CompSci 260P Spring 2023 Project 4: Greedy Approximations (TSP)
Due May 24, 2023, at 6:00 PM, although late submissions are allowed. Note that the stated due date here is later than the one listed in the syllabus.

Getting Started

Choosing a project partner (or even two)

You have the option to work with a second person for this assignment. If you do so, I expect you to work via pair programming. That is, you may not split the assignment, such as by having one person implement the approximation algorithms while the other person writes the report. I reserve the right to ask any non-empty subset of the project partners about the implementation and adjust the score accordingly if I believe they do not understand what was submitted.

Similarly, any academic dishonesty arising from a group will be treated as an offense by all involved.

All partners must fill out the following survey to register the partnership. It is not enough for one to do so. Be sure to include your UCINetID and your partner’s UCINetID. Failure to do so may cause some of you to not get credit. Make sure you know what your UCINetID is. It is probably not just numbers and it almost certainly does not include the @ symbol.

All partnerships must be registered by 11 pm Irvine time on Friday, May 19.

https://docs.google.com/forms/d/e/1FAIpQLSfQjizQblInqDQ7k9CfdLr4gBuU7Ex3OW4y6D2iiL5RaaWt5NA/viewform

There are two parts of this project: programming and an experimental analysis of the algorithm.
Programming Requirements

For this project, you are going to experimentally analyze the 2-approximation algorithm for TSP that we saw in lecture. You are required to create a project on gitlab called project4. Note the spelling and case sensitivity: that is vital for my script to collect what you submit. If you are working as a partnership, the person indicated as the submitter must be the one who owns the project, similar to how this was dealt with in project two.

In the past, I have required students to do this in C++ and graded their program correctness with automated test cases, similar to what I did for projects 0 through 3. While I will not be doing that for this assignment, you must do the following:

- Your program must be in C++, Go, Java, or Python3.
- You must include a README.txt file at the top level of your project 4 directory with instructions for how I can run your program to test it for correctness if I choose to do so. In order to get credit for the assignment, it must be very clear how I can do the following in the VM.
  - If you are using a language that is not installed on the VM already, you must include command-line instructions for how I would install a compiler for your language of choice in a new VM.
  - When I run your program, I must be able to easily run it in such a way that it computes a minimum spanning tree of a graph from a file. The file will describe a graph in the following format:
    - The first line will be an integer, indicating the number of vertices, which I will refer to later as numbered 0..n-1, where n is the number of vertices.
    - The second line will be an integer, indicating the number edges, m
    - The remaining m lines will each be three integers, indicating an undirected edges. They will be three space-separated integers, representing first the two endpoints and then a positive integer weight for that edge.
  - Your program, taking such an input, will then output only a single integer that is the cost of the minimum spanning tree.
  - Your program, given a 500 vertex complete graph, must be able to compute the cost of the MST within three minutes\(^1\) on the same computer I use to grade other projects. Note that this does not mean your experiment should limit itself to 500 vertex graphs.
- Your code should be reasonably well written and commented.

---

\(^1\) Technically, I use processor minutes for this, so (a) if I have something else running, it doesn’t count against you when it’s on the processor, and (b) someone cannot get around this by spinning up extra threads. There are instructions in a previous project for how I dealt with this towards grading.
For any language you choose, you may use standard libraries as appropriate, unless there is one that makes solving this problem trivial. I am unaware of any such part of the library. The standard priority queue class in C++ lives in <queue> for some reason. However, it is by default a max priority queue while you want a min priority queue if you are implementing Jarnik’s Algorithm.

You are explicitly permitted to use C++ standard library container classes (std::vector, std::set, etc). You are welcome to ask anything you want about these libraries, or to look up material about them online. Information about how to use an explicitly-permitted library may always be shared among classmates, but refrain from telling one another how you solved a problem in the assignment with them. For example, answering “how do I check if an element is in a std::set?” is great and encouraged, while answering “what did you use std::set for in your project?” is not.

A good reference for the STL container classes (such as those listed above, including std::map) is http://www.cplusplus.com/reference/map/map/.

If you would like to reuse some or all of your code (not that of someone else) from project 0 or project 3 for part of this project, you are welcome to do so. If you have a partnership, you may take code from either or both partners’ versions of project 0 or project 3, but not from anyone else’s. If you take from a previous version, please leave a comment saying whose and which project. Note that the TSP portions of projects 0 and 3 assumed much smaller graphs than you will need for this project.

Remember that the purpose of this project isn’t to find an implementation of TSP, but rather to code it yourself. Submitting work that isn’t yours (for any reason) is a decidedly bad idea and one of the very few ways to do poorly in this class. If I find that you submitted code you found online, your grade will be worse than if you hadn’t turned in this project at all.

The Report

In addition to having a programming portion, you (and your group, if applicable) need to put together a short report. In this report, you will perform an experimental analysis of the algorithm(s) you implemented. You will need to convert your report, when done, to a PDF for upload to GradeScope. Your report must be typed. Your group for the report must be the same set of people with whom you did the programming if you had a group. However, if you worked alone on the programming portion, you may elect to partner with someone else (or a total of three people, yourself included) who also worked alone to produce a single report. In this case, you should run the same experiments with the code each of you produced and, if there are differences in behavior, discuss this in the report. You do not need to modify your code based on this, although you are allowed to do so. Different approaches to generating the approximate tour might produce different answers that are correct, after all.
If you worked alone on the programming but with a partner for the report, you do not need to (nor should you) submit the form above. One of you should submit the report to GradeScope and list both of you as authors/partners for it.

Your report should address the following things:

- How did you test your code to ensure that it behaves the way you intended?
  - For this project, I explicitly authorize the sharing of test cases between groups for purposes of testing the minimum spanning tree portion. That is, you may generate a graph, compute the MST cost, and share the graph with another group to compare if they get the same cost. If you do this, include that in your report about how you tested your code. Tell me who you shared yours with and what you found. It is okay if you find that one of you has incorrect code and fixed it accordingly. State that in the project report.
  - How did you ensure that your MST code was correct?
  - What, other than correctness of MST, were you measuring, and how did you do so?

- How did you test the code for effectiveness? There should be test cases beyond the ones provided in the starting test cases.
  - Remember, you may not share test cases for the TSP portion beyond your group for this project.
  - Describe in general how you came up with test cases for TSP and how you ensured that they were reasonable. Remember, these are for large graphs, and you do not know the optimal TSP solution on them.
  - How did you ensure the input graphs you have obeyed triangle inequality? Recall that this is important for the theoretical guarantees of the algorithm.
  - Describe the purpose of 2-3 of them. You can refer to them by name -- you don’t need to copy the specifics of the test case to your report (in fact, please don’t copy/paste the specifics). Just make sure they’re in the collected code you submit (if you are using the provided code, I believe they would be in the /gtest/ folder).

- How did you measure the approximation ratio produced in test cases? How accurate of a bound do you think you measured?
  - I realize you all know the “correct answer” is that this is a 2-approximately and that worst-case scenarios exist that produce near this. I am more interested in how you use the experiment to analyze typical behavior than the worst case. If your experiments find that for a lot of graphs, your approximation ratio is better than 2, that’s fine. If you find a lot of graphs where the approximation ratio is 2, that’s also fine.
  - You should make it clear how you estimate the approximation ratio for your graphs, since a good experiment should include several graphs that are very large and thus ones for which we do not know the optimal TSP cost. Limiting yourself to graphs on which project 2’s algorithm can run, even with a week to run, would be a bad experimental choice.
A side note: Spring 2023 is only the third time I am requiring a report for this project. I am mostly interested in seeing how you went about this project and if you got the “big picture” ideas about experimental evaluation that I have tried to convey. Please treat it seriously and not as a last-minute addendum to programming. I plan to make a note of which reports are particularly insightful, which will help me add to any letters of recommendation I write for the authors in the future.

When you submit the report to GradeScope, it will ask you to “tag” the pages that the “response” is on. Please tag each page.

Each group should submit ONE REPORT TOTAL. Select a group member and ask that person to upload the report to GradeScope. Be sure to declare the group within GradeScope: there is a mechanism to do so. Just to be sure, I would recommend also writing each group member’s name in the report PDF.

Report limit is five pages, must be typed, reasonable font size. If you require more pages, please let me know -- exceptions will be granted, although I’d prefer that you try to limit it to five pages, as I will have many to read in a short period of time.

**Deliverables**

Ensure that you have committed and pushed your code to your project4 repo.

**Can I submit after the deadline?**
Yes, it is possible, subject to the late work policy for this course, which is described in the section titled Late work in the course reference and syllabus.

If you do not submit on time, your submission time is the later of when you submit the late form and when you submit the project report on GradeScope.