CS 3005: Programming in C++

The Traveling Salesman Problem (Part 2 - Good Cycle)

The Traveling Salesman Problem (TSP) is a classic NP-Complete problem. Solutions to the problem can be applied to many fields.

There are several variations of the problem. For this assignment we will use the following description. As you search for more information and solutions, be sure that they apply to this version.

Definition

Given a complete graph G, with n vertices, v_1, v_2, ..., v_n, and edge weights w(u, v), such that w(u, v) == w(v, u) and the edge weights obey the triangle inequality, find the lowest weight cycle that visits all vertices.

The triangle inequality means that it is never more costly to go from vertex a to c, than it is to go from a to b then to c.

A cycle that visits all vertices must follow a path from the starting location, v_1, and visit every vertex exactly once, then return to the starting location v_1.

As mentioned above, finding an optimal solution to this problem is known to be NP-Complete. The consequence of this is that for any graph with more than a few tens of nodes, it is not feasible to find an optimal solution.

Assignment

Since finding an optimal solution is not feasible, in this step of the assignment, you will be required to find a good cycle of a graph.

There are many known approximation algorithms for TSP. You may implement any one algorithm you find, including the two described below. However, your chosen algorithm must be able to achieve a cycle quality of at least 0.80 on all of the sample graphs. Also, the algorithm must take no more than 60 CPU seconds on any one of the sample graphs, when run on oxygen or nitrogen.

Algorithm 1 - Hill-Climbing-TSP

Hill-Climbing is a general algorithm technique where a random solution is found and evaluated. Then, neighbor solutions (similar solutions) are found and evaluated. If any of the neighbors are better than the original, the original is replaced. This process is repeated until no better neighbors are found.

Pseudo-Code

```
set best evaluation to infinitely bad, and best cycle to empty
set current evaluation and current cycle from a random cycle
while current evaluation is better than best evaluation
    update best evaluation and best cycle from current evaluation and current cycle
    repeat as much as desired
        set tmp evaluation and tmp cycle from a neighbor of best cycle
        if tmp evaluation is better than current evaluation
            set current evaluation and current cycle from tmp evaluation and tmp cycle
return best cycle
```

Be careful, [repeat as much as desired], may cause your program to not find a good enough solution, or may cause your program to run too long.

Algorithm 2 - Greedy-TSP

Greedy is a general algorithm technique where the next step in the algorithm is chosen to make the best advancement given the current state. Then, the state is updated, and the process is repeated until a solution is found. It usually does not lead to an optimal solution.

Pseudo-Code

```
set solution collection of edges to be empty
```
sort all edges by weight in ascending order
for each edge
    if either node in the edge already has 2 edges in the solution,
        skip this edge
    else if adding the edge to the solution would produce a cycle in the solution,
        skip this edge
    else
        add the edge to the solution
return the solution

Before you implement this algorithm, you may want to spend a few minutes thinking about the data structures and algorithms you will need to answer the questions in the two if statements.

**Requirements**

- All requirements from part 1.
- Program must run on any sample graph in no more than 60 CPU seconds.
- Program must obtain a cycle quality at least 0.80 on all sample graphs.
- Program executable file must be called `TSP-GOOD`.

**Timing Files**

- `zip`
- `tgz`

**Show Off Your Work**

To receive credit for this assignment, you must upload the source code (.h and .cpp files) and the Makefile to the CIT submission system linked at the top of the page.

Additionally, the program must build and run. Any incorrect performance or memory errors will be counted against the assignment score.