Advanced Algorithm Analysis

Project #1

Introduction

Shortest-path problem. Map apps face a massive challenge: how do you respond to huge numbers of shortest-path queries for highway routing, when the graph you're searching on is huge (10's of millions of vertices and edges: `https://networkrepository.com/road-road-usa.php`)? For this task, direct algorithms like bidirectional Dijkstra's and A* are hopelessly slow. Instead of finding shortest paths directly, map apps do something clever: they *preprocess* large chunks of the road graph in various ways, resulting in a data structure which summarizes the information needed to find actual shortest paths in an extremely short amount of time. Road networks are special in a couple of convenient ways:

- They don't change rapidly (obvious).
- They have hierarchical structure (maybe less obvious?).

For various reasons which we'll explore in class, this second point is central to why these preprocessing algorithms work so well. In general, preprocessing is a technique that takes your input data and processes it offline into an auxiliary dataset. The hope is that once you have this auxiliary dataset ready to go, the algorithmic question can be answered quite quickly by running "queries" on the auxiliary dataset. It is risky:

- It might result in a giant auxiliary dataset.
- It might take too long to compute the auxiliary dataset.
- Queries on the auxiliary dataset might still be slow.

If you balance the risks correctly, you may end up with a very nice system where you can preprocess offline (i.e., before you put your system out into the world) and then respond to queries very quickly and efficiently.

This project

In this project, you'll explore various aspects of the preprocessing problem for giant graphs (e.g., 10^6 to 10^7 nodes/edges). In collaboration with your work group and the professor, you'll decide on a plan for how you'd like to study preprocessing algorithms for giant graphs. Once you've decided on the plan, each person in the group will submit a writeup of their approach to this plan. To give you a sense of what I'm looking for, consider the following problem (with no preprocessing):

Example Deliverables: Direct Shortest-path algorithms

I define *Direct Shortest-path algorithms* as ones which include no preprocessing. The algorithm is given a graph, a source, and a target, and it must find a source-target shortest path without ever having seen the graph before. Examples are Dijkstra's algorithm (with a target node), bidirectional Dijkstra, A*, and (for educational purposes) the Bellman-Ford algorithm (with a target node).

If I had assigned this project to study direct shortest-path algorithms, here are a few example project deliverables you could complete for this problem:

1. **Potential Deliverable #1:** Pick 2 of these algorithms and compare them empirically/numerically on real-world highway networks. For example, use the Python package `osmnx` to download graph representations of the city of Los Angeles, and then find out which of your two algorithms is faster on average on that network. "Faster on average" could mean that you randomly choose many source/target pairs, and then record the average time it takes to find a shortest source/target path over all those pairs. Is there a clear winner? Does your choice of A*'s heuristic matter?

2. **Potential Deliverable #2:** Pick 1 or 2 of these algorithms and study its performance empirically/numerically on different types of graphs. Is there a type of graph (grid, erdos-renyi, barabasi-albert, geometric etc etc) on which Bidirectional Dijkstra is very fast? Here, you'll need to be careful to compare apples-to-apples: the worst-case performance of these algorithms is a function of the number of vertices and edges in a graph, but this might not hold for average-case once you include a target node.

3. **Potential Deliverable #3:** Consider each of these algorithms and *derive* graphs which are (a) specialized to be very good for a specific one of the algorithms, and are (b) specialized to be very bad for a specific one of the algorithms. For instance, routing in a straight line on a grid graph is A*'s absolute specialty. What is A*'s nemesis? Can you think of a general way to create graphs which extremely bad for A* and extremely good for bi-Dijkstra?

This Actual Project: Your Assignment

In collaboration with your work group and with the professor, write 2 problem statements ("deliverables") which mirror the ones I listed above as "Potential Deliverable." Everybody in a work group should complete the same 2 deliverables; this will give provide you with flexibility but also closer working partners as you go through the semester. Each of the deliverables should address a well-posed scientific question and explore some aspect of shortest-path preprocessing algorithms. There are lots and lots of options for these, be creative! Here are a couple to get you thinking:

- In Contraction Hierarchies (CH), what is the effect of choosing various heuristics for defining the ordering of the nodes? (random, rank by edge difference, rank by degree, online/offline, etc).
- In Transit Node Routing (TNR), what is the effect of different neighborhood sizes? You could even do a parameters sweep of the neighborhood size.
- What kinds of graphs are *bad* for these preprocessing algorithms? I.e., come up with a family of graphs for which your selected preprocessing algorithm(s) result in slow queries, or very large auxiliary data, or extremely slow preprocessing times.
- Pick 2 algorithms and test them on real-world networks, perhaps like those you might get from `osmnx`.
- Pick a few algorithms and derive what kinds of networks and/or design choices would make them work very badly. Why is the hierarchical structure important?
- Create an algorithm to measure or estimate the highway dimension of a network.

While the project is active, your team will give regular weekly updates in class and I may occasionally ask for demos of the project. When the project is finished, each person in your group will submit a written report which summarizes your findings, and we will supplement the weekly work group update with brief verbal reports from each person.

You are welcome to use and adapt this example code which I provided to illustrate Dijkstra's algorithm: `https://github.com/descon-uccs/shortest-path`