.031</td>
<td>4648.9</td>
<td>56.7</td>
<td>15510.0</td>
<td>0.011</td>
</tr>
<tr>
<td>360.0</td>
<td>150.0</td>
<td>0.014</td>
<td>48788.9</td>
<td>0.034</td>
<td>8648.0</td>
<td>238.0</td>
</tr>
<tr>
<td>864.0</td>
<td>0.007</td>
<td>386.0</td>
<td>390.0</td>
<td>29.5</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>864.0</td>
<td>0.007</td>
<td>386.0</td>
<td>390.0</td>
<td>29.5</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>1859.0</td>
<td>170.0</td>
<td>0.011</td>
<td>1712.0</td>
<td>294.0</td>
<td>0.011</td>
<td></td>
</tr>
</tbody>
</table>

(a) = agent moves until the goal is reached per test case; (b) = total search time per test case (in milliseconds); (c) = search episodes per test case; (d) = time per search episode (in milliseconds); (e) = total cell expansions per test case; (f) = cell expansions per search episode.

**Figure 5:** Gridworlds used in the experimental evaluation.