

1. *This part is based on the A* matlab demo (A_Star). You need to understand how this demo works and answer the following questions. (6 marks)*
 - (a) You are required to implement Greedy Search and Uniform Cost Search algorithms based on the A* code (In the report, show what changes you have made and explain why you make these changes. You can use screenshot to demonstrate your code verification). (2 marks)
 - (b) You are required to use the matlab basics from the first lab session to show the evaluation results of the three searching methods (hint: bar/plot) with respect to the **'total path cost'**, **'number of nodes discovered'** and **'number of nodes expanded'**. Explain how you can extract the related information from data stored in variable **'QUEUE'** (2 marks)
 - (c) Design and implement another heuristic h2 which is different from the one (h1) is used in the A* matlab code, explain how h2 works and show what changes you have made to change the heuristic function from h1 to h2. Is h2 optimal? Why? Which heuristic is better? Why? (2 marks)
2. This part is based on the maze generator demo (MazeGeneration-master). The maze generator is a project written by some student using Matlab. He has adopted depth-first approach to randomly generate a maze with user defined size and difficulty. (9 marks)
 - (a) In the demo code, show which line(s) of code is used to implement depth-first approach, explain the logic the student adopts to generate the maze. (2 marks)
 - (b) Identify the problem of this maze generator if there is any. (1 marks)
 - (c) Write a maze solver using A* algorithm. (6 marks)
 - i) The solver need be called by command **'AStarMazeSolver(maze)'**
 - ii) The maze solver should be able to solve any maze generated by the maze generator
 - iii) The maze solver should be able to find the optimal solution.
 - iv) Your code need display the all the routes that A* has processed with RED color.
 - v) Your maze solver should be about to display the final result BLACK color.

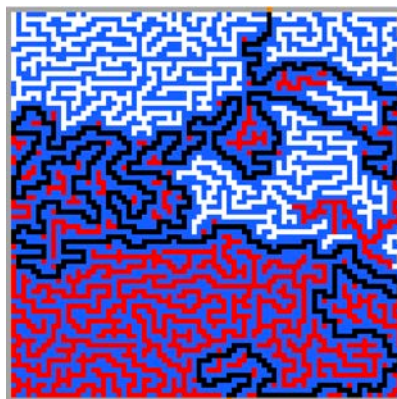
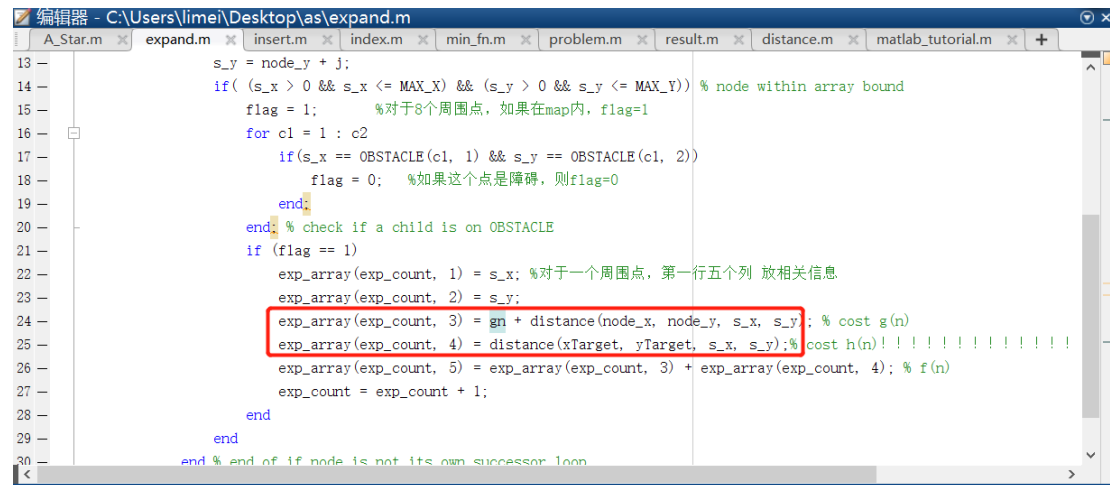


Figure 1. Sample output

Report

Q1. (a),



```
13 s_y = node_y + j;  
14 if ( (s_x > 0 && s_x <= MAX_X) && (s_y > 0 && s_y <= MAX_Y)) % node within array bound  
15     flag = 1; %对于8个周围点, 如果在map内, flag=1  
16     for c1 = 1 : c2  
17         if (s_x == OBSTACLE(c1, 1) && s_y == OBSTACLE(c1, 2))  
18             flag = 0; %如果这个点是障碍, 则flag=0  
19         end;  
20     end; % check if a child is on OBSTACLE  
21     if (flag == 1)  
22         exp_array(exp_count, 1) = s_x; %对于一个周围点, 第一行五个列 放相关信息  
23         exp_array(exp_count, 2) = s_y;  
24         exp_array(exp_count, 3) = gn + distance(node_x, node_y, s_x, s_y); % cost g(n)  
25         exp_array(exp_count, 4) = distance(xTarget, yTarget, s_x, s_y); % cost h(n)!!!!!!!!!!!!!!!!!!!!  
26         exp_array(exp_count, 5) = exp_array(exp_count, 3) + exp_array(exp_count, 4); % f(n)  
27         exp_count = exp_count + 1;  
28     end  
29 end  
30 end % end of if node is not its own successor loop
```

As the above picture shows, I change the code in red rectangle.

For **UCS** :

I change the “**exp_array(exp_count,4)**” in expand.m fold to equal to **0**.

For **Greedy search**:

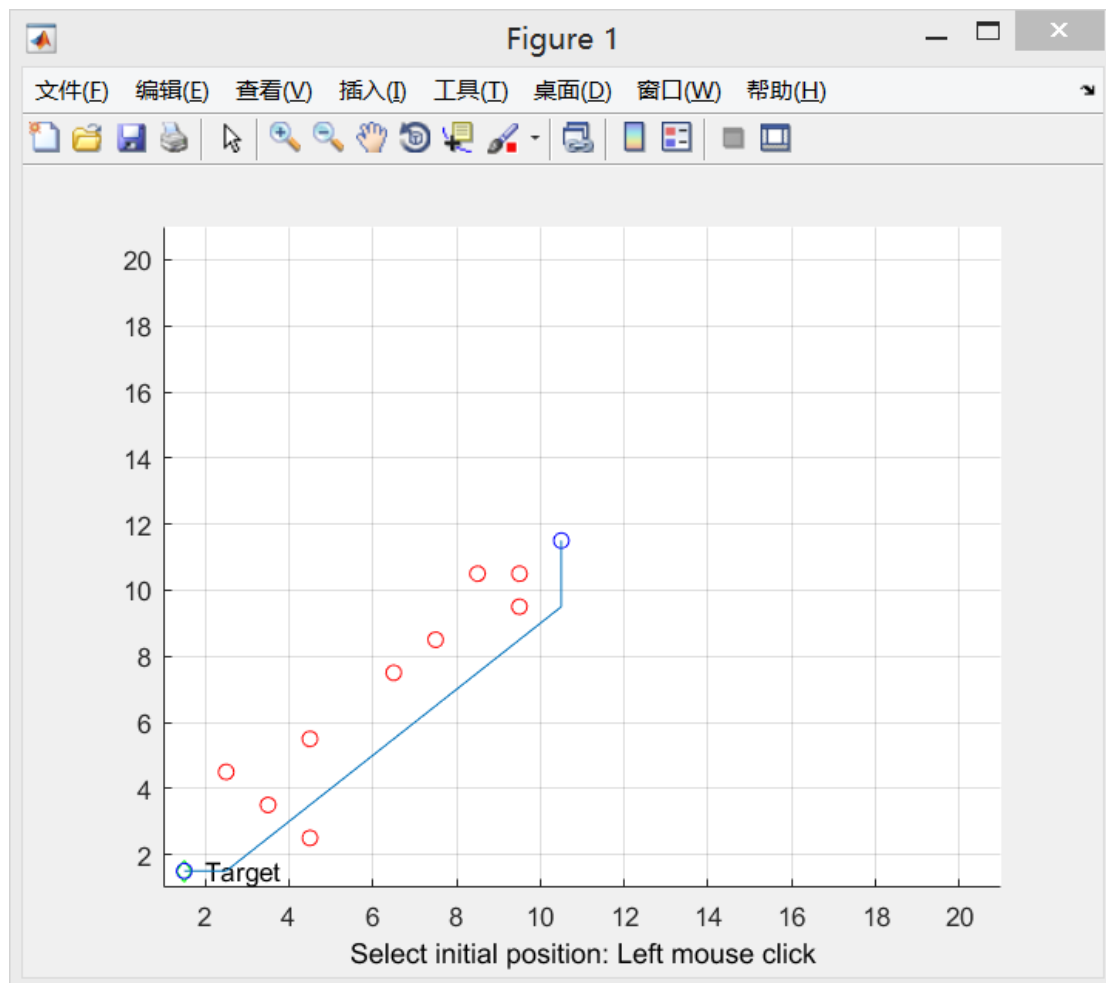
I change the “**exp_array(exp_count,3)**” in expand.m fold to equal to **0**.

The reason: A* search algorithm to find the shortest path to a goal state using a heuristic formula: **f = g + h**.

If **h=0**, the A* search algorithm becomes UCS search algorithm

If **h** is too large that can dominate g, which means **g=0**, the A* search algorithm becomes greedy search.

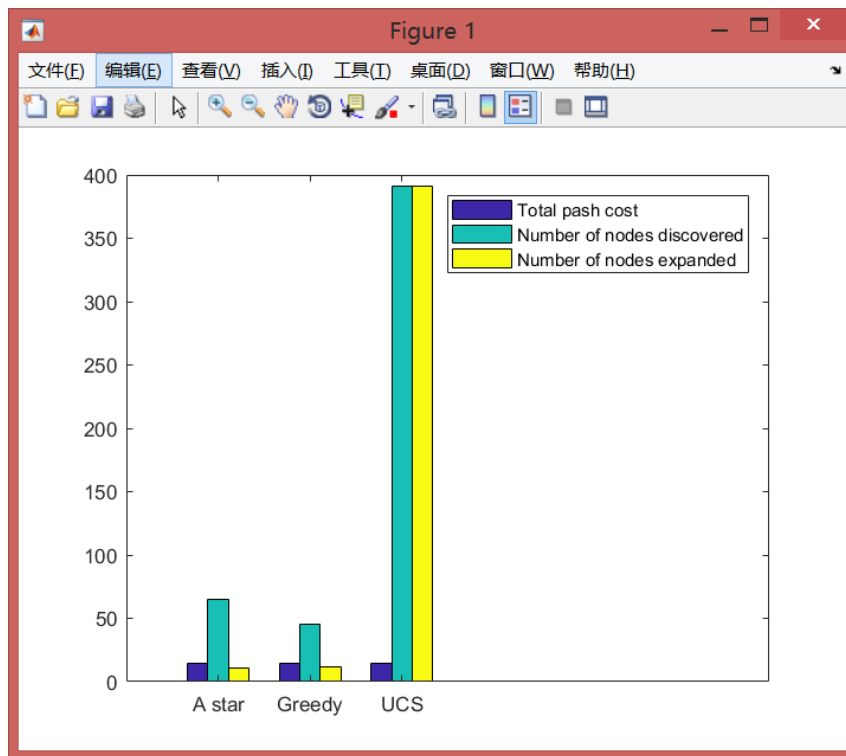
(b)



This is the graph I used to test.

The data :

	Total path cost	Number of nodes discovered	Number of node expanded
A*	14.3137	65	11
Greedy	14.3137	45	12
UCS	14.3137	391	391



(NOTE: it's noticeable that “the number of nodes discovered” and “the number of nodes expanded” is same in UCS, I have checked and there is no problem in my code, I think it must be the particularity of this point circumstance.)

The “**total path cost**” is the **$g(n)$** of QUEUE, which is the 6th column.

The “**Number of nodes discovered**” is the “**QUEUE_COUNT**” of QUEUE, which is the number of rows.

The “**Number of nodes expanded**” is the number of rows which **the first column is 0** in QUEUE.

(c),

The basic principle of h_2 is same as h_1 . However, the $h(n)$ of h_2 isn't the **straight-**

line distance as h_1 , it is the **Manhattan distance**, which is two points in the north-south distance plus the distance in the east-west direction.

```

编辑 - C:\Users\limei\Desktop\as\distance.m
A_Star.m distance.m expand.m index.m insert.m min_fn.m problem.m re
1 % This function calculates the distance between any two cartesian coordinates.
2 % Copyright 2009-2010 The MathWorks, Inc.
3
4 function dist = distance(x1, y1, x2, y2)
5 - dist = (abs ( x1 - x2 ) + abs ( y1 - y2 )) %sqrt((x1 - x2)^2 + (y1 - y2)^2);
6

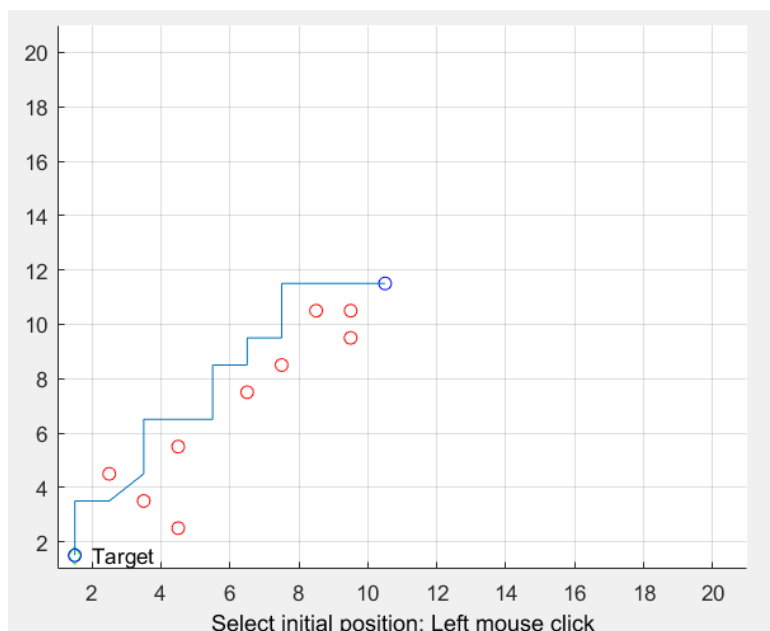
```

The new code

the original code

Which is optimal?

h_2 is optimal, because h_2 is still a A-star algorithm. When h_2 search a new node, it still finds the node which $f(n)$ is smallest. So h_2 can find the optimal path.



Which is better?

h1 is better than h2. Because in h2, the $h(n)$ overestimates the cost to the goal point.

So h2 will cost more path than h1.

Q2, (a)

The code:

```
while numel(nodes) > 0 %nodes最后还会删掉
    [maze, position, nodes] = move(maze, position, nodes, difficulty);
    dispMaze(maze);
end

if checkNode(futurePosition, previousPosition) == 1%:
    nodes(1, end + 1) = position.row;%end指目前为止的
    nodes(2, end) = position.col;
end

if any(directions) == 0 %如果四个方向都不能走
    if same(position, nodes) == 1 %如果现在的位置在nodes
        % Remove last node because all positions are explored
        % nodes是一个node坐标的list
        nodes = nodes(:, 1 : end - 1);
    else
        position = point(nodes(1, end), nodes(2, end));%
    end
else
```

The logic:

First, the program randomly generates the ends and the starting points on the top of bottom of two boundaries in the maze. Then set the point above the start point to become a path point and put it in nodes list. The search is unfolded

at this point.

Search method: If the position has 1-3 directions that can walk to, a new point around the position is randomly extended according to the difficulty of the design and whether there is a path point around it or other conditions. If the new generated point doesn't coincide with the path point around the location, the coordinate of this position is imported to the nodes list and the new position will be the present position. If there is no direction to go, the last point will be deleted in nodes list. At this time, the position is assigned to the previous location and continues the recursion.

In the main function, if node list is empty, which means all points can't expand a new point around itself, the maze is generated.

Besides, the maze generator also has an insurance mechanism which can make sure the path can connect to the final point. It is in "adjustEnd.m". the program will check whether there is a path point around the point which is under the final point. If there is no path point, the adjustEnd will automatically generate a path point around that point.

(b)