CSCI561 – Introduction to Artificial Intelligence
Instructor: Dr. K. Narayanaswamy
Assignment 2 – Greedy and A* Search Algorithms
Euclidean Travelling Salesman Problem
Due: 10/11/2013 11:59:59pm

1. Introduction

In this project you will solve the Euclidean traveling salesman problem using Greedy and A* search algorithm with two heuristics: 1) straight line distance 2) minimum spanning tree.

Our version of the TSP problem is defined as follows: Given a set of cities and distances between every pair of cities, and starting from a particular city, find the shortest way of visiting all the cities exactly once and returning to the starting city.

You can imagine the cities as nodes in a completely connected graph, and distances as the edge cost between the nodes representing cities. Now the TSP problem is transformed to finding the shortest tour of the graph. TSP is a well known NP-Complete problem.

In this assignment, only a list of cities with their coordinates is given as input. Thus, you need to construct a graph by yourself. An image below shows how to construct a graph from a city list.

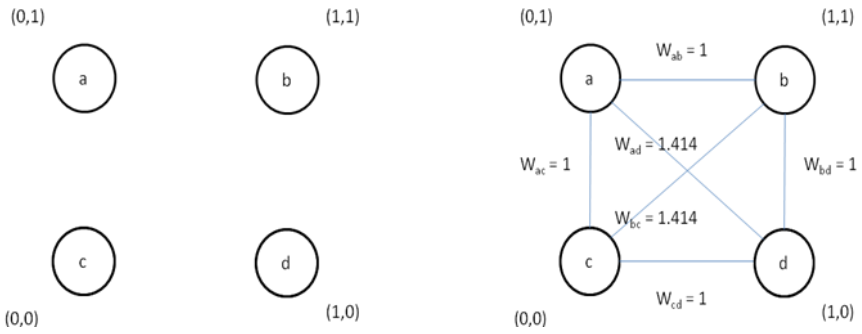

Figure1

There are 6 edges in this example. For simplicity, let us assume that weight of each edge is given by the length of the straight line between the corresponding cities. Assume that p and q are cities, with the location of p= (p1,p2) and location of q = (q1,q2). The weight of the edge between p and q can be computed by

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}.$$

For example, a straight line between a = (0,1) and d= (1,0) is 1.414

An example of desired output is shown in the image below. The shortest way of visiting all the cities exactly once and returning to the starting city is shown in the right image.

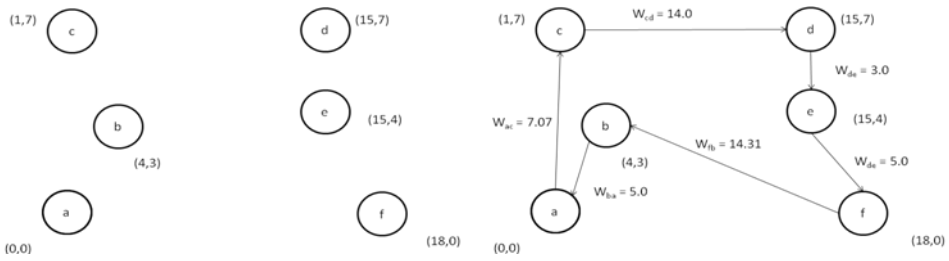

Figure 2

2. Tasks

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

2.1 Find a tour using Greedy search

2.2 Find the optimal tour with A* search using **straight line distance** as a heuristic

2.3 Find the optimal tour with A* search using **minimum spanning tree** as a heuristic

3. Illustrative Examples

3.1 Greedy search: Assuming that the initial state is "a"

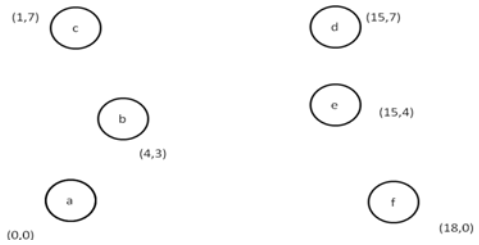

Figure 3

Greedy Search will pick the **nearest unvisited node** to "a" as the next node. In this case, it will pick "b" as the next node because "b" is nearest unvisited node to "a" ($W_{ab} = 5.0 < W_{a*}$). Then, it will pick "c" as the next node because "c" is nearest unvisited node to "b" ($W_{bc} = 5.0 < W_{b*}$).

In task 2 and 3, assuming we define a state as below. You are free to define a state using any data structure you see fit. This example is only used for demonstration.

```
class State{

    String current_city;
```

List<String> visted_cities;

double g; \\g(n) = path cost

double h; \\h(n) = heuristic

}

3.2 A* search using straight line distance as a heuristic. A straight line is a straight line between current node to the goad node.

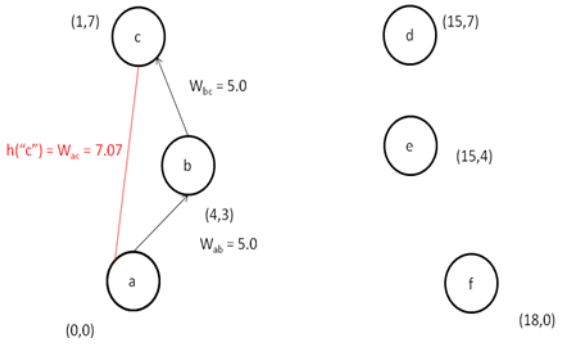

Figure 4

Figure3: Say that the state is (current_city = "c", visited_cities = {"a", "b"}, g = 10, h= 7.07). The goal node is "a". h("c") is given by the straight line between "c" and the goal node ("a").

3.3 A* search using minimum spanning tree as a heuristic. To use the minimum spanning tree heuristic, you must construct a minimum spanning tree of **unvisited** cites (in this case {"d", "e", "f"}) including **a current node** ("c") and **a goal node** ("a"). An example of a minimum spanning tree would be as follows:

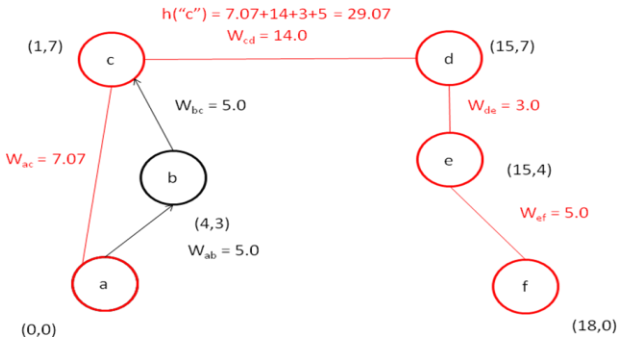

Figure 5

Figure 4: Say the state is (current_city = "c", visited_cities = {"a", "b"}, g = 10, h= (14+3+5+7.07)) where h("c") is a cost of minimum spanning tree of {"a","c","d","e","f"}. Therefore, in Figure 4, h("c") is 7.07+14+3+5 = 29.07. This gives you a method to compute the heuristic estimate for nodes at every stage of the A* algorithm.

3.3.1 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 for finding a minimum spanning tree are Prim's algorithm and Kruskal's Algorithm, which you can find discussed in online resources such as [3][4]. You are free to use any algorithm for finding a minimum spanning tree – just document which one you are using. The Prim's Algorithm pseudo code provided below is from [4]. From 3.3, if you want to find a MST of the current state, your graph G in the below 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. Input

There are three inputs for your programs;

4.1 Which task to perform: there are three possible values

1 ➔ greedy search

2 ➔ A* using SLD heuristic

3 ➔ A* using MST heuristic

4.2 The start node

Single letter between a and z – case insensitive

4.3The city list: Each line represents a city. All cities will be indicated by letters of the alphabet. All cities are assumed to be fully connected. You need to construct a graph by yourself. A graph is a fully connected undirected graph. Weight between two cities can be computed by Euclidean distance between two cities as shown in the section 1. The nodes in the figure 2 will be represented in a input file as

a,0,0

b,4,3

c,1,7

d,15,7

e,15,4

f,18,0

5. 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 will produce.

5.1 A tour: 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.

5.2 A traverse log: 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, output1_ tlog_t3.txt, output2_tlog_t1.txt, output2_tlog_t2.txt, output2_tlog_t3.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

6 Program Specifications

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

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

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

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

7. 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 5 arguments, 3 for inputs and 2 for outputs.

7.1 C/C++

The grader will execute this command:

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

Example:

tsp-t 1 -s a -i input1.txt -op output1_path_bfs.txt -ol output1_tlog_bfs.txt

7.2 JAVA

The grader will execute this command:

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

Example:

java tsp -t 1 -s a -i input1.txt -op output1_path_bfs.txt -ol output1_tlog_bfs.txt

7.3 Arguments:

6.3.1 <task> : there are 3 possible values 1, 2 and 3.

6.3.2 <start_node>: the start node

6.3.4 <input_file>: location of an input file.

6.3.5 <output_path>: location of an path output.

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

Thus, you should interpret the example:

```
tsp -t 1 -s a -i input1.txt -op output1_path_greedy.txt -ol output1_tlog_ greedy.txt
```

The first task is chosen. The start node is "a". 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_greedy.txt. Finally, the location of traverse log output file is same as your program and its name is output1_tlog_ greedy.txt.

8. Grading Policy: (Total Points: 100)

8.1 Programming (90 pts):

Your program must implement the three search algorithms.

8.1.1 Correct outputs for task 1: 20 points

8.1.2 Correct outputs for task 2: 30 points

8.1.3 Correct outputs for task 3: 30 points

8.1.4 Your analysis of similarities/differences between task2 and task3 heuristics. Your explanation must be included as part of readme.txt: 10 points

8.2 Readme.txt (10 pts):

8.2.1 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)

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

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

8.2.4 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.

7.2.5 Remember to also include the explanation of outputs (Part 7.1.4)

9. 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.

9.1 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.

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

9.3 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.

10. 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