CS 456 : Advanced Algorithms

Programming Assignment #03

Total Points: 150

Assigned Date : Wednesday, April 04, 2018
Due Date : Wednesday, April 18, 2018 @ 02:59:59 p.m.

Overview

For your final programming assignment, you will implement algorithms to solve the famous NP-Hard **Traveling Salesman Problem (TSP)**. You will implement four algorithms: naive brute-force, branch-and-bound, dynamic programming (i.e. Held-Karp Algorithm), and a polynomial time approximation algorithm using Minimum Spanning Trees (e.g. Christofides Algorithm). See `https://goo.gl/FN99ig`.

Your input will be a set of points in 2D space. You can travel from any point to any point in any order you wish. You must then determine the shortest Hamiltonian Circuit of these points where cost is simply the Euclidean Distance.

Your output will be a simple ordered list of vertices that is the solution and its total length.

If you can come up with a polynomial time algorithm that is guaranteed to produce the optimal path, you will win a million dollars since you have shown that P=NP. `https://en.wikipedia.org/wiki/Millennium_Prize_Problems`.

Instructions

- This is an individual assignment. **Do your own work**.
- **Start early!!**
- **Take backups of your code often!!**.
- You may use any programming language of your choice. However, you **must** make sure that your code compiles and runs on a typical Linux machine.
- It is **highly** recommended that you test your program's compilation and execution on the home server, `home.cs.siue.edu`, before submitting.
- Absolutely **DO NOT** include executables with your submissions.
- You **MUST** submit a makefile. If your program does not need to be compiled, then have the makefile output instructions on how to execute your program instead.
- The report part of your solution must be produced using a word processor. I highly recommend LaTeX. The report must be in `.pdf` format.
- Any figures, graphs, plots, etc., should also be produced using appropriate computer applications.
- Graphs/plots should be properly labeled.

- Your final report must be in **.pdf** format. No exceptions.
- Follow a good coding standard. Use the Google C++ coding standard found here `http://goo.gl/1rC1o`, if you don't already follow one.
- Include only the things necessary to run your program in your tarball.
- Include any additional instructions in a readme file if needed.
- Each algorithm should be their own program that takes a **command line argument which is a filepath** to an input file. Output can simply be a print to the console of the complete path and the total cost of that path.

Deliverables

The due date of this assignment is **Wednesday, April 18, 2018 @ 02:59:59 p.m.** A dropbox will be opened for submission on Moodle before the due date. A complete submission includes:

- Report (in `.pdf` format)
- A compressed tarball that comprised of the `Makefile` and your source code (directory)

For example, submission to moodle:

`lastname.tar.gz lastname.pdf`

The tarball should be such that when it is decompressed it will create a directory that is named your last name. Your makefile should be in the base directory created. Executables (created by issuing your `Makefile`) should end up in the same directory as the `Makefile`.

DO NOT use any source code found online. The grader will actively check for possible plagiarism. You can use pseudocode found online with proper citations in your report.

Report [**120 points**]

Your report should be structured as follows:

- Motivation and background of the experiment [**5 points**].
- For each algorithm, in their own sections, [**25 points**] per algorithm:
  - Pseudocode of the algorithm [**5 points**].
  - Correctness Proof [**5 points**].
  - Problems Encountered/Key insights [**5 points**].
  - Test results. This should include a graph that shows the execution runtimes of that particular program. You should use input sizes that are big enough to insure that the main factor in the runtime is the problem size, and also use enough inputs that the graph shows a clear pattern. Part of this section for the MST approximation algorithm should include comparison and justification of runtime versus solution accuracy as well as a numerical summary of the solution accuracy for all of your tests. Is the trade off worth it for this particular algorithm?[**5 points**]
  - Observations on and justification of results. If something seems odd, reason about it and offer an explanation about why it is the way it is. [**5 points**]
- Conclusion and performance comparisons. This includes comparing the different algorithms against each other in terms of speed as well as accuracy in this case since the approximation algorithm will most likely not generate an optimal result. as well as how what difference the two data

structures made. A graph comparing the run times to each other would be most appropriate.[**20 points**].

Executables [30 points]

A compressed tarball of the directory containing your source codes and Makefile. Do not include executables in this tarball; we will do a fresh compile of your code using your Makefile. Do not include test files, we will use our own. To create a compressed tarball of the directory source, use the following command: `tar -zcvf name.tar.gz source/`. Change name to your last name. The only files it should have is a Makefile and source code files.

- Program correct accepts command line input [**2.5 x 4 points**]
- Program produces correct output [**2.5 x 4 points**]
- Good coding practices e.g. naming conventions, readable code, commenting, etc. [**2.5 x 4 points**]

Extra Credit

Being able to utilize parallelization in program execution is an important ability, especially today when processing speedups are coming from the inclusion of multiple processing cores instead of simply faster processing cores.

For extra credit, create a pseudo code for each algorithm that shows how to successfully execute all or part of each algorithm in parallel. [**5x4 points**]
The format for the input file will be one point per line, starting with the x value, then a space, then the y value. The ID for the points will simply be the line it is on, starting at 0.

Input Example

With an input file of:
0 0
1 3
4 10

We will have point 0 at (0,0), point 1 at (1,3), and point 2 at (4,10).