

Coding 2: Travelling Salesman Problems

(Last Submission Date: Thursday December, 1st at 11:59pm)

Goal: Practice implementation of an algorithm using techniques learned in CS3xx courses to compare algorithms learned in CS483.

Activity: Given a graph solve the TSP problem by (1) testing every possible permutation of edges and (2) by using a simple approximation algorithm.

Teamwork: Pairs within the same section of CS483 are permitted, however... the assignment is slightly different for pairs [see below]. *One set of code should be produced, but both students must create a submission and upload that submission to Blackboard for credit.*

Details

Motivation

The Travelling Salesman Problem is a classic NP-Hard problem: find the minimum cost path that visits every node in a graph. Even in its “decision problem” form it is NP-Complete (“is there a path that visits every node with at most cost k ”). The simplest brute force solution to the problem tries every valid permutation of vertices to visit and sums up the cost (for a runtime of approximately $n!$). A number of improvements exist that reduce it down near-ish to $O(2^n)$, but that is still an intractable problem.

We’re going to work on a Euclidean TSP problem where (1) the “vertices” in the graph represent cities with a physical location in the world, and therefore (2) they satisfy the triangle inequality (it is faster or equally as fast to go directly to your destination, than to go somewhere else and then to your destination). This is a very common variation of the problem which is still hard, but has a simple approximation algorithm that we can implement with the tools we’ve discussed in CS483: use Prim’s Minimum Spanning Tree algorithm to get a MST, perform a Depth First Traversal of that tree and visit the cities in the order you encounter them (making it a pre-order DFT).

In this project you’re going to implement both the simple brute force try-every-possibility algorithm (we’ll call this TSP-Permutation) as well as the approximation algorithm (we’ll call this TSP-Approx) in a larger framework that will allow you to visualize the algorithm steps. I’m providing some guidance in this document, but there are very specific instructions in the code base to allow you to complete this project.

The Simulator

This project has a large code base where you will be inserting your contribution. The majority of this code is there to provide you with a visual display of your work once you are done. As it is, the code base can be compiled and run with the following commands from your user directory on Windows:

```
javac -cp .;483libs.jar *.java
java -cp .;483libs.jar SimGUI
```

Or the following commands on MacOS/Linux:

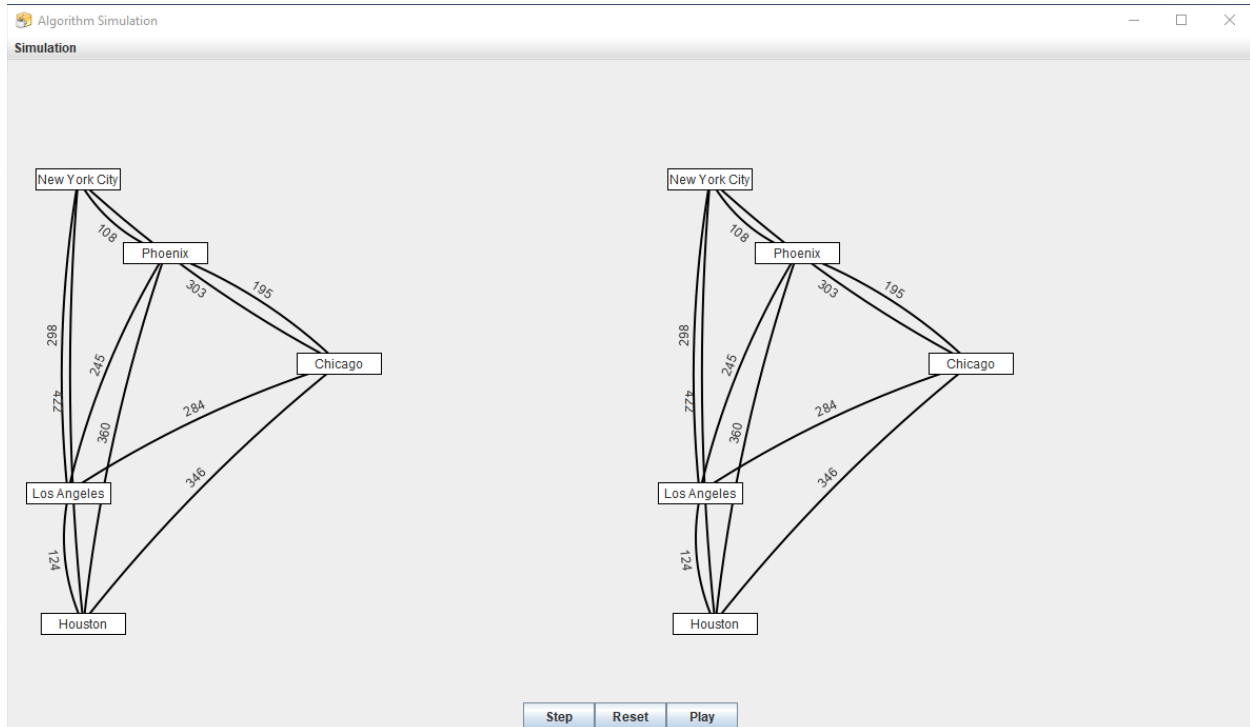
```
javac -cp .:483libs.jar *.java
java -cp .:483libs.jar SimGUI
```

Approach

Why is there extra stuff in the command? The `-cp` is short for `-classpath` (meaning "where the class files can be found"). The `.;483libs.jar` or `.:483libs.jar` has the following components:

- the current directory
- `;` or `:` the separator for Windows or Linux/MacOS respectively
- `483libs.jar` the provided jar file which contains the library code for JUNG (more later)

If you compile and run, you should see the simulator open, even though you haven't written any code yet:



A randomly generated graph will be displayed twice (once for each algorithm), with random city names and edge distances appropriate to the location where they were first created (as shown). You can click and drag cities around for better viewing, but this does not alter the distances.

There is a "step" button and "play" button which will play the algorithms (but only after you write them!). The "reset" button should work out-of-the-box, it will just give you a new random graph of the same type.

The following three commands will also work to run the simulator with more parameters:

```
java -cp .;483libs.jar SimGUI [number-of-cities]
java -cp .;483libs.jar SimGUI [number-of-cities] [rng-seed]
java -cp .;483libs.jar SimGUI [number-of-cities] [rng-seed] [ms-delay]
```

[number-of-cities] can be between 2 and 10, [rng-seed] can be any integer value, and [ms-delay] controls the simulation drawing-delay (minimum 1ms). For example, if you want 6 cities, generated with a random number generator seeded with 1034, and a 1ms delay for drawing, you'd use:

```
java -cp .;483libs.jar SimGUI 6 1034 1
```

Calling with no parameters gives defaults of 5, 0, and 100 to the above parameters respectively.

The JUNG Library and Provided Code Base

The JUNG library (Java Universal Network/Graph Framework, <http://jung.sourceforge.net>) provides a lot of cool visualization tools for graphs, such as automatic layouts, an easy interface for creating/editing graphs, and much more. The remainder of the code base (except `PermutationIterator`) was written by us utilizing the JUNG library heavily. `PermutationIterator` has a header with information on its origin and licensing information for those interested.

Below is an overview of the code base provided:

- `SimGUI` – The simulator itself.
- `GraphNode` – a general-purpose graph node class for the simulator
- `City` – extends graph node class for city-specific things (like city names)
- `GraphEdge` – a general-purpose graph edge class for the simulator
- `Flight` – extends graph node class for flight-specific things
- `FourEightyThreeAlg` – an interface for all CS483 algorithms for use in the simulator
- `TSPAlg` – an abstract class with a lot of helper code for TSP algorithms (esp. displaying them)
- `TSPPermutation` – code to display the TSP-by-permutation an algorithm step-by-step
- `PermutationIterator` – a helper class for `TSPPermutation`
- `TSPApprox` – code to display the TSP-by-approximation an algorithm step-by-step

The Algorithms (Your Code)

Your task is to complete the implementation of `TSPPermutation` and `TSPApprox`. Each of these files has “YOUR CODE HERE” comments which indicate where you need to put your work, and there are detailed instructions in those files as well. However, below is a quick summary of what you need to do:

1. TSP-Permutation – Generate all permutations of cities the salesman can visit, and test them to find the minimum value path.
2. TSP-Approx – Generate a minimum spanning tree and do a depth-first traversal to determine the approximate path the salesman should take.

IMPORTANT: Read over the provided code and all the comments in `TSPAlg`, `TSPPermutation`, and `TSPApprox` before starting, there is a lot of work you are not doing.

Teamwork Option

If you want to work in teams, the only change in the assignment is that you need to run a series of three “interesting” experiments with your final results. For each of the three experiments, you need to choose a set of parameters (# cities and seed #) and set the ms-delay to something small enough to run the simulation 10 times (like ms-delay of 1). Each of the experiments should be *sufficiently different* so as to be interesting (hint: do not just change one of these parameters once).

Here’s what you’ll need to do:

1. Change the provided **optional-team-report.xlsx** to remove the word optional and add the GMU username of each team member, e.g.: **team-report-USERNAME1-USERNAME2.xlsx**
2. Open the team report and examine what you need to put in it:
 - a. At the top, enter the explanation of why you think the experiments will be interesting (write this before you run them!).
 - b. For each experiment, enter the parameters at the top and run the experiment on TEN different graphs using the “reset” button to get to each graph. For example, running the

simulator from the command line generates the graph for Run #1, hitting “reset” will give you the graph for Run #2, etc.

- c. Record the time TSP-Approx and TSP-Permutation took for a run as well as the path cost found (NOTE: this is already output on the command line after the algorithm runs, **you do not need extra code to do this but be careful about the time units!** You may want to alter the code in printClockTime() from TSPAlg for convenience...)
- d. Next, calculate the ratio between the two algorithms for time and cost (TSP-A divided by TSP-P format please), and determine the average, min, and max for each column to complete the chart given:


| Experiment 1 | | | | | | |
|-------------------------------------|------------|------------|------------------------|------------|------------------------|------------|
| Parameters: #cities, seed#, msdelay | | | | | | |
| Run # | TSP-A Time | TSP-P Time | Time Ratio (TSP-A Cost | TSP-P Cost | Cost Ratio (TSP-A Cost | TSP-P Cost |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |
| 6 | | | | | | |
| 7 | | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| Average: | | | | | | |
| Min: | | | | | | |
| Max: | | | | | | |

- e. At the bottom, enter the explanation of what was actually turned out to be interesting.

3. Make sure **both** of you submit **everything including the report!!**

Grading Rubric

No credit will be given for non-submitted assignments, assignments submitted after the last submission date, non-compiling assignments, or non-independent work (“pairs” are still required to work independently of other students/pairs). Otherwise your score will be as follows:

 For individual students:

- Full credit: 5pts for TSP-Permutation, 5pts for TSP-Approx
- Partial credit: Up to 5pts from manual inspection indicating a valid attempt at doing the project

For pairs of students:


- Full credit: 3.5pts for TSP-Permutation, 3.5pts for TSP-Approx, 3pts for Report
- Partial credit: Up to 4pts from manual inspection indicating a valid attempt at doing the project

I don't care who is “responsible” for what part of the project. If you're working in a pair, all parts need to be completed. You may NOT start working as a pair and then split to try to get credit for only the first half. All cases of similar code being submitted by two students *not claiming to be on a team* will be considered an honor code violation.

5% will be subtracted if you don't submit in the correct zip→folder→files format listed in the submission instructions (because it will slow down grading for the entire class).

Extra credit for early submissions: 1% extra credit rewarded for every 24 hours your submission made before the last submission date (up to 5% extra credit). Your latest submission before the due time will be used for grading and extra credit checking. You CANNOT choose which one counts.

Submission

- Make a backup copy of your user folder (in case you mess this up), then remove all test files, jar files, class files, output folders, etc. You're just submitting your java files and the readme.txt
- **EVERYONE:**
-  Complete the readme.txt file in the user folder (instructions are in the file).
- **INDIVIDUALS:**
 - Zip your user folder (***not just the files***) and name the zip username.zip.
 - The submitted file should look something like this for INDIVIDUALS:

```
iavramo2.zip --> iavramo2 --> City.java
                                Flight.java
                                ...
                                readme.txt
                                ...
```

That's a zip file containing a folder containing your code.

- **TEAMS:**
 - Zip your user folder (***not just the files***) and name the zip team-username1.zip.
 - The submitted file should look something like this for EACH TEAM MEMBER:

```
team-iavramo2.zip --> iavramo2 --> City.java
                                Flight.java
                                ...
                                readme.txt
                                ...
                                team-report-UN1-UN2.xlsx
                                ...
```

That's a zip file which starts with the word "team-" containing a folder for YOU containing your team's code. BOTH STUDENTS need to create SEPARATE zip submissions in this format. So the *second* person on the team creates a *separate* file that would look something like this:

```
team-othername.zip --> othername --> City.java
                                Flight.java
                                ...
                                readme.txt
                                ...
                                team-report-UN1-UN2.xlsx
                                ...
```

The code and readme.txt files are the same for both people, only the folder and zip are different.

- **EVERYONE:**
 - Go to the Coding 2 submission link on Blackboard and submit your assignment. You can resubmit as many times as you'd like, only the last submission will be graded.
 - CHECK WHAT YOU UPLOADED. Wrong submissions will receive a 0.