Ryan Bridges - A09393354
Cole Stipe - A10632377
3/14/15
CSE 101
Professor Kahng

Implementation Project

System Specification

CPU: Intel Core i7-950 3.8 GHz
Platform: Windows 7 Ultimate 64Bit
Compiler: Microsoft Visual Studio 2013

Our secret Numbers: W = 12705, H = 15640, N = 8146

Deliverable A

(1) Algorithm description in English for constructing the MST

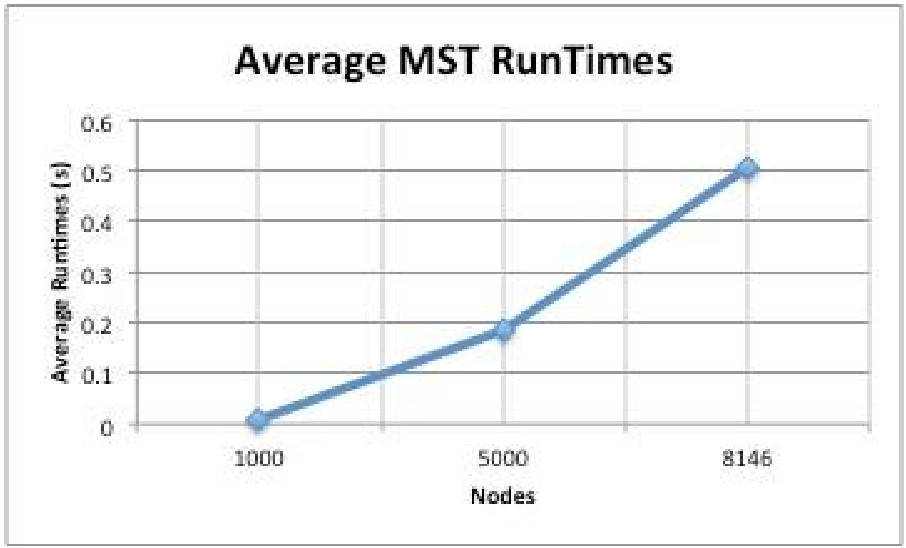
We use Prim's Algorithm to construct our MST. Prim's algorithm begins with a single vertex in a forest of unconnected vertices and builds the MST by always selecting the minimum weight edge that connects a vertex already in the tree until the MST is built.

(2) Running time analysis (big-O and empirical) based on (1)

The runtime for Prim's Algorithm is $O(|V|^2)$ when using an adjacency matrix.

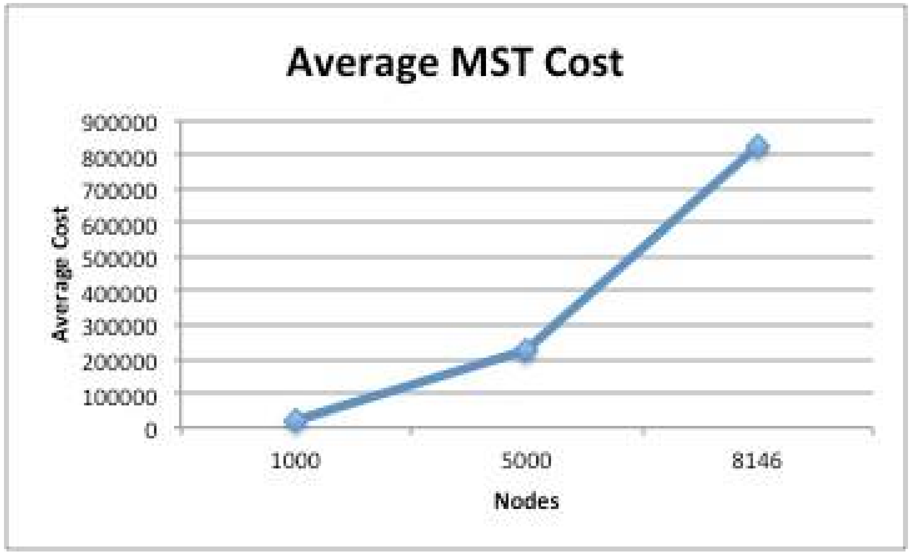
Below is a graph of the average MST runtimes for our secret number as well as 1000 vertices and 5000 vertices for comparison.

- Mean Runtime 8146 Vertices: 0.5023 seconds
- Sample Standard Deviation Runtime 8146 Vertices: 0.0216363 seconds
- Number of Trials: 100



(3) Mean and standard deviation of MST cost for given W, H, N

- Mean Cost 8146 Vertices: 825139
- Sample Standard Deviation Cost 8146 Vertices: 3366.12
- Number of Trials: 100



(4) Description of methodology for implementation

We used an adjacency matrix to select a node to start from. We then proceed greedily, growing the tree by one edge each iteration by searching the adjacency matrix for the next smallest edge. We repeat this process until all vertices are included in the MST, and we are sure to never add an edge that would create a cycle

Deliverable B

(1) Algorithm Description in English For TSP2 Calculation Code

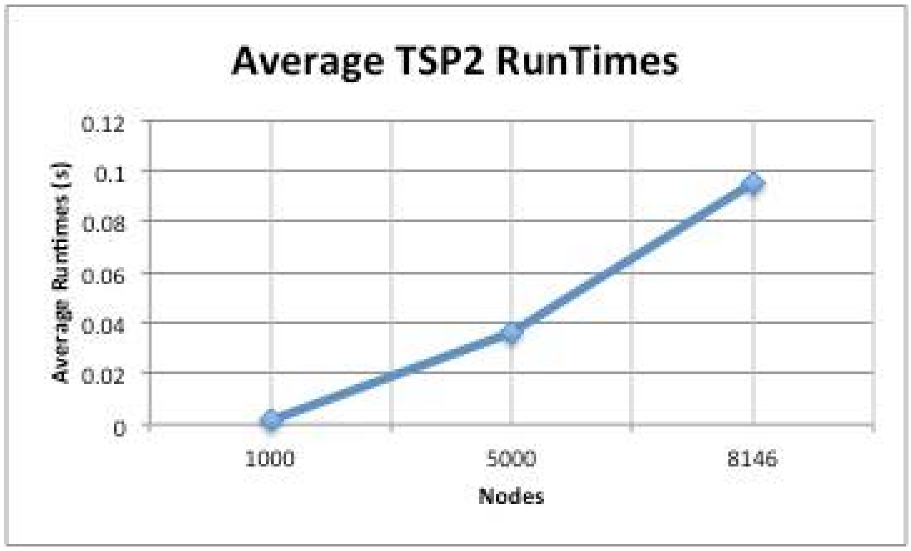
To compute the heuristic TSP cost of the Eulerian tour generated by DFS of the MST, we run a modified version of the Depth First Search algorithm on the MST. The algorithm adds edges in the path taken by DFS unless it is forced to backtrack. When it is forced to backtrack, it finds a shortcut in the graph to the next unvisited vertex and adds that edge instead.

(2) Running time analysis (big-O and empirical) based on (1)

The run time for the DFS shortcut algorithm is O(|E| + |V|) because the algorithm runs DFS on the graph and only performs an extra step when it determines that a shortcut must be found. The shortcut and be computed in constant time. Therefore, the overall runtime is unaffected and remains O(|E| + |V|).

Below is a graph of the average runtimes for for our secret number as well as 1000 vertices and 5000 vertices for comparison..

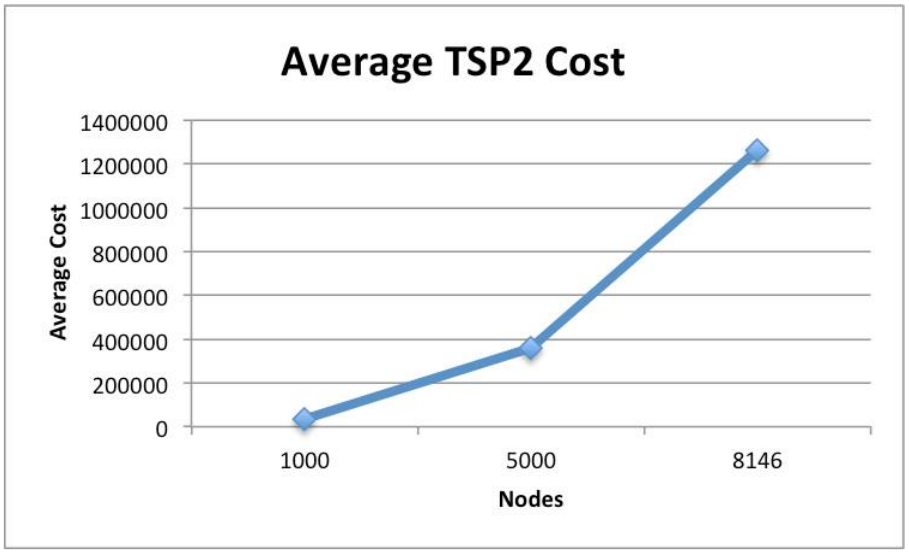
- Mean Runtime 8146 Vertices: 0.09526 seconds
- Sample Standard Deviation Runtime 8146 Vertices: 0.000579446 seconds
- Number of Trials: 100



(3) Mean and standard deviation of TSP2 cost for given W, H, N

Below is a graph of the average costs for our secret number as well as 1000 vertices and 5000 vertices for comparison.

- Mean Cost 8146 Vertices: 1.26401e+006
- Sample Standard Deviation Cost 8146 Vertices: 13715
- Number of Trials: 100



(4) Description of methodology for implementation

The modified DFS uses recursion to reach a leaf node in the MST. Along the path to a leaf node, we add the edges we take to the heuristic TSP cost. When a leaf node is encountered, we know that we will be forced to go back up the tree. Therefore, we set a "shortcut" flag to true, store the leaf node, and do not add edge lengths as we recurse back up the stack. Only when we encounter a node in DFS that has not yet been visited do we add another edge weight to our heuristic TSP cost. Since the "shortcut" flag is set to true when we encounter an unvisited node, we compute the edge cost between the stored leaf node and the new node. Then we add this to our heuristic TSP cost and reset the "shortcut" flag back to false. We repeat the process of finding leaf nodes, adding edge weights, and shortcutting until the entire graph is traversed. Then we return the heuristic TSP cost.

Deliverable C

(1) Algorithm Description in English For TSP1p5 Code

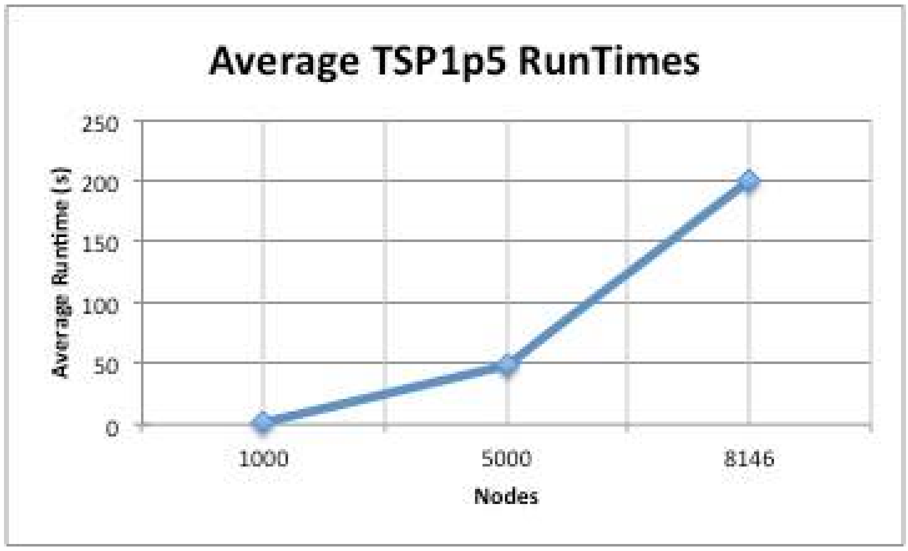
To compute the heuristic cost for TSP1p5, we locate the odd vertices in the MST and find the minimum-weight perfect matching between them. We then add these edges to the MST and run a modified version of Fleury's algorithm that uses shortcutting to compute the heuristic.

(2) Running time analysis (big-O and empirical) based on (1)

The runtime for the TSP1p5 code consists of the time it takes to identify odd degree vertices, perform the minimum-weight perfect matching, combine the graphs, and then run Fleury's algorithm with shortcutting. Identifying the odd degree vertices is O(|V|) because there must be fewer odd degree vertices than all the vertices in the graph. According to the documentation for the Blossom V algorithm, the current runtime is O(|V|(|E| + |V|*log(|V|))). Combining the perfect matching with the MST can be done in O(|E|) time since no more than |E| edges can be added to the MST. Finally Fleury's algorithm takes O(|V|*(|V| + |E|)) time since we run DFS on every vertex to determine the next bridge. Therefore, the overall time complexity is that of Blossom V, which is O(|V|(|E| + |V|*log(|V|))).

Below is a graph of the average runtimes for our secret number as well as 1000 vertices and 5000 vertices for comparison.

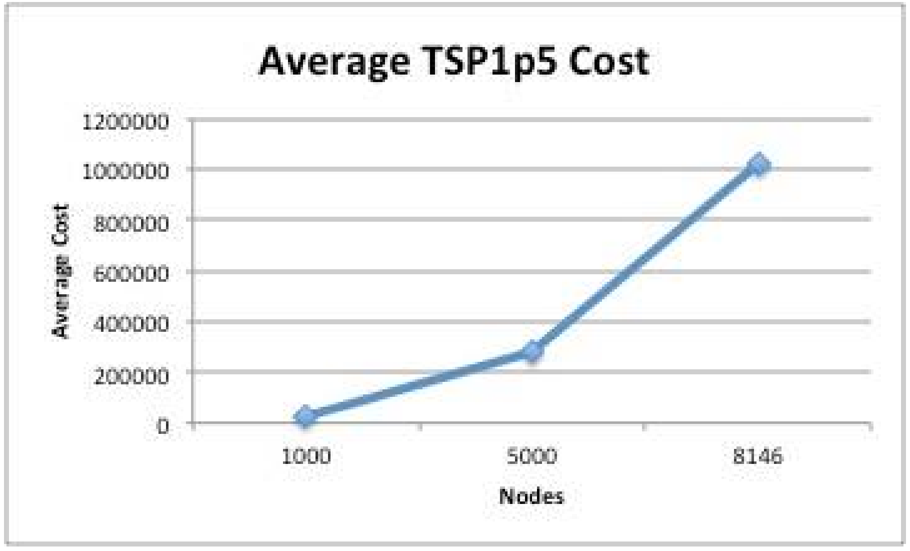
- Mean Runtime 8146 Vertices: 200.748 seconds
- Sample Standard Deviation Runtime 8146 Vertices: 3.49179 seconds
- Number of Trials: 100



(3) Mean and standard deviation of TSP1p5 cost for given W, H, N

Below is a graph of the average costs for our secret number as well as 1000 vertices and 5000 vertices for comparison.

- Mean Cost 8146 Vertices: 1.01821e+006
- Sample Standard Deviation Cost 8146 Vertices: 8621.69
- Number of Trials: 100



(4) Description of methodology for implementation

For TSP1p5, we start by finding the odd degree vertices in the MST we already generated. We then use the original adjacency matrix to create a fully connected graph from the subset of odd vertices in the MST. We pass this new graph to the code that finds the minimum-weight perfect matching. When the minimum weight perfect matching is algorithm is complete, we add the edges to our MST to create a combined graph where each vertex is now of even degree. Then we run a modified version of Fleury's algorithm to find an Eulerian tour within the combined graph and shortcut whenever we encounter a node we have already visited. The algorithm is as follows. We begin at an arbitrary start vertex, *v*, and scan *v*'s adjacent vertices. For each edge, (*v,u*), we check if how many vertices would be reachable from *u* if we were to remove (*v,u*). This is done with a DFS traversal starting from *u*. If removing (*v,u*) does not affect the number of vertices reachable from *u*, we travel across the edge to *u*, mark *v* as visited, remove the edge (*u,v*) from the graph, and the weight of edge (*u,v*) to our TSP1p5 heuristic cost. However, if *u* has already been visited in our Eulerian Tour, we store vertex *v* so we know to shortcut it later. Then we remove the edge (*u,v*) from the graph and follow the path to *u*, but we **do not** add the edge weight to our heuristic. Instead, we let Fleury's algorithm continue until it reaches a vertex which has not yet been visited in its traversal. When it reaches a vertex that has not been visited, *k*, we shortcut everything in between and add the weight of edge (*v,k*) to your TSP1p5 heuristic.

References

[1] Prim’s Algorithm, http://en.wikipedia.org/wiki/Prim%27s_algorithm
[2] Depth First Search, http://en.wikipedia.org/wiki/Depth-first_search
[3] Blossom V, http://mpc.zib.de/index.php/MPC/article/viewFile/11/4
[4] Fleury’s Algorithm, http://en.wikipedia.org/wiki/Eulerian_path#Fleury.27s_algorithm