Assignment 1 - Fast Trajectory Replanning

Pritish Sahu - Serena De Stefani

Introduction to AI - CS 520 - 10/14/2015

In this assignment we implement and compare the algorithms Repeated Forward A*, Repeated Backward A* and Adaptive A*.

1 Understanding the method

1.1 a)

Question. Explain in your report why the first move of the agent for the example search problem from Figure 8 is to the east rather than the north given that the agent does not know initially which cells are blocked.

The first move of the agent is to the east (cell E3 on Figure 8 in the assignment) because the first step of A* is to proceed along the "shortest presumed unblocked path". The agent "knows" that the cell E3 is unblocked, because it is included in the original knowledge about the environment available to the agent.

1.2 b)

Question. Give a convincing argument that the agent in finite gridworlds indeed either reaches the target or discovers that this is impossible in finite time. Prove that the number of moves of the agent until it reaches the target or discovers that this is impossible is bounded from above by the number of unblocked cells squared.

Since the agent is in a finite gridworld, *in the worst case scenario*, it will be able visit all the unblocked nodes (unless they are not reachable because surrounded by blocked nodes; in this case, they would be same as blocked). Once the agent has visited all the nodes, either the target has been visited and it has been added to the open list (therefore the path has been found) or the target has not been visited and the open list is empty (because all the nodes have been visited and sorted by they respective f-values). Once the open list is empty, the algorithm stops and there is no way to proceed further, therefore the target cannot be found.

But in fact, there is no actual need for the agent to visit each single cell to reach this conclusion. Let's assume the target is completely surrounded, on all sides, by blocked cells. The agent will get as close as possible. Once he finds the most direct path blocked, the agent will explore all the other possible paths drawing a "circle" around the blocked target. The agent won't move far away to

explore all the surrounding unblocked cells, because the agent will always only pick the cells for which $f(s) \leq f(goal) = g(goal)$.

Let's now consider the cells for which $g(goal)=f(goal) \geq f(s)$.

They will not been expanded.

Once all the cells for which this condition holds have been checked, the agent will stop, even if it hasn't visited all the cells. If the target is still not in the open list, the agent discovers that it is not possible to find it in finite time.

The number of moves is bounded by the number of expanded cells, and this in turn will be bounded by:
(max number of cells expanded) X (max number of cells in a single expansion). In the worst case scenario, given a certain number of unblocked nodes n in a certain disposition, the algorithm will have to expand each of the n nodes each time (we need to remember that A* search expands all states at most *once each*. Therefore, in the worst case scenario, the numbers above will be both n, expanding n nodes n times, such that we have n squared maximum moves.

2 The effect of Ties

Note. For the scope of this analysis and of all the following ones, we built a gridworld composed of fifty "101 by 101" grids. The grids were generated randomly with a Deep Search First algorithm; each cell had 70% probability to be unblocked and 30% probability to be blocked. In each grid, we placed the start point at the upper left corner (node 1,1) and the target in the middle of the grid (node 50, 50). We considered only grids where the path from start to target was unblocked. The fifty grids were saved in text files that the program could access during the test implementation; this way each trial was performed in the same gridworld.

Question. Implement and compare both versions of Repeated Forward A* with respect to their runtime or, equivalently, number of expanded cells. Explain your observations in detail, that is, explain what you observed and give a reason for the observation.

We implemented both version of Repeated Forward A* with respect to both their number of expanded cells (the most direct indicator of performance) and their runtime in seconds (**Figure 1** and **Table 1**). If we compare the two pathfinder algorithms by the number of expanded nodes, we see that the version of Repeated Forward A* breaking ties with lower G-values is faster, since its mean performance in the fifty 101 X 101 gridworld is of 207.98 nodes expanded per search, while Repeated Forward A* breaking ties with higher G-values has a mean performance of 253.48 node expanded per search. The former algorithm is

also more reliable, since its performance has less variance (sd= 52.94) compared to the latter one (sd = 70.89). Comparing performances as the mean number of seconds necessary to reach the target yields similar results.

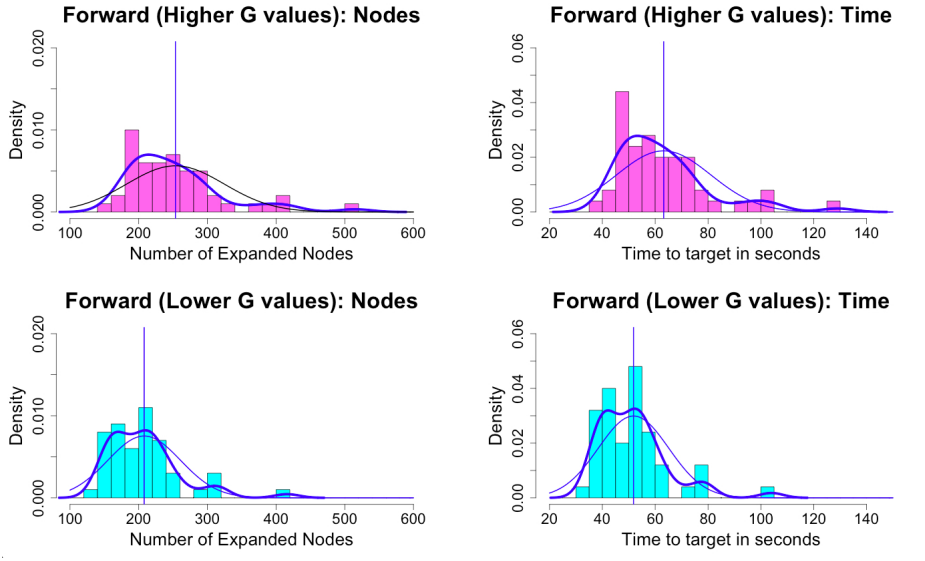

Fig. 1. Frequency histogram comparing Forward A* breaking ties at higher and lower G-values. The frequencies on the y-axis are computed as probabilities. The vertical line marks the meanpoint. The area delimited by the blue line represents the density distribution. The area delimited by the black line represent a normalized version of the density distribution

3 Forward vs. Backward

Question. Implement and compare Repeated Forward A* and Repeated Backward A* with respect to their runtime or, equivalently, number of expanded cells.

We implemented Repeated Forward A* and and Repeated Backward A* with respect to both their number of expanded cells and their runtime in seconds (**Figure 2** and **Table 1**), considering both algorithms breaking ties at higher G-values. If we compare the two pathfinder algorithms by the number of expanded nodes, we see that Repeated Backward A* is faster, since its mean performance is of 233.22 nodes expanded to complete the search, while Repeated Forward A* has a mean performance of 253.48 nodes expanded. The former algorithm is also more reliable, since its performance has less variance (sd= 51.80) compared to the latter one (sd = 70.89). If we compare performances as the mean number of seconds necessary to reach the target we obtain equivalent results.

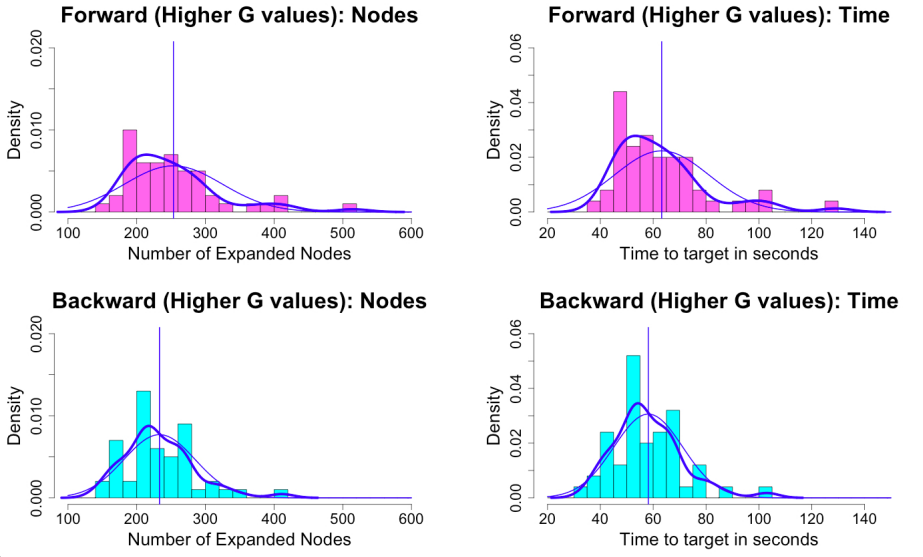

Fig. 2. Frequency histogram comparing Forward A* and Backward A* breaking ties at higher G-values. The frequencies on the y-axis are computed as probabilities. The vertical line marks the meanpoint. The area delimited by the blue line represents the density distribution. The area delimited by the black line represent a normalized version of the density distribution

4 Heuristic in the Adaptive A*

Question. The project argues that the Manhattan distances are consistent in gridworlds in which the agent can move only in the four maijn compass directions. Prove that this is indeed the case.

The Manhattan distances are consistent because they are computed summing up the shortest possible horizontal and vertical distances. This means that the Manhattan distances between two points will always be the same in any combinations of horizontal and vertical steps. Since the agent can move only in the four main compass directions, and cannot move in diagonal,this heuristic can never overestimate the cost of reaching the target. In the case in which the agent can move in diagonal, the Manhattan distance would not be consistent anymore because, if the target is reachable diagonally, the heuristic would overestimate the target.

Question. Furthermore, it is argued that "The h-values $h_{new}(s)$... are not only admissible but also consistent." Prove that adaptive A* leaves initially consistent h-values consistent even if action costs can increase.

We are still considering a gridworld in which the agent can move only in the four main compass directions. First I will prove the h-values $h_{new}(s)$ are consistent, then that they remain consistent even if action costs can increase.

H-values $h_{new}(n)$ are consistent.

Concerning notation, we consider $h(s)$ as values that follow the Manhattan Heuristic and $h_{new}(s)$ as values calculated by the difference between *g(goal)* and *g(s)*. Furthermore, n will will be a cell and n' will be the successor cell. The cost to move from n to n' — *c(n, a, n')* — will be one.

First let's consider the triangle inequality: $h(n) \leq$h(n') + *c(n, a, n')*.

The fact that this inequality holds for $h(s)$ tells us that those h-values are consistent. Now we want to prove that $h_{new}(n) \leq$h$_{new}(n')$ + *c(n, a, n')*.To do so we substitute $h_{new}(n)$ = *g(goal)-g(n)* and $h_{new}(n')$ = *g(goal)-g(n')*. So we have *g(goal) - g(n)* ≤ *g(goal)-g(n')+c(n,a,n')*.

We can eliminate *g(goal)* from both sides, then reverse the inequality changing the sign of all terms, obtaining: *g(n)* ≥ *g(n')* − *c(n,a,n')*.

It is easy to see that if the agent can move only in the four main compass directions this equation is always true. *c(n,a,n')* is one. If *g(n')* is smaller than *g(n)*, then subtracting one from it will make it even smaller. *g(n')* is greater than *g(n)*, then subtracting one from it will make them equal (because their distance is always one). Therefore the triangle inequality is satisfied and the $h_{new}(s)$ values are consistent.

H-values $h_{new}(n)$ remain consistent even if action costs can increase.

Let's consider again the triangle inequality: $h_{new}(n) \leq$h$_{new}(n')$ + *c(n, a, n')*. Let's assume that there is an action cost increase, where

c: action cost before increase

c': action cost after increase

Then: $h_{new}(n) \leq$h$_{new}(n')$ + *c(n, a, n')* ≤h$_{new}(n')$ + *c'(n, a, n')*. Therefore the heuristic remains consistent.

5 Heuristic in the Adaptive A*

Question. Implement and compare Repeated Forward A* and Adaptive A* with respect to their runtime. Explain your observations in detail, that is, explain what you observed and give a reason for the observation.

We implemented Repeated Forward A* and and Adaptive A* with respect to both their number of expanded cells and their runtime in seconds (**Figure 3** and **Table 1**), considering both algorithms breaking ties at higher G-values. If we compare the two pathfinder algorithms by the number of expanded nodes, we see that Adaptive A* is faster, since its mean performance is of 221.30 nodes expanded to complete the search, while Repeated Forward A* has a mean performance of 253.48 nodes expanded. The former algorithm is also more reliable, since its performance has less variance (sd= 51.28) compared to the latter one (sd = 70.89). Comparing performances as mean time necessary to reach the target provides results that are equivalent.

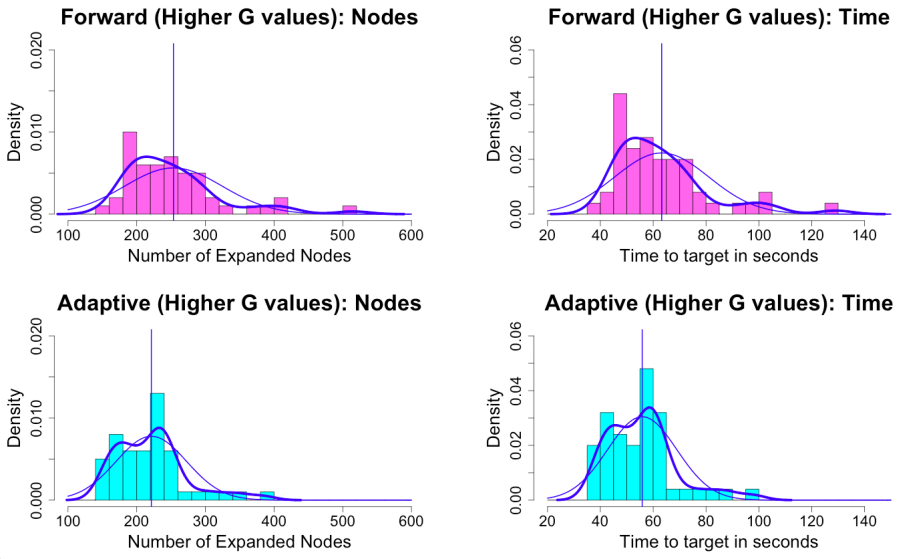

Fig. 3. Frequency histogram comparing Forward A* and Adaptive A* breaking ties at higher G-values. The frequencies on the y-axis are computed as probabilities.The vertical line marks the meanpoint. The area delimited by the blue line represents the density distribution. The area delimited by the black line represent a normalized version of the density distribution

Table 1. Performances for the four algorithms considered are compared

	Forward HighG	Forward LowG	Backward HighG	Adaptive HighG
Mean (Nodes expanded)	253.48	207.98	233.22	221.30
SD (Nodes expanded)	70.89	52.94	51.80	51.28
Mean (seconds)	63.22	51.80	58.14	55.84
SD (seconds)	17.84	13.33	13.02	13.09

6 Memory Issues

Question. Suggest additional ways to reduce the memory consumption of your implementations further.

Our current implementation uses a parent pointer (4 bytes) for each node, keeping position for x and y (8 bytes total). In addition we store the G-score (4 bytes)

and the H-score (4 bytes) for each node. Finally we use the boolean "visited" with memory of one byte to keep track of the nodes already seen by the agent. Therefore, for each node we are using a total of 21 bytes.
To optimize memory use, we can start allocating only two bits (approximately one byte) to the parent pointer, using a boolean system where for example 00 indicates the node to the right, 01 the node to the left, 11 the node to the south and 10 the node to the north of the designated node.
In addition, we can stop storing H-scores (recalculating them each time instead). Regarding the G-scores, we can dynamically allocate memory based on the requirements. That means we can avoid storing the G-scores for every single cell, considering instead only the cells that have been expanded. In our tests runs we saw that the number of expanded cells in each grid never exceeded 5% of the total numbers of cells in the grid. In the end we will allocate only two bytes for each node (one for the pointer, rounding off the two bits to one byte, and one for the boolean "visited") instead of 21, and we will also allocate four bytes to G-scores only in 5% of all cells.

Question. Calculate the amount of memory that they need to operate on gridworlds of size 1001 X 1001 and the largest gridworld that they can operate on within a memory limit of 4 MBytes.

4 MBytes is equivalent to 4,194,304 bytes. As we saw, we have 95% of the nodes that needs 2 bytes of memory allocated, and 5% of nodes (those that we expand) that require 6 bytes of memory. We need to solve the following equation:

$$2(95/100)x + 6(5/100)x = 4MBytes = 4,194,304Bytes \quad (1)$$

Solving for x we obtain 1,999,287 nodes; taking the square root we obtain the side of the grid: 1413 nodes. That means that 4MBytes of memory support a grid of approximately 1400 X 1400 nodes.

References

Koenig and Likhachev, Adaptive A* [Poster Abstract], Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 1311-1312, 2005