CSCI561 – Introduction to Artificial Intelligence
Instructor: Dr. K. Narayanaswamy
Assignment 2 – A* Search Algorithms
Robo Tours Problem
Due: 2/28/2014 11:59:59pm

1. Introduction

In this project you will solve a variant of the traveling salesman problem (TSP) using A* search algorithm with two heuristics: 1) Manhattan distance 2) minimum spanning tree.

Our variant of the TSP problem is defined as follows: Given a robot on a map, a) construct a shortest path graph by calculating the pairwise distance for all checkpoints on the map, and b) then find the shortest way for the robot to visit all the checkpoints exactly once and return to its starting checkpoint.

To solve the robot problem as explained above, there are two main parts of this assignment. 1) Construct the shortest path graph. 2) Solve TSP problem on the shortest path graph.

2. Shortest Path Graph

This section is for task 1 (computing the shortest path graph). Given the map, to construct the shortest path graph, you need to find a shortest path between each pair of the checkpoints. For example,

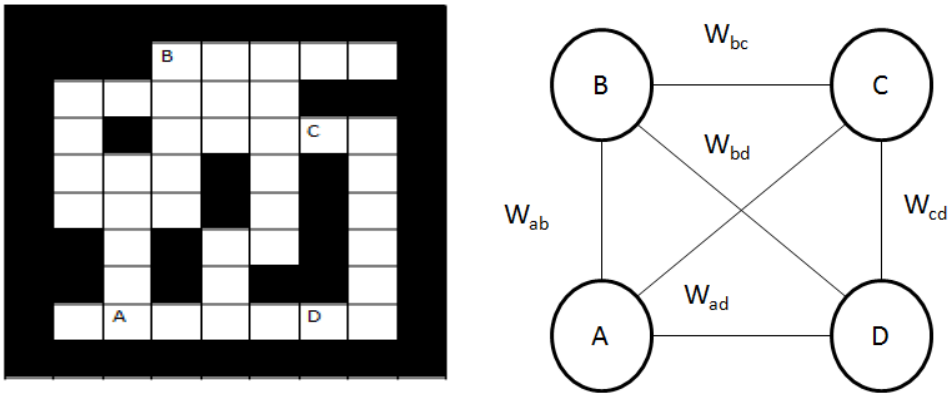
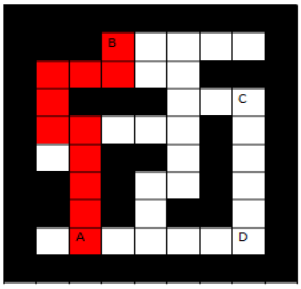
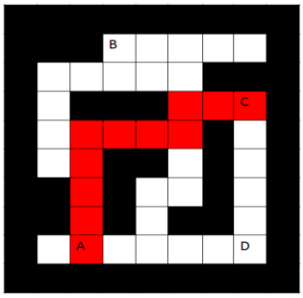


Figure 1

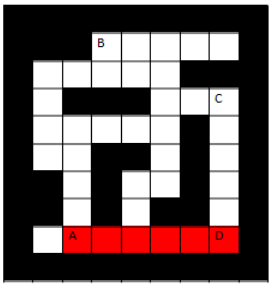
The value of W_{ab}, W_{ac}, W_{ad}, W_{bc}, W_{bc}, W_{cd} can be determined by finding the shortest path between two checkpoints. For example,



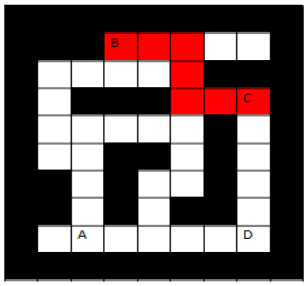
$W_{ab} = 10$



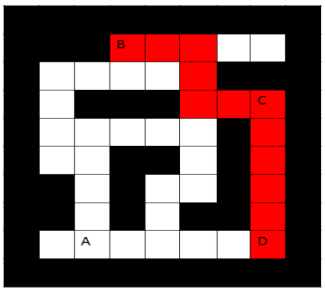
$W_{ac} = 10$



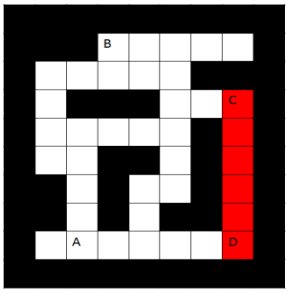
$W_{ad} = 5$



$W_{bc} = 6$



$W_{bd} = 11$

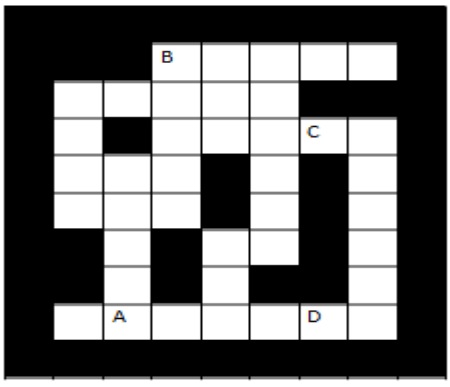


$W_{cd} = 5$

Figure 2

To find the shortest path between two check points, you need to use A* search algorithm with Manhattan distance between two points as a heuristic. Please note that: the robot can move only in four directions: ***Up, Right, Down, Left***. You can assume that the graph is fully connected – meaning that ALL checkpoints are reachable from all other checkpoints.

Give the left map, the shortest path graph should be



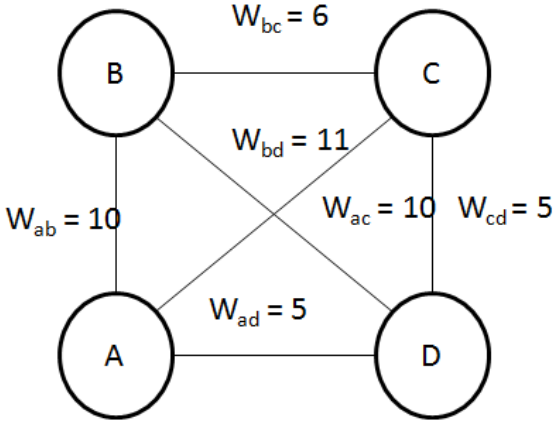


Figure 3

2.1 Tiebreaking for task 1: if there are multiple equivalent choices about which node goes next, selecting the node that is first in positional order manner on the image below.

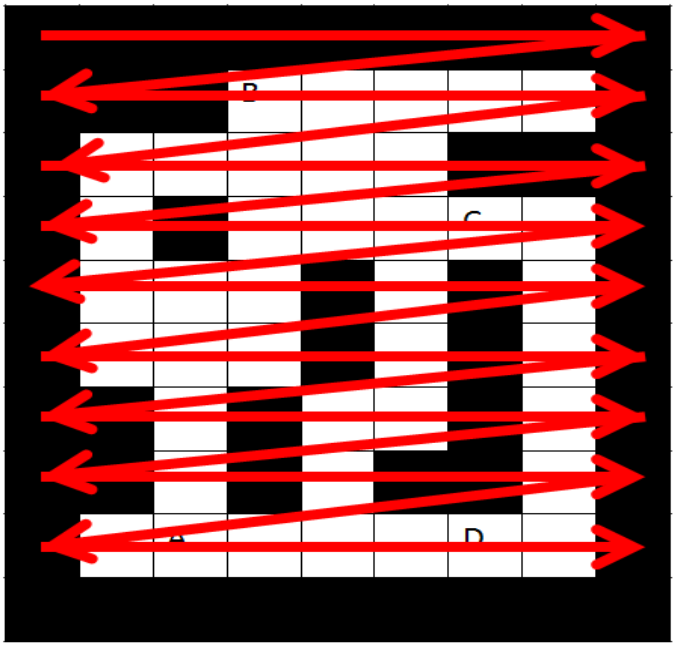
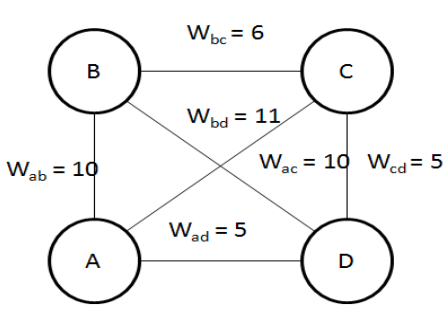
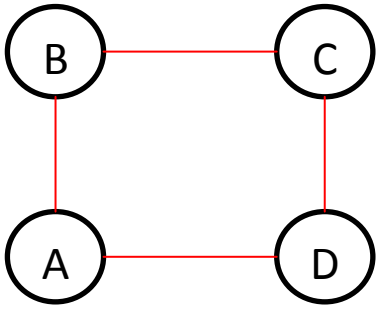


Figure 4

3. Traveling Salesman Problem (TSP)

This section is for task 2: computing the shortest tour starting at checkpoint A. Once you construct the shortest path graph, the problem is transformed to the standard TSP. TSP is a well known NP-Complete problem. Given the fully connected graph from previous section, and suppose the robot is always at checkpoint A in the first place, you need to find a TSP tour that the robot can visit all the checkpoints exactly once and returning to the starting checkpoint (A) with minimal distance. Note that the grader can run task2 independent of task1 – which means before or after task1. Thus, when the grader is evaluating task2, your program MUST compute the shortest path graph first, before solving TSP.





Starting checkpoint for task 2: The starting checkpoint for the robot for Task 2 is always at **'A'**.

For Task 2, you will use A* search and Minimum Spanning Tree (MST) Heuristic. This heuristic is explained in the next subsection.

3.1 The Minimum Spanning Tree Heuristic

Assuming we define a state as below. You are free to define a state using any data structure you think is appropriate. The following example is only provided as a suggestion or demonstration of what might be done.

```
class State{
        String current_ checkpoint;
        List<String> visted_checkpoints;
        double g; \\g(n) = path cost
        double h; \\h(n) = heuristic
    }
```

We want to find the optimal tour in the shortest path graph of the given map in figure 5.

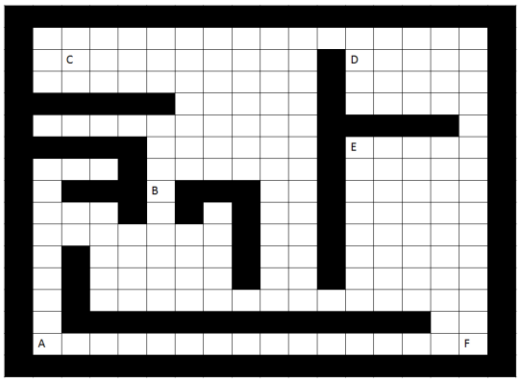
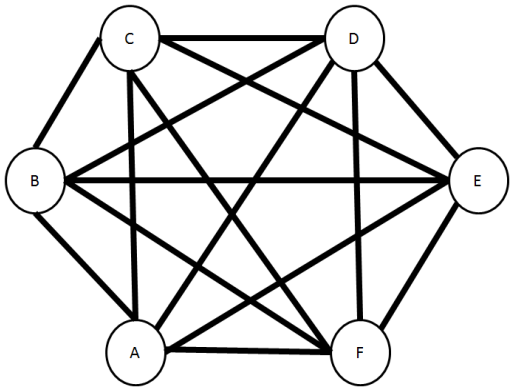
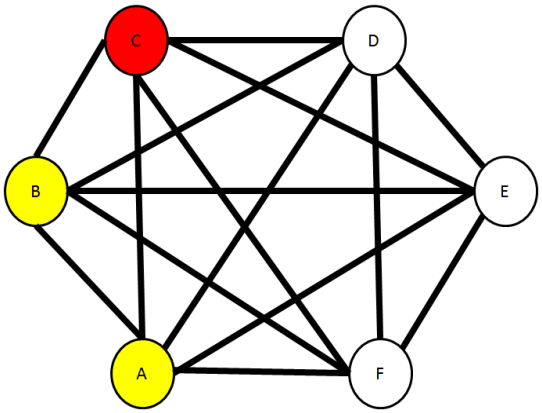


Figure 5

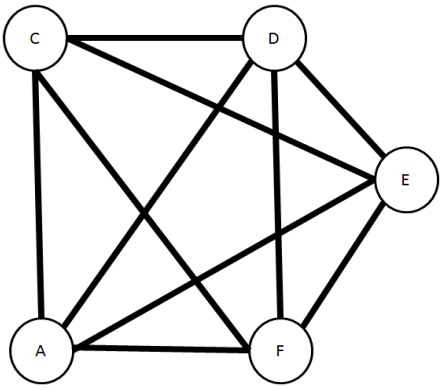
First, we construct the shortest path graph.



The **start** and **goal** node of the TSP are the **same**. Assuming the **start** and **goal** node of this TSP is 'A'. That the current state is **" current_ checkpoint"** = 'C' and **"visted_checkpoints"** = {'A','B'}. Thus, **unvisited checkpoints** is {'d', 'e', 'f' }.



To use the minimum spanning tree heuristic, you must construct a minimum spanning tree of **unvisited** checkpoints (in this case {"d", "e", "f"}) including **a current node** ("c") and **a start/goal node** ("a"). In other word, you compute the MST from the sub graph of union of **unvisited checkpoints**, **current node** and **start node**. In this setting, if the MST heuristic of the current state can be computed from the sub graph from the image below.



An example of a minimum spanning tree would be as follows:

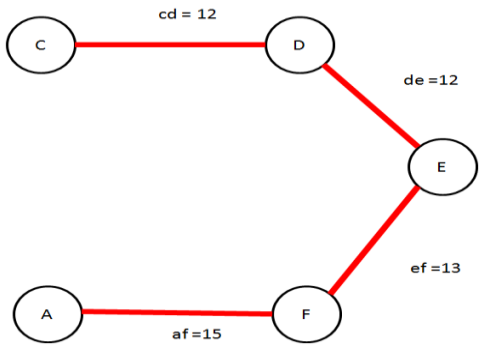


Figure 6

Figure 6: Say the state is (current_ checkpoint = "c", visited_ checkpoints = {"a", "b"}, g = 22, h= (12+12+13+15)) where h("c") is a cost of minimum spanning tree of {"a","c","d","e","f"}. Therefore, in Figure 5, h("c") is 12+12+13+15= 52. This gives you a method to compute the heuristic estimate for nodes at every stage of the A* algorithm.

3.2 How to construct a minimum spanning tree

A spanning tree of a graph is a subset of n-1 edges that form a tree and connects all the vertices together, where n is the number of vertices [1]. A minimum spanning tree (MST) is then a spanning tree with weight less than or equal to the weight of every other spanning tree[2]. The common algorithms to find a minimum spanning tree are Prim's algorithm and Kruskal's Algorithm, which you can find in online resources such as [3][4]. You are free to use any algorithm to find a minimum spanning tree – just specify which one you are using. Prim's Algorithm pseudo code provided below is from [4]. From 3.1, if you want to find a MST of the current state, your graph G in the following pseudo code should be a fully connected graph of vertices {"a","c","d","e","f"}.

- Input: A non-empty connected weighted graph G with vertices V and edges E
- Initialize: $V_{new} = \{x\}$, where x is an arbitrary node (starting point) from V, $E_{new} = \{\}$
- Repeat until $V_{new} = V$:
    - Choose an edge $\{u, v\}$ with minimal weight such that u is in V_{new} and v is not
    - Add v to V_{new}, and $\{u, v\}$ to E_{new}
- Output: V_{new} and E_{new} describe a minimal spanning tree

4. Tasks

In this assignment, you will write a program to implement the following tasks.

4.1 Construct a shortest path graph with A* using **manhattan distance** as a heuristic

4.2 Using a graph from 4.1. Find the optimal tour with A* search using **minimum spanning tree** as a heuristic. Note that the grader can run task2 independent of task1 (meaning before or after task1). Thus, when the grader is evaluating task2, your program MUST compute the shortest path graph first before proceeding to solve TSP.

5. Input

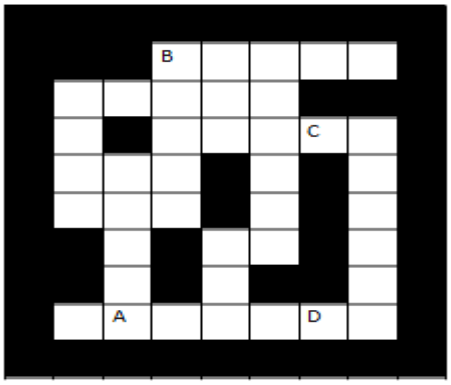
There are two inputs for your programs;

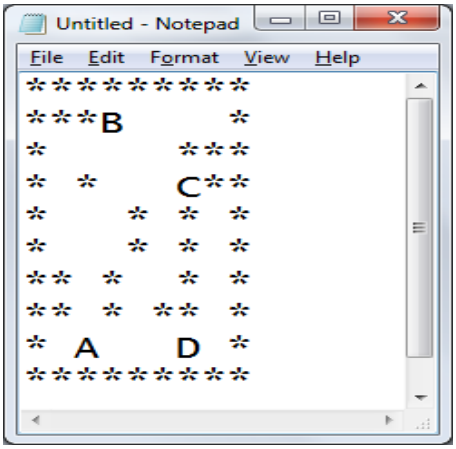
5.1 Which task to perform: there are two possible values

1 ➔ Construct a shortest path graph

2 ➔ A* using MST heuristic

5.2 The map file:





6. Output

In this assignment, we are interested in both how your search algorithms traverse the graph and the path. There are two output files that your program need to produce.

6.1 A shortest path graph (task 1): in a adjacency list format. Each line contain three columns: node1, node2 and weight. Each column is separated by commas. Print the results in alphabetic order.

Example:

a,b,10

a,c,11

a,d,5

b,c,6

b,d,10

c,d,5

6.2 A tour (task 2): List the names of the nodes in the tour (separated by new lines) in order.

Example:

a

b

c

d

e

f

a

Total Tour Cost: 50.0

Please see the complete examples in output1_path_t1.txt, output1_path_t2.txt, output1_path_t3.txt, output2_path_t1.txt, output2_path_t2.txt, output2_path_t3.txt.

6.3 A traverse log(task 1-2):: List tours with the values of g(n) – the path cost to "n" from the start node, h(n) – the heuristic estimate from "n" to the goal state, and f(n) = g(n) + h(n) at each step in the order traversed by your program (separated by new lines). Please see the complete examples in output1_tlog_t1.txt, output1_tlog _t2.txt, output2_tlog_t1.txt, output2_tlog_t2.txt.

Example:

tour,g,h,f

a,0.0,42.69,42.69

ab,5.0,42.69,47.69

ac,7.07,42.69,49.76

acb,12.07,37.69,49.76

7 Program Specifications

7.1 Your program must be written in either Java or C++.

7.2 Your program MUST compile and run on aludra.usc.edu

7.3 Write your own code. Files will be compared and any cheating will be reported.

7.4 Your program name must be "tsp" (without quotation mark).

8. Execution Details

Your program will be tested on aludra.usc.edu on unseen input files that meet the input file specifications. Your program will receive 2 arguments, 2 for inputs and 2 for outputs.

8.1 C/C++

The grader will execute this command:

tsp -t <task> -i <input_file> -op <output_path> -ol <output_log>

Example:

tsp -t 1 -i input1.txt -op output1_path_sgraph.txt -ol output1_tlog_ sgraph.txt

8.2 JAVA

The grader will execute this command:

java tsp -t <task> -i <input_file> -op <output_path> -ol <output_log>

Example:

java tsp -t 1 -i input1.txt -op output1_path_sgraph.txt -ol output1_tlog_ sgraph.txt

8.3 Arguments:

8.3.1 <task> : there are 2 possible values 1 and 2.Note that the grader can run task2 before task1. Thus, when the grader grade task2, your program MUST compute the shortest path graph before solving TSP.

8.3.2 <input_file>: location of an input file.

8.3.3 <output_path>: location of an path output.

8.3.4 <output_log>: location of an traverse log output.

Thus, you should interpret the example:

tsp -t 1 -i input1.txt -op output1_path_sgraph.txt -ol output1_tlog_ sgraph.txt

The first task is chosen. The location of the input file is same as your program and its name is input1.txt. The location of path output file is same as your program and its name is

output1_path_sgraph.txt. Finally, the location of traverse log output file is same as your program and its name is output1_tlog_ sgraph.txt.

9. Grading Policy: (Total Points: 100)

9.1 Programming (90 pts):

Your program must implement the three search algorithms.

9.1.1 Correct outputs for task 1: 40 points

9.1.2 Correct outputs for task 2:40 points

9.1.3 What will happen to the number of iterations (i.e., nodes explored) if we use Euclidean (straight line) distance as the heuristic (rather than Manhattan distance) for Task 1? Explain the reasons for your conclusion: 10 points

9.2 Readme.txt (10 pts):

9.2.1 A brief description of the program structure and any other issues you think will be helpful for the grader to understand your program. (5 pts)

9.2.2 Instructions on how to compile and execute your code. (5 pts)

9.2.3 Please include your name, student ID and email address on the top.

9.2.4 You must submit a program in order to get any credit for the Readme.txt. In short, if you submit ONLY a Readme.txt file you will get 0.

9.2.5 Remember to also include the explanation of Part 9.1.3

10. Submission Guidelines

Your program files will all be submitted via blackboard. You MUST follow the guidelines below. Failure to do so will incur a -25 point penalty.

10.1 Compress and zip ALL your homework files (this includes the Readme.txt and all source files) into one .zip file. Note that only .zip file extensions are allowed. Other compression extensions such as .tar, rar, or 7z will NOT be accepted.

10.2 Name your zip file as follows: firstname_lastname.zip. For example, John_Smith.zip would be a correct file name.

10.3 To submit your assignment, simply select the appropriate assignment link from the Assignments subsection of the course blackboard website. Upload your zip file and click submit (clicking send is not enough).

Please make sure ALL source files are included in your zip file when submitted. Errors in submission will be assessed –25 points. A program that does not compile as submitted will be given 0 points.

Only your FINAL submission will be graded.

For policies on late submissions, please see the Syllabus from the course home page.

11. References

1. http://en.wikipedia.org/wiki/Minimum_spanning_tree

2. http://mathworld.wolfram.com/MinimumSpanningTree.html

3. http://en.wikipedia.org/wiki/Kruskal%27s_algorithm

4. http://en.wikipedia.org/wiki/Prim%27s_algorithm