Adaptive A*

Sven Koenig*
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
skoenig@usc.edu

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

ABSTRACT

Agents often have to perform repeated on-line searches as they gain additional knowledge about their environment. We describe an incremental version of A*, called Adaptive A*, that solves series of similar search problems faster than running A* repeatedly from scratch because it updates its heuristics between search episodes. It is simpler than other incremental versions of A* and thus likely easier to extend and adapt to new applications.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Graph and tree search strategies*

General Terms

Algorithms

Keywords

A*, D* Lite, Heuristic Search, Incremental Search, Planning with the Freespace Assumption

1. INTRODUCTION

Assume that one has to perform several A* searches with consistent heuristics in the same state space and with the same goal states but possibly different start states. Our idea is to make the heuristics more informed after each A* search in order to speed up future A* searches. Assume that s is a state that was expanded during such an A* search. One can obtain an admissible estimate of the goal distance of state s as follows: The distance from the start state to any goal state via state s is equal to the distance from the start state to state s plus the goal distance of state s. It is no smaller than the goal distance of the start state. Thus, the goal distance of state s is no smaller than the goal distance of the start state (= the f-value of the goal state that was about to be expanded when the A* search terminates) minus the distance from the start state to state s (= the g-value of state s when the A* search terminates). Consequently, this difference provides an admissible estimate of the goal distance of state s and can be calculated quickly. Our Adaptive A* obtains more informed heuristics by calculating and assigning this difference to each state that was expanded during the A* search and thus is in the closed list when the A* search terminates. (The states in the open list are not updated since the distance from the start state to these states can be smaller than their g-values when the A* search terminates.) Adaptive A* is simpler than other incremental versions of A*, such as D* Lite [1], and thus likely easier to extend and adapt to new applications.

*This research has been supported in part 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 NSF or the U.S. government.

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

2. ADAPTIVE A*

The task of Adaptive A* is to repeatedly find cost-minimal paths to a given set of goal states in a given state space with positive action costs. The searches can differ in their start states. Also, the action costs of an arbitrary number of actions can increase between searches by arbitrary amounts. Adaptive A* uses informed h-values to focus its searches. The initial h-values are provided by the user and must be consistent for the initial action costs. Adaptive A* updates its h-values after each search to make them more informed and focus its searches even better. An iteration of Adaptive A* proceeds as follows: It first updates the action costs, if necessary, to reflect any increases in action costs. It then picks a start state and runs a forward A* search to find a cost-minimal path from the start state to any state in the given set of goal states. Assume that the search determined that the cost of the cost-minimal path is g^*. Let $CLOSED$ be the set of states that were expanded during the search. Then, Adaptive A* executes $h[s] := g^* - g[s]$ for all states $s \in CLOSED$, where $g[s]$ is the g-value and $h[s]$ is the h-value of state s after the search. It then starts a new iteration. One can prove that the h-values of the same state are monotonically nondecreasing over time and thus indeed become more informed. One can also prove that the h-values remain consistent and Adaptive A* thus continues to find cost-minimal paths over time without having to re-expand states during the same search.

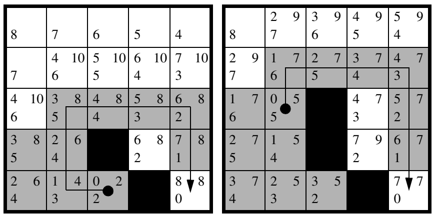

Figure 1: Forward A* Searches

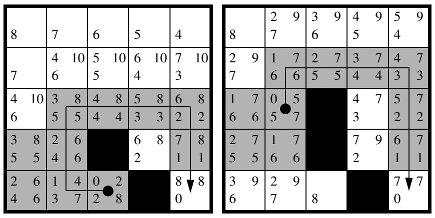

Figure 2: Adaptive A*

3. AN APPLICATION

Consider characters in real-time computer games such as Total Annihilation or Warcraft. These real-time situated agents often do not know the terrain in advance but automatically observe it within a certain range around them and then remember it for future use. To make the agents easy to control, the users can click on some position in known or unknown terrain and the agents then move autonomously to this position. We discretize the terrain into cells that are either blocked or unblocked and assume for simplicity that the agents can only move in the four main compass directions with unit action costs and thus operate on four-connected grids. As heuristic estimate of the distance of two cells we use the consistent Manhattan distance. The agents initially do not know which cells are blocked. They always know which (unblocked) cells they are in, sense the blockage status of their four neighboring cells, and can then move to any one of the unblocked neighboring cells. Their task is to move to a given goal cell. Our agents use planning with the freespace assumption [2] to solve the task. They find and then follow a cost-minimal presumed unblocked path from their current cell to the given goal cell, where a presumed unblocked path is one that does not pass through cells that are known to be blocked. Whenever the agents observe additional blocked cells during execution, they add them to their map. If such cells block their current path, they find and follow another cost-minimal presumed unblocked path from their current cell to the given goal cell, and repeat the process until they either reach the given goal cell or all paths to it are blocked. Figure 1 shows the resulting A* searches and Figure 2 shows the resulting Adaptive A* searches for a simple navigation example. The black circle is the agent. Black cells have been observed as blocked. The arrows show the planned paths from the current cell of the agent to its goal cell, which is in the lower right corner. All 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: right, down, left and up. All cells have their h-value 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 Adaptive A*, expanded cells have their updated h-values in the lower right corner. Note that A* re-expands the three cells in the bottom row because the h-values are misleading. Adaptive A* avoids these re-expansions since it updates the h-values.

Expanded Cells μ	(σ of μ)	Planning Time (μsec) μ	(σ of μ)
A* with Binary Heap			
53,084.27	(1,229.42)	16,514.75	(357.13)
Adaptive A* with Binary Heap			
41,593.55	(735.96)	14,355.81	(262.40)
D* Lite with Binary Heap			
11,416.37	(116.94)	8,505.72	(88.01)
Adaptive A* with Buckets			
41,063.69	(541.11)	7,051.12	(96.86)

Table 1: Experiments in Random Mazes

4. EXPERIMENTAL RESULTS

We performed experiments in randomly generated four-connected mazes of size 201×201 that were solvable. We generated their random corridor structure with a depth-first search and then removed 750 walls. We averaged over 5000 random mazes with randomly chosen start and goal cells on a SUN workstation with an AMD Opteron Processor 150 with 1 GByte of RAM. We compared A* against two incremental versions of A*, namely Adaptive A* and the optimized final version of D* Lite as published in [1]. All three search methods used standard binary heaps as priority queues and broke ties between cells with the same f-values in favor of cells with larger g-values and remaining ties randomly. They are guaranteed to determine the same paths (module tie breaking), which are on average 931.64 moves long. Table 1 shows both the average number of expanded cells and the average planning time until the agent reached the goal cell as well as the standard deviation of the mean (to show statistical significance). Adaptive A* expanded more than 20 percent fewer cells than A* but needed about 10 percent more time per cell expansion. Overall, Adaptive A* ran more than 10 percent faster than A*. (Note, however, that the planning times of all three search methods are implementation, compiler, and computer dependent.) D* Lite was even faster than Adaptive A* despite its large time per cell expansion. However, Adaptive A* can be made more efficient. For example, it can update the h-values incrementally when it encounters them during a future search rather than all at once after the current search. This modification can speed it up if it expands a large number of states during its current A* search that it is not going to encounter during future A* searches but does not make it faster in our mazes. Adaptive A* can also incorporate most of the methods used to speed up A*. For example, it can use buckets as priority queue instead of a binary heap. Buckets speed up Adaptive A* substantially in our mazes (more than A* since A* is more susceptible to the loss of secondary tie breaking than Adaptive A*) and makes it faster than D* Lite, that currently cannot be sped up in the same way. Of course, the results might be different in different domains.

5. REFERENCES

[1] S. Koenig and M. Likhachev. D* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483, 2002.

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