A Heuristic Algorithm for Finding Attractive Fixed-Length Circuits in Street Maps

R. Lewis

School of Mathematics, Cardiff University, CF24 4AG, Wales.
LewisR9@cf.ac.uk

Abstract. In this paper we consider the problem of determining fixed-length routes on a street map that start and end at the same location. We propose a heuristic for this problem based on finding pairs of edge-disjoint shortest paths, which can then be combined into a circuit. Various heuristics and filtering techniques are also proposed for improving the algorithm's performance.

1 Introduction

The task of finding fixed-length routes on a map has various practical applications in everyday life. For example, we may want to go on a 10 km run, organise a cycling tour, or we may need to quickly determine a walk from our house in order to complete our daily number of steps as determined by our fitness tracker.

In practical circumstances, routes of a specific length are often quite easy to determine. As a trivial 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 paper, 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. Two examples of such routes are shown in Figure 1, using Times Square in New York City as a starting point.

To define this problem more formally, it is useful to first review some standard definitions from graph theory. Let $G = (V, E)$ be an undirected edge-weighted graph with n vertices and m edges, and let $w(u, v)$ denote the weight (or length) of an edge $\{u, v\} \in E$.

Definition 1. *A* walk *is a series of incident edges in a graph. A* trail *is a walk with no repeated edges. A* path *is a trail with no repeated edges or vertices.*

It is also usual to add the prefix u-v to the above terms to signify a walk/trail/path that starts at vertex u and finishes at vertex v. The following terms can then also be used in cases where $u = v$.

Definition 2. *A u-v-walk/trail/path is considered* closed *whenever $u = v$. Closed trails are usually known as* circuits*; closed paths are usually known as* cycles*.*

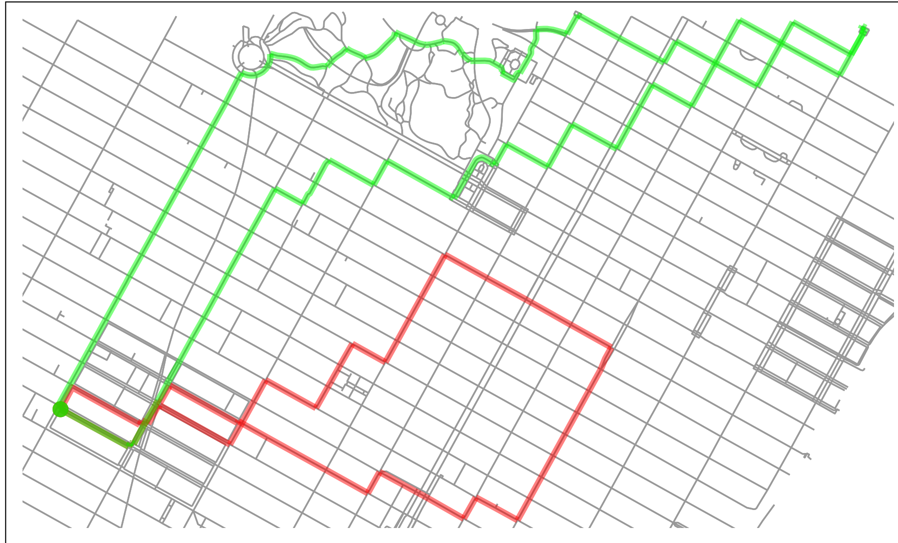

Fig. 1. Starting from Times Square (bottom-left) in New York City, the above map shows a 5 km (red) circuit and an 8 km (green) circuit. Central Park is at the top of the figure; the East river is at the bottom right.

Similarly to the above, the term u-circuit (cycle) can be used to denote a circuit (cycle) that contains the vertex u. We can also extend this to multiple vertices: for example, a u-v-circuit is a circuit seen to contain vertices u and v.

The problem considered in this paper can now be stated as follows:

Definition 3. *Let $G = (V, E)$ be an undirected edge-weighted graph, $s \in V$ be a* source *vertex and k be a required length. The k* circuit problem *involves determining an s-circuit C in G such that the total length of its edges $L(C)$ does not exceed k, and $k - L(C)$ is minimal.*

In this paper, note that 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. We also define our problem so that circuits with lengths exceeding k are disallowed. This again comes from practical considerations in that it is often easier to add a little extra distance to a route (e.g. by walking up and down a street), than it is to shorten it.

The examples in Figure 1 show optimal solutions for $k = 5000$ m and 8000 m using Times Square as the source vertex. These particular cases are not cycles because certain vertices (intersections on the map) are visited more than once; however, they are both circuits, in that no edge (road section) is traversed more than once.

From a computational perspective very little work seems to have been conducted on the problem of finding fixed-length circuits and cycles in edge-weighted

graphs. One recently suggested heuristic for cycles is due to Willems et al. [11] who use an adaptation of Yen's algorithm [12]. The basic idea is to calculate the shortest path between two vertices, followed by the second shortest path, the third shortest path, and so on. The algorithm then halts when a path close to the required length has been identified. To calculate a cycle containing a specific source vertex v, a special dummy vertex v' is added to the graph. Appropriate v-v'-paths are then sought.

Complexity results are also known. 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 $O(n^{2.3})$ [4], so for large values of k the resultant complexity can be quite high at $O(kn^{2.3})$. Basagni et al. [1] have also noted that the problem of calculating a u-v-walk of length k can be solved in polynomial time when using unweighted graphs, providing that $k = n^{O(1)}$; however, the problem is NP-hard with edge-weighted graphs.

For circuits and cycles, similar complexity results exist. The task of identifying a circuit in a graph G can be seen as the problem of identifying an Eulerian subgraph in G (recall that an Eulerian graph is a connected graph in which the degrees of all vertices are even). However, the problem of identifying the *longest* Eulerian subgraph in G is known to be NP-hard, both for weighted and unweighted graphs [8]. This tells us that our k circuit problem is also NP-hard, since it is equivalent to the longest Eulerian subgraph problem whenever $k \geq \sum_{\{u,v\} \in E} w(u,v)$. Similar reasoning can also be applied to the problem of finding cycles of length k in a graph, due to its relationship with the NP-hard Hamiltonian cycle problem.

In this paper we propose a heuristic for the k circuit problem with a particular focus on tackling graphs resembling maps of roads and footpaths. Our intention is for this heuristic to be fast while also producing accurate and visually pleasing solutions. In the next section we develop the overall framework of our algorithm. Section 3 then discusses methods for processing problem instances. In Section 4 we then analyse the performance of our methods, before showing how this performance can be further improved in Section 5.

2 Forming Circuits

In this paper our proposed strategy for the k circuit problem is to construct solutions by generating a pair of edge-disjoint paths between the source vertex s and a particular target vertex t. In an undirected graph, the union of these two s-t-paths forms an s-t-circuit. If these paths also happen to be vertex disjoint, then their union will also be an s-t-cycle. 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 closest to, but not exceeding k.

A single path between a pair of vertices can be formed in various different ways. One strategy is to use depth first search, though this can often produce

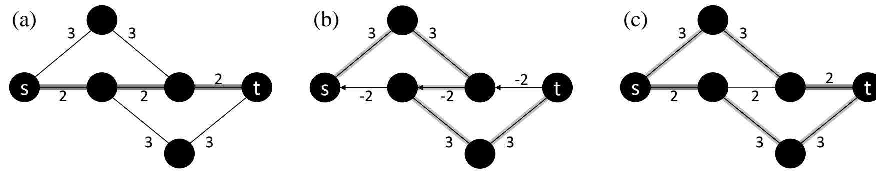

Fig. 2. (a) The shortest s-t-path in an example graph G; (b) a modified version of G and the corresponding shortest s-t-path (determined via Modified-Dijkstra); (c) the resultant s-t-circuit (cycle) formed by "unweaving" the two paths.

long meandering paths that, for this application, may well be unattractive to the user. A better alternative is breadth first search, which generates paths between vertices containing the minimum numbers of edges. In our case, however, we choose to focus on using the *shortest* paths between vertices (in terms of the sum of the edge-weights within the path), as doing so seems to result in simpler-looking paths that involve less crisscrossing. The production of such paths can be achieved via various well-known polynomial time algorithms, such as those of Bellman-Ford [3] and Moore [6], which both feature a complexity of $O(nm)$. The approach we follow here, though, is the more efficient method of Dijkstra [5], which has a complexity of $O(m \lg n)$ when a binary heap is used for its priority queue [3].

An obvious way of determining two paths between s and t is to produce a single s-t-path, remove this path's edges from the graph, and then find a second s-t-path. However, this approach has faults. Figure 2(a), for example, shows a small edge-weighted graph and its corresponding shortest s-t-path. Removing the edges of this path then disconnects s and t, preventing a second s-t-path from being formed. However, it is obvious that two disjoint s-t-paths exist, as shown in Figure 2(c). Better techniques are therefore needed.

Previously, Suurballe [9] and Bhandari [2] have proposed methods for finding the pair of edge-disjoint s-t paths whose edge-weight sums are minimal. An outline of Bhandari's method is given in Figure 2. As shown, in the first step, the shortest s-t-path is found. In the second step, the graph is then modified by adding directions on this path so that its edges point towards s. The weights on these edges are then negated and the shortest s-t-path in this new graph is calculated. In the final step, the graph is reset, and the two paths are "unweaved" to form the final pair of paths, as demonstrated in Figure 2(c). This unweaving process involves taking the symmetric difference of the two paths, resulting in a set of edges defining an Eulerian circuit.

Note that while Dijkstra's algorithm is sufficient for producing the first s-t-path in this method, it cannot be used for the second path because it is known to be incorrect for graphs featuring negatively weighted edges. Bhandari [2] proposes a modified version of Dijkstra's algorithm for this purpose. This Modified-Dijkstra algorithm operates using the following steps. In this

description $L(v)$ is used the denote the length of the path between the source s and a vertex v, and $P(v)$ gives the vertex that precedes v in the shortest s-v-path. The method halts as soon as the shortest s-t-path has been established.

1. For all $v \in V$, set $L(v) = \infty$ and $P(v) = $ NULL.
2. Let $S = \emptyset$ and set $L(s) = 0$.
3. Let $u \in S$ such that $L(u)$ is minimal among all vertices currently in S. If $u = t$ then exit; otherwise, remove u from S and go to Step 4.
4. For all neighbouring vertices v of u, if $L(u) + w(u, v) < L(v)$ then: (a) set $L(v) = L(u) + w(u, v)$, (b) set $P(v) = u$, and (c) insert v into S. Now return to Step 3.

Note that MODIFIED-DIJKSTRA differs from Dijkstra's original algorithm in that vertices can be inserted and removed from the set S of visited vertices multiple times; however, Segewick and Wayne [7] note that this brings run times that are exponential in the worst case. As an alternative, Bhandari [2] also suggests a modified version of Moores algorithm that is able to halt as soon as the shortest s-t-path is identified. This features a more desirable complexity of $O(nm)$. Despite this, in our experimentation, we still found that MODIFIED-DIJKSTRA generally gave shorter run times, as shown in Section 4. It is therefore used in all test unless specified otherwise.

Having now reviewed methods for producing s-t-circuits, our overall algorithm framework GEN-k-CIRCUIT is described as follows.

1. Let $T = V - \{s\}$ be the set of target vertices to check and let $C^* = \emptyset$ be the best circuit observed during the run.
2. Use Dijkstra's algorithm to determine the shortest path tree rooted at s. This gives the shortest paths between s and all other vertices in the graph.
3. Randomly select and remove a target vertex t from T. Using the shortest s-t-path found in Step 2, employ the methods of Bhandari together with MODIFIED-DIJKSTRA to form an s-t-circuit C.
4. If $L(C) \leq k$ and $L(C) > L(C^*)$ then set $C^* = C$.
5. If $L(C^*) = k$ or $T = \emptyset$ then return C^* and end; else go to Step 3.

As shown, the basic idea in these steps is to take each vertex $t \in V - \{s\}$ in turn and generate the shortest s-t-circuit. The observed circuit C^* whose length is closest to but not exceeding k is then returned. In the worst case, this involves $n - 1$ separate iterations of the algorithm (i.e., applications of MODIFIED-DIJKSTRA). In the following sections, however we will discuss various ways in which this number can be reduced while not affecting the accuracy of the algorithm.

3 Problem Generation and Preprocessing

As mentioned, in this paper we want to focus on graphs resembling real-world networks of roads and footpaths. However, to help analyse behaviour, we also want to be able to alter their edge densities. We therefore started by looking

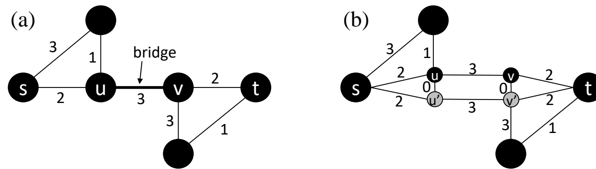

Fig. 3. (a) An example of a graph with an edge connectivity of one (due to the presence of the bridge $\{u, v\}$); (b) the modified graph, featuring an edge connectivity of two.

at the central districts of five large cities, namely Amsterdam, Kolkata, London, Melbourne and New York. These were found to have approximately 400 nodes (intersections) per square km. We then generated our own large problem instances that emulated these features.

To generate a single instance we started by taking a 10 km by 10 km square and placing 400,000 nodes within it at random coordinates. A Delaunay triangulation was then generated from these vertices, and a subset of this triangulation's edges was randomly selected to form a connected planar graph. Edge weights were then set to the Euclidean distances between end points, rounded up to the nearest metre. To allow circuits in any direction, the source vertex was also placed at the centre of the square at coordinate $(5000, 5000)$. For our experiments we considered three types of graphs: sparse, medium, and dense, featuring 500,000, 750,000 and 1000,000 edges respectively. Twenty such graphs were generated in each case.

Before producing circuits with these graphs, two preprocessing steps are required. Firstly, observe that any vertex v whose distance is more than $k/2$ units from the source can be removed from the graph since, in such cases, all s-v-circuits will be longer than k. For small values of k this can drastically reduce the number of edges and vertices, making computation much faster.

Our second preprocessing step is used to deal with any bridges in the graph (recall that a bridge is any edge whose removal increases the number of graph components). The presence of bridges in a graph can severely limit the number of circuits that are available and can therefore affect the quality of solution – in Figure 3(a), for example, we see that only one s-circuit is possible. To avoid these issues, graphs containing bridges are extended so that their edge connectivity is raised to two, making circuits between all pairs of vertices possible. Our method of doing this is illustrated in Figure 3(b). First, all bridges in the graph are identified[1], and the endpoints of these edges are inserted into a set A. For each vertex $v \in A$, a dummy vertex v' is then added to the graph and its neighbourhood is set to be equal to vertex v. The zero-weighted edge $\{v, v'\}$ is then also added.

Note that when executing our Gen-k-Circuit algorithm, a dummy vertex v' of a vertex v does not need to be added to the set of target vertices T. This is

[1] This can be achieved in $O(m)$ time using the algorithm of Tarjan [10].

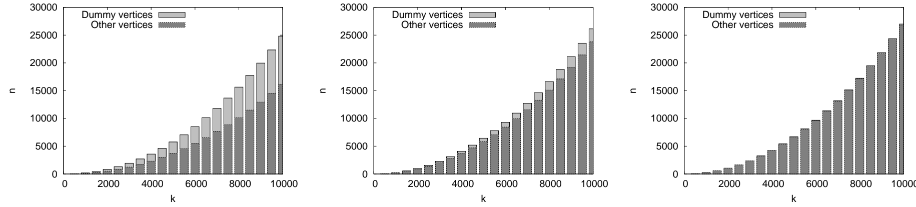

Fig. 4. Number of vertices in the graphs formed for differing values of k using sparse, medium, and dense problem instances respectively. All points are averaged across twenty problem instances.

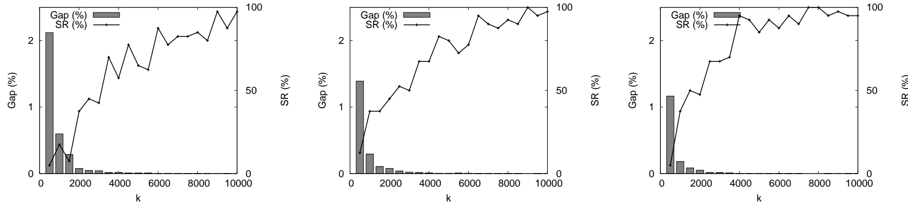

Fig. 5. Accuracy of our approach for differing values of k using sparse, medium, and dense problem instances respectively. All points are averaged across twenty problem instances.

because the generated s-v'-circuit will have the same length as the s-v-circuit. An instance containing n' dummy vertices therefore requires a maximum of $n-n'-1$ iterations in total.

The effects of these preprocessing steps are illustrated in Figure 4. In all cases, we see that the value for k has a large effect on the total number of vertices in the resultant graph. For the sparse graphs we also see that significant numbers of dummy vertices need to be added to eliminate the bridges that are present. Fewer additions are needed with denser graphs, however.

4 Basic Algorithm Performance

Our first set of results shows the accuracy of our approach for differing k values.[2] Here, accuracy is reported in two ways: (a) the percentage of instances for which $L(C^*) = k$ was achieved (the success rate), and (b) the percentage gap between $L(C^*)$ and k, calculated as $(1 - L(C^*)/k) \times 100$.

These results are summarised in Figure 5. For the smallest values of k, the gap between k and $L(C^*)$ seems to be around one to two percent and success rates are low. These gaps exist because the graphs are small and, as a result, very

[2] All algorithms were written in C++ and executed on a 3.3 GHz CPU with 8 GB of RAM.

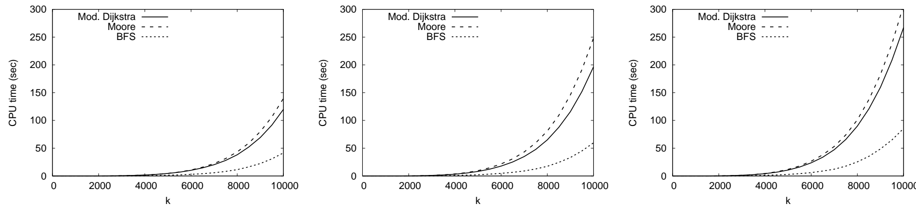

Fig. 6. Run times for different variants of GEN-k-CIRCUIT and differing values of k using sparse, medium, and dense problem instances respectively. All points are averaged across twenty problem instances.

few s-circuits are generated by the algorithm, giving us fewer options to choose between. For the same reason, larger values of k result in a much higher accuracy. Accuracy also improves slightly with denser instances because, as shown in Figure 4, these tend to feature more vertices, but fewer dummy vertices. This also gives a greater number of s-circuits to choose from.

In Figure 6 we show the run times of GEN-k-CIRCUIT with these instances. To reduce noise in these timings, we executed the algorithm for the maximum number of iterations – that is, we did not halt early if a circuit of length k was found. To contrast these results, we also include the times for two different variants. In the first of these, applications of MODIFIED-DIJKSTRA were replaced by Bhandari's modification of Moore's $O(nm)$ algorithm [2]. This variant therefore produces solutions of identical quality to the original. In the second variant, the shortest path algorithms used in Steps 2 and 3 of GEN-k-CIRCUIT were replaced by the $O(m)$ breadth first search (BFS) algorithm. This variant produces different solutions to the others because it determines s-t-circuits with the minimum number of edges as opposed to minimum lengths.

Figure 6 shows that the time requirements of these variants increase for denser graphs and larger graphs, with run times reaching up to 300 seconds in places. We see that using Moore's algorithm gives slightly higher run times compared to MODIFIED-DIJKSTRA, while the use of BFS shortens run times quite considerably. An issue with BFS, however, is that its preference for circuits with minimal numbers of edges can result in solutions that, subjectively, are sometimes less attractive to the user. An example of this is shown in Figure 7.

5 Improving Algorithm Performance

In this section we show how the performance of GEN-k-CIRCUIT can be improved by (a) using information collected during a run to filter out members of T that cannot improve solution quality, and (b) strategically selecting members of T that are more likely result in high-quality solutions being found earlier in a run. For (a) we first give the following theorem.

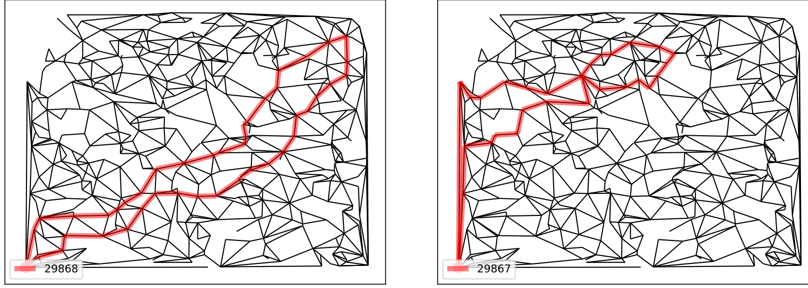

Fig. 7. Circuits produced by Modified-Dijkstra (left) and BFS (right) using a planar graph with $n = 400$ and $m = 800$, and a desired length of $k = 30000$. The source appears at the bottom-left corner.

Theorem 1. *Let $G = (V, E)$ be an edge-weighted graph with no negative weights. In addition, let P_1 be the shortest s-v-path in G, and let C be the shortest s-v-circuit, determined using P_1 together with the methods of Bhandari (Section 2).*

(i) If $u \in C$, then the shortest s-u-circuit has a length of at most $L(C)$.
(ii) If u is a descendent of v in the shortest path tree rooted at s, then the lengths of all s-u-circuits in G equal or exceed $L(C)$.

Proof. Part (*i*) is trivial: if $u \in C$ then C also defines an s-u-circuit; hence an s-u-circuit of length $L(C)$ is known to exist.

To prove Part (*ii*), let P_2 be the shortest v-u-path in G. The shortest s-u-path therefore has length $L(P_1)+L(P_2)$, where $L(P_2)$ is nonnegative. In addition, using the methods of Bhandari let P_1' and P_2' be the second paths generated from s to v and s to u respectively. We now need to show that $L(C) = L(P_1)+L(P_1') \leq L(P_1) + L(P_2) + L(P_2')$ or, in other words, $L(P_1') \leq L(P_2) + L(P_2')$. To do this, assume the opposite, giving $L(P_1') > L(P_2) + L(P_2')$. This now implies that the shortest s-v-circuit has length $L(P_1) + L(P_2) + L(P_2')$, which is a contradiction.

The findings of Theorem 1 allow us to filter out members of T through the application of the following two rules, which are applied between Steps 3 and 4 of Gen-k-Circuit. Recall that at this point in the algorithm, C is the shortest s-t-circuit in G.

1. If $L(C) \leq k$, then remove all vertices $u \in C$ from T. (All s-u-circuits will be equal or inferior in quality compared to C.)
2. If $L(C) \geq k$, then remove from T any descendents u of t in the shortest path tree rooted at s. (The lengths of all s-u-circuits in G will equal or exceed $L(C)$.)

Instead of removing just one vertex from T in each iteration of the algorithm, these rules therefore allow the removal of multiple vertices, thereby speeding up the algorithm while not compromising the quality of solution produced. Note,

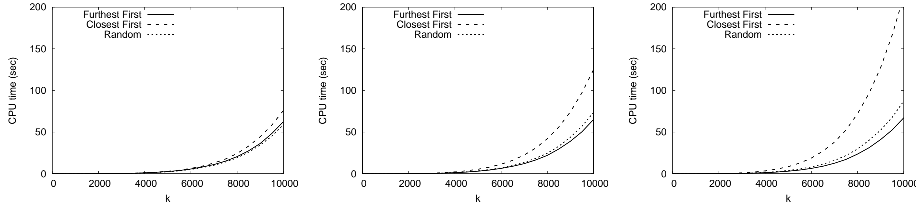

Fig. 8. Run times for GEN-k-CIRCUIT using filtering and heuristic selection. Results are summarised for differing values of k using sparse, medium, and dense problem instances respectively. All points are averaged across twenty problem instances.

however, that these rules rely on the use of shortest paths and circuits – they are therefore not suitable for the BFS variant of the algorithm.

Our second strategy for improving algorithm performance is to modify Step 3 of GEN-k-CIRCUIT so that t is chosen according to some heuristic. We suggest three strategies here: furthest-first, where the $t \in T$ furthest from the source s is selected; closest-first, where the $t \in T$ closest to s is selected; and the original random selection.

Figure 8 shows that these augmentations significantly reduce the run times of the algorithm. In general, the furthest-first rule seems to give the shortest run times because, in early stages of the run, it tends to produce overly long circuits with many vertices, allowing many elements to be removed from T according to the first rule above. Note that the second filtering rule is never actually applied using this heuristic because descendents of a vertex t will have already been considered and removed from T. Perhaps because of this, the random heuristic is often able to produce similar results in that it is able to remove elements from T according to both filtering rules.

As with our previous experiments, note that the run times shown in Figure 8 were gained by executing GEN-k-CIRCUIT for the maximum number of iterations. That is, the algorithm did not halt early in cases where a circuit of length k was found. Doing so, however, can drastically cut run times. In Figure 9 we show the times at which the best observed solution was found in each run with each instance. The best performance again comes when using the furthest-first rule, because solutions with lengths close to k tend to be considered in early stages of the run. On occasion, the random heuristic produces better results, but we found its run times to be subject to a much higher variance, making its behaviour less predictable. In contrast the closest-first heuristic is clearly the worst because, early in the run, it tends to consider very short circuits whose lengths are far from k. This also seems to result in fewer vertices being filtered from T.

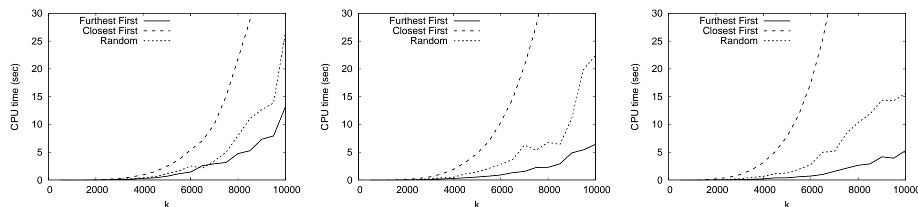

Fig. 9. Time at which the best solution was observed in runs of GEN-k-CIRCUIT using filtering and heuristic selection. Results are summarised for differing values of k using sparse, medium, and dense problem instances respectively. All points are averaged across twenty problem instances.

6 Conclusions

This paper has proposed a fast-acting heuristic algorithm for the NP-hard k circuit problem. Our method is based on finding pairs of edge-disjoint shortest paths between the source vertex s and a suitable target vertex t. The best observed performance comes when the targets that are furthest from s are considered first, which seems to result in higher quality solutions being identified earlier in the run, while also allowing larger numbers of vertices to be filtered out of the set of potential targets T.

There are a number of extensions to this problem that might be considered in future work. These include identifying fixed-length circuits that remain within a given distance of the starting point (encouraging routes that "stay local"), or adding required destinations within a circuit, such as a local shop or a friend's house.

References

1. S. Basagni, D. Bruschi, and S. Ravasio. On the difficulty of finding walks of length k. *Theoretical Informatics and its Applications*, 31(5):429–435, 1997.
2. R. Bhandari. *Survivable Networks.* Kluwer Academic Publishers, 1999.
3. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 3rd edition, 2009.
4. A. Davie and A. Stothers. Improved bound for complexity of matrix multiplication. *Proceedings of the Royal Society of Edinburgh*, 143(2):351369, 2013.
5. E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(269-271), 1959.
6. E. Moore. The shortest path through the maze. In *Proceedings of the International Symposium on the Theory of Switching, 1957, Part II*, pages 285–292. Harvard University Press, 1959.
7. R. Sedgewick and K. Wayne. *Algorithms.* Pearson Education, 4th edition, 2011. isbn: 9780 321 573513.
8. S. Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, chapter Eulerian Cycles, pages 192–196. Addison-Wesley, Reading, MA., 1990.

9. J. Suurballe. Disjoint paths in a network. *Networks*, 4:125–145, 1974.
10. R. Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161, 1974.
11. D. Willems, O. Zehner, and S. Ruzika. On a technique for finding running tracks of specific length in a road network. In N. Kliewer, J. Ehmke, and R. Borndörfer, editors, *Operations Research Proceedings 2017*, pages 333–338, Cham, 2018. Springer International Publishing.
12. J Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17(11):661–786, 1971.