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@ucsc.cl

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.

*We would like to thank Nathan Sturtevant from the University of Alberta for making maps from World of Warcraft available to us. This material is based upon work supported by, or in part by, projects under contract TIN2006-15387-C03-01 and Fondecyt-Chile #11080063, the U.S. Army Research Laboratory and the U.S. Army Research Office under contract W911NF-08-1-0468 and by NSF under contract 0413196. 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.

Notation

S is the finite set of states (vertices). $A \subset S \times S$ is the finite set of actions (edges). Executing action $a = (s, s')$ moves from state $s \in S$ to state $s' \in S$ with cost $c(s, s') > 0$. $Succ(s) := \{s' \in S | (s, s') \in A\}$ is the set of successor states of state $s \in S$. Assume that we are given a cost-minimal path *Path*$(s_0, s_n) = (s_0, \ldots, s_n)$ from state $s_0 \in S$ to state $s_n \in S$. Then, *nextstate*$(s_i) = s_{i+1}$ is the state after state s_i on the path. (*nextstate*$(s_n) =$ *null*.) *backstate*$(s_i) = s_{i-1}$ is the state before state s_i on the path. (*backstate*$(s_0) =$ *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 $s \in S$ to state $s' \in S$. $H(s, s')$ denotes the user-provided H-value, a lower approximation of $d(s, s')$ that satisfies the triangle inequality.

For our application, the states correspond to the cells of a gridworld. The agent always senses whether the adjacent cells in the main four compass directions are blocked. The actions correspond to moving from a cell to an adjacent cell in the main four compass directions. The cost is infinity if at least one cell is known to be blocked and one otherwise. The h-values $H(s, s')$ are the Manhattan distances. s_{start} denotes the initial cell of the agent (and is updated as the agent moves) and s_{goal} 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

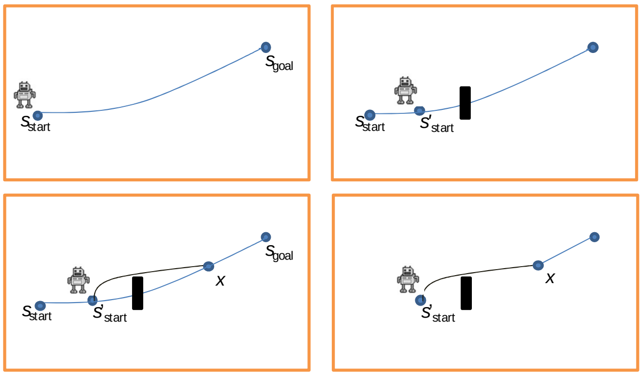

Figure 1: Top-left: the agent follows $Path(s_{start}, s_{goal})$; top-right: the agent is at s'_{start} when it discovers that $Path(s_{start}, s_{goal})$ is blocked; bottom-left: the agent performs an A* search until it expands state $x \in Path(s_{start}, s_{goal})$; bottom-right: the agent follows $Path(s'_{start}, s_{goal})$, the concatenation of $Path(s'_{start}, x)$ and $Path(x, s_{goal})$.

the same goal state potentially faster than A* (Koenig and Likhachev 2005). The start state can change and the costs of one or more actions can increase from search to search. AA* with the initial h-values $h(s) := H(s, s_{goal})$ finds cost-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. Assume, for example, that AA* performed an A* search from state s_{start} to state s_{goal}. It then updates the h-values of all states s expanded during the A* search (that is, the states in the closed list) by assigning $h(s) := f(s_{goal}) - g(s)$. The updated h-values again satisfy the triangle inequality.

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 and then follows a cost-minimal path from its current state to the goal state. If the agent senses 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.

Formally, we define the reusable path as follows: Assume that the agent follows $Path(s_{start}, s_{goal})$ and is at s'_{start} when it discovers that the cost of at least one action on the path increased. Assume that the cost of action (s, r) on the path increased but the costs of no actions on $Path(r, s_{goal})$ increased. Then, $Path(r, s_{goal})$ is the reusable path.

When the next A* search is about to expand a state x on the reusable path $Path(r, s_{goal})$ it terminates. Path-AA* updates the cost-minimal path by concatenating the path found by the A* search from state s'_{start} to state x and the part of the reusable path from state x to state s_{goal}, see Figure 1. It then updates the h-values of all states s expanded during the A* search by assigning $h(s) := f(x) - g(s)$.

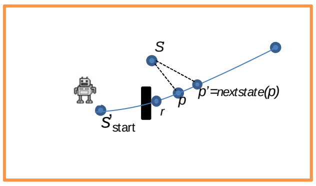

Figure 2: Optimized tie breaking.

Path-AA* is correct: Consider a state s that is on a cost-minimal path from s_{start} to s_{goal} found during a previous A* search. That A* search expanded state s and AA* thus updated its h-value to $h(s) = f(s_{goal}) - g(s) = d(s_{start}, s_{goal}) - d(s_{start}, s) = d(s, s_{goal})$. 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'_{start} to s_{goal} is about to expand state x with f-value $f(x)$. It holds that $f(x) = g(x) + h(x) = g(x) + d(x, s_{goal}) = g(x) + d(x, x') + d(x', s_{goal}) = g(x') + h(x') = f(x')$. 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_{goal} in sequence and would terminate when it is about to expand state s_{goal} with f-value $f(s_{goal}) = f(x)$. AA* would then update the h-values of the expanded states s on the reusable path from x to s_{goal} by assigning $h(s) := d(x, s_{goal})$ since these states are again on a cost-minimal path. This update would not change their h-values and thus could be omitted. AA* would update the h-values of all other expanded states s by assigning $h(s) := f(s_{goal}) - g(s) = f(x) - g(s)$, resulting in exactly the operations performed by Path-AA*.

Optimization of Tie Breaking

How the A* searches performed by Path-AA* break ties among states with the same f-value determines how many 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

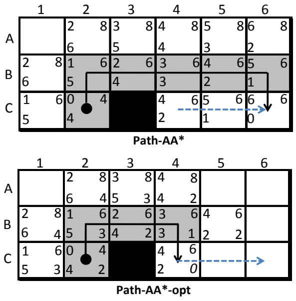

Figure 3: Tie breaking example.

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 Yeoh 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} or a state x on the reusable path *Path*(r, s_{goal}) (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.

```
1  procedure InitializeState(s)
2  if search(s) = 0
3      h(s) := H(s, s_goal);
4      g(s) := ∞;
5  else if search(s) ≠ counter
6      if g(s) + h(s) < pathcost(search(s))
7          h(s) := pathcost(search(s)) − g(s);
8      g(s) := ∞;
9  search(s) := counter;

10 procedure MakePath(s)
11 while s ≠ s_start
12     s_aux := s;
13     s := parent(s);
14     nextstate(s) := s_aux;
15     backstate(s_aux) := s;

16 procedure CleanPath(s)
17 while backstate(s) ≠ null
18     s_aux := backstate(s);
19     backstate(s) := null;
20     nextstate(s_aux) := null;
21     s := s_aux;

22 procedure ComputePath()
23 while OPEN ≠ ∅
24     delete a state s with the smallest f-value g(s) + h(s) from OPEN;
25     if s = s_goal or nextstate(s) ≠ null
26         pathcost(counter) := g(s) + h(s);
27         CleanPath(s);
28         MakePath(s);
29         return true;
30     for all s′ ∈ Succ(s)
31         InitializeState(s′);
32         if g(s′) > g(s) + c(s, a)
33             g(s′) := g(s) + c(s, s′);
34             parent(s′) := s;
35             if s′ is in OPEN then delete it from OPEN;
36             insert s′ into OPEN with f-value g(s′) + h(s′);
37 return false;

38 procedure Main()
39 counter := 1;
40 for all s ∈ S
41     search(s) := 0;
42     nextstate(s) := null;
43     backstate(s) := null;
44 while s_start ≠ s_goal
45     InitializeState(s_start);
46     InitializeState(s_goal);
47     g(s_start) := 0;
48     OPEN := ∅;
49     insert s_start into OPEN with f-value g(s_start) + h(s_start);
50     if ComputePath() = false
51         return "goal is not reachable";
52     while nextstate(s_start) ≠ null
53         s_start := nextstate(s_start);
54         update the increased action costs (if any);
55         for all increased action costs c(s, s′)
56             if backstate(s′) = s
57                 CleanPath(s′);
58     counter := counter + 1;
```

Figure 4: Path-Adaptive A*.

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 Yeoh 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.

	(a)	(b)	(c)	(d)	(e)	(f)	(a)	(b)	(c)	(d)	(e)	(f)
	200×200 Gridworlds with 20% Blocked Cells						200×200 Gridworlds with 40% Blocked Cells					
AA*	**170.0**	0.5	**33.0**	0.016	2,777.8	84.2	1,511.0	26.2	**386.0**	0.068	162,956.3	422.2
Path-AA*	**170.0**	0.3	**33.0**	0.009	1,360.0	41.2	1,512.0	13.1	**386.0**	0.034	88,788.9	230.0
Path-AA*-opt	174.0	**0.2**	34.0	**0.007**	**1,003.0**	**29.5**	**1,501.0**	**8.2**	390.0	**0.021**	58,383.0	149.7
D* Lite	175.0	1.1	34.0	0.031	4,648.9	136.7	1,515.0	12.3	396.0	0.031	**52,395.4**	**132.3**
	1000×1000 Gridworlds with 20% Blocked Cells						1000×1000 Gridworlds with 40% Blocked Cells					
AA*	**869.0**	25.2	**167.0**	0.151	66,938.5	400.8	12,799.0	1,969.8	3,294.0	0.598	7,386,199.9	2,242.3
Path-AA*	**869.0**	9.4	**167.0**	0.056	28,445.1	170.3	12,791.0	638.1	**3,289.0**	0.194	2,842,248.5	864.2
Path-AA*-opt	893.0	**5.8**	171.0	**0.034**	**18,592.0**	**108.7**	**12,705.0**	**256.9**	3,336.0	**0.077**	1,457,498.4	436.9
D* Lite	888.0	46.0	171.0	0.269	120,118.2	702.4	12,862.0	295.0	3,391.0	0.087	**873,869.4**	**257.7**
	151×151 Cyclic Mazes with 150 Blocked Cells Removed						151×151 Acyclic Mazes					
AA*	1735.0	16.3	678.0	0.024	112,435.9	165.8	5,904.0	86.1	1,832.0	0.047	594,910.9	324.7
Path-AA*	1738.0	6.8	680.0	0.010	50,725.1	74.6	5,916.0	29.4	1,835.0	0.016	221,338.3	120.6
Path-AA*-opt	1712.0	**4.0**	673.0	**0.006**	31,429.1	**46.7**	5,844.0	**18.2**	1,824.0	**0.010**	147,561.6	80.9
D* Lite	**1659.0**	6.7	**561.0**	0.012	**28,952.1**	51.6	**5,738.0**	19.7	**1,794.0**	0.011	**84,088.0**	**46.9**
	130×130 Game Maps						676×676 Game Maps					
AA*	**136.0**	1.2	**37.0**	0.033	7,337.1	198.3	**941.0**	51.1	**273.0**	0.187	208,528.8	763.8
Path-AA*	**136.0**	0.7	**37.0**	0.018	4,067.7	109.9	942.0	25.7	**273.0**	0.094	117,862.2	431.7
Path-AA*-opt	137.0	**0.6**	38.0	**0.017**	**3,765.8**	**99.1**	946.0	**23.2**	294.0	**0.079**	**107,986.2**	**367.3**
D* Lite	**136.0**	1.4	38.0	0.037	6,913.4	181.9	979.0	46.0	295.0	0.156	174,175.7	590.4

(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

Table 1: Experimental results.

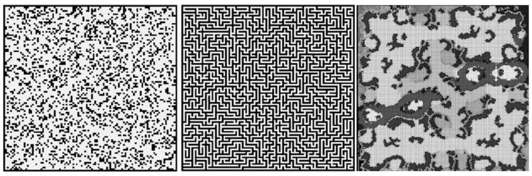

Figure 5: Gridworlds used in the experimental evaluation.

Conclusions and Future Work

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), LSS-LRTA* (Koenig and Sun 2008) and LRTA*LS(k, d) (Hernandez and Meseguer 2008), to speed up real-time search.

References

Bulitko, V., and Lee, G. 2006. Learning in real time search: A unifying framework. *Journal of Artificial Intelligence Research* 25:119–157.

Hernandez, C., and Meseguer, P. 2008. Combining lookahead and propagation in real-time heuristic search. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics*.

Holte, R.; Mkadmi, T.; Zimmer, R.; and MacDonald, A. 1996. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence* 85(1–2):321–361.

Koenig, S., and Likhachev, M. 2002. D* Lite. In *Proceedings of the AAAI Conference of Artificial Intelligence*, 476–483.

Koenig, S., and Likhachev, M. 2005. Adaptive A*. In *Proceedings of of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 1311–1312.

Koenig, S., and Likhachev, M. 2006a. A new principle for incremental heuristic search: Theoretical results [poster abstract]. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 402–405.

Koenig, S., and Likhachev, M. 2006b. Real-Time Adaptive A*. In *Proceedings of of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 281–288.

Koenig, S., and Sun, X. 2008. Comparing real-time and incremental heuristic search for real-time situated agents. *Journal of Autonomous Agents and Multi-Agent Systems* 18(3):313–341.

Koenig, S.; Tovey, C.; and Smirnov, Y. 2003. Performance bounds for planning in unknown terrain. *Artificial Intelligence* 147:253–279.

Pearl, J. 1985. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In *Proceedings of the International Conference on Robotics and Automation*, 3310–3317.

Sun, X.; Yeoh, W.; Chen, P.; and Koenig, S. 2009. Simple optimization techniques for A*-based search. In *Proceedings of of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 931–936.

Sun, X.; Koenig, S.; and Yeoh, W. 2008. Generalized Adaptive A*. In *Proceedings of of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 469–476.