Average-case behavior of k-shortest path algorithms

Alexander Schickedanz[1], Deepak Ajwani[2], Ulrich Meyer[3], and Pawel[4] Gawrychowski

[1] Goethe University, Frankfurt am Main, Germany, alex@ae.cs.uni-frankfurt.de
partially supported by the DFG grant ME 2088/3-2.
[2] Nokia Bell Labs, Dublin, Ireland deepak.ajwani@nokia-bell-labs.com
[3] Goethe University, Frankfurt am Main, Germany, umeyer@ae.cs.uni-frankfurt.de
[4] Instytut Informatyki, Uniwersytet Wroclawski, Wroclaw, Poland
gawry@cs.uni.wroc.pl

Abstract. The k-shortest path problem is a generalization of the fundamental shortest path problem, where the goal is to compute k simple paths from a given source to a target node, in non-decreasing order of their weight. With numerous applications modeling various optimization problems and as a feature in some learning systems, there is a need for efficient algorithms for this problem. Unfortunately, despite many decades of research, the best directed graph algorithm still has a worst-case asymptotic complexity of $\tilde{O}(kn(n+m))$. In contrast to the worst-case complexity, many algorithms have been shown to perform well on small diameter directed graphs in practice. In this paper, we prove that the average-case complexity of the popular Yen's algorithm on directed random graphs with edge probability $p = \Omega(\log n)/n$ in the unweighted and uniformly distributed weight setting is $O(km \log n)$, thus explaining the gap between the worst-case complexity and observed empirical performance. While we also provide a weaker bound of $O(km \log^4 n)$ for sparser graphs with $p \geq 4/n$, we show empirical evidence that the stronger bound should also hold in the sparser setting. We then prove that Feng's directed k-shortest path algorithm computes the second shortest path in expected $O(m)$ time on random graphs with edge probability $p = \Omega(\log n)/n$. Empirical evidence suggests that the average-case result for the Feng's algorithm holds even for $k > 2$ and sparser graphs.

Keywords: k-Shortest Path Algorithms, Average Case Analysis, Yen's Algorithm, Feng's Algorithm

1 Introduction

An important problem in the analysis of complex networks is to find structural relationships between the nodes. In many applications, one is interested in finding many short *simple* paths between two nodes to reveal deeper semantic insights into the relationship between the nodes. For instance, in areas such as financial fraud analysis, law-enforcement and journalistic investigations, a query such as "What are the different ways in which the person A and person B have indirectly communicated/dealt with/transacted with?" can reveal important connections between individuals that can then be further investigated. For the problem of recommending people to connect in a social network (a problem crucial for the growth of a social network), statistics from short simple paths can be used as a feature in prediction systems as well as in presenting a visual evidence for convincing a user that he/she should connect to the recommended person.

The problem of finding many short paths has applications well beyond finding structural relationship between nodes. For instance, Shih and Parthasarathy [31] used an approach for finding many short simple paths to identify the potential regulatory pathways in a Yeast gene network.

A natural way to find many short paths is to model it as the k-shortest simple path problem [9]. This is a generalization of the fundamental shortest path problem, where the goal is to compute not just one, but k loop-less paths between a given source and target node, in non-decreasing order of their weight. In this variant of k-shortest path, all shortest paths (and not just the first) are constrained to be cycle-free (simple). This is because cycles rarely add any insight when studying structural relationships between nodes, particularly in applications such as semantic search and recommending people to connect to. Note that in this variant, the resultant paths need not be node or edge-disjoint.

The best asymptotic complexity of algorithms for the loop-less variant of the k-shortest path on the weighted directed graphs is $O(kn(m+n\log\log n))$ [17]. There is considerable evidence [2, 32] that the asymptotic worst-case bound can't be improved significantly. This poor asymptotic complexity in the directed graph setting is in stark contrast with the significantly better asymptotic complexity of $O(k(m+n\log n))$ [21] in the undirected setting. This difference in asymptotics discourages the use of exact k-shortest path algorithms in applications where structural relationship between nodes in directed graphs needs to be computed efficiently.

A recent empirical study [3] has found that despite the difference in asymptotics between algorithms for directed and undirected graphs, they are fairly competitive in average on random graphs and that some of the directed graph k-shortest path algorithms scale well on random graphs. We hypothesize that two of the popular directed graph k-shortest simple path algorithms with poor worst-case asymptotic complexity, by Yen [33] and Feng [13], have near-linear average-case asymptotic complexity (explaining their scalability in practice). In this paper, we provide considerable theoretical and empirical evidence to support this hypothesis. Thus, our work bridges this important gap between the worst-case complexity and the empirical performance of these algorithms and motivates the usage of these algorithms in many network analytics applications that have strong efficiency requirements.

We first show that the average-case asymptotic complexity of Yen's algorithm is $O(km\log n)$ on random directed graphs with uniform random weights, in the setting where $k = O(n)$ and $m = \Omega(n \cdot \log n)$. This setting is motivated by the facts that (i) choosing $k = \omega(n)$ would result in running times of $\omega(n^2)$ which are infeasible for huge values of n, (ii) the average-case computational savings on random graphs over general graphs are more significant when $m \gg n$, and (iii) we can rely on certain structural properties (lacking in very sparse random graphs) which significantly simplify our proofs. While we also provide a weaker bound of $O(km\log^4 n)$ for sparse random graphs, we show empirical evidence indicating that the stronger bound for the denser case even holds for very sparse random graphs.

Next, we show that the average-case complexity of computing the second shortest path with Feng's algorithm [13] is even better, namely $O(n+m)$ on unweighted random undirected graphs. Our experimental results suggest that similar bounds hold for $k > 2$ as well, even with uniform and exponential edge weights.

Remark: Note that in this paper, the average-case complexity refers to the expected complexity of computing k-shortest paths from a randomly chosen source node to a randomly chosen target node on a directed random graph from the class $\mathscr{D}(n, p, [0,1])$. The class $\mathscr{D}(n, p, [0,1])$ comprises directed graphs on n nodes, where each edge exists with probability p independent of the presence of all other edges, and the edge weights are uniformly distributed between 0 and 1.

2 Related Work

In this paper, our focus is on the average-case complexity of *exact* algorithms for the *loop-less* k-shortest path problem. From a worst-case asymptotic complexity perspective, the best algorithm for this problem in the weighted case is by Gotthilf and Lewenstein [17], that achieves a bound of $O(kn(n \log\log n + m))$. For directed *unweighted* graphs, Roditty and Zwick [29] have improved the bound to $O(km\sqrt{n}\log^2 n)$.

Due to fine-grained conditional complexity results by Williams and Williams [32] and Agarwal and Ramachandran [2], it is unlikely that the asymptotic worst-case complexity of k-shortest path on directed weighted graphs can be improved to $O(n^{3-\delta})$ for some $\delta > 0$ or $o(nm)$, respectively.

In this paper, we focus on a very popular k-shortest simple path algorithm, originally proposed by Yen [33]. Yen's algorithm considers all possible $O(n)$ deviations from each of the previously computed paths. This takes $O(kn(n \log n + m))$ time when using Dijkstra's SSSP algorithm [11] with Fibonacci heaps [14] as a kernel.

There has also been considerable work on practical improvements of the directed k-shortest simple path algorithm (e.g., [23, 27, 25, 24, 30, 31, 19, 13, 22]). In particular, Feng's algorithm [13] claims to be the empirically fastest approach on many graph classes, while still matching the worst-case time complexity of Yen's algorithm. In this paper, we present theoretical and empirical evidence to explain the performance of this algorithm in an average-case setting.

The best k-shortest path algorithm for *undirected* graphs [21] has a significantly better asymptotic worst-case complexity of $O(k(m + n \log n))$, compared to directed graph algorithms. Despite the big gap between the worst-case asymptotic complexity of directed and undirected graph algorithms, a recent experimental paper [3] showed comparable performance of these algorithms on the random graphs, motivating this theoretical study.

Our current study on average-case complexity of exact k-shortest simple path algorithms excludes the work done on approximating k-shortest simple paths [5, 15], all-pair k-shortest simple paths [1], loopy k-shortest path algorithms [12, 4] as well as distance oracles for computing replacement paths [10, 6].

3 Algorithms

3.1 Yen's Algorithm

Given a graph $G(V,E)$, a source node s and a target node t, Yen's algorithm [33] starts by computing the shortest path from s to t, namely $P_0 = (s,\ldots,t)$. The algorithm then maintains a priority queue C of candidates for the next shortest paths. Initially, the queue C consists of only P_0 with the priority $weight(P_0)$.

We then iteratively compute the shortest paths P_i for $i = 1,\ldots,k-1$ and update C with candidate paths for the next iteration. The path P_i is obtained by simply extracting the minimum weight path from C. Let $P_i = (v^i_1 = s, v^i_2, \ldots, v^i_d, \ldots, v^i_l = t)$ and let P_j be the path with maximum common prefix $(v^j_1 = v^i_1, v^j_2 = v^i_2, \ldots, v^j_d = v^i_d)$ among all paths $\{P_1,\ldots,P_{i-1}\}$. Further, let $dev(P_i) = v^i_d$ be the node at which P_i deviated from P_j (we define $v^1_d = s$ for P_1). Then, the new candidate paths to be inserted into C are the shortest paths deviating from P_i at nodes $\{v^i_d,\ldots,v^i_{l-1}\}$. To compute the deviation path from a node v^i_f, we first compute the shortest path P^f from v^i_f to t in the graph G' obtained by removing nodes $\{v^i_1,\ldots,v^i_{f-1}\}$, incident edges and all edges in E^f from G, where $E^f = \{(v^h_f, v^h_{f+1}) : 1 \leq h \leq i$ and $v^h_1 = v^i_1, \ldots, v^h_f = v^i_f\}$. The shortest deviation path from P_i at a node v^i_f is simply the path $\{v^i_1, v^i_2, \ldots, v^i_{f-1}\}$ appended by P^f. We insert it into C with its weight as priority. See Algorithm 1 for a pseudo-code of the algorithm.

3.2 Feng's Algorithm

Feng's algorithm [13] optimizes the computation of deviations in Yen's algorithm by significantly pruning the subgraph for deviation computation. We refer to the pruned subgraph, involved in the computation of the deviation from v^i_f from path P_i, as the yellow graph $Y^i_f(V^{i,f}_y, E^{i,f}_y)$. We next describe the set of nodes and edges in this graph and then modify the description for efficient computation of yellow graphs.

Let T be the reverse shortest path tree rooted in t. When computing the deviation from path P_i at node v^i_f, we conceptually colour the nodes into three colours: Nodes $v^i_1,\ldots,v^i_f$ are coloured red, all nodes whose shortest path to t in T goes through a red node are yellow and the remaining nodes are green. The node set $V^{i,f}_y$ of the yellow graph Y^i_f consists of all yellow nodes, only one red node v^i_f and only one green node t. The set E_y consists of all edges between the yellow nodes and only outgoing edges of v^i_f. All edges $(u,v) \in E$ where u is a yellow node and v is a green node are replaced by edges (u,t) in E_y. The weight of the edge (u,t) in E_y equals the sum of the weight $w(u,v)$ of edge (u,v) in E and the weight of the path $d(v)$ from v to t in T.

We modify the original description of Feng's algorithm [13] to compute the yellow graphs efficiently. The reverse shortest path tree T from t is computed only once, before the deviation computation commences. As part of the shortest path tree computation, we also store the distance $d(v)$ of a node v from t in T in addition to all the in-edges to v. To compute $V^{i,f}_y$, we traverse T from each of the red nodes r to identify the subtree of yellow nodes rooted at r in T. A simple BFS traversal from r in T that skips the other red nodes (and the subtrees rooted at them) suffices for this purpose. The union of nodes

Algorithm 1 Yen's Algorithm

```
Require: $G(V,E)$, $s$, $t$, $K$                                  ▷ source, target, # paths
 1: $K_{\text{paths}} = \emptyset$
 2: $P_0 = \textbf{sssp}(G,s)$, $\alpha_0 = 0$, $j_0 = 0$
 3: $C = \emptyset$
 4: $k = 1$
 5: while $k < K$ and $C \neq \emptyset$ do
 6:     $P_{root} = (v_0^{k-1}, ..., v_{\alpha_{k-1}-1}^{k-1})$
 7:     $G'(V',E') = G[V \setminus P_{root}]$                        ▷ $P_{root}$ as a set.
 8:     for $i = 0, ..., k-2$ do
 9:         if $j_i == j_{k-1}$ or $i == j_{k-1}$ then
10:             $E' = E' \setminus \{(v_{\alpha_i}^i, v_{\alpha_i+1}^i)\}$
11:         end if
12:     end for
13:     for $n = \alpha_{k-1}, ..., q_{k-1} - 1$ do
14:         $E' = E' \setminus \{(v_n^{k-1}, v_{n+1}^{k-1})\}$
15:         $P = \textbf{sssp}(G', n)$
16:         $P = P_{root} \frown P$, $j = k-1$, $\alpha = n$
17:         $C = C \cup \{(P, j, \alpha)\}$
18:         $G'(V',E') = G[V \setminus \{v_n^{k-1}\}]$
19:     end for
20:     $(P_k, j_k, \alpha_k)$ = shortest path in C
21:     $C = C \setminus \{(P_k, j_k, \alpha_k)\}$
22:     $K_{\text{paths}} = K_{\text{paths}} \cup \{P_k\}$
23:     $k = k+1$
24: end while
Ensure: $K_{\text{paths}}$
```

in all these subtrees, together with v_f^i and t, constitutes $V_y^{i,f}$. Since the yellow nodes are disjointly partitioned among the different subtrees and each red node is skipped in exactly one traversal, these traversals touch $O(n_y^i(f) + n_r^i(f))$ nodes and edges altogether, where $n_y^i(f) = |V_y^{i,f}|$ and $n_r^i(f)$ is the number of red nodes. Skipping can be performed efficiently by storing the red nodes in a hash table, resulting in the overall complexity of $O(n_y^i(f) + n_r^i(f))$ for computing $V_y^{i,f}$.

For computing the set $E_y^{i,f}$, we consider all outgoing edges of yellow nodes. For an outgoing edge (y,u) of a yellow node y, if u is a yellow node, we insert (y,u) in $E_y^{i,f}$. If instead u is a green node, we insert (y,t) in $E_y^{i,f}$ with weight equal to $d(u) + w(u,v)$. Otherwise, we ignore it. Clearly, this requires touching $O(m_y^i(f) + m_r^i(f))$ edges, where $m_y^i(f) = |E_y^{i,f}|$ and $m_r^i(f)$ is the number of edges from yellow nodes to red nodes. This is because each outgoing edge from a yellow node either results in an edge in $E_y^{i,f}$ or leads to a red node.

In addition, we add all edges from v_f^i to yellow nodes in $E_y^{i,f}$. In the computation of the i^{th} shortest path, each node in the graph can appear at most once as v_f^i for some deviation. Thus, summing over all k shortest paths, this requires touching $O(km)$ edges in total. Therefore, the total complexity of creating the yellow graphs over all deviations (excluding the initial SSSP computation) is $O(km + \sum_{i=1}^{k-1} \sum_{f=1}^{l} \mathfrak{Y}^i(f))$ for

$$\mathfrak{Y}^i(f) := n_y^i(f) + m_y^i(f) + n_r^i(f) + m_r^i(f) \tag{1}$$

where the inner summation is taken over all deviations in the computation of i^{th} shortest path.

4 Average-Case Analysis of Yen's Algorithm

Theorem 1. *For random directed graphs from $\mathscr{D}(n,p,[0,1])$ with $p = \Omega(\frac{\ln n}{n})$, the average-case complexity of Yen's k-SP algorithm with $k = O(n)$ is $O(km\log n)$ w.h.p.*

Proof. Let us consider the work done by Yen's algorithm when going from the i^{th} shortest path P_i to the $(i+1)^{th}$ shortest path: It computes deviation paths from nodes $dev(P_i)$ in path P_i to t. Each deviating path finding boils down to removing $O(m)$ edges, which can be done in $O(n+m)$ complexity, and subsequently running a SSSP computation. Using Dijkstra's algorithm [11] with Fibonacci heaps [14], the shortest-path computation itself takes $O(n\log n + m)$ time. By Chernoff bounds and the lower bound on the edge probability p we have $m = \Omega(n\log n)$ w.h.p., thus implying that each shortest-path computation will take at most $O(n+m)$ time w.h.p. It remains to show that for $k = O(n)$, the overall number of deviations required by Yen's k-SP algorithm is $O(k\log n)$ w.h.p. In order to see this note in Yen's algorithm, at most one deviation is computed from each node in the i^{th} shortest path when computing the $(i+1)^{th}$ shortest path. Thus, it suffices to prove that the number of hops in the i^{th} shortest path is $O(\log n)$ w.h.p. for $i < k$. To this end we will leverage the following properties proved by Priebe [28]:
(P1) If $p = \Omega(\frac{\ln n}{n})$ for sufficiently large constants then the diameter (i.e., the maximum shortest path weight among all n^2 pairs of vertices) of generated random graphs in $\mathscr{D}(n,p,[0,1])$ is bounded by $O(\frac{\log n}{n \cdot p})$ w.h.p. (P2) If $p = \Omega(\frac{\log n}{n})$ for sufficiently large constants then shortest paths in $\mathscr{D}(n,p,[0,1])$ consist of $O(\log n)$ edges w.h.p.[5]

The proof for (P2) in [28] is based on the observation that with high probability random weighted graphs from $\mathscr{D}(n,p,[0,1])$ do not contain simple paths of total weight at most $\delta := C \cdot (\log n)/(n \cdot p) < 1$ that at the same time consist of more than $D \cdot \log n$ edges for appropriately chosen constants C and D. Since the threshold δ from [28] is just an upper bound on the diameter given in (P1), (P2) is also applicable for the i^{th} shortest path, $i < k = O(n)$, as long as its path weight stays below δ. Thus, in the remainder we only need to argue that $\mathscr{D}(n,p,[0,1])$ contains $\Omega(n)$ edge-disjoint $s-t$ paths of weight less than δ with high probability:

Prior to actually generating the input graph G according to $\mathscr{D}(n,p,[0,1])$ let us arbitrarily partition its nodes into two equally sized sets while ensuring that s and t belong to

[5]In fact, Priebe showed (P1) in the form of his Lemma 3.4 and (P2) in the form of his Lemma 3.10, for any edge weight distribution function F that satisfies the following requirements: F is concentrated on $[0,\infty)$, $F(0) = 0$ and that $F'(0)$ exists and is strictly positive. Since the uniform edge weight distribution between 0 and 1 is compatible with these requirements, (P1) and (P2) hold for $\mathscr{D}(n,p,[0,1])$, too.

different sets. Let the graphs induced by the two partitions be G_s and G_t, respectively. By construction G_s and G_t can be considered to stem from $\mathscr{D}(n/2, p, [0,1])$. Hence, w.h.p., for $p = \Omega(\frac{\ln n}{n})$ both subgraphs are strongly connected [18] and (P1) & (P2) also hold for them. For each node x in G_s let I_x denote the random binary indicator variable for the event that there is at least one edge from x with a target node y in G_t and edge weight at most $\frac{1}{np}$. There are $n/2$ potential edges from x into G_v each independently showing up with probability p. Hence, the probability for I_v being 1 is at least $1-(1-p\cdot\frac{1}{np})^{n/2} = 1-(1-\frac{1}{n})^{n/2} \geq 1-e^{-1/2} > 1/4$. Thus, $E[I_v]$ is a constant between 0 and 1 and therefore we can use Chernoff bounds to show that $\sum_{x\in G_u} I_v$ is $\Theta(n)$ w.h.p. This implies that there are $\Theta(n)$ alternative paths from s to t via edges (x_i, y_i). Due to (P1) each of these paths consists of at most $2\cdot D\cdot\log(n/2)+1 = O(\log n)$ edges w.h.p. and due to (P2) their respective paths lengths are at most $O(2\cdot\frac{\log(n/2)}{(n/2)\cdot p}+\frac{1}{n\cdot p}) = O(\frac{\log n}{n\cdot p})$ w.h.p., which is below the threshold δ for sufficiently large p. □

Theorem 2. *For* sparse *random directed graphs from $\mathscr{D}(n, p, [0,1])$ with $\frac{4}{n} \leq p \leq \frac{\Theta(\ln n)}{n}$, the average-case complexity of Yen's k-SP algorithm with $k = O(n)$ and both s and t being part of the giant strong component*[6] *is bounded by $O(km\log^4 n)$ w.h.p.*

Proof. This rather crude upper bound can be shown with a similar proof like the one we used for Theorem 1. One extra log factor results from the now dominating priority queue operation costs within Dijkstra's algorithm[7]. Furthermore, the underlying proof technique for (P2) does not seem to be fully transferable to the sparse setting where the expected weights for shortest s-t paths are higher: by subdividing these paths into logarithmically many subpaths with smaller weights, we can reuse the old analysis but this incurs an extra multiplicative log factor in our upper bound for the time. Similarly, for sparse random graphs the relative node degree fluctuation is likely to be larger than on their dense counterparts, which accounts for at most another logarithmic factor in case one wants to stick to the old analysis structure. □

Theorems 1 and 2 carry over to the respective *unweighted* cases. The proofs can be streamlined since shortest path distances now coincide with the number of edges on these paths. As a positive consequence, with high probability, we now find even more equally suited alternative shortest paths between the two graph partitions, which is why the results also hold for higher values of k up to $\Theta(n^2\cdot p) = \Theta(m)$.

4.1 Empirical Results on Average-Case Behavior of Yen's Algorithm

The bound of $O(km\log n)$ in Theorem 1 was proved under the assumption of $p > 2\ln n/n$ (and $k = O(n)$). Next, we provide empirical evidence that similar bound is likely to hold for the setting of even sparser graphs, even though the bound proven in Theorem 2 is off by a poly-logarithmic factor. For this, we consider $n = 2^j\cdot 10^6$ for

[6]For $p \geq 4/n$ the size of the giant strong component is $\Omega(n)$ w.h.p. [20].

[7]We are aware that there exist improved SSSP algorithms with linear average-case time (e.g., [16, 26]) for initially uniformly distributed edge weights. Unfortunately, for the particular subgraphs on which we would like to apply these better SSSP algorithms it seems difficult to prove that their overall edge weight distribution remains sufficiently uniform

$j = 0, \dots, 7$, $m = 4n$ and generate directed random graphs with n nodes, m edges following the $G(n,m)$ model [7] without self-loops[8]. For the edge weights, we consider two settings, (i) unweighted and (ii) uniform random edge-weight distribution in $[0,1]$. After generating a graph for a fixed n, we draw the source and target nodes uniformly at random and average the results over 10 different source-target pairs.

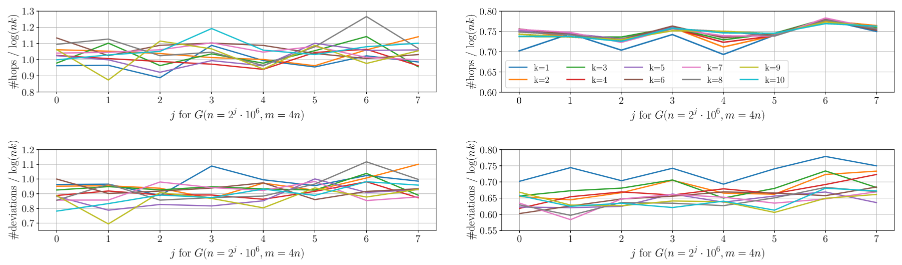

Fig. 1. The number of hops (1st row) of the k^{th} shortest path and the number of deviations (2nd row) to compute the $(k+1)^{th}$ shortest path from the k^{th} with respect to $\log(nk)$ for $G(n,4n)$ with uniform random edge weights (left) and unweighted (right).

Figure 1 show that the number of hops in the k^{th} shortest path and the number of deviations required to compute the $(k+1)^{th}$ shortest path is quite well approximated by $\log(nk)$, indicating that even in this sparse setting, the average-case complexity bound of $O(km\log(nk))$ ($O(km\log n)$ for $k = O(n)$) is likely to hold. Figure 2 shows the same for larger values of k.

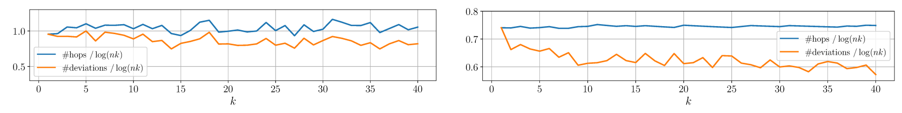

Fig. 2. The number of hops of the k^{th} shortest path and deviation computed from it for the $(k+1)^{th}$ shortest path with respect to $\log(nk)$, computed for a directed random graph from class $G(n = 32 \cdot 10^6, 4n)$ with uniform random edge weights (left) and unweighted (right).

5 Average-Case Analysis of Feng's Algorithm

Theorem 3. *On unweighted random undirected graphs with $p = \Omega(\frac{\ln n}{n})$, the average-case complexity of Feng's algorithm for computing the second shortest path is $O(m)$.*

[8]We note that for convex properties, the $G(n,p)$ and $G(n,m)$ random graph models are equivalent up to lower order terms, provided $m \approx p \cdot N$ (where N is the maximum number of edges insertable in a graph, $N = n(n-1)$ for directed graphs) [7]. Thus, the empirical results shown for the $G(n,m)$ model should also hold for the $G(n,p)$ model.

Proof. Consider the three main steps involved in Feng's algorithm: (i) Computing SSSP from t: This requires $O(n+m)$ time using BFS for unweighted graphs. (ii) Computation of yellow graphs: As argued in Section 3, this requires $O(km+\sum_{i=1}^{k-1}\sum_{f=1}^{l}\mathfrak{Y}^i(f))$ operations, where the second summation is taken over all deviations in the computation of i^{th} shortest path. (iii) SSSP computations in the yellow graph: Every shortest path computation can be performed in $O(\mathfrak{Y}^i(f))$ time using BFS. Once again, the summation over all deviations yields a total bound of $O(\sum_{i=1}^{k-1}\sum_{f=1}^{l}\mathfrak{Y}^i(f))$.

Thus, the total complexity of Feng's algorithm for computing the k shortest paths is $O(n+m+\sum_{i=1}^{k-1}\sum_{f=1}^{l}\mathfrak{Y}^i(f))$, which is $O(m+\sum_{f=1}^{l}\mathfrak{Y}^1(f))$ for the second shortest path. Lemma 1 shows that $\mathbf{E}[\sum_{f=1}^{l}\mathfrak{Y}^1(f)] = O(n+m)$, making the average-case complexity of this algorithm to be $O(m)$.

Lemma 1. *If $p \geq C\ln n/n$ for some constant $C > 2$, $\mathbf{E}[\sum_{f=1}^{l}\mathfrak{Y}^1(f)] = O(n+m)$.*

Proof. We first show that $\sum_{f=1}^{l} n_y^1(f) + n_r^1(f) = O(n)$. Then we will argue that $\sum_{f=1}^{l} m_y^1(f) + m_r^1(f) = O(m)$.

Consider the i^{th} node $t(i)$ on the shortest path from s to t. We want to show that, in expectation, the number of nodes u such that the shortest path from s to u goes through $t(i)$ decreases roughly exponentially in i. To this end, we analyse the structure of the BFS tree rooted at s. Let Γ_i denote the set of nodes at distance i from s, thus for any $y \in \Gamma_{i+1}$ there exists $x \in \Gamma_i$ such that (x,y) is an edge. We assume that the parent of y in the BFS tree is randomly chosen among all such x, if there is more than one possibility.

The depth of the BFS tree is $O(\log n)$ w.h.p. [28] and, by Lemma 8 from [8], for each $1 \leq i \leq i_0 = 2/3\log n/\log(np)$, with probability at least $1-o(1/n)$, we have $|\Gamma_i| = \Omega((np)^i)$. We would like to have a lower bound on $|\Gamma_i|$ for $i > i_0$ as well. We choose the smallest i_1 such that $|\bigcup_{i=0}^{i_1}\Gamma_i| > n - 2\sqrt{n}$. We claim that now, for any $i = i_0 + 1,\ldots,i_1-1$, $|\Gamma_i| \geq \sqrt{n}$ with probability $1-1/n$. Assume that $|\Gamma_i| < \sqrt{n}$, then we have a partition of the nodes into three sets U,S and V with the following properties: $|S| < \sqrt{n}$, $|U|,|V| \geq 2\sqrt{n}$, and there are no edges between U and V. But the probability of such a partition to exist can be bounded for $|U| = x$ as follows:

$$\binom{n}{\sqrt{n}}\binom{n}{x}(1-p)^{x(n-\sqrt{n}-x)} \leq (\sqrt{n}e)^{\sqrt{n}}(ne/x)^x e^{-p\cdot x(n-\sqrt{n}-x)}$$
$$\leq e^{\sqrt{n}\ln n - x\ln x + x - x\ln n + 2c\ln n} \leq e^{\sqrt{n}\ln n - x\ln x + (2c-x/2)\ln n}$$

which is at most $1/n^2$ for sufficiently large n and $x \geq 2\sqrt{n}$. Summing over all possible $|U|$, the probability is bounded by $1/n$.

We rephrase the random procedure used to create the graph. Instead of fixing every edge, we work in phases corresponding to the layers of the BFS tree. Having chosen Γ_i, we select (with appropriate probability) the nodes of Γ_{i+1}, but do not fix the edges between $x \in \Gamma_i$ and $y \in \Gamma_{i+1}$ yet. Also, we terminate after having obtained Γ_{i_1}, thus there might be still up to $2\sqrt{n}$ unvisited nodes. To fully determine the relevant part of the BFS tree we still need to choose, for every $y \in \Gamma_{i+1}$, for $i = 0,1,\ldots,i_1-1$, its parent $x \in \Gamma_i$. We assume that $u \in \Gamma_{i_1}$.

We need to bound the probability that, for a randomly chosen node u, the shortest path from s to u goes through $t(i)$. By union bound, this is at most $1/|\Gamma_i| + 1/|\Gamma_{i+1}| +$

$\ldots + 1/|\Gamma_{i_1-1}|$, because for the shortest path from s to u and s to t to meet for the first time in some node $x \in \Gamma_j$ it must hold that two nodes $y, y' \in \Gamma_{j+1}$ have chosen the same parent, which holds with probability $1/|\Gamma_j|$. Summing over all i we obtain that w.h.p. the expected number of such nodes u is

$$\begin{aligned}\sum_{i=0}^{i_1-1} n \sum_{j=i}^{i_1-1} 1/|\Gamma_j| &\le O(n\log^2 n/\sqrt{n} + \sum_{i=0}^{i_0}\sum_{j=i}^{i_0} n/(np)^j) \\ &\le O(n\log^2 n/\sqrt{n} + \sum_{i=0}^{i_0} n/(np)^i) = O(n).\end{aligned}$$

It remains to argue that $\sum_{f=1}^{l} m_y^1(f) + m_r^1(f) = O(m)$. The expected degree of a node is $np \ge c\ln n$, so by standard Chernoff bounds with probability $1 - 1/n$ the degree of every node is $O(np)$. Thus, $\sum_{f=1}^{l} m_y^1(f) + m_r^1(f) = O(n^2 p)$ with probability $1 - 1/n$. Finally, the expected value of m is $n^2 p/2$, so by Chernoff bounds m is less than $n^2 p/4$ with probability at most $1/n$ for sufficiently large n. Thus, with probability $1 - O(1/n)$, $\sum_{f=1}^{l} m_y^1(f) + m_r^1(f) = O(n^2 p) = O(m)$.

5.1 Empirical Results on Feng's Average-Case Behavior

Theorem 3 proves that the average-case complexity of computing second shortest path using Feng's algorithm is $O(n+m)$ for $p \ge c\ln n/n$. This is based on Theorem 1 that shows that in this setting, $\mathfrak{Y}^i(f)$ grows exponentially as a function of the relative distance of deviation node to target node and $\sum_{f=1}^{l} \mathfrak{Y}^i(f) = O(n+m)$. In this section, we show experimental results which indicate that the results are likely to be true for the (i) k^{th} shortest path for $k > 2$ and (ii) for sparser graphs.

The experimental setting is similar to that of the empirical analysis of Yen's algorithm. Consider the computation of candidate deviation paths from the i^{th} shortest path with #*hops* hops. Let f be the deviation node as numbered from t ($f = 0$ for s and $f = l - 1$ for the predecessor of t). Figure 3 shows that $\mathfrak{Y}^i(f)$ grows exponentially the closer the deviation node is to the target node (so while f/l gets closer to 1), even when the results are averaged over 10 shortest paths (and not just for the second shortest path). This is true both for unweighted and uniformly random edge weight setting.

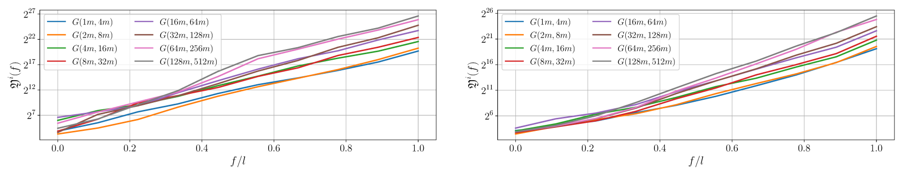

Fig. 3. $\mathfrak{Y}^i(f)$ averaged over the first $k = 10$ shortest paths for $G(n, 4n)$ with uniform random edge weights (left) and unweighted (right). Since all of the k-shortest paths have a different number of hops, the size of the yellow graph is plotted with respect to the number of hops.

Finally, Figure 4 shows that $\sum_{f=1}^{l} \mathfrak{Y}^i(f)/(n+m)$ is close to being a constant, even for larger values of k (for both unweighted and uniformly random edge weights). Again, this suggests that Theorem 1 is likely to hold for $k > 2$ as well.

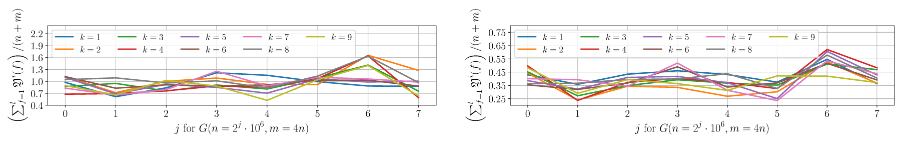

Fig. 4. $(\sum_{f=1}^{l} \mathfrak{Y}^i(f))/(n+m)$, the total size of all yellow graphs with respect to n and m during the computation of the $(k+1)^{th}$ shortest path for $G(n,4n)$ with uniform random edge weights (left) and unweighted (right).

6 Conclusion

We have shown that in the not-too-sparse setting, the average-case complexity of Yen's algorithm is $O(km\log n)$ in contrast to its worst-case complexity of $O(kmn)$. For Feng's algorithm, we found that the considerable pruning of the subgraphs for computing deviations is not only a fast heuristic in practice, but it also enables the proof of a better average-case complexity bound of $O(km)$ for the second shortest path on unweighted graphs. Proving that this bound still holds for $k > 2$ and in the sparser settings remains an open combinatorial problem.

Acknowledgements. We are grateful to Erika Duriakova for providing us the code for the implementation of Yen's algorithm, the SSSP subroutines and her generous help with debugging our usage of her code.

References

1. Agarwal, U., Ramachandran, V.: Finding k simple shortest paths and cycles. In: Proc. 27th ISAAC, LIPIcs, pp. 8:1–8:12 (2016)
2. Agarwal, U., Ramachandran, V.: Fine-grained complexity for sparse graphs. In: Proc. 50th STOC, pp. 239–252. ACM (2018)
3. Ajwani, D., Duriakova, E., Hurley, N., Meyer, U., Schickedanz, A.: An empirical comparison of k-shortest simple path algorithms on multicores. In: Proc. 47th ICPP (2018)
4. Akiba, T., Hayashi, T., Nori, N., Iwata, Y., Yoshida, Y.: Efficient top-k shortest-path distance queries on large networks by pruned landmark labeling. In: Proc. 29th AAAI, pp. 2–8 (2015)
5. Bernstein, A.: A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In: Proc. 21st SODA, pp. 742–755 (2010)
6. Bernstein, A., Karger, D.R.: A nearly optimal oracle for avoiding failed vertices and edges. In: Proc. 41st STOC, pp. 101–110 (2009)
7. Bollobás, B.: Models of Random Graphs, 2 edn., p. 3459. Cambridge Studies in Advanced Mathematics. Cambridge University Press (2001)

8. Chung, F., Lu, L.: The diameter of sparse random graphs. Advances in Applied Mathematics **26**(4), 257 – 279 (2001). DOI https://doi.org/10.1006/aama.2001.0720
9. Clarke, S., Krikorian, A., Rausen, J.: Computing the N best loopless paths in a network. Journal of the Society for Industrial and Applied Mathematics **11(4)**, 10961102 (1963)
10. Demetrescu, C., Thorup, M., Chowdhury, R.A., Ramachandran, V.: Oracles for distances avoiding a failed node or link. SIAM J. Comput. **37**(5), 1299–1318 (2008)
11. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische mathematik **1**(1), 269–271 (1959)
12. Eppstein, D.: Finding the *k* shortest paths. SIAM J. Comput. **28**(2), 652–673 (1998)
13. Feng, G.: Finding k shortest simple paths in directed graphs: A node classification algorithm. Networks **64**(1), 6–17 (2014)
14. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM (JACM) **34**(3), 596–615 (1987)
15. Frieder, A., Roditty, L.: An experimental study on approximating *k* shortest simple paths. Journal of Experimental Algorithmics (JEA) **19**, 1–5 (2015)
16. Goldberg, A.V.: A practical shortest path algorithm with linear expected time. SIAM J. Comput. **37**(5), 1637–1655 (2008)
17. Gotthilf, Z., Lewenstein, M.: Improved algorithms for the k simple shortest paths and the replacement paths problems. Information Processing Letters **109**(7), 352–355 (2009)
18. Graham, A.J., Pike, D.A.: A note on thresholds and connectivity in random directed graphs. Atlantic Electronic Journal of Mathematics **3**(1), 1–5 (2008)
19. Hershberger, J., Maxel, M., Suri, S.: Finding the k shortest simple paths: A new algorithm and its implementation. ACM Transactions on Algorithms (TALG) **3**(4), 45 (2007)
20. Karp, R.M.: The transitive closure of a random digraph. Random Struct. Algorithms **1**(1), 73–94 (1990)
21. Katoh, N., Ibaraki, T., Mine, H.: An efficient algorithm for k shortest simple paths. Networks **12**(4), 411–427 (1982)
22. Kurz, D., Mutzel, P.: A sidetrack-based algorithm for finding the k shortest simple paths in a directed graph. CoRR/Arxiv **abs/1601.02867** (2016)
23. Lawler, E.L.: A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. Manage. Sc. **18**(7), 401–405 (1972)
24. Martins, E., Pascoal, M.M., Santos, J.: Deviation algorithms for ranking shortest paths. International Journal of Foundations of Computer Science **10**(3), 247–262 (1999)
25. Martins, E.Q., Pascoal, M.M.: A new implementation of Yens ranking loopless paths algorithm. Journ. Belgian, French and Italian Oper. Res. Societies **1**(2), 121–133 (2003)
26. Meyer, U.: Average-case complexity of single-source shortest-paths algorithms: lower and upper bounds. J. Algorithms **48**(1), 91–134 (2003)
27. Perko, A.: Implementation of algorithms for k shortest loopless paths. Networks **16**(2), 149–160 (1986)
28. Priebe, V.: Average-case complexity of shortest-paths problems. Ph.D. thesis, Saarland University (2001). URL http://scidok.sulb.uni-saarland.de/volltexte/2007/1180/
29. Roditty, L., Zwick, U.: Replacement paths and k simple shortest paths in unweighted directed graphs. ACM Transactions on Algorithms (TALG) **8**(4), 33 (2012)
30. Sedeño-Noda, A.: An efficient time and space K point-to-point shortest simple paths algorithm. Applied Mathematics and Computation **218**(20), 10,244–10,257 (2012)
31. Shih, Y.K., Parthasarathy, S.: A single source k-shortest paths algorithm to infer regulatory pathways in a gene network. Bioinformatics **28(12)**, i49–i58 (2012)
32. Williams, V.V., Williams, R.: Subcubic equivalences between path, matrix and triangle problems. In: Proc. 51th FOCS, pp. 645–654 (2010)
33. Yen, J.Y.: Finding the *k* shortest loopless paths in a network. Management Science **17**(11), 712–716 (1971)