Assignment 3: Efficient Route Planning

Riko Jacob and Martin Aumüller

Hand-in date: 2024-11-06 11:59

The assignment is to be solved in groups of 1–3 people.

1 Introduction

Your task is to implement a variant of the contraction hierarchies algorithm by Geisberger et al. [GSSV12] for efficient route planning in road networks. With this task comes a real-world graph that depicts the road network in Denmark with $n = 569\,586$ vertices and $m = 2\,419\,072$ edges. This file is available on learnIT and will be the basis for running experiments.

You will need to test your implementations for correctness, perform experiments where you determine the average search time and the average number of edges that were inspected by the algorithm. Furthermore, you will have to detail your choices on how you implemented contraction hierarchies and report on its preprocessing and search time.

2 Dataset

Your implementation is supposed to read files (through standard input) that have the following structure. The first line contains two integers n, the number of vertices, and m, the number of edges. Each of the following n lines contain three numbers: an integer that identifies the vertex, and two floating point values representing the longitude and latitude of this vertex. Finally, m lines follow representing the edges of the graph. Each line contains three integers: the first two represent the identifiers of the two nodes it connects, and the last one is the cost of the edge. We simplify the setting and assume that the graph is undirected, i.e., does not contain one-way roads.

3 Basic (Bidirectional) Dijkstra

In this first part, your task is to implement a basic version of Dijkstra's algorithm. Please feel free to re-use the graph data structure and Dijkstra implementation from your "Algorithms and Data Structures" lecture, see `https://algs4.cs.princeton.edu/code/`.

Algorithm 1 Bidirectional Dijkstra(s, t)

```
$d_l \leftarrow \langle\infty, \dots, \infty\rangle, d_r \leftarrow \langle\infty, \dots, \infty\rangle, d \leftarrow \infty$
settled $\leftarrow$ ⟨False, ..., False⟩
$d_l[s] \leftarrow 0, d_r[t] \leftarrow 0$
$Q_l \leftarrow \{(s, 0)\}, Q_r \leftarrow \{(t, 0)\}$ //PQs, $l$ from start, $r$ from target
**while** $Q_l \neq \emptyset$ or $Q_r \neq \emptyset$ **do**
    **if** $Q_l \neq \emptyset$ and $\min(Q_l) \leq \min(Q_r)$ **then**
        $i \leftarrow l$
    **else**
        $i \leftarrow r$
    **end if**
    $(u, \cdot) \leftarrow Q_i$.deleteMin()
    **if** settled[$u$] **then**
        //This vertex was settled by the other instance
        **break**
    **end if**
    settled[$u$] $\leftarrow$ True
    **for all** $(u, v) \in E$ **do**
        **if** $d_i[u] + c(u, v) < d_i[v]$ **then**
            $d_i[v] \leftarrow d_i[u] + c(u, v)$
            Update $Q_i$ with $(v, d_i[v])$
        **end if**
        $d \leftarrow \min(d, d_l[v] + d_r[v])$ //Update minimum dist between $s$ and $t$
    **end for**
**end while**
**return** $d$
```

Solve the following tasks:

1. Write a method that reads a file as described above from standard input and builds a graph data structure from it. You are free to choose which graph data structure to use. Add a unit test that reads a small file and tests that the graph is built correctly.

2. Modify your Dijkstra implementation to use the "early stopping criteria." (Stop as soon as you identified the correct distance from s to t.) Add a test case to your unit test suite that checks whether the correct output is produced on your test graph.

3. Using a fixed seed, choose 1 000 random (s, t) pairs in the Denmark graph for testing. Report on the average running time and the average

number of relaxed edges for solving the 1 000 searches. Your algorithm should only compute the distance of each (s, t) pair and not recover an actual shortest path between s and t.

4. Create an implementation of bidirectional Dijkstra, as described in Algorithm 1. Make sure to add test cases that show that your implementation behaves as intended.

5. Using the same 1 000 (s, t) pairs, compare the performance of bidirectional Dijkstra to your previous implementation and report on the same performance metrics.

4 Contraction Hierarchies

In this second part, you will implement Contraction Hierarchies [GSSV12]. Provide unit tests on a small example graph that demonstrate that you correctly implemented the contraction phase and the query algorithm. Add a short paragraph to your report to discuss how you carried out your testing.

Solve the following tasks:

1. Implement the preprocessing phase of contraction hierarchies. That is, contract each node one-by-one and add shortcut edges as described in the lecture and in Section 3 of the paper [GSSV12]. Since this might take a long time, save the resulting graph for future use.

   In particular, write a program that reads a graph as described in Section 2, and outputs the augmented graph after the preprocessing phase has finished. The augmented graph consists of the original graph, all shortcuts added during the preprocessing phase, and the ranks of the vertices. The augmented graph should be stored in a text file that has the following format: The first line contains the number of vertices and edges. (The former should be the same as the input graph, the latter is potentially larger due to added shortcuts). Then follow n lines representing the vertices, which should include the *rank* of the node, i.e., at which point it was contracted. Finally, m lines follow representing the edges of the graph. Each line should include the identifier of the contracted node if it is shortcut, or -1 if it is not a shortcut. Report on the preprocessing time it took to run this step on the Denmark graph and include the resulting number of edges. Make all design choices explicit in your report. As a minimum, these include:

- How did you order the vertices? As a suggestion, try to use "lazy updates" and the "edge difference heuristic" mentioned on Page 8 of [GSSV12].
- How did your graph data structure support the `contract` operation? This includes a discussion of how you added and removed edges, and how you stored the rank of a vertex. As a suggestion, try to not remove edges, but rather ignore them. The paper [GSSV12] contains many ideas for an efficient implementation.
- How did you decide that during the contracting of a vertex v, the edge (u, w) should be added as a shortcut? (Which means that the shortest path (u, v, w) was unique.) As discussed in the lecture, this is a time-crucial operation, and you need to make certain choices to speed this up. Can it happen in your implementation that you add an unnecessary shortcut? The paper [GSSV12] contains many ideas of speeding up the process, but you do not have to implement them. However, to be able to finish the contraction process on the Denmark graph, you will need to care about an efficient way of resetting visited nodes after each Dijkstra iteration.[1]

2. Implement the contraction hierarchies bidirectional search algorithm (Algorithm 1 in [GSSV12]). As in the previous tasks, report on its performance on the same 1 000 pairs on the Denmark graph, using the same performance metrics.

5 Bonus

The tasks described above represent the minimum viable product for getting the assignment approved. For your final project, you will get additional tasks that will allow you to be considered for getting a better grade, like 10 or 12. Please find some suggestions of such additional tasks in the following. Please report on these tasks in your report: These tasks *do not count* towards the page limit.

Proper experimentation of the pre-processing phase. Implement and evaluate at least two different node ordering strategies for the pre-

[1]Just initializing the `distTo` array takes time $\Theta(n)$, and you would do it n times, yielding a running time of $\Theta(n^2)$ just for initializing arrays. For the Denmark graph, this is too large.

processing phase of contraction hierarchies. Furthermore, evaluate the impact the resulting augmented graphs have for the search performance.

Recover the path. Implement and evaluate an option to recover the shortest path from the shortest path tree that (most likely) include contracted edges.

Higher resolution graph with one way streets. As you have seen in the lecture demo, the graph actually ignores small residential streets. Write a parser to retrieve a more detailed graph from `https://download.geofabrik.de/europe/denmark.html`, and evaluate the performance of your implementation on this larger graph.

Your own idea. Think about something else to do! This could for example be a visualization of your results.

6 Report

Write a report in LaTeX, following the advice given on the lectures and the feedback you got from your report in the first two assignments. The report must not exceed **4 pages**, **excluding possible tables, figures, and references**. Hand in the report and the code you produced as a **single zip file** through LearnIT. Your group only needs to hand in one report.

References

[GSSV12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transp. Sci.*, 46(3):388–404, 2012.