Determining Fixed-Length Paths in Directed and Undirected Edge-Weighted Graphs

Daniel Hambly
School of Mathematics, Cardiff University, Wales, UK

Rhyd Lewis
School of Mathematics, Cardiff University, Wales, UK

Padraig Corcoran
School of Computer Science and Informatics, Cardiff University, Wales, UK

Abstract

In this paper, we examine the $\mathcal{NP}$-hard problem of identifying fixed-length s-t paths in edge-weighted graphs – that is, a path of a desired length k from a source vertex s to a target vertex t. Many existing strategies look at paths whose lengths are determined by the number of edges in the path. We, however, look at the length of the path as the sum of the edge weights. Here, three exact algorithms for this problem are proposed: the first based on an integer programming (IP) formulation, the second a backtracking algorithm, and the third based on an extension of Yen's algorithm. Analysis of these algorithms on random graphs shows that the backtracking algorithm performs best on smaller values of k, whilst the IP is preferable for larger values of k.

2012 ACM Subject Classification Theory of computation → Backtracking; Theory of computation → Integer programming; Information systems → Fixed length attributes

Keywords and phrases Graphs, paths, backtracking, integer programming, Yen's algorithm

Digital Object Identifier 10.4230/LIPIcs.SEA.2024.15

Supplementary Material *Software (Source Code)*: `https://zenodo.org/doi/10.5281/zenodo.11059135` [22]

Funding *Daniel Hambly*: EPSRC

1 Introduction

In this paper, we consider the problem of finding fixed-length paths in directed and undirected edge-weighted graphs. That is, we want to find a path in a network from one point to another where the total length of the path, in terms of the sum of its edge weights, is as close to possible as some prescribed length k. A particular application of interest is that of exercise. A street network can be conceptualised as a graph, where intersections and dead ends act as vertices, and roads as edges. Designing a workout route for activities like walking, running, or cycling, involves determining a path from a starting point to a destination within a specified number of steps, calories to be burned, or distance to cover. The selection of criteria, such as steps, calories, or distance, would define the weights assigned to the edges, making the task akin to identifying a path of a fixed length. Routing services are a feasible option for finding a solution, although they usually focus on routes with minimum length rather than a specific length [28]. Fixed-length paths can also be used in the combinatorial optimisation problem of determining a fixed-length cycle. Adding a dummy vertex to a graph, with edges and edge weights equal to those that leave the source vertex, setting it as the target vertex, and finding a path from the source to the target would be equivalent to finding a fixed-length cycle from the source. This is applicable for determining exercise routes for fixed-length

circuits and cycles [28, 29, 40]. Further applications of fixed-length paths include identifying interaction pathways between proteins [24], and detecting signaling pathways in protein interaction networks [34].

We now review some definitions and results relevant to this work:

▶ **Definition 1.** *In a graph, a walk is a sequence of vertices and edges; a trail is a sequence of vertices with no repeated edges; and a path is a trail with no repeated vertices.*

To indicate that a walk/trail/path starts at a vertex u and ends at a vertex v, the prefix u-v is added to the terms above. In the case when u=v, the following terms are used:

▶ **Definition 2.** *A u-v walk/trail/path is called closed if u=v, with closed trails and paths labelled as circuits and cycles respectively.*

Similarly to before, a u-circuit (cycle) can be used to describe a circuit (cycle) that starts and ends at a vertex u.

We can now formally define the k-length path problem (KPP):

▶ **Definition 3.** *Let $G = (V, E)$ be a directed or undirected edge-weighted graph with n vertices and m edges, let $k \geq 0$ be a desired length, and let $w(u, v)$ be a nonnegative weight for each edge $\{u, v\} \in E$. The k-length path problem involves determining a path $P = (v_1, v_2, ..., v_l)$ from a source vertex $s = v_1$ to a target vertex $t = v_l$ in G such that its length $L(P) = \sum_{i=1}^{l-1} w(v_i, v_{i+1})$ minimises $|k - L(P)|$.*

The problem of determining the shortest or longest s-t paths is also included in our definition by setting $k = 0$ or $k \geq \sum_{\forall\{u,v\}\in E} w(u, v)$ respectively.

In the next section, we will look at some bounds and existing solutions for finding fixed-length paths in graphs. In Section 3, we then establish three different methods for the KPP, whilst Section 4 analyses the performance of these algorithms. Finally, Section 5 concludes the paper and discusses ideas for future work.

2 Literature Review

Basagni et al. [2] show that determining a walk of length k in directed and undirected edge-weighted graphs is an $\mathcal{NP}$-complete problem, though it can be solved in polynomial time if the graph is unweighted and $k = n^{\mathcal{O}(1)}$. The number of k-length walks in unweighted graphs can be found by taking the kth exponent of the binary adjacency matrix of the graph [26]; specifically, the number of k-length s-t walks, where the length is equal to the number of edges, is equal to $(A^k)_{ij}$ [39]. Eppstein [14] showed that the K shortest trails can be found in $\mathcal{O}(m + n\log(n) + K)$ time using a variation of Yen's algorithm (discussed in greater detail in Subsection 3.3).

The shortest path problem is a well-studied problem that is known to be solvable in polynomial time with a variety of different algorithms [36]. The related problem of finding the longest s-t path in a graph is known to be $\mathcal{NP}$-hard [36] and fixed-parameter tractable on unweighted graphs [8], though it can be solved in polynomial time on directed acyclic graphs [35]. The KPP is a generalisation of the longest path problem when k is large enough and is, therefore, also $\mathcal{NP}$-hard. Moreover, akin to the longest path problem, the KPP can be solved in polynomial time on trees and directed acyclic graphs [23]. Further, Williams [41] notes that the KPP can be solved in polynomial time when $k \leq \log(n)/\log(\log(n))$. The problems of enumerating the total number of k-length paths or s-t paths are $\#\mathcal{W}[1]$-*complete* [15] and $\#\mathcal{P}$-*complete* [38] respectively.

Various bounds on the lengths of paths are noted in the literature. A trivial lower bound on the length of an s-t path in an edge-weighted graph is to find the shortest s-t path. This can be computed in polynomial time, with the time complexity of the algorithm dependent on the algorithm chosen. A theorem from Dirac [13] states that, if $\deg(v) \geq d \ \forall v \in V$ for an undirected unweighted graph G and $d \geq 0$, then there is a path of length at least d in G. This was extended by Bondy and Fan [9] to edge-weighted graphs to form the following theorem where $\deg^w(u) = \sum_{v \in \Gamma(u)} w(u,v)$, and $\Gamma(u)$ is the neighbourhood of u.

▶ **Theorem 4.** *Let $G = (V, E)$ be a connected undirected edge-weighted graph, $u \in V$, and $d \geq 0$. If $\deg^w(v) \geq d \ \forall v \in V \setminus \{u\}$ and $w(u,v) \geq 0 \ \forall \{u,v\} \in E$, then G contains a path from u with length at least d.*

Gallai et al. [20] theorise that, for a biconnected undirected unweighted graph G with $u, v \in V$ and $d \geq 0$, if $\deg(x) \geq d \ \forall x \in V \setminus \{u, v\}$, then there is a u-v path of length at least d in G. Again, this was extended by Bondy and Fan [9] to edge-weighted graphs:

▶ **Theorem 5.** *Let $G = (V, E)$ be a biconnected undirected edge-weighted graph, $u, v \in V$, and $d \geq 0$. If $\deg^w(x) \geq d \ \forall x \in V \setminus \{u, v\}$, then G contains a u-v path with length at least d.*

Frieze et al. [19] have proved a conjecture from [9] that provides a further lower bound on the length of a path:

▶ **Theorem 6.** *Let $G = (V, E)$ be an undirected edge-weighted graph. Then G contains a path with length at least $\frac{2\sum_{\{u,v\} \in E} w(u,v)}{n}$.*

Fomin et al. [17] have also broadened the results from [13] and [20] to obtain the following corollary and theorem respectively:

▶ **Corollary 7.** *Let $G = (V, E)$ be a connected graph, $B \subseteq V$, $k \geq 0$, and let $\delta(G)$ denote the minimum vertex degree $\forall v \in V$. Deciding whether G contains a path of length at least $\min\{2\delta(G - B), |V| - |B| - 1\} + k$ is solvable in $2^{\mathcal{O}(k+|B|)} n^{\mathcal{O}(1)}$ time.*

▶ **Theorem 8.** *Let $G = (V, E)$ be a biconnected graph, $B \subseteq V$, $s, t \in V$, $k \geq 0$, and let $\delta(G)$ denote the minimum vertex degree $\forall v \in V$. Deciding whether G contains an s-t path of length at least $\delta(G - B) + k$ is solvable in $2^{\mathcal{O}(k+|B|)} n^{\mathcal{O}(1)}$ time.*

Several upper bounds for the length of the longest path are trivial to compute. For example, in an undirected edge-weighted graph G where $e_1, ..., e_n$ are the n edges with the largest weights in G, an upper bound is $\sum_{i=1}^{n} w(e_i)$. Lewis et al. [29] further provide an upper bound, of $\left\lfloor \left(\sum_{v \in V} \tau(v) \right) /2 \right\rfloor$, for the length of the longest cycle in a biconnected undirected edge-weighted graph G. Here, $\tau(v)$ signifies the sum of the weights of the two heaviest edges connected to a vertex v. This also serves as an upper bound for the longest path in the graph.

In terms of solution strategies for the KPP and related problems, Monien [30] was amongst the first to provide an algorithm that could determine if a path between two vertices has a length equal to k. The algorithm constructs a matrix $D^{(k)}(G)$, where each entry $d_{ij}^{(k)}$ for $i, j \in V$ is either equal to k if a path of length k between i and j exists, or λ otherwise. However, the construction of the matrix takes exponential time $\mathcal{O}(k!nm)$ to compute. Bodlaender [8] has shown that there exists an algorithm that can prove the existence of a path or an s-t path of length $\geq k$ in $\mathcal{O}(2^k k!n)$ and $\mathcal{O}(2^{2k}(2k)!n + m)$ time respectively. Nonetheless, these algorithms are only feasible in terms of run-time for very

small values of k. Bax et al. [4] created a finite-difference method to count the number of k-length s-t paths in directed graphs. However, this algorithm also runs in exponential time $\mathcal{O}(2^n \text{poly}(n))$. Papadimitriou et al. [32] initially conjectured the possibility of finding paths with lengths $\log(n)$ in polynomial time. This was subsequently validated by Alon et al. [1], who additionally introduced a colour-coding method to identify k-length paths in $\mathcal{O}(2^k m \log(n))$ and $\mathcal{O}(2^k n \log(n))$ time for directed and undirected unweighted graphs, respectively.

Kneis et al. [25] introduced the divide-and-colour method, combining divide-and-conquer and colour-coding techniques to locate paths of k vertices in unweighted graphs. Operating with a time complexity of $\mathcal{O}(3^{\log(k)}4^k)$ and an exponentially small error probability, this approximation algorithm also featured a deterministic counterpart capable of finding k-vertex paths in $\mathcal{O}(4^k k^2 \log^2(n))$ time. Around the same time, Chen et al. [12] proposed a divide-and-conquer approximation algorithm and a deterministic algorithm for this problem in unweighted graphs, with runtimes of $\mathcal{O}(4^k k^{3.42} m)$ and $\mathcal{O}(12.8^k mn)$, respectively. Chen et al. [11] extended this research to achieve an approximation and deterministic algorithm of $\mathcal{O}(4^k k^{2.7} m)$ and $\mathcal{O}(4^{k+\mathcal{O}(\log^3(k))} nm)$ time on undirected edge-weighted graphs, respectively. Koutis [27] introduced an $\mathcal{O}^*(2^{3k/2})$ approximation algorithm for this problem on directed unweighted graphs, which was then expanded to achieve an $\mathcal{O}^*(2^k)$ approximation algorithm [41]. Björklund et al. [7] realised the first approximation algorithm with an exponent base below 2 for determining the existence of k-vertex paths in undirected graphs in $\mathcal{O}^*(1.66^k)$ time, whilst Fomin et al. [18] developed a deterministic algorithm on directed and undirected graphs with time complexities of $\mathcal{O}(2.619^k n \log(n))$ and $\mathcal{O}(2.619^k m \log(n))$, respectively. Zehavi [43] improved upon this by introducing an $\mathcal{O}^*(2.597^k)$ time deterministic algorithm that uses divide-and-colour to find k-vertex paths with length at most W on directed edge-weighted graphs. This work was then extended to accomplish an $\mathcal{O}^*(2.554^k)$ time deterministic algorithm for determining k-vertex s-t paths on directed unweighted graphs [37]. Fomin et al. [16] then also presented a $2^k n^{\mathcal{O}(1)}$ randomised algorithm for the same problem on undirected unweighted graphs.

In other work, Bax [3] employed inclusion-exclusion algorithms to enumerate all paths and s-t paths in graphs. However, both algorithms exhibit the same exponential runtimes as observed in [4]. Roberts and Kroese [33], meanwhile, utilised a Monte Carlo simulation to estimate the number of s-t paths in both directed and undirected unweighted graphs, whilst Björklund et al. [6] introduced an algorithm applying a linear transformation on a function, enabling the computation of the total number of s-t paths in directed edge-weighted graphs based on both length and the number of vertices $0 \leq k \leq n-1$ in the path. The latter algorithm runs in $\mathcal{O}^*(\exp(H(k/2n)n))$ time, where $H(p) = -p\log(p) - (1-p)\log(1-p)$ and $0 \leq p \leq 1$. Similarly, Birmelé et al. [5] devised an algorithm listing all s-t paths in undirected unweighted connected graphs in $\mathcal{O}(m + \sum_{\pi \in \mathcal{P}_{st}(G)} |\pi|)$ time. Here, $\mathcal{P}_{st}(G)$ represents the set of all s-t paths in G, π is an s-t path, and $|\pi|$ is the number of edges in π. A different approach, due to Montanari et al. [31], used Markov chains to sample s-t paths in undirected unweighted planar graphs, considering path length as the number of edges. Longer paths have a small acceptance probability, whilst shorter paths are always accepted. More recently, a general purpose algorithm was introduced, capable of finding all paths up to length k in $\mathcal{O}(n + m + (k^\omega + k\Delta)|S_k|)$ time on a directed edge-weighted graph G, where k is a specified number of vertices, ω is the exponent of matrix multiplication, Δ is the maximum degree vertex in G, and $|S_k|$ is the number of connected induced subgraphs [21]. This algorithm is known to be fixed-parameter tractable if $|S_k| = \mathcal{O}(f(k)\text{poly}(n))$.

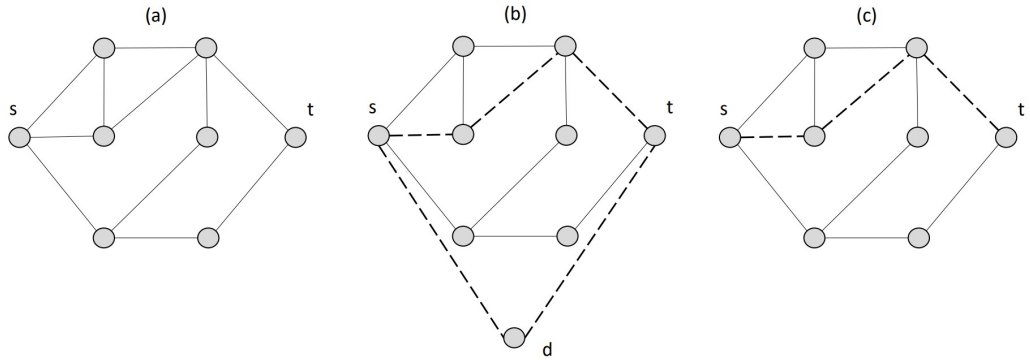

Figure 1 An example of adding a dummy vertex. Part (a) shows the graph and the source and target vertices s and t respectively. Part (b) shows the addition of the dummy vertex d with edges to s and t. The dashed edges show a possible d-cycle. Part (c) shows the s-t path formed from the d-cycle with d removed.

3 Proposed Solutions

Most of the methods reviewed in Section 2 considered unweighted graphs in which the length of the path is defined as the number of edges it contains. For this paper, we look at edge-weighted graphs and consider the length of the path as the sum of the edge weights. Over the course of the upcoming subsections, we will examine three different exact methods to solve the KPP – that is, given excess time, they guarantee the optimal solution. The first is an IP formulation, of which we believe is the first for determining k-length paths. The second is a backtracking algorithm that has an in-built pruning mechanism for reducing the size of the solution space. This is an improvement on similar techniques noted in Section 2 as, unless k is close in length to that of the longest paths, not all s-t paths need to be considered to find the optimal solution. The final is an extension of Yen's algorithm.

3.1 Integer Programming (IP) Formulation

In this subsection, we introduce an IP formulation for the KPP. This formulation is adapted from a model for finding fixed-length cycles in undirected edge-weighted graphs [29]. Specifically, we introduce a dummy vertex d with edges to s and t and edge weights of 0. By running the aforementioned model to find a fixed-length cycle from d, we effectively achieve the task of determining a fixed-length path from s to t. Figure 1 gives an example of how adding a dummy vertex to the graph can work.

For a graph $G = (V, E)$, let $V = \{v_0, v_1, v_2, ..., v_n\}$, and set v_0 as the dummy vertex d with edges to s and t and edge weights equal to 0. In the following formulation, the parameters $A_{n+1\times n+1}$ and $W_{n+1\times n+1}$, and the variable $X_{n+1\times n+1}$ denote the adjacency matrix, the weight matrix, and the binary decision variables respectively. $A_{ij} = 1$ if $\{v_i, v_j\} \in E$ and 0 otherwise, $W_{ij} = w(v_i, v_j)$ if $\{v_i, v_j\} \in E$ and ∞ otherwise, whilst $X_{ij} = 1$ if there is an edge from v_i to v_j in the d-cycle, and 0 otherwise. The objective is to therefore minimise $|k - \sum_{i=0}^{n} \sum_{j=0}^{n} A_{ij} W_{ij} X_{ij}|$. As this is not linear, we introduce a variable Z and Constraints (1) and (2) to linearise it. The full model is to now minimise Z subject to:

$$k - \sum_{i=0}^{n}\sum_{j=0}^{n} A_{ij}W_{ij}X_{ij} \leq Z \tag{1}$$

$$\sum_{i=0}^{n}\sum_{j=0}^{n} A_{ij}W_{ij}X_{ij} - k \leq Z \tag{2}$$

$$X_{ij} \leq A_{ij} \qquad \forall i \in \{0,\ldots,n\},\ \forall j \in \{0,\ldots,n\} \tag{3}$$

$$\sum_{i=0}^{n} X_{ij} - \sum_{i=0}^{n} X_{ji} = 0 \qquad \forall j \in \{0,\ldots,n\} \tag{4}$$

$$\sum_{i=0}^{n} X_{ij} + \sum_{i=0}^{n} X_{ji} \leq 2 \qquad \forall j \in \{0,\ldots,n\} \tag{5}$$

$$\sum_{i=0}^{n}\sum_{j=0}^{n} A_{ij}X_{ij} \geq 3 \tag{6}$$

$$0 \leq Y_{ij} \leq nX_{ij} \qquad \forall i \in \{1,\ldots,n\},\ \forall j \in \{0,\ldots,n\} \tag{7}$$

$$2\sum_{\substack{j=0\\ j\neq i}}^{n} Y_{ij} - 2\sum_{\substack{j=1\\ j\neq i}}^{n} Y_{ji} - \sum_{\substack{j=0\\ j\neq i}}^{n} X_{ij} - \sum_{\substack{j=1\\ j\neq i}}^{n} X_{ji} = 0 \qquad \forall i \in \{1,\ldots,n\} \tag{8}$$

$$A_{ds}, A_{sd}, A_{dt}, A_{td}, X_{ds}, X_{td} = 1 \tag{9}$$

$$X_{ij} \in \{0,1\} \qquad \forall i \in \{0,\ldots,n\},\ \forall j \in \{0,\ldots,n\} \tag{10}$$

$$Y_{ij}, Z \in \mathbb{R} \qquad \forall i \in \{0,\ldots,n\},\ \forall j \in \{0,\ldots,n\} \tag{11}$$

In this formulation, Constraint (3) stipulates that an edge is eligible to be part of the cycle only if its corresponding entry in the adjacency matrix is set to 1. Constraints (4) and (5) ensure a vertex is only in the d-cycle if it has one edge both entering and leaving. Constraint (6) ensures the cycle must contain a minimum of three edges (the shortest cycle possible is (d,s,t,d), which contains three edges). Due to the structure of the graphs generated in this paper and Constraints (4) and (9), a d-cycle will always contain at least 3 edges, rendering Constraint (6) redundant. We have still included it here, however, for completeness. Constraints (7) and (8) incorporate the auxiliary variables Y_{ij} to ensure that the solution consists of only a single d-cycle. A proof of this can be found in [10]. Finally, Constraint (9) simply adds edges, in both directions, from d to s and t. It also guarantees that the edges $\{d, s\}$ and $\{t, d\}$ are present in the cycle, since we want the path to start at s and end at t.

3.2 Backtracking Algorithm

Our second exact method for the KPP is a backtracking algorithm. This is based on an extension to depth-first search that can enumerate all s-t paths. Initially, the neighbourhood $\Gamma(s)$ is added to a list, and a vertex $u \in \Gamma(s)$ is considered. If u is not already in the current path, it is given a label of the length of the path from s to u and added to the current path. All vertices of $\Gamma(u)$ are then added to the back of the list and the vertices in this neighbourhood are then considered. Therefore, the neighbours of a vertex that have been added to the back of the list are considered before those preceding it. In this sense, the list is a stack. If t is considered in a neighbourhood, the s-t path and its length are output. Then, instead of adding t to the path and $\Gamma(t)$ to the list, the algorithm considers the next vertex of the neighbourhood at the back of the list. This method continues until an s-t path of length k or all s-t paths have been found.

To shorten execution times, this algorithm also uses a cutoff value $\overline{k}$, which represents the length of the shortest observed s-t path whose length exceeds k. Initially, $\overline{k}$ is set to ∞. When an s-t path with a length greater than k but less than $\overline{k}$ is found, $\overline{k}$ is then updated. During execution, if an s-u path is encountered whose length exceeds $\overline{k}$ then, due to the assumption that all edge weights are nonnegative, no extensions to this s-u path need to be evaluated. If $\overline{k}$ assumes the value of k during execution, then an optimal solution has been found. If $\overline{k}$ was not included in this way, the algorithm would simply find all s-t paths.

In this method, each s-t path is computed in $\mathcal{O}(n+m)$ time. Given a graph with $|\mathcal{P}|$ such paths, the overall runtime is $\mathcal{O}(|\mathcal{P}|\,(n+m))$ in the worst-case. Since $|\mathcal{P}|$ can approach the exponential with regards to graph size, this may be infeasible for large and/or dense graphs. As the algorithm considers the neighbourhoods of a vertex at the back of the list instead of the front, DFS will start at one end of the graph and systematically work its way around the rest of the graph. If the optimal solution is one of the last s-t paths to be considered, the algorithm may run for very long periods.

3.3 Yen's Algorithm

Our final approach for this problem is based on an extension of Yen's algorithm [42], which is a method for outputting the sequence of K shortest s-t paths in length order. Upon the production of each s-t path, Yen's algorithm operates by modifying the graph based on the previously observed paths, allowing the creation of the next path in the sequence. If Dijkstra's algorithm is employed using a Fibonacci heap, the overall time complexity of this algorithm is $\mathcal{O}(Kn(m+n\log(n)))$. Here, it is necessary to set $K=\infty$, though we can include a clause that terminates the algorithm once an s-t path with a length greater than or equal to k is found. For large graphs and cases where the length of the shortest s-t path is significantly lower than k, the algorithm may need to produce a restrictively large number of paths, leading to infeasible runtimes and memory requirements.

4 Analysis

In this section, we analyse and compare the performance of our three approaches on a set of random Euclidean graphs. In our trials, each graph was generated by assigning n vertices to random positions in a $(10,000 \times 10,000)$-unit square. Edges were then created between each pair of vertices with probability 0.5, with edge weights then set to the Euclidean distance between the endpoints. Our IP formulation was implemented on Gurobi version 10.0.2, and the backtracking algorithm and Yen's algorithm were written in C++.[1] For the algorithms in C++, adjacency lists were used to store the graphs, whilst for the IP formulation, adjacency matrices were required. All trials were executed on Windows 10 machines with 3.5 GHz and 8.0 RAM. A time limit of 1800 seconds was used for each algorithm.

Figure 2 shows the mean runtimes of executions that terminated in less than our time limit. In all cases, the considered k values used in these charts range from zero to the longest observed s-t paths for each graph. It is clear that for these values of n, Yen's algorithm can only find the optimal solution in the given time limit for the smallest values of k. This is because there are usually very many s-t paths with lengths greater than the shortest s-t path that will need to be enumerated by the algorithm before a path of sufficient length can be found. The backtracking algorithm has the shortest runtimes for most values of k;

[1] Source code is available at `https://zenodo.org/doi/10.5281/zenodo.11059135`.

Figure 2 Runtimes for $n = 50$ and 100 (respectively) on random graphs for various k-values. Each point is the mean across 20 graphs, with shaded areas indicating one standard deviation on either side.

Table 1 Values of k where the algorithms returned optimal solutions within the time limit in all runs. Here, $\leq \infty$ indicates that the algorithm returned solutions in all cases for all values of k, including when k was larger than the length of the longest s-t path. (For $n = 50$ and 100 the longest paths had lengths of approximately 370,000 and 750,000 respectively).

Algorithm	n	
	50	100
IP	$\leq \infty$	$\leq \infty$
Backtracking	$\leq$ 240,000	$\leq$ 450,000
Yen's	$\leq$ 15,000	$\leq$ 10,000

however, it also features a rapid increase in run times when k exceeds a certain threshold (as detailed in Table 1), producing unpredictable runtimes and a high standard deviation. As k eventually reaches the lengths of the longest s-t paths in the graph, many of the paths found have lengths lower than the cutoff value. Therefore, the cutoff value is rarely updated and more paths are discovered. Since the total number of paths can approach the exponential, the algorithm cannot search the whole graph in the time limit. Note that, unlike Yen's algorithm, which enumerates paths in length order, the ordering of paths observed by the backtracking algorithm is arbitrary. Hence, unlike the former, it has the potential to evaluate long paths early in the run. In addition, the cutoff value reduces the solution space and prevents costly solutions from being considered. These features, combined with its lower complexity, brings better results.

In contrast, although the IP method tends to have longer runtimes for lower values of k, it also features better scaling-up properties than both other algorithms, with optimal solutions being found within the time limit over a larger range of k values. Indeed, it was able to find the optimal solutions for all tested values of k.

Table 1 shows the maximum values for k for which the algorithms had 100% success rates. Yen's algorithm had a 100% success rate for the smallest values of k, with k decreasing further as n increased. The backtracking algorithm had 100% success rates on most values of k, the exception being when k was close to the maximum s-t path length. In contrast, the IP had 100% success rates for all values of k, with the optimal solution found in all cases. This includes the longest s-t paths in the graphs.

5 Conclusion

This paper has analysed three different exact methods on the $\mathcal{NP}$-hard problem of finding k-length s-t paths in directed and undirected edge-weighted graphs. One method is an adaptation of an IP formulation on the related problem of finding k-length cycles, the second is a backtracking algorithm that contains a cutoff value to prevent expensive solutions from being explored, and the third is an extension of Yen's algorithm.

The backtracking algorithm was shown to perform best on most values of k; however, it struggled to produce the optimal solution on higher values of k in the given time limit. Yen's algorithm could only find optimal solutions for the smallest values of k. In contrast, the IP was able to find the optimal solutions for all values of k, including when k was greater than the length of the longest s-t path.

Several avenues can be explored to further the work on the KPP. For the backtracking algorithm, adding a heuristic could help with finding t earlier in the run. Studying the literature to see if any other methods could be used to compare against the IP model would be beneficial. For example, heuristics such as local search or tabu search could be considered, where many different neighbourhood operators could be employed. The inclusion of some preprocessing of the graphs before any algorithms are run would be helpful to further reduce the search space and speed up the runtimes of the algorithms. In terms of the analysis in this paper, future work could come in the form of using randomised distance for the edge-weights instead of the Euclidean distance, or extending the analysis for both directed and undirected graphs. Finally, analysing the algorithms over different graph topologies that represent real-life networks (for example, planar graphs or grid graphs) would establish supportive evidence for which algorithm operates best in a real-world setting.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Stefano Basagni, Danilo Bruschi, and F. Ravasio. On the difficulty of finding walks of length k. *RAIRO Theor. Informatics Appl.*, 31(5):429–435, 1997. doi:10.1051/ita/1997310504291.

3 Eric T. Bax. Algorithms to count paths and cycles. *Inf. Process. Lett.*, 52(5):249–252, 1994. doi:10.1016/0020-0190(94)00151-0.

4 Eric T. Bax and Joel Franklin. A finite-difference sieve to count paths and cycles by length. *Inf. Process. Lett.*, 60(4):171–176, 1996. doi:10.1016/S0020-0190(96)00159-7.

5 Etienne Birmelé, Rui A. Ferreira, Roberto Grossi, Andrea Marino, Nadia Pisanti, Romeo Rizzi, and Gustavo Sacomoto. Optimal listing of cycles and st-paths in undirected graphs. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1884–1896. SIAM, SIAM, 2013. doi:10.1137/1.9781611973105.134.

6 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The fast intersection transform with applications to counting paths. *CoRR*, abs/0809.2489, 2008. doi:10.48550/arXiv.0809.2489.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.

8 Hans L. Bodlaender. On linear time minor tests with depth-first search. *J. Algorithms*, 14(1):1–23, 1993. doi:10.1006/jagm.1993.1001.

9 J.A. Bondy and G. Fan. Optimal paths and cycles in weighted graphs. In *Graph Theory in Memory of G.A. Dirac*, volume 41 of *Annals of Discrete Mathematics*, pages 53–69. Elsevier, 1988. doi:10.1016/S0167-5060(08)70449-7.

10 David Chalupa, Phininder Balaghan, Ken A. Hawick, and Neil A. Gordon. Computational methods for finding long simple cycles in complex networks. *Knowl. Based Syst.*, 125:96–107, 2017. doi:10.1016/j.knosys.2017.03.022.

11 Jianer Chen, Joachim Kneis, Songjian Lu, Daniel Mölle, Stefan Richter, Peter Rossmanith, Sing-Hoi Sze, and Fenghui Zhang. Randomized divide-and-conquer: Improved path, matching, and packing algorithms. *SIAM Journal on Computing*, 38(6):2526–2547, 2009.

12 Jianer Chen, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Improved algorithms for path, matching, and packing problems. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, volume 7, pages 298–307. Citeseer, SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283415.

13 Gabriel Andrew Dirac. Some theorems on abstract graphs. *Proceedings of the London Mathematical Society*, 3(1):69–81, 1952.

14 David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998. doi:10.1137/S0097539795290477.

15 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

16 Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, Kirill Simonov, and Giannos Stamoulis. Fixed-parameter tractability of maximum colored path and beyond. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3700–3712. SIAM, SIAM, 2023. doi:10.1137/1.9781611977554.ch142.

17 Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. Algorithmic extensions of dirac's theorem. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 406–416. SIAM, SIAM, 2022. doi:10.1137/1.9781611977073.20.

18 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

19 A. Frieze, C. McDiarmid, and B. Reed. On a conjecture of Bondy and Fan. *Ars Combin.*, 33:329–336, 1992.

20 T Gallai et al. On maximal paths and circuits of graphs. *Acta Math. Acad. Sci. Hungar*, 10:337–356, 1959.

21 Pierre-Louis Giscard, Nils M. Kriege, and Richard C. Wilson. A general purpose algorithm for counting simple cycles and simple paths of any length. *Algorithmica*, 81(7):2716–2737, 2019. doi:10.1007/s00453-019-00552-1.

22 Daniel Hambly, Rhyd Lewis, and Padraig Corcoran. Code and Data for the Paper "Determining Fixed-Length Paths in Directed and Undirected Edge-Weighted Graphs". Software, version 1.0. (visited on 2024-06-28). URL: https://zenodo.org/doi/10.5281/zenodo.11059135.

23 Magnus Lie Hetland. *Python Algorithms: mastering basic algorithms in the Python Language*. Apress, 2014.

24 Brian P Kelley, Roded Sharan, Richard M Karp, Taylor Sittler, David E Root, Brent R Stockwell, and Trey Ideker. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proceedings of the National Academy of Sciences*, 100(20):11394–11399, 2003.

25 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Divide-and-color. In Fedor V. Fomin, editor, *Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG 2006, Bergen, Norway, June 22-24, 2006, Revised Papers*, volume 4271 of *Lecture Notes in Computer Science*, pages 58–67. Springer, Springer, 2006. doi:10.1007/11917496_6.

26 W. Kocay and D.L. Kreher. *Graphs, Algorithms, and Optimization, Second Edition.* Discrete Mathematics and Its Applications. CRC Press, 2016.

27 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, Springer, 2008. doi:10.1007/978-3-540-70575-8_47.

28 Rhyd Lewis and Padraig Corcoran. Finding fixed-length circuits and cycles in undirected edge-weighted graphs: an application with street networks. *J. Heuristics*, 28(3):259–285, 2022. doi:10.1007/s10732-022-09493-5.

29 Rhyd Lewis, Padraig Corcoran, and Andrei V. Gagarin. Methods for determining cycles of a specific length in undirected graphs with edge weights. *J. Comb. Optim.*, 46(5):29, 2023. doi:10.1007/s10878-023-01091-w.

30 Burkhard Monien. How to find long paths efficiently. In *North-Holland Mathematics Studies*, volume 109, pages 239–254. Elsevier, 1985.

31 Sandro Montanari and Paolo Penna. On sampling simple paths in planar graphs according to their lengths. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 493–504. Springer, Springer, 2015. doi:10.1007/978-3-662-48054-0_41.

32 Christos H. Papadimitriou and Mihalis Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. *J. Comput. Syst. Sci.*, 53(2):161–170, 1996. doi:10.1006/jcss.1996.0058.

33 Ben Roberts and Dirk P. Kroese. Estimating the number of s-t paths in a graph. *J. Graph Algorithms Appl.*, 11(1):195–214, 2007. doi:10.7155/jgaa.00142.

34 Jacob Scott, Trey Ideker, Richard M. Karp, and Roded Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *J. Comput. Biol.*, 13(2):133–144, 2006. doi:10.1089/cmb.2006.13.133.

35 R. Sedgewick. *Algorithms in Java, Part 5: Graph Algorithms, 3rd Edition.* Addison-Wesley Professional, 2003.

36 Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition.* Addison-Wesley, 2011.

37 Dekel Tsur. Faster deterministic parameterized algorithm for k-path. *Theor. Comput. Sci.*, 790:96–104, 2019. doi:10.1016/j.tcs.2019.04.024.

38 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. doi:10.1137/0208032.

39 Piet Van Mieghem. *Graph spectra for complex networks.* Cambridge university press, 2023.

40 David Willems, Oliver Zehner, and Stefan Ruzika. On a technique for finding running tracks of specific length in a road network. In Natalia Kliewer, Jan Fabian Ehmke, and Ralf Borndörfer, editors, *Operations Research Proceedings 2017, Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Freie Universiät Berlin, Germany, September 6-8, 2017*, Operations Research Proceedings, pages 333–338. Springer, Springer, 2017. doi:10.1007/978-3-319-89920-6_45.

41 Ryan Williams. Finding paths of length k in $o^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009. doi:10.1016/j.ipl.2008.11.004.

42 Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

43 Meirav Zehavi. Mixing color coding-related techniques. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 1037–1049. Springer, Springer, 2015. doi:10.1007/978-3-662-48350-3_86.