Compsci 330 Design and Analysis of Algorithms
Case Study, Fall 2023 Duke University

Due Date: Tuesday, November 21, 2023

Preface

You can work on the case study in a group of up to 4 students. If you typically work with a homework partner, consider teaming up with another pair, perhaps from your recitation section. If you would like help finding others to work with, please contact your recitation leaders.

The case study is different from a regular homework assignment. The problems will be described at a higher and more open-ended level. How you should model the problem and evaluate your approach may not be immediately obvious and is a substantial part of the exercise, reflecting the reality that real-world problems are often complex and ill-defined. Substantial flexibility is left for you to consider and to argue these things in your report—just like if you were working in the real world.

1 Introduction

With your brand new bachelors degree in computer science from Duke University, you have taken a job in the "Data-Driven Algorithms Design Group" of the new company NotUber (NU for short). NU runs a service in New York City where passengers request rides through a mobile app. Drivers registered with NU sign into the app whenever they want to drive.

So far NU has been subcontracting to another taxi dispatcher to make real-time assignments of drivers to passengers. Your group has been tasked with designing the next generation algorithmic-system for automating these decisions.

2 Problem Formulation

All trips will take place on a road network, modeled as a weighted undirected graph where edges represent road segments. Pickup and drop-off coordinates may not be vertices of the graph and may not even be on the road network, in which case trips should be from the closest network vertices to these locations.

2.1 Event-Driven Decisions

A matching algorithm can make a decision whenever an event happens, at which point it can either wait (do nothing) or assign an unmatched driver to an unmatched passenger. Once a driver is matched to a passenger, the driver travels to the passenger pickup location and then to their drop-off location. The events that can change the state of the world and result in a match opportunity for your algorithm include:

- New unmatched passengers may appear (along with a pickup and desired drop-off location).
- New unmatched drivers may appear, along with a current location, in two ways.
  - A brand new driver may appear at an arbitrary location,
  - Or, a matched driver with passenger may arrive at the drop-off location. The passenger then exits. With some probability, the driver may also exit and no event takes place. Otherwise, the driver becomes a new unmatched driver at their current location (note this was the drop-off location of the passenger who just exited).
  - Imagine that most drivers will typically want to finish several rides over a period of a few hours before exiting.

2.2 Desiderata and Constraints

Your algorithm has to balance at least three fundamental desiderata (desirable properties) listed below as D1-D3.

- **D1.** Passengers want to be dropped off as soon as possible, that is, to minimize the amount of time (in minutes) between when they appear as an unmatched passenger, and when they are dropped off at their destination.
- **D2.** Drivers want to maximize ride profit, defined as the number of minutes they spend driving passengers from pickup to drop-off locations minus the number of minutes they spend driving to pickup passengers. For simplicity, there is no penalty for the time when a driver is idle.
- **D3.** Your algorithms should be empirically efficient and scalable.

You should try to optimize these deisderata subject to the following constraints C1-C3.

- **C1.** A driver can only have one passenger onboard at any given time.
- **C2.** Passengers can only be picked up or dropped off at their requested pickup and drop-off locations.
- **C3.** Once you assign a driver to a passenger, that match continues until the passenger is dropped off at their correct location - matches made by your algorithm cannot be canceled.

3 Algorithm Design and Implementation Tasks

We work empirically with a dataset representing the NYC road network as a graph in an adjacency list format. In addition to vertex coordinates, neighbors, and edge lengths, the dataset includes information about the average speed along each road segment, which varies. Speed estimates are included for different hours of the day on both weekdays and weekends.

We also have a dataset with the location of drivers logging on at different times and passengers, along with their pickup and drop-off locations, requesting rides at different times. Both are sorted by increasing order of date-time. Formatting details for these datasets are provided in accompanying `readme` files.

Address the following tasks.

- **T1 (10 points).** Consider a very simple baseline that just keeps track of queues of available drivers and unmatched passengers and assigns the longest waiting passenger to the first available driver, whenever possible. Implement this algorithm and conduct an experiment measuring the performance of the algorithm in terms of desiderata D1-D3.
- **T2 (10 points).** Consider a slightly improved baseline that, whenever there is a choice, schedules the driver-passenger pair with minimum **straight-line distance** between the driver and pickup location of the passenger. Implement this algorithm and conduct an experiment measuring the performance of the algorithm in terms of desiderata D1-D3.
- **T3 (20 points).** Consider a further improvement that, whenever there is a choice, schedules the driver-passenger pair with minimum **estimated time** for the driver to traverse the network to the passenger pickup location. Explain how you will do this algorithmically. Then implement this algorithm and conduct an experiment measuring the performance of the algorithm in terms of desiderata D1-D3. Your implementation does not have to be optimized for efficiency (see T4 below).
- **T4 (30 points).** Improve the runtime efficiency while maintaining the same general scheduling strategy as in T3. To accomplish this, consider the following two optimizations:

  (i) Explain how to preprocess the nodes of the road network so that you can efficiently find the closest (in terms of straight-line distance) node to a given query point (at least approximately). By efficient, your algorithm should scale sub-linearly with $|V|$ (i.e., it should not simply loop over all nodes of the road network).

  (ii) Describe an algorithm to compute shortest paths between two nodes of the road network in a way that is more efficient than just Dijkstra's algorithm. Again, you may consider possible pre-processings, and you might consider computing approximate rather than exact shortest paths. Keep in mind that speed on road segments may vary by day and time.

  Conduct experiments and provide evidence that both of your optimizations are at least approximately correct empirically. Also conduct an experiment to measure the the performance of the algorithm in terms of desiderata D1-D3, particularly with a focus on the improvement in query runtimes over the strategy from T3.
- **T5 (30 points).** Describe and implement your own alternative scheduling algorithm to improve over the baseline algorithms. The improvement might be on any one or multiple of the desiderata D1-D3, or on some other important desirable properties that you propose. Conduct experiments to demonstrate the improvement.

4 Additional Bonus Questions

You are highly encouraged, but not required, to consider the following questions extending the case study. We will award small amounts of bonus points for thoughtful and detailed conceptual responses to these questions, and more bonus points for possible worked solutions with experimental results. It is possible to get over a 100% on the case study, up to a maximum possible score of 115%.

- **B1 (up to 5 bonus points).** So far we have considered passenger performance by arrival time, perhaps the total or average. What about when there is a surge in demand, and there

are many more passengers requesting rides than you can match to drivers? How could you design your algorithms to be fair to passengers and cope with such high demand?

- **B2. (up to 5 bonus points).** We have primarily emphasized passengers and runtime efficiency. Suppose you have a competitor NNU (Not-NotUber) that opens for business in the same city. NNU claims that its scheduling algorithm is better for drivers and many of your drivers leave to join NNU. NNU claims that its algorithm does a better job of optimizing D2, the driver ride profit, *and* they claim it is also more fair to drivers, ensuring that drivers get a more equitable number of rides assigned to them. How could you modify your algorithm to compete with NNU for drivers while still maintaining good performance for passengers?
- **B3. (up to 5 bonus points).** Suppose your service becomes so popular that half of all road traffic comes from your drivers. You notice an issue where sometimes your algorithm assigns too many drivers at once to the same routes causing congestion and traffic stops on the road network. How could you modify and expand your algorithm to account for congestion caused by your own drivers?
- **B4. (up to 5 bonus points).** After your service has been running for several years, your data science team decides to optimize your scheduling and routing algorithms based on years of historical data to make more informed decisions. How will you incorporate historical data into your algorithm?

5 Coding, Writing, and Submitting

Your code should be in Java or Python 3, and we encourage you to use Python 3. You are welcome to use any standard library features, but you should not use any external packages. You should submit all code implemented for the case study (including for running any experiments) under *Case Study Code* on Gradescope (there is no autograder—this is for reference). Be sure to add your group members on gradescope.

For a report of your findings you should create a deck of slides (for example, powerpoint or google slides) with the following outline. You can describe algorithms in English or with bullets and do not need to provide pseudocode or code-level detail. Anywhere you are asked to provide experimental results, you should use well-labeled tables or figures with explanations (possibly as captions). You should also explain any modeling decisions you made that would be helpful for interpreting your results.

1. Title slide with group members.
2. Executive Summary slide that summarizes your project and your findings.
3. Slides labeled for each of tasks T1-T5 with requested explanations, any modeling details, algorithms, and experimental results. Use at least 1 and at most 5 slides per task.
4. (optional bonus) Slides labeled for each of tasks B1-B4 if you choose to consider the bonus questions. Again, at least 1 and at most 5 slides per question.

Finally convert your slides to a pdf and submit to gradescope under *Case Study Report*. Be sure to add your group members on gradescope.