§CS 520: Assignment 1
Fast Trajectory Replanning

Xiaoyang Xie
167008240

Yikun Xian
168000142

Department of Computer Science
Rutgers University, New Brunswick, NJ

09 October 2015

Abstract

Heuristic search algorithms like A* can be adapted to solving path planning problems of directed goal and unknown environment. In this report, we mainly discuss and evaluate three variants of A* algorithms, namely Repeated Forward A*, Repeated Backward A* and Adaptive A*. In the experiment, we first generate two sets of 50 random grid-based maps, where one follows the assignment requirement containing approximate 30% obstacles and 70% roads, and the other is the set of corridor-like mazes generated by DFS with randomly expanded nodes. Then we compare three algorithms by such evaluation indexes as number of expanded nodes, explored nodes, moves, optimal moves, etc. The result shows that 1) original Repeated Forward A* expands states 30 times more than the modified one with $c \times f - g$; 2) Repeated Backward A* expands 15 times more states than Repeated Forward A*; 3) Adaptive A* expands 22% less states than Repeated Forward A* only in complicated mazes. Finally, we discuss and calculate how to optimize data structures to store states as many as possible within only 4M memory. This is the practical problem in the situation where computational resources are rare and precious.

Part 0 Setup Environment

We simulate all path-finding processes based on the framework of GridWorld[1], an AP case study project from CollegeBoard [1]. It provides graphical user interface based on Java AWT where visual objects can interact and perform customized actions in a two-dimensional grid map. In the next part, we will first illustrate original GridWorld framework and our enhancement of displaying colored path.

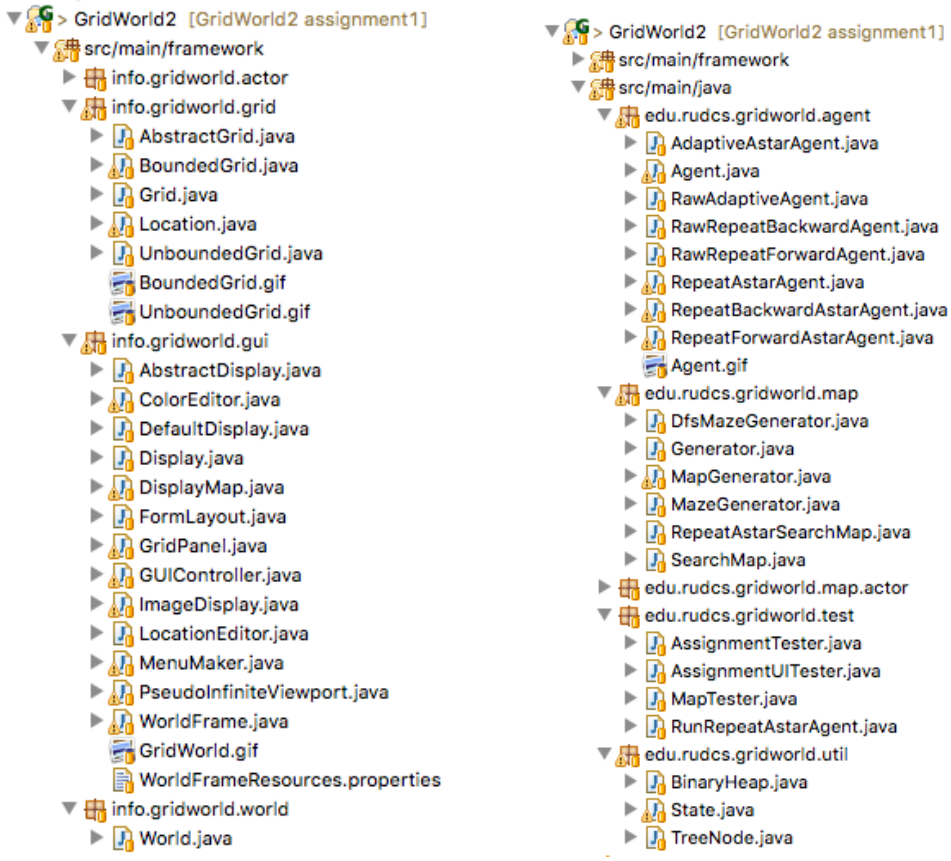
0.1 GridWorld Architecture and Modification

The source code of original GridWorld project is placed in $src/main/framework$ folder and its structure can be divided into four parts, as shown in Figure 1a.

The *actor* package contains objects whose behavior on the map can be arbitrarily defined by rewriting *act* method in each inherited class:

```
@Override
public void act()
```

[1] https://www.collegeboard.org/



(a) GridWorld structure (b) Project structure

Figure 1: Structure of GridWorld framework and path-finding project

The *grid* package defines features on bounded and unbounded grid map as well as connections between map and actors. The *gui* package encapsulates low-level Java AWT to provide APIs for visualization and interactions of map and actors. The *world* package provides high-level integration of actors and world.

In order to visualize the presumed unblocked path and the set of nodes expanded and explored after each planning, we enhance the *GridPanel* class in the GUI package by implementing the method below:

```
private void drawColoredLocations(Graphics2D g2)
```

Meanwhile, following five abstract methods related to colored grid are added to *Grid* interface and classes like *BoundedGrid* and *UnboundedGrid* are responsible to implement color configuration on each grid.

```
ArrayList<Location> getColoredLocations();
```

```
Color getColor(Location loc);
void putColor(Location loc, Color color);
void removeColor(Location loc);
void resetColors();
```

The source code related to assignment is placed in *src/main/java* folder, as shown in Figure 1b, which are divided into five packages. The *agent* package defines the simulated agents equipped with specific navigation algorithm for solving goal-directed path-finding problem. The *map* package provides APIs for generation of random map and corridor-like maze. The *map.actor* package defines some static object on the map like obstacles and goal. The *test* package contains simulation program for experiment and testing. The *util* package includes some self-defined data structures to be used in storing intermediate result of algorithm.

0.2 Data Structures

In the project, we implement some useful data structures from scratch, like binary heap and tree, by referring to CLRS[2]. The detailed interfaces provided by these data structures are listed in Figure 2.

0.3 How to Run

Our project is built and packed by Gradle[2], an open-source build automation tool. To run the project, you should first install and setup Gradle environment on your computer, and install Gradle plugin for Eclipse, called "Gradle Integration for Eclipse". Then, download complete source code of the project, which is attached in Sakai or available on the Github (`https://github.com/orcax/GridWorld2`). Finally, inside Eclipse, click *File* $\rightarrow$ *Import* $\rightarrow$ *Gradle* $\rightarrow$ *Gradle Project* to import project. All runnable programs are placed in *test* package.

0.4 Maze Generation Algorithm

In the experiment, we mainly use two kinds of grid map, namely random map with discrete obstacles and random maze with consecutive obstacles. The random map is generated according to the assignment requirement, so for each map, there are about 30% obstacles and 70% roads. The start position and the goal position can be placed either randomly or on diagonal corners. Difference between these

[2]http://gradle.org/

```
protected void buildHeap(int length){
protected void sort(){
public void add(T data){
protected int left(int i){
protected int right(int i){
private void heapify(int i){
private void heapify(int i, int size){
public T top(){
public T extractTop(){
private void swap(int i , int j){
public boolean contains(T data){
public boolean remove(T data){
public boolean update(T data){
public void clear(){
public boolean isEmpty(){
public T peek(){
public T poll(){
```

(a) Interfaces for Binary Heap

```
public List<TreeNode<T>> getChildren() {
public TreeNode<T> getChild(T data) {
public void addChild(TreeNode<T> node) {
public void removeChild(TreeNode<T> node) {
public TreeNode<T> getParent() {
public void setParent(TreeNode<T> node) {
public T getData() {
public void setData(T data) {
public int getDepth(T data) {
private int depth(T data, int d) {
```

(b) Interfaces for Tree

Figure 2: Interfaces of Data Structures

two ways lies in the absolute distance between start and goal, although, the experimental result implies that this initial distance in this type of random map has nearly no influence on the performance of three algorithms. So in this case, we simply collect maps that exist a path between the start position and goal position.

Random maze is the other type of map tested in the experiment. It is generated according to DFS by expanding neighbors randomly. The generation process is described as Algorithm 1.

In this algorithm, the map is initialized as two-dimensional array and set all cells as obstacles (WALL). Roads will be expanded randomly and paved only on the cells of odd index. To generate roads, we first begin with randomly picking a cell on the map and set it as road. Then, randomly select one of its non-road neighboring nodes (of odd index), set it and the node between them as roads. Repeat this procedure until there is no valid neighboring nodes. If this happens, traceback to its parent node and do the same thing again. At last, all nodes of odd index are

Algorithm 1 Maze Generation Algorithm by DFS

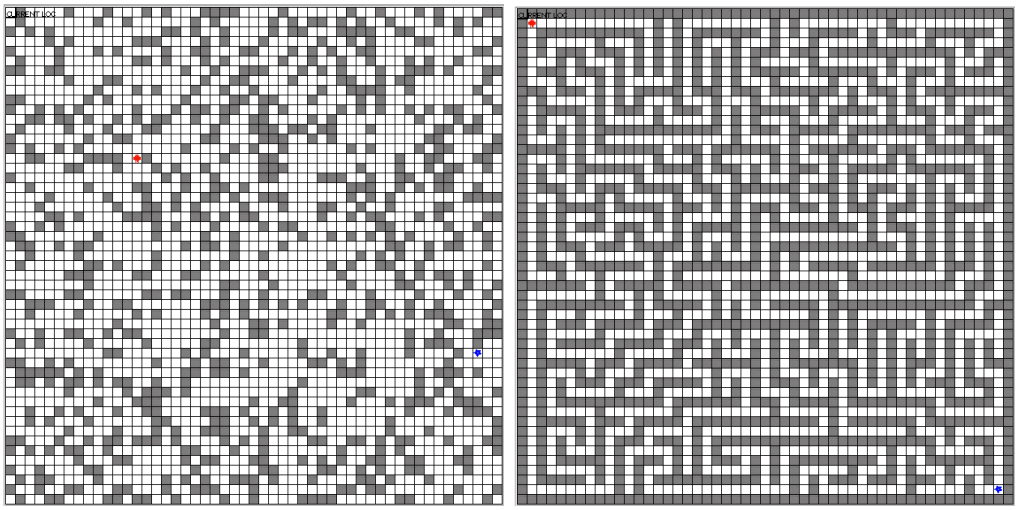
```
1: function GENERATE-BY-DFS(map)
2:     set all cells in map as WALL
3:     r = random(⌊map.rows/2⌋) * 2 + 1
4:     c = random(⌊map.cols/2⌋) * 2 + 1
5:     map[r][c] = ROAD
6:     call RANDOM-DFS(map, r, c)
7:     map[1][1] = START
8:     map[map.rows − 2][map.cols − 2] = GOAL

1: function RANDOM-DFS(map, row, col)
2:     (row′, col′)=RANDOM-NEIGHBOR(map, row, col)
3:     while (row′, col′) ≠ NULL do
4:         map[row′][col′] = ROAD
5:         map[(row + row′)/2][(col + col′)/2] = ROAD
6:         call RANDOM-DFS(map, row′, col′)
7:         (row′, col′)=RANDOM-NEIGHBOR(map, row, col)

1: function RANDOM-NEIGHBOR(map, row, col)
2:     initialize neighbors as empty list
3:     if row − 2 > 0 and map[row − 2][col] ≠ ROAD then
4:         neighbors.add((row − 2, col))
5:     if row + 2 < map.rows and map[row + 2][col] ≠ ROAD then
6:         neighbors.add((row + 2, col))
7:     if col − 2 > 0 and map[row][col − 2] ≠ ROAD then
8:         neighbors.add((row, col − 2))
9:     if col + 2 < map.cols and map[row][col + 2] ≠ ROAD then
10:        neighbors.add((row, col + 2))
       return neighbors[random(neighbors.size)]
```

traversed and some nodes of even index are set as road to connect two adjacent nodes. Finally, without loss of generality, we simply set top-left odd bottom-right respectively as start position and goal position.

It is easy to prove that there is only one depth first tree generated during one complete search. The connectivity of single depth first tree make the maze always reachable, namely there always exists one path from start to goal. Example of two kinds of map of size 50*50 is shown in Figure 3.



(a) Map with discrete obstacles (b) Corridor-like maze

Figure 3: Example of random map and random maze

Part 1 Understanding the Methods

1.1 Problem A

Problem Explain in your report why the first move of the agent for the example search problem from Figure 4 is the east rather than then north given that then agent does not know initially which cells are blocked.

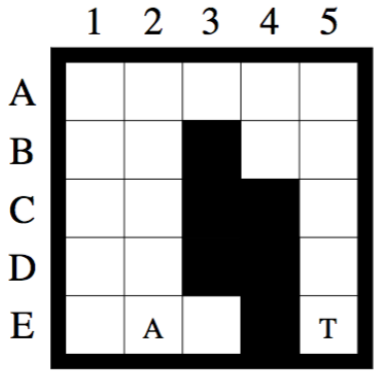


Figure 4: Second Example Search Problem

Solution In Figure 4, the start state A is at $(E, 2)$ and goal state T is at $(E, 5)$. Let's denote A_N as the north cell $(D, 2)$ of A, so $g(A_N) = 1$ and $h(A_N) = 4$, where $h(A_N)$ is calculated as Manhattan distance from A_N to T. Then, we have

$$f(A_N) = g(A_N) + h(A_N) = 5$$

Equivalently, let's denote A_E as the east cell $(E, 3)$ of A, so $g(A_E) = 1$, and $h(A_E) = 2$. Then

$$f(A_E) = g(A_E) + h(A_E) = 3$$

For each iteration in ComputePath, the state with smaller $f - value$ will be chosen from the $OPEN$ list. In this case, A_E is chosen since $f(A_E) < f(A_N)$, the agent will first move to the east rather than to the north.

1.2 Problem B

Problem This project argues that the agent is guaranteed to reach the target if it is not separated from it by blocked cells. 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.

Solution For the first question, a finite gridworld can be seen as an undirected graph, which is divided by the blocked cells into several connected components. The start cell of agent and the goal must be assigned to any of these connected components. In one connected component, every cell is reachable from all other cells in the component. This means an agent can use search algorithms like DFS or A* to traverse all the cells in the same connected component in finite time. If start cell and goal cell lie on the same component, the agent can definitely reach the goal, otherwise, it is impossible for agent to reach the goal after all the cells are traversed.

For the second question, suppose there are totally K unblocked cells in a map. Now, let's consider the number of moves in each iteration. For the 1st iteration, the agent will just move from the start cell to a nearby cell, so the number of moves is 1. For the 2nd iteration, the agent may either move to an adjacent new cell or trace back to the start cell and move to the other new cell next to the start cell, so the largest number of moves is 2. Generally, for the ith iteration, the agent will visit exact one new cell, where a "new cell" is said to be the cell that has never

been visited in previous $i-1$ iterations. Therefore, the largest number of moves is i. Since there are K unblocked cells, there are only $K-1$ iterations. The worst case is that in each iteration, the agent always traverse the longest path to reach the new cell. Let's denote $N(k)$ as the total number of moves after ith iteration and we have

$$N(k) = \sum_{i=1}^{i=k-1} = \frac{k(k-1)}{2} = O(k^2)$$

If the start cell and the goal cell are in the same connected component of the map, $k = K$ so that $N(k) = N(K) = O(K^2)$, otherwise, $k < K$ so that $N(k) < N(K) = O(K^2)$. Therefore, in the end, the number of moves of the agent is bounded from above by the number of unblocked cells squared.

Part 2 The Effects of Ties

Problem Repeated forward a* needs to break ties to decide which cell to expand next if several cells have the same smallest f-value. It can either break ties in favor of cells with smaller g-values or in favor of cells with larger g-values. 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.

Solution After running two algorithms respectively for 50 random maps of size 101×101 (with discrete obstacles), the result is shown in Table 1. (Please refer to Appendix A for full list of results.)

Table 1: Result of two Repeated Forward A* algorithms

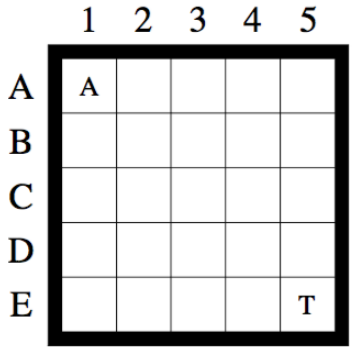
Algorithm	Aver. Expa/M	aver. Expa/P	Aver. Expl/M	Aver. Expl/P
Raw RFA*	264141.78	3058.58	285757.42	3308.32
Modified RFA*	8847.88	92.44	25461.42	267.68
Algorithm	Aver. Moves	Aver. Count	Aver. Optimal	Aver. Ratio
Raw RFA*	335.72	88.2	198.6	1.6852
Modified RFA*	367.56	96.04	198.6	1.8458

In the table, raw RFA* is the RFA* algorithm that breaks ties by f value and the modified RFA* is the one that breaks ties by $c \times f - g$. The meaning of each column is listed as below:

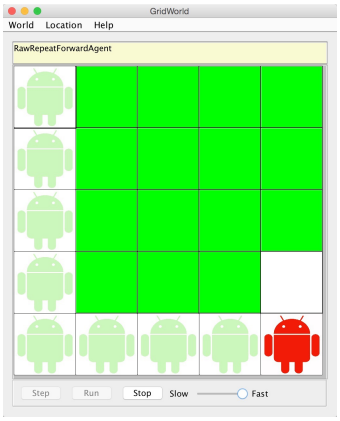
- Expa/M: total number of cells expanded in each map;
- Expa/P: average number of cells expanded in each plan;
- Expl/M: total number of cells explored in each map;
- Expl/P: average number of cells explored in each plan;
- Moves: total number of actual moves in each map;
- Count: total number of planning in each map;
- Optimal: number of optimal (minimum) moves in each map;
- Ratio: ratio of moves/optimal;

As we can see, the average number of cells expanded in each map by raw RFA* is almost 30 times as that by modified RFA*. Similarly, the average number of cells expanded at each planning (call computepath) by raw RFA* is also 33 times as that by modified RFA*. This indicates that modified RFA* expands much less cells than that by modified RFA*. This is similar to the number of explored cells, which represents all the cells that are added to *open* list and some of them may not be extracted to expand. We can see that the average number of explored in each map by raw RFA* is more than 10 times as that by modified RFA*, and ratio of the number for each planning reaches over 12 times. However, when we inspect the average number of moves, the situation is quite different. Thanks to the large-scale nodes expanded, raw RFA* enable agent to move nearly 9% less steps than modified RFA* either in each map or for each planning. This indiates that the move part of raw RFA* is more likely to approach optimal.

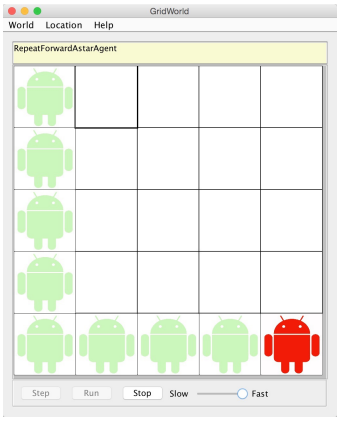
To explain in detail, let's introduce a simple example as shown in figure 5a. It is a 5×5 grid map where the start cell is at $(A, 1)$ and the goal cell is at $(E, 5)$. We can see that raw RFA* in Figure 5b almost expands all cells on the map while modified RFA* in Figure 5c only expands some useful cells. The reason for this difference depends on the way they break ties. In Figure 5b, the algorithm only use f-value to determine the order of expansion which causes a lot of tie cases. For example, at first, the agent will explore its neighbors $(A, 2)$ and $(B, 1)$, whose g-values are both set as 1, h-values are 7 and f-values are 8. The next node to expand is either $(A, 2)$ or $(B, 1)$, so without loss of generality, let's pick $(B, 1)$. After that, cell $(C, 1)$ is explored whose $g = 2$, $h = 6$ and $f = 8$. (Just forget $(B, 2)$ temporarily.) It has the same f-value as $(A, 2)$. In this case, raw RFA* will pick one arbitrarily, and if it picks $(A, 2)$, $(A, 3)$ will be explored. But for modified RFA*, it will inspect g value to break ties between $(A, 2)$ and $(C, 1)$ and pick the



(a) Third Example of Search Problem



(b) Raw RFA* breaking ties by f



(c) Modified RFA* breaking ties by $(c \times f - g)$

Figure 5: Results of two versions of Repeated Forward A*

cell with smaller g, that is $(C, 1)$. Repeatedly, modified RFA* will expand much less cells than raw RFA*. Therefore, this is the effect of breaking ties by using $c \times f - g$ value.

Part 3 Forward vs. Backward

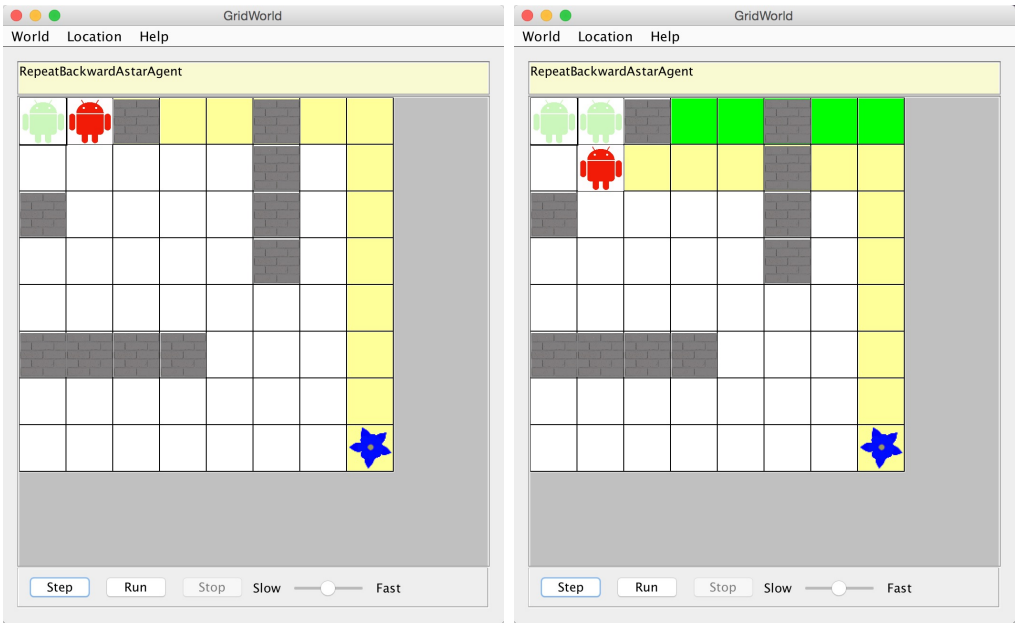
Problem Implement and compare Repeated Forward A* and Repeated Backward A* with respect to their runtime or, equivalently, number of expanded cells. Explain your observations in detail, that is, explain what you observed and give a reason for the observation. Both versions of Repeated A* should break ties among cells with the same f-value in favor of cells with larger g-values and remaining ties in an identical way, for example randomly.

Solution As Table 2 shows, the Repeated Backward A* expands more cells than RFA*. On average, for each map, RBA* takes time (namely the expanded cells) 15 times more than RFA*. In addition, in each procedure of computing path, RBA* also expands cells over 10 times more than RFA*. So the efficiency of RBA* is much worse than that of RFA*.

The reason is that, at the beginning of search, RFA* can use the knowledge from moving to discover which cells are blocked, thus it can immediately avoid calculating those possible shortest paths that are blocked by obstacles on the way. In contrast, RBA* searches from the goal, and cells near the goal are still unknown.

Table 2: Result of Repeated Forward A* and Repeated Backward A*

Algorithm	Aver. Expa/M	aver. Expa/P	Aver. Expl/M	Aver. Expl/P
RFA*	8847.88	92.44	25461.42	267.68
RBA*	133043	1272.54	266050.76	2551.44
Algorithm	Aver. Moves	Aver. Count	Aver. Optimal	Aver. Ratio
RFA*	335.72	88.2	198.6	1.6852
RBA*	390.88	103.4	198.6	1.9626

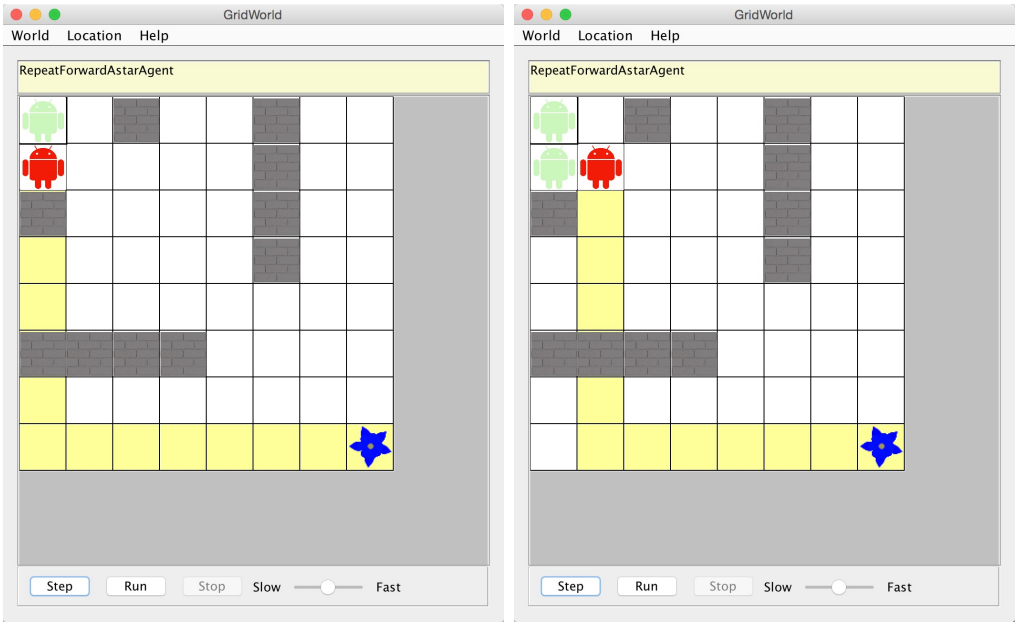


(a) RBA* before computing (b) RBA* after computing

Figure 6: Example of RBA* procedure for computing path

So RBA* will find blocked cells very late, which may result in the defect that RBA* has to compute another path from an early cell it has expanded. Thus RBA* will cost more time and expand more cells to find the shortest path.

For example, in the Figure 6, denote each cell as $c(i, j)$, where i, j is the row number and column number from 1 to 8. In the Figure 6a, the agent encounters a blocked cell $c(1, 3)$, then it has to compute the path again. Using RBA* search, it will compute the path from goal. When it reaches $c(1, 4)$ along the yellow path near the agent, it will find its adjacent cell $c(1, 3)$ is blocked. Then it has to turn down to $c(2, 4)$. However, when the cell $c(2, 8)$ was expanded, its left cell $c(2, 7)$



(a) RFA* before computing (b) RFA* after computing

Figure 7: Example of RFA* procedure of computing path

was added to $OPEN$ list. Then we have

$$f(c(2,7)) = h(c(2,7)) + g(c(2,7)) = 6 + 7 = 13$$

and

$$f(c(2,4)) = h(c(2,4)) = 3 + 12 = 15$$

So, the procedure will choose $c(2,7)$ to expand and there will be a new path in this procedure of computing path, like the Figure 6.

In contrast, as Figure 7 shows, when RFA* computes the path, it will immediately find $c(3,1)$ is blocked, thus it will avoid adding it to the shortest path at once. Then it will not expand extra cells.

Part 4 Heuristics in the Adaptive A*

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

Solution Assume that Manhattan distances are not consistent in gridworlds, then there must exist some cell n, which $h(n) > h(n') + c(n, n')$, where n' is the next cell generated by n and c is the cost from n to n'. In the gridworlds where the agent can move only in the four main compass directions, the $c(n, n')$ is always 1. Then in order to make the inequality above hold true, difference of $h(n)$ and $h(n')$ must be larger than 1. However, if the agent can move only in the four main compass directions, the cell n and n' must be adjacent, the different of their heuristics values must be 1. It is conflict. So the Manhattan distances are consistent in this case.

Problem Furthermore, it is argued that "The h-value $h_{new}(s)$... are only admissible but also consistent." Prove that the Adaptive A* leaves initially consistent h-value consistent even if action costs can increase.

Solution By referring to [3], we conclude that for every cell n and every successor n' of n, when the h value update, there are 3 cases.

First, if both n and n' were expanded, it means that $h_{new}(n) = g(goal) - g(n)$ and $h_{new}(n') = g(goal) - g(n')$. Then $h_{new}(n) - h_{new}(n') = g(goal) - g(n) - g(goal) + g(n') = g(n') - g(n) \leq c$, because n and n' are adjacent. So $h_{new}(n) \leq h_{new}(n') + c$.

Second, n was expanded but n' was not. Then we have $h_{new}(n) = g(goal) - g(n)$, $g(n') \leq g(n) + c$, according to case 1, $f(goal) \leq f(n')$, since n' was generated but not expanded. Thus,

$$\begin{aligned} h_{new}(n) &= g(goal) - g(n) \\ &= f(goal) - g(n) \\ &\leq f(n') - g(n) \\ &= g(n') + h(n') - g(n) \\ &= g(n') + h_{new}(n') - g(n) \\ &\leq g(n') + h_{new}(n') - g(n') + c \\ &= h_{new}(h) + c \end{aligned}$$

Third, n was not expanded,which implies that $h_{new}(n) = h(n)$. In the meantime, $h(n') \leq h_{new}(n')$. Thus $h_{new}(n) = h(n) \leq h(n') + c \leq h_{new}(n') + c$.

Therefore, the Adaptive A* leaves initially consistent h-value consistent even if action costs can increase.

Part 5 Heuristics in the Adaptive A*

Problem 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. Both search algorithms should break ties among cells with the same f-value in favor of cells with larger g-values and remaining ties in an identical way, for example randomly.

Solution Firstly, we use 50 maps of size 101×101 with discrete obstacles to evaluate performance of two algorithms. The result is shown in Table 3 and complete experimental data for RTAA* is listed in Appendix C. As we can see, the average number of cells expanded per map in RFA* is almost equivalent to that in RTAA* (Real-Time Adaptive A*). Moreover, rest of evaluation indexes also reflect no significant difference between these algorithms. Therefore, can we draw the conclusion that the running time of RTAA* is approximately equal to RFA*?

Table 3: Result of Repeated Forward A* and Adaptive A* in Maps

Algorithm	Aver. Expa/M	aver. Expa/P	Aver. Expl/M	Aver. Expl/P
RFA*	8847.88	92.44	25461.42	267.68
RTAA*	8749.2	91.32	25413.66	266.72
Algorithm	Aver. Moves	Aver. Count	Aver. Optimal	Aver. Ratio
RFA*	335.72	88.2	198.6	1.6852
RTAA*	367	96.18	198.6	1.843

THE ANSWER IS NO!!!

And this is why we generate another kind of gridworld, namely the maze. As previously said, any maze in this case is always a connected undirected graph, so the optimal path from start to goal is quite long, full of twists and turns. It is this characteristic of maze that favors the RTAA* to performs better than RFA*. The result is shown in Table 4, and complete experimental data is listed in Appendix C. The average number of expanded cells in each map for RTAA* is about 22.3% less than that for RFA*, and the average number of expanded cells in each planning for RTAA* is about 19.09% less than that for RFA*. Meanwhile, the average number of explored cells for RTAA* is almost the same as that for RFA* because both algorithms will place all successors into $OPEN$ list. The average number of moves either in each map or in each planning for RTAA* is 9.3% larger than that for RFA*. Compared this ratio to that of expanded cells, we can find RTAA* performs better in running time of expansion by the price of moving more steps.

Table 4: Result of Repeated Forward A* and Adaptive A* in Mazes

Algorithm	Aver. Expa/M	aver. Expa/P	Aver. Expl/M	Aver. Expl/P
RFA*	148454.72	148.32	338600.96	349.84
RTAA*	115318.38	120	301468.68	319.16
Algorithm	Aver. Moves	Aver. Count	Aver. Optimal	Aver. Ratio
RFA*	2848.6	932.54	1078.48	2.87
RTAA*	2854.56	929.44	1078.48	2.8758

The advantage of RTAA* lies in its constant update for heuristic value of expanded states. The heuristics of the same state in different planning procedures are monotonically nondecreasing over time, and thus become more informed[3]. Since in every iteration, the algorithm will choose the cell with the smallest f-value in the $OPEN$ list, the cell with updated larger heuristic, namely larger f-value, implies that it will be less probable to be extracted from the $OPEN$ list. Instead, some other cells may be firstly selected by RTAA*, which is selected later by RFA*. On the other hand, the heuristic value in RTAA* can be regarded composed of two parts. The first part is calculated according to from gained knowledge in previous planning, and the second the part is the ordinary estimate goal distance, like Manhattan distance in the case. The first part can benefit RTAA* from selecting a more informed cell from $OPEN$ list, so that less nodes are needed to be expanded, whereas RFA* expands more redundant cells during planning.

Part 6 Memory Issues

Problem You performed all experiments in gridworlds of size 101×101 but some real-time computer games use maps whose number of cells is up to two orders of magnitude larger than that. It is then especially important to limit the amount of information that is stored per cell. For example, the tree-pointers can be implemented with only two bits per cell. Suggest additional ways to reduce the memory consumption of your implementations further. Then, calculate the amount of memory that they need to operate on gridworlds of size 1001×1001 and the largest gridworld that they can operate on within a memory limit of 4 MBytes.

Solution The data structure of states in our project show below.

State			
type	name	bits	description
Integer	column	10	the column of state in grid
	row	10	the row of state in grid
Boolean	bSearch	1	reflect whether the cell of this state have been found
Boolean	canMove	1	reflect whether the cell of this state is viable

Let's denote n as the memory size of each state and N as the memory size of the array of states. In this data structure, $n = 22$ bits. In the map of size 1001×1001, then

$$N = n * 1001 * 1001 = 2.6MBytes.$$

Denote $g(s)$ as the cost from start state to state s. During each iteration, namely the procedure of computing path, the array of g values need store all cells which have been explored.

Denote $h(s)$ as the estimated cost from state s to goal state. During each iteration, namely the procedure of computing path, the array of h values need store all cells which have been explored.

Denote $search(s)$ as the explore times of state s. During each iteration, the array of $search$ values need store all cells which have been explored.

Denote $close(s)$ as the state which have been expanded. During each iteration, the array of $close$ values need store all states which have been expanded.

Denote $open(s)$ as the priority heap that stores states s to be expand sorted by the sum of heuristic value and $g(s)$. The heap $open$ need store nearly all cells which have been explored.

The type of the value of each element in the data structure above is integer.

Assume the start state and goal state are in the opposite endpoints in the diagonal line of the map. Then for each iteration, the number of expanded cells will be close to the sum of $rows$ and $cols$, where $rows$ is the total number of rows in map and $cols$ is the total number of columns. So the number of expanded cells ≈ 2000. Then in the map of size 1001×1001,

$$
\begin{aligned}
g &= 4 * 2000 * (32 + 34) \approx 65KByte \\
h &= 4 * 2000 * (32 + 34) \approx 65KByte \\
search &= 4 * 2000 * (32 + 34) \approx 65KByte \\
close &= 2000 * 34 \approx 8KByte \\
open &= 4 * 2000 * 34 \approx 65KByte \\
tree &= 4 * 2000 * (32 + 34) \approx 65KByte
\end{aligned}
$$

So, the total memory cost is :

$$
S_{total} = S + g + search + close + open + tree \approx 3MBytes < 4MBytes
$$

Then the project can operate within a memory limit of 4 Mbyte, using a map of size 1001×1001.

References

[1] CollegeBoard. AP central gridworld case study. `http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/151155.html`. [Online; accessed 2-October-2015].

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[3] Sven Koenig and Maxim Likhachev. Real-time adaptive a*. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 281–288. ACM, 2006.

[4] Wikipedia. Maze generation algorithm — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Maze_generation_algorithm&oldid=679876968`. [Online; accessed 4-October-2015].

Appendix A Experimental Result in Part 2

The following data shows the result of Repeated Forward A* that breaks ties by f-value, running on 50 gridworlds of size 101×101 with discrete obstacles.

Map	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
01	193939	2731	210778	2968	308	72	198	155%
02	241247	3133	260651	3385	324	78	202	160%
03	290740	3303	313775	3565	340	89	208	163%
04	261493	3076	282159	3319	364	86	198	183%
05	265125	3535	285301	3804	260	76	200	130%
06	269247	3496	289484	3759	310	78	198	156%
07	294643	3202	317934	3455	340	93	198	171%
08	381789	3601	410470	3872	414	107	206	200%
09	233863	3118	252433	3365	318	76	196	162%
10	157708	2160	172349	2360	282	74	198	142%
11	329602	4336	353612	4652	322	77	198	162%
12	329680	2536	358172	2755	454	131	198	229%
13	269377	2960	291730	3205	308	92	198	155%
14	305818	4023	330092	4343	332	77	196	169%
15	323755	3205	350503	3470	386	102	196	196%
16	311526	3943	335212	4243	284	80	198	143%
17	293787	2937	317747	3177	340	101	198	171%
18	203746	2546	221013	2762	332	81	196	169%
19	263649	3337	285326	3611	296	80	196	151%
20	248162	3141	268052	3393	324	80	198	163%
21	300202	2943	325008	3186	336	103	196	171%
22	299406	3402	323624	3677	370	89	196	188%
23	212412	2441	230413	2648	340	88	198	171%
24	254674	3690	274313	3975	248	70	198	125%
25	287253	2762	310868	2989	364	105	198	183%
26	247032	2839	268384	3084	348	88	202	172%
27	186408	2662	202645	2894	276	71	196	140%
28	261635	2400	284418	2609	390	110	200	195%
29	248697	3552	268251	3832	298	71	196	152%
30	234020	3250	253314	3518	286	73	196	145%
31	357994	3729	387464	4036	368	97	196	187%
32	225317	3086	244718	3352	284	74	202	140%
33	330790	2876	359311	3124	426	116	198	215%
34	245950	3616	264951	3896	278	69	196	141%

Map	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
35	268510	3274	289327	3528	316	83	202	156%
36	292034	3281	316049	3551	340	90	200	170%
37	213698	3339	230924	3608	274	65	196	139%
38	310991	2853	336212	3084	402	110	200	201%
39	294104	3770	316165	4053	260	79	202	128%
40	353257	2993	383021	3245	416	119	204	203%
41	251062	3138	271256	3390	290	81	196	147%
42	229942	2947	248723	3188	306	79	200	153%
43	188577	2449	205809	2672	340	78	196	173%
44	320741	2741	346617	2962	440	118	198	222%
45	237946	2087	259449	2275	408	115	198	206%
46	238556	2981	258069	3225	320	81	200	160%
47	237402	2498	257667	2712	378	96	198	190%
48	189984	2288	207542	2500	318	84	204	155%
49	169421	2144	184717	2338	346	80	196	176%
50	250178	2579	271849	2802	382	98	198	192%

The following data shows the result of Repeated Forward A* that breaks ties by $c \times f - g$ value, running on 50 gridworlds of size 101×101 with discrete obstacles.

Map	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
01	7448	87	21738	255	326	86	198	164%
02	10555	89	30036	254	446	119	202	220%
03	9513	106	27765	311	358	90	208	172%
04	7750	94	22063	269	380	83	198	191%
05	10434	98	29746	280	424	107	200	212%
06	9880	90	27833	255	422	110	198	213%
07	6112	88	17808	258	306	70	198	154%
08	10858	100	31267	289	436	109	206	211%
09	9791	89	28195	258	444	110	196	226%
10	5626	82	16487	242	284	69	198	143%
11	7331	83	21034	239	296	89	198	149%
12	8462	82	24072	236	414	103	198	209%
13	10084	92	29369	269	346	110	198	174%
14	8001	91	23474	269	332	88	196	169%
15	15108	126	43961	369	410	120	196	209%
16	6856	92	20034	270	328	75	198	165%
17	7860	92	23003	270	322	86	198	162%

Map	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
18	6264	79	17840	225	330	80	196	168%
19	7386	85	21424	249	352	87	196	179%
20	7664	97	22480	284	316	80	198	159%
21	7039	95	20654	279	290	75	196	147%
22	9170	95	26622	277	376	97	196	191%
23	7522	67	21360	190	462	113	198	233%
24	7174	96	21094	285	312	75	198	157%
25	11622	91	31781	250	504	128	198	254%
26	10035	98	29101	285	412	103	202	203%
27	7505	92	21883	270	322	82	196	164%
28	8132	88	23801	258	322	93	200	161%
29	8670	88	25175	256	354	99	196	180%
30	8197	87	24013	255	342	95	196	174%
31	10383	92	30040	268	436	113	196	222%
32	9311	99	27209	289	362	95	202	179%
33	13378	107	36611	292	464	126	198	234%
34	6546	99	19222	291	290	67	196	147%
35	8897	93	25370	267	344	96	202	170%
36	8929	105	26054	306	352	86	200	176%
37	11307	97	29247	252	464	117	196	236%
38	8363	85	23831	243	396	99	200	198%
39	9750	105	28543	310	346	93	202	171%
40	8332	106	24554	314	310	79	204	151%
41	8016	86	23377	251	352	94	196	179%
42	10959	116	32145	341	320	95	200	160%
43	7234	92	21262	272	290	79	196	147%
44	8182	81	23190	229	394	102	198	198%
45	8440	83	23530	232	384	102	198	193%
46	11833	97	34218	282	396	122	200	198%
47	9418	85	26692	242	416	111	198	210%
48	7789	82	22661	241	360	95	204	176%
49	8799	87	25613	256	368	101	196	187%
50	8479	86	24589	250	366	99	198	184%

Appendix B Experimental Result in Part 3

The following data shows the result of Repeated Backward A* that breaks ties by $c \times f - g$ value, running on 50 gridworlds of size 101×101 with discrete obstacles.

Map	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
01	134906	1260	270094	2524	394	108	198	198%
02	135787	1385	275869	2814	338	99	202	167%
03	200046	1818	400515	3641	404	111	208	194%
04	40174	441	84564	929	302	92	198	152%
05	184030	1437	364682	2849	452	129	200	226%
06	122975	1366	247853	2753	338	91	198	170%
07	80545	767	160536	1528	402	106	198	203%
08	411531	2236	810202	4403	690	185	206	334%
09	103712	886	206155	1762	476	118	196	242%
10	38155	393	80253	827	362	98	198	182%
11	136259	1285	271111	2557	416	107	198	210%
12	53781	672	111394	1392	296	81	198	149%
13	237452	1562	460195	3027	654	153	198	330%
14	204787	1750	406341	3473	444	118	196	226%
15	72982	1027	147807	2081	320	72	196	163%
16	176802	1155	349723	2285	552	154	198	278%
17	121503	1278	247019	2600	314	96	198	158%
18	62020	747	125790	1515	338	84	196	172%
19	107999	1421	218249	2871	316	77	196	161%
20	180443	1409	360588	2817	482	129	198	243%
21	90132	804	179443	1602	410	113	196	209%
22	135365	1410	271375	2826	374	97	196	190%
23	144744	964	278297	1855	578	151	198	291%
24	135593	1102	269679	2192	488	124	198	246%
25	78912	710	157808	1421	420	112	198	212%
26	254975	2628	509428	5251	384	98	202	190%
27	42123	569	86567	1169	324	75	196	165%
28	124467	1072	244979	2111	488	117	200	244%
29	58542	760	118093	1533	336	78	196	171%
30	130137	1606	259602	3204	364	82	196	185%
31	136634	1485	269412	2928	368	93	196	187%
32	104565	1015	213367	2071	342	104	202	169%
33	85245	1469	173786	2996	248	59	198	125%
34	133248	1200	266490	2400	424	112	196	216%

Map	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
35	173719	1867	350207	3765	388	94	202	192%
36	170068	1868	343798	3778	316	92	200	158%
37	118598	972	237036	1942	434	123	196	221%
38	146021	1872	292571	3750	308	79	200	154%
39	222513	1986	441990	3946	428	113	202	211%
40	213087	1805	423234	3586	426	119	204	208%
41	101709	1255	206998	2555	296	82	196	151%
42	142643	1333	285098	2664	414	108	200	207%
43	124403	1704	251151	3440	318	74	196	162%
44	62355	820	126476	1664	320	77	198	161%
45	91722	1007	187141	2056	310	92	198	156%
46	94290	1047	191813	2131	304	91	200	152%
47	135220	1365	269816	2725	392	100	198	197%
48	245313	1817	491518	3640	424	136	204	207%
49	105437	1285	214535	2616	308	83	196	157%
50	44481	535	91890	1107	320	84	198	161%

Appendix C Experimental Result in Part 5

The following data shows the result of Adaptive A* that breaks ties by $c \times f - g$ value, running on 50 gridworlds of size 101×101 with discrete obstacles.

Map	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
01	7428	87	21721	255	326	86	198	164%
02	10365	87	29863	253	446	119	202	220%
03	9473	106	27739	311	358	90	208	172%
04	7651	88	22077	256	382	87	198	192%
05	10403	95	30078	275	428	110	200	214%
06	9696	88	27686	251	422	111	198	213%
07	6110	88	17807	258	306	70	198	154%
08	10777	99	31190	288	436	109	206	211%
09	9700	88	28115	257	444	110	196	226%
10	5625	82	16486	242	284	69	198	143%
11	7237	82	20935	237	296	89	198	149%
12	8449	80	24298	231	416	106	198	210%
13	10040	92	29327	269	346	110	198	174%
14	7987	91	23462	269	332	88	196	169%
15	14831	125	43415	367	408	119	196	208%
16	6837	92	20019	270	328	75	198	165%
17	7814	91	22958	270	322	86	198	162%
18	6131	77	17708	224	330	80	196	168%
19	7372	85	21415	249	352	87	196	179%
20	7469	95	21930	281	310	79	198	156%
21	7008	94	20625	278	290	75	196	147%
22	9994	98	29125	288	382	102	196	194%
23	7040	69	20228	198	406	103	198	205%
24	7169	96	21094	285	312	75	198	157%
25	10980	87	31075	246	518	127	198	261%
26	10153	98	29589	287	412	104	202	203%
27	7490	92	21878	270	322	82	196	164%
28	8104	88	23783	258	322	93	200	161%
29	8601	87	25110	256	354	99	196	180%
30	8185	87	24004	255	342	95	196	174%
31	10662	92	31062	270	444	116	196	226%
32	9264	98	27159	288	362	95	202	179%
33	12763	100	36428	286	466	128	198	235%
34	6546	99	19223	291	290	67	196	147%

Map	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
35	8689	91	25121	264	344	96	202	170%
36	8906	104	26043	306	352	86	200	176%
37	9882	87	27282	241	458	114	196	233%
38	8373	85	24075	245	402	99	200	201%
39	9705	105	28509	309	346	93	202	171%
40	8327	106	24549	314	310	79	204	151%
41	7984	85	23351	251	352	94	196	179%
42	10919	116	32108	341	320	95	200	160%
43	7224	92	21252	272	290	79	196	147%
44	8017	79	23046	228	394	102	198	198%
45	8097	80	23130	231	380	101	198	191%
46	11704	96	34100	281	396	122	200	198%
47	9294	83	26653	240	416	112	198	210%
48	7766	82	22640	240	360	95	204	176%
49	8793	87	25670	254	370	102	196	188%
50	8426	85	24542	250	366	99	198	184%

The following data shows the result of Repeated Forward A* that breaks ties by $c \times f - g$ value, running on 50 mazes of size 101×101.

Maze	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
01	113162	120	312238	333	2812	937	872	322
02	261787	168	580154	373	4924	1554	1608	306
03	234329	198	457259	387	3968	1180	932	425
04	110743	121	284572	313	2678	909	1280	209
05	28712	85	78835	235	676	336	676	100
06	46137	99	128970	276	972	467	836	116
07	79847	99	197909	247	1528	800	1528	100
08	101044	121	254925	307	3020	830	656	460
09	189733	220	410538	477	2464	860	1264	194
10	314339	243	552230	427	4492	1292	1284	349
11	98389	133	264182	358	2292	737	1184	193
12	115882	156	281395	379	2268	743	1172	193
13	188766	189	387569	389	3488	997	804	433
14	53954	99	142863	264	1112	542	952	116
15	170156	122	434035	311	4740	1394	1168	405
16	113123	145	294392	378	2132	779	888	240
17	61174	109	162781	292	1580	558	980	161

Maze	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
18	150685	150	359740	358	2832	1005	1160	244
19	105854	126	271584	324	2876	838	1020	281
20	268251	151	618040	349	7084	1770	732	967
21	132127	135	347725	357	3078	973	1040	295
22	365597	204	758615	424	6184	1788	1320	468
23	108928	118	289651	316	1716	917	1660	103
24	170866	135	447369	354	4492	1264	772	581
25	16987	89	49198	258	436	191	412	105
26	91726	103	237607	268	1716	887	1716	100
27	58865	104	159896	284	1072	564	1072	100
28	79056	138	202254	354	2020	572	816	247
29	79756	116	204305	297	1748	688	976	179
30	160135	144	375752	339	3824	1107	1052	363
31	168598	160	371971	354	3824	1049	780	490
32	340000	209	695160	428	5268	1623	2060	255
33	169781	158	385602	360	2256	1072	1740	129
34	77927	117	209634	315	1852	666	916	202
35	65238	116	186760	334	1016	559	1008	100
36	131240	126	331618	318	1868	1041	1860	100
37	271774	226	573871	477	4296	1203	592	725
38	221740	210	455669	432	3756	1055	1012	371
39	246527	232	466120	440	3528	1060	1100	320
40	137534	149	333373	361	3268	922	892	366
41	267697	188	554892	389	5256	1424	700	750
42	62614	116	176167	328	1116	538	1040	107
43	210315	185	414978	366	4028	1133	904	445
44	141534	165	352205	410	2776	858	948	292
45	49383	98	137829	274	952	503	952	100
46	95581	149	229496	359	1872	639	988	189
47	108930	156	270586	388	2192	697	660	332
48	129061	156	300302	363	2782	827	852	326
49	365743	246	702717	474	4372	1482	1692	258
50	91409	114	234515	294	1928	797	1396	138

The following data shows the result of Adaptive A* that breaks ties by $c \times f - g$ value, running on 50 mazes of size 101×101.

Maze	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
01	108492	116	305911	329	2812	930	872	322
02	232013	150	548825	355	4924	1544	1608	306
03	131808	113	310476	266	3996	1166	932	428
04	100189	110	275397	303	2690	908	1280	210
05	27860	83	78333	233	676	336	676	100
06	44360	95	127012	273	972	465	836	116
07	69969	89	186463	237	1528	785	1528	100
08	91469	110	244982	295	3028	829	656	461
09	151760	175	385864	447	2464	864	1264	194
10	186438	143	409121	315	4564	1296	1284	355
11	91828	125	256765	351	2292	731	1184	193
12	99335	134	266983	360	2268	742	1172	193
13	117553	119	280922	284	3488	987	804	433
14	50072	92	138177	255	1112	541	952	116
15	156198	113	420313	304	4740	1381	1168	405
16	109001	140	288636	372	2132	776	888	240
17	56417	102	157046	285	1584	552	980	161
18	123429	123	326987	327	2832	999	1160	244
19	100151	118	268474	317	2876	846	1020	281
20	230629	129	593502	333	7088	1778	732	968
21	124986	126	346622	350	3094	991	1040	297
22	223936	125	562667	316	6244	1781	1320	473
23	99188	108	278106	304	1716	914	1660	103
24	158829	125	438559	345	4492	1270	772	581
25	16898	88	49111	258	436	191	412	105
26	84083	95	231149	263	1716	877	1716	100
27	56155	99	158001	280	1072	565	1072	100
28	69104	123	190875	341	2020	560	816	247
29	73545	106	201054	291	1748	690	976	179
30	138284	124	354775	319	3840	1111	1052	365
31	125125	120	323743	310	3824	1043	780	490
32	232809	140	594080	358	5300	1660	2060	257
33	139585	131	357634	337	2256	1062	1740	129
34	73797	111	205739	310	1852	664	916	202
35	64000	114	185490	333	1016	558	1008	100
36	115070	110	313265	301	1868	1039	1860	100
37	177257	149	458444	387	4296	1183	592	725
38	169422	158	416849	391	3756	1067	1012	371

Maze	Expa/M	Expa/P	Expl/M	Expl/P	Moves	Count	Optimal	Ratio
39	151326	139	388291	357	3540	1086	1100	321
40	116031	126	310302	337	3280	921	892	367
41	177263	128	437975	316	5268	1385	700	752
42	60613	113	173629	324	1116	536	1040	107
43	132104	118	324613	290	4028	1117	904	445
44	125397	146	339681	397	2788	856	948	294
45	47898	95	136071	272	952	501	952	100
46	82595	128	222670	345	1872	646	988	189
47	90654	131	250838	363	2192	692	660	332
48	95113	115	254107	308	2780	825	852	326
49	184981	128	475834	331	4372	1438	1692	258
50	80900	102	223071	283	1928	787	1396	138