Finding Attractive Exercise Circuits in Street Maps

Lewis, R.[*]

School of Mathematics, Cardiff University, Wales.

February 8, 2021

Summary
We consider the problem of determining fixed-length routes on a map that start and end at the same location. We discuss two heuristic algorithms for this problem. The first is based on finding pairs of edge-disjoint paths that are then combined; the second uses local search techniques. In real-world applications, we might also be interested in establishing routes that look attractive, are safe to use, and do not stray too far from the starting point. A short discussion on these issues is also conducted.

KEYWORDS: Shortest paths, fixed-length circuits; exercise routes; heuristics.

1. Introduction

The task of finding fixed-length routes on a map has various practical applications. For example, we may want to go on a 5 km run, organise a cycling tour, or we may need to quickly determine a walk from our house to complete our daily number of steps as determined by our fitness tracker. In practical circumstances, routes of a specific length can be quite easy to determine. For example, we may choose to travel back and forth on the same street repeatedly until the required distance has been covered. Similarly, we might also choose to perform "laps" of a city block.

In this study, we want to consider more attractive routes that avoid repetition. Specifically, we want to avoid routes that ask the user to travel along a street or footpath more than once. As an example, Figure 1 shows two routes from Times Square, Manhattan. These satisfy our requirements because, as we see, street sections are never used more than once.

2. Problem Description and Complexity

The considered problem can be modelled through graph-theoretic principles. Let G = (V, E) be an edge-weighted graph with a vertex set V and edge set E. In maps, edges correspond to roads and paths, while vertices correspond their intersections. The "weights" of edges then give to the length of the road/path between two adjacent intersections. Given a "source" vertex $s \in V$, our task is to now establish a circuit that contains s and whose total edge-weight equals (or comes close to equalling) a target k. In this current work we focus on undirected graphs only. This is appropriate for practical applications that involve determining jogging and walking routes, though it is not sufficient in applications involving one-way streets.

From a computational perspective, very little work has been conducted on the problem of finding fixed-length circuits in edge-weighted graphs. For unweighted graphs, the number of walks of length k between pairs of vertices can be found by taking the (binary) adjacency matrix of a graph G and raising it to the kth power. Currently, the best known algorithms for matrix multiplication operate in approximately $O(n^{2.3})$ (Davie and Stothers, 2013), so for large values of k the resultant complexity can be quite high at $O(kn^{2.3})$. Basagni et al. (1997) have also noted that the problem of calculating a u-v-walk of length k is NP-hard with edge-weighted graphs. For circuits, similar complexity results are known. The task of identifying a circuit in a graph G can be seen as the problem of identifying an Eulerian subgraph in G. However, the problem of identifying the longest Eulerian subgraph in G is known to be NP-hard, both for weighted and unweighted graphs (Skiena, 1990). This tells us that our

[*] LewisR9@cardiff.ac.uk

current problem is also NP-hard. This implies that we cannot expect to be able to solve this problem exactly in polynomial time; instead we are compelled to turn towards heuristic and approximation-based methods. Two heuristics are now described.

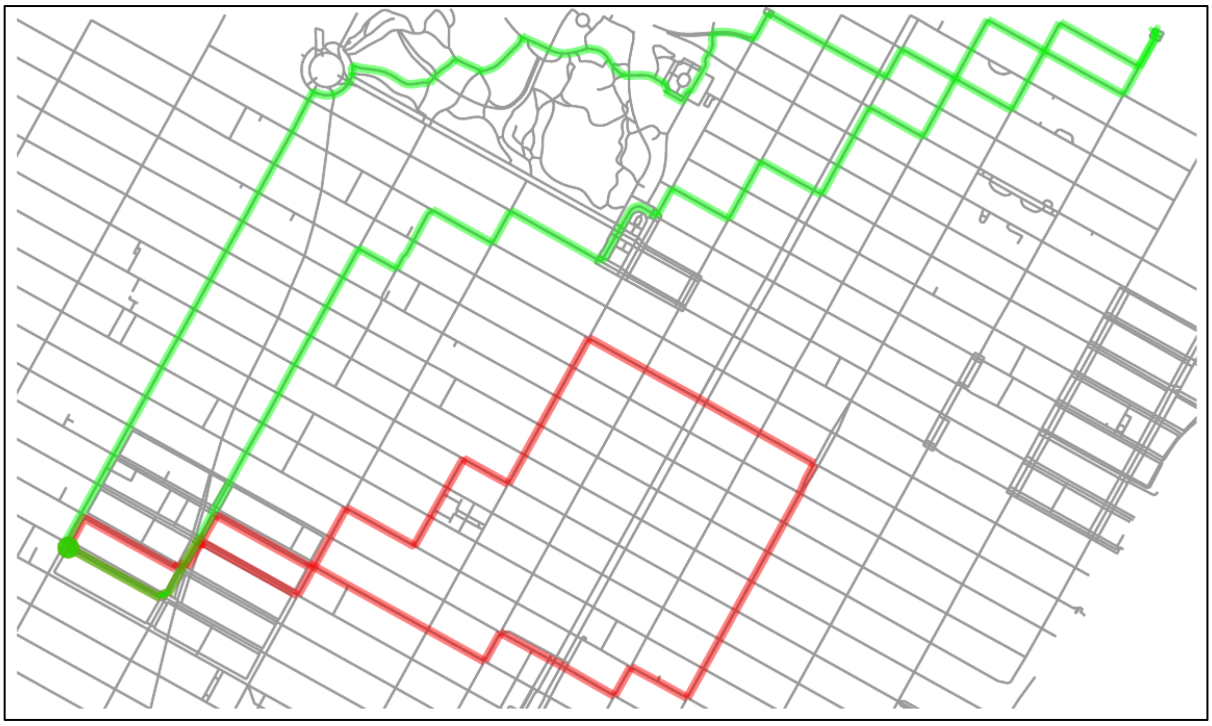

Figure 1. Starting from Times Square (bottom-left) in New York City, the above map shows a 5 km (red) circuit and an 8 km (green) circuit. Central Park is at the top of the figure.

3. Method 1: Finding Pairs of Shortest-Paths

Our first method is designed specifically for graphs resembling maps of roads and footpaths and was originally described by Lewis (2020). The intention is for this method to be fast while also producing accurate and visually pleasing solutions. Solutions are generated by producing a pair of edge-disjoint paths between the source vertex s and a particular "target" vertex t. The union of these two s-t-paths then forms a circuit containing s, as required. The problem now involves identifying the most appropriate target vertex – that is, the vertex $t \in V$ for which the sum of the lengths of the two s-t-paths is as close to k as possible.

Paths between two vertices in a graph can be formed in various ways. One strategy is to use depth first search, though this can often produce long and winding paths that look unattractive on a map. Another alternative is to use breadth first search, which generates paths between vertices that contain the minimum numbers of edges. In our case, we focus on the *shortest* paths between vertices (in terms of the sum of the edge weights within the path), as doing so seems to result in simple, logical-looking paths that involve less crisscrossing. Such paths can be achieved by standard polynomial-time algorithms such as those of Bellman-Ford, Moore, and Dijkstra.

Bhandari (1999) has shown how shortest path algorithms can be extended to form a pair of edge-disjoint s-t paths whose edge-weight sum is minimal. The method starts with a single shortest s-t-path and then uses this to modify the directions and weights of certain edges in the graph. The shortest s-t-path in this new graph is then calculated, and the two paths are combined to form the final circuit.

The overall algorithm works as follows. A set of target vertices $T = V - \{s\}$ is first defined. Elements t of T are then selected and removed from T one by one, and the two shortest edge-disjoint s-t-paths are calculated and combined to form a circuit. The best circuit among all those considered is then returned. In the worst case, this algorithm involves 2n applications of an O(mn) shortest path algorithm. This gives an overall complexity of $O(mn^2)$, though this bound can be considered conservative because:

- Any target vertex t whose shortest s-t-path is more than length k/2 does not need to be considered by the algorithm.
- During a run, multiple elements can often be removed from T in a single iteration. For example, if a circuit is seen to have a length of less than k, then all vertices in this circuit can also be removed from T.
- The algorithm can prioritise the selection of certain members of T, meaning that high quality solutions are usually encountered earlier in the run. The effects that different methods of prioritisation have on run times is illustrated in Figure 2.

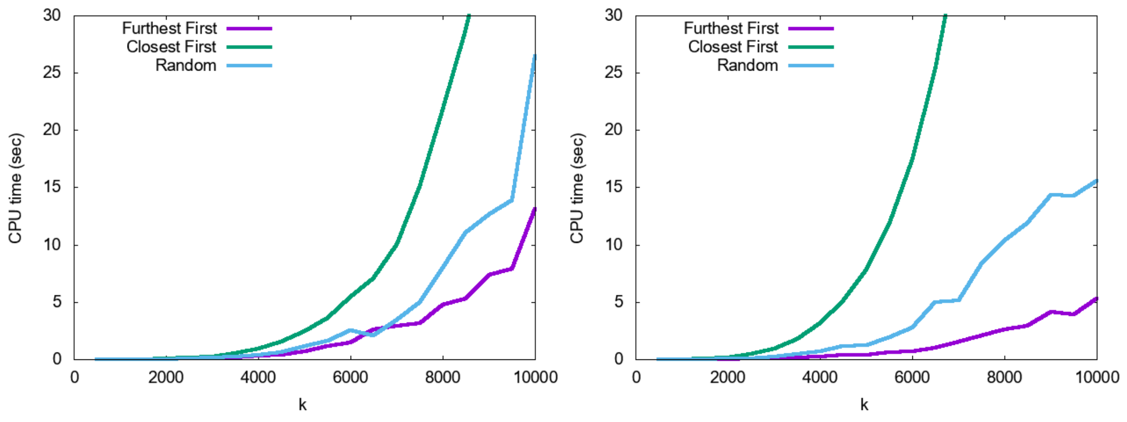

Figure 2. Run times for three different variants of Method 1 using differing values of k. All points are averaged across twenty randomly generated planar graphs with n = 40,000 vertices and m = 50,000 edges (left) and m = 100,000 edges (right). All algorithms were written in C++ and executed on a 3.3 GHz CPU with 8 GB of RAM.

4. Method 2: Local Search

Our second method is based on local search principles. Local search is a general-purpose method for combinatorial problems that involves making a series of adjustments to a candidate solution in order to try and optimise a particular objective function. For this problem the objective function is simply the difference between the length of the current circuit and the target length k. Circuits are then adjusted via specialised neighbourhood operators. The first of these involves removing a section of the circuit between two vertices u and v, and then replacing it with a new u-v-path. The second operator takes a vertex v in the circuit and then seeks a new circuit that starts and ends at v. This new circuit is then spliced into the current circuit.

5. Comparison of Methods

Our experiments with these two algorithms have focussed on real-world road networks (generated using the osmnx Python library) and randomly generated planar graphs. In general, Method 1 has shorter run times and returns more accurate solutions; however, its solutions tend to be less appealing visually. An example of this is shown in Figure 3. We see that Method 1 tends to produce paths that are more logical and regular in shape, whereas Method 2's solutions are more irregular and cluttered.

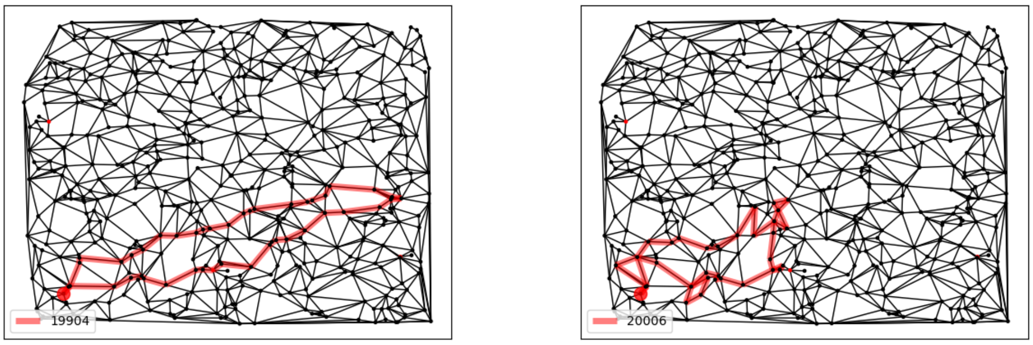

Figure 3. Example solutions produced on a small planar graph using Methods 1 (left) and 2 (right).

6. Discussion

The two methods described here are fast-acting and surprisingly accurate; however, issues remain. While the solutions returned by Method 1 are often more logical in appearance, they can also be quite elongated in shape, with the two shortest paths running almost in parallel to each other. In real-world applications this can mean that two shortest paths will go through a lot of the same geographical regions. The solutions of Method 2 can also seem quite illogical, as shown in Figure 3.

One advantage of Method 2 is that other factors can also be incorporated into the objective function to make solutions more appealing to users. Some relevant questions are now listed.

- *Aesthetics*. What exactly makes a circuit look more attractive on a map? Should it be convex in shape? Should we seek to minimise the number of intersections? Should we avoid long and winding roads?
- *Safety*. How important is route safety? Should we seek to avoid busy roads? Is street furniture an issue? Should we avoid steep hills? What about streets perceived to have higher risks of Covid-19 transmission?
- *Convenience*. Is it preferable to use routes that do not stray too far from their starting points? Should footpaths and cycle lanes be prioritised? What if users want to stop at certain locations en route?

References

Basagni, S., Bruschi, D., and Ravasio, S. (1997). On the difficulty of finding walks of length k. *Theoretical Informatics and its Applications*, 31(5):429–435.

Bhandari, R. (1999). *Survivable Networks*. Kluwer Academic Publishers.

Davie, A. and Stothers, A. (2013) Improved bound for complexity of matrix multiplication. *Proceedings of the Royal Society of Edinburgh*, 143(2):351-369.

Lewis, R. (2020). A Heuristic Algorithm for Finding Attractive Fixed-Length Circuits in Street Maps. In *Computational Logistics* (LNCS vol. 12433), Springer, pp. 384-395.

Skiena. S. (1990). *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Chapter: Eulerian Cycles, pages 192–196. Addison-Wesley, Reading, MA, 9.

Biography

Rhyd Lewis is a reader in Mathematics at Cardiff University. His main interests are algorithmic graph theory and heuristics. He is the author of the book *A Guide to Graph Colouring: Algorithms and Applications*, Springer (2015).