5611 HW3: The Travelling Salesman Problem

Shane Harding
10309405

February 16, 2015

1   Introduction

This report details everything that I did in this assignment and why I did things the way I did.

2   Task 1

There are multiple C files to this task that are then all compiled together via the makefile.

The first thing I did was write a function that computes the distance between two points on a two dimensional plane (written in `dist.c`). This does not give the proper Euclidean distance since I do not compute the square root (just $x^2 + y^2$ is calculated). The reason for this is that square root calculations are expensive and not necessary in this case. They aren't necessary because we are just comparing distances and if d_1^2 is bigger than d_2^2 then d_1 is bigger than d_2.

In my main file, the first thing I do is deal with the command line arguments and flags. So only one or zero flags are allowed. If there are no flags then the number of cities is simply set to 10. The `-f` flag allows for a configuration of cities to be loaded from a specified file. The `-n` flag generates a specified number of random cities on a grid.

Once all the coordinates of the cities are loaded/generated a distance matrix is set up. This puts the distance between cities i and j in the matrix entry (i, j). The initial ordering of the cities set to be $0, 1, ..., n-1$. The permutation function is then called to generate all the possible routes. At this stage it is important to note that I am solving this problem for closed paths. So the first point and the last point are connected. This means that our starting city does not make any difference (because the path taken is always going to be a closed loop). Which then means that we don't have to move it in our permutations. And we also have $(n-1)!$ possible paths.

There is a swap function that simply swaps two elements in an array. This is needed in my function that generates all the permutations of the n cities. The permutation function writes all of the permutations to a file. The reason for this is that the permutation function is called recursively within itself. This

made returning a value from it very difficult and I found it much easier to just write all the permutations to a file and then read them from the file when they were required.

Also, since I wrote to the file and then wanted to read from it. It meant that I had to close if and open it again, because I couldn't find a way to just open it in both read and write mode; it had to be one or the other.

Once all the possible permutations (routes) are read, the distances of each is calculated (by simply adding the relevant elements of the distance matrix) and the shortest one is returned. This is done in the `shortest_route.c` function. The shortest path is then printed to the terminal. And all the mallocs are then freed.

3 Task 2

4 Task 3

5 Task 4

Task 4 is this report. And I added the section before realising how stupid adding it was, and had a good laugh at myself.