# Faster Detours in Undirected Graphs

**Shyan Akmal**
MIT EECS and CSAIL, Cambridge, MA, USA

**Virginia Vassilevska Williams**
MIT EECS and CSAIL, Cambridge, MA, USA

**Ryan Williams**
MIT EECS and CSAIL, Cambridge, MA, USA

**Zixuan Xu**
MIT Mathematics, Cambridge, MA, USA

## Abstract

The $k$-Detour problem is a basic path-finding problem: given a graph $G$ on $n$ vertices, with specified nodes $s$ and $t$, and a positive integer $k$, the goal is to determine if $G$ has an $st$-path of length exactly $\mathsf{dist}(s,t)+k$, where $\mathsf{dist}(s,t)$ is the length of a shortest path from $s$ to $t$. The $k$-Detour problem is NP-hard when $k$ is part of the input, so researchers have sought efficient parameterized algorithms for this task, running in $f(k)\,\mathrm{poly}(n)$ time, for $f(\cdot)$ as slow-growing as possible.

We present faster algorithms for $k$-Detour in undirected graphs, running in $1.853^k\,\mathrm{poly}(n)$ randomized and $4.082^k\,\mathrm{poly}(n)$ deterministic time. The previous fastest algorithms for this problem took $2.746^k\,\mathrm{poly}(n)$ randomized and $6.523^k\,\mathrm{poly}(n)$ deterministic time [Bezáková-Curticapean-Dell-Fomin, ICALP 2017]. Our algorithms use the fact that detecting a path of a given length in an undirected graph is easier if we are promised that the path belongs to what we call a "bipartitioned" subgraph, where the nodes are split into two parts and the path must satisfy constraints on those parts. Previously, this idea was used to obtain the fastest known algorithm for finding paths of length $k$ in undirected graphs [Björklund-Husfeldt-Kaski-Koivisto, JCSS 2017], intuitively by looking for paths of length $k$ in randomly bipartitioned subgraphs. Our algorithms for $k$-Detour stem from a new application of this idea, which does not involve choosing the bipartitioned subgraphs randomly.

Our work has direct implications for the $k$-Longest Detour problem, another related path-finding problem. In this problem, we are given the same input as in $k$-Detour, but are now tasked with determining if $G$ has an $st$-path of length *at least* $\mathsf{dist}(s,t)+k$. Our results for $k$-Detour imply that we can solve $k$-Longest Detour in $3.432^k\,\mathrm{poly}(n)$ randomized and $16.661^k\,\mathrm{poly}(n)$ deterministic time. The previous fastest algorithms for this problem took $7.539^k\,\mathrm{poly}(n)$ randomized and $42.549^k\,\mathrm{poly}(n)$ deterministic time [Fomin et al., STACS 2022].

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** path finding, detours, parameterized complexity, exact algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.7

**Related Version** *Full Version*: https://arxiv.org/abs/2307.01781 [1]

**Funding** *Shyan Akmal*: Supported in part by NSF grant CCF-2127597.
*Virginia Vassilevska Williams*: Supported by NSF CCF-2129139.
*Ryan Williams*: Supported by NSF CCF-1909429.

**Acknowledgements** The first author thanks Nicole Wein for several helpful discussions, and Fedor V. Fomin for answering a question about previous algorithms for $k$-Longest Path.





## 1 Introduction

The $k$-Path problem is a well-studied task in computer science:

> $k$-Path
> **Given:** $k \in \mathbb{N}^+$, a graph $G$, nodes $s$ and $t$.
> **Determine:** Does $G$ contain a simple path of length $k$ from $s$ to $t$?

For graphs $G$ with $n$ nodes, this problem can be easily solved in $O(kn^k)$ time by enumerating all sequences of $k$ vertices. In the 1980s, Monien [13] showed that the $k$-Path problem is actually fixed-parameter tractable (FPT) in $k$, presenting a $k!\,\mathrm{poly}(n)$ time algorithm solving $k$-Path. Since then, significant research has gone into obtaining faster algorithms for $k$-Path, with better dependence on $k$ (see [3, Table 1] for an overview of the many results in this line of work). This research culminated in the work of Koutis and Williams [11, 16, 12], who showed that $k$-Path can be solved in $2^k\,\mathrm{poly}(n)$ (randomized) time, and Björklund, Husfeldt, Kaski, and Koivisto [3, Section 2], who proved that in undirected graphs, $k$-Path can be solved even faster in $1.657^k\,\mathrm{poly}(n)$ (randomized) time. *Throughout this paper, we assume that algorithms are randomized (and return correct answers with high probability in the stated time bounds), unless otherwise specified.*

The $k$-Path problem is a parameterized version of the NP-complete Longest Path problem, but it is not the only natural parameterization. Various other parameterizations of $k$-Path have been proposed and studied, which we consider in the present paper.

1. **Finding a path of length at least $k$.** Instead of looking for a path of length *exactly* $k$ from $s$ to $t$, one can try to determine the existence of an $st$-path of length *at least* $k$:

   > $k$-Longest Path
   > **Given:** $k \in \mathbb{N}^+$, a graph $G$, nodes $s$ and $t$.
   > **Determine:** Does $G$ contain a simple path of length ***at least*** $k$ from $s$ to $t$?

   Observe that in the $k$-Longest Path problem, the length of a solution path is not necessarily bounded as a function of $k$. However, it is known that $k$-Longest Path is also FPT: work of Zehavi [17] and Fomin, Lokshtanov, Panolan, and Saurabh [9] implies that $k$-Longest Path can be solved in $4^k\,\mathrm{poly}(n)$ time. More recently, Eiben, Koana, and Wahlström [7, Section 6.3] proved that over undirected graphs, $k$-Longest Path can be solved in $1.657^k\,\mathrm{poly}(n)$ time, matching the fastest known runtime for $k$-Path.

2. **Finding an $st$-path longer than a polynomial-time guarantee.** Another parameterization for $k$-Path is motivated by the fact that one can find a *shortest path* from $s$ to $t$ in polynomial time. If the shortest path distance $\mathsf{dist}(s,t)$ happens to already be long, then it is actually "easy" to find a long path from $s$ to $t$. Therefore, it is natural to consider the parameterized complexity of searching for an $st$-path that is $k$ edges *longer* than the shortest path length from $s$ to $t$. Our work focuses on these so-called "detour" variants of the path detection problems discussed above.

   > $k$-Detour (a.k.a. $k$-Exact Detour)
   > **Given:** $k \in \mathbb{N}^+$, a graph $G$, nodes $s$ and $t$.
   > **Determine:** Does $G$ contain a simple path of length $\mathsf{dist}(s,t)+k$ from $s$ to $t$?

Since $k$-Path efficiently reduces to solving a single instance of $(k-1)$-Detour,[1] the $k$-Detour problem is at least as hard as the classical $k$-Path problem.
The $k$-Detour problem was introduced by Bezáková, Curticapean, Dell, and Fomin [2], who showed that it can be solved by calling polynomially many instances of $\ell$-Path, for path lengths $\ell \leq 2k+1$. Employing the fastest known $k$-Path algorithms, this implies that $k$-Detour can be solved in $2^{2k}\operatorname{poly}(n) = 4^k \operatorname{poly}(n)$ time in general, and even faster over undirected graphs in $1.657^{2k}\operatorname{poly}(n) \leq 2.746^k \operatorname{poly}(n)$ time.

The two parameterizations above can be combined to produce the following problem:

$k$-Longest Detour
**Given:** $k \in \mathbb{N}^+$, a graph $G$, nodes $s$ and $t$.
**Determine:** Does $G$ contain a simple path of length ***at least*** $\mathsf{dist}(s,t)+k$ from $s$ to $t$?

Observe that $k$-Longest Detour is at least as hard as $k$-Longest Path. Unlike the problems discussed above, $k$-Longest Detour over directed graphs is not known to be FPT: in fact, it remains open whether $k$-Longest Detour is in P even for the special case of $k = 1$! However, in undirected graphs, Fomin, Golovach, Lochet, Sagunov, Simonov, and Saurabh [8] showed that $k$-Longest Detour can be reduced to solving $p$-Detour for $p \leq 2k$, and then solving polynomially many instances of $\ell$-Longest Path, for $\ell \leq 3k/2$. Employing the fastest known algorithms for $k$-Detour and $k$-Longest Path as subroutines, this implies that $k$-Longest Detour can be solved over undirected graphs in $7.539^k \operatorname{poly}(n)$ time.

The algorithms for $k$-Detour and $k$-Longest Detour discussed above are significantly slower than the fastest known algorithms for the analogous $k$-Path and $k$-Longest Path problems. This motivates the questions: can $k$-Detour be solved as quickly as $k$-Path, and can $k$-Longest Detour be solved as quickly as $k$-Longest Path? Given the extensive and influential line of work that has gone into finding faster algorithms for $k$-Path and $k$-Longest Path, obtaining faster algorithms for these detour problems as well is an interesting open problem in parameterized complexity and exact algorithms.

## Our Results

The main result of our work is a faster algorithm for $k$-Detour on undirected graphs.

▶ **Theorem 1.** *In undirected graphs, $k$-Detour can be solved in $1.853^k \operatorname{poly}(n)$ time.*

This marks a significant improvement over the previous fastest $2.746^k \operatorname{poly}(n)$ time algorithm for $k$-Detour (and shows, for example, that this problem can be solved in faster than $2^k \operatorname{poly}(n)$ time, which is often a barrier for parameterized problems). Since the fastest known algorithms for $k$-Longest Detour over undirected graphs have a bottleneck of solving $2k$-Detour, Theorem 1 implies the following result.

▶ **Theorem 2.** *In undirected graphs, $k$-Longest Detour can be solved in $3.432^k \operatorname{poly}(n)$ time.*

Again, this is a significant improvement over the previous fastest algorithm for $k$-Longest Detour on undirected graphs, which ran in $7.539^k \operatorname{poly}(n)$ time.

Our algorithm for Theorem 1 applies the fact that $k$-Path is easier to solve on undirected graphs which have a prescribed vertex partition into two sets, where we constrain the path to contain a particular number of nodes from one set, and a particular number of edges

[1] Given an instance of $k$-Path, add an edge from $s$ to $t$. Then a solution to $(k-1)$-Detour in this new graph corresponds to a solution to $k$-Path in the original graph.

whose vertices are in the other set. Formally, we consider the $(\ell, k_1, \ell_2)$-Bipartitioned Path problem: given a graph $G$ on $n$ nodes, whose vertices are partitioned into parts $V_1$ and $V_2$, with distinguished vertices $s$ and $t$, the goal is to determine if $G$ contains a simple path from $s$ to $t$ of length $\ell$, which uses exactly $k_1$ vertices from $V_1$, and exactly $\ell_2$ edges whose endpoints are both in $V_2$. A careful application of the following result from [3] is the main source of the speed-up in our algorithm for $k$-Detour.

▶ **Lemma 3** ([3, Section 2]). *Let $\ell, k_1, \ell_2$ be nonnegative integers satisfying the inequality $\ell + 1 \geq k_1 + 2\ell_2$. Then over undirected graphs, the $(\ell, k_1, \ell_2)$-Bipartitioned Path problem can be solved in $2^{k_1+\ell_2}\,\mathrm{poly}(n)$ time.*

Although this "Bipartitioned Path" problem may appear esoteric at first, Lemma 3 plays a crucial role in obtaining the fastest known algorithm for $k$-Path in undirected graphs [3], and an analogue of Lemma 3 for paths of length at least $k$ is the basis for the fastest known algorithm for $k$-Longest Path in undirected graphs [7]. Proofs of Lemma 3 can be found in [3, Section 2], [5, Section 10.4], and in the full version of this paper in [1, Appendix B].

In Section 3, we provide an intuitive overview of how Lemma 3 helps us obtain our algorithm for $k$-Detour.

The fastest known algorithms for the path and detour problems discussed above all use randomness. Researchers are also interested in obtaining fast *deterministic* algorithms for these problems. We note that a simplified version of our algorithm for $k$-Detour implies faster deterministic algorithms for these detour problems over undirected graphs.

▶ **Theorem 4.** *The $k$-Detour problem can be solved over undirected graphs by a deterministic algorithm in $4.082^k\,\mathrm{poly}(n)$ time.*

Prior to this work, the fastest known deterministic algorithm for $k$-Detour on undirected graphs ran in $6.523^k\,\mathrm{poly}(n)$ time [2].

▶ **Theorem 5.** *The $k$-Longest Detour problem can be solved over undirected graphs by a deterministic algorithm in $16.661^k\,\mathrm{poly}(n)$ time.*

Prior to this work, the fastest known deterministic algorithm for $k$-Longest Detour on undirected graphs ran in $42.549^k\,\mathrm{poly}(n)$ time [8].

In summary, we obtain new randomized and deterministic algorithms for $k$-Detour and $k$-Longest Detour over undirected graphs, whose runtimes present significant advances over what was previously known for these problems.

## Organization

The remainder of this paper presents our new algorithms $k$-Detour. A thorough discussion of additional related work can is included in the full version of this paper in [1, Appendix A].

In Section 2, we introduce relevant notation, assumptions, and definitions concerning graphs. In Section 3, we provide an overview of our algorithm for $k$-Detour. In Section 4, we present the details of our algorithm, and prove its correctness. The runtime analysis for our algorithm (and thus the formal proofs of Theorems 1, 2, 4, and 5, given correctness of our algorithm) are presented in Section 5. In Section 6, we highlight some open problems.

## 2 Preliminaries

Given positive integers $a$ and $b$, we let $[a] = \{1, 2, \ldots, a\}$, and $[a, b] = \{a, a+1, \ldots, b\}$. Given an integer $a$ and a set of integers $S$, we define $a + S = \{a + s \mid s \in S\}$.

Throughout, we let $G$ denote the input graph. We assume that $G$ is undirected, has vertex set $V$ with $|V| = n$, and, without loss of generality, is connected.[2] Throughout, we let $s$ and $t$ denote the two distinguished vertices that come as part of the input to the $k$-Detour problem. Given a vertex $u$, we let $d(u) = \mathsf{dist}(s, u)$ denote its distance from $s$. This distance is well-defined, since $G$ is connected. Given a path $P$ containing vertices $u$ and $v$, we let $P[u, v]$ denote the subpath from $u$ to $v$ on $P$.

Given an edge $e = (u, v)$ from $u$ to $v$, we say $e$ is ***forward*** if $d(v) = d(u) + 1$, ***backwards*** if $d(v) = d(u) - 1$, and ***stable*** if $d(v) = d(u)$. In an undirected graph, by triangle inequality and symmetry of distance, adjacent vertices $u$ and $v$ have $|d(u) - d(v)| \leq 1$, so every edge in a path falls into one of these three categories.

Given two vertices $u, v \in V$, let $G_{(u,v]}$ denote the induced subgraph of $G$ on the set $\{u\} \cup \{w \in V \mid d(u) < d(w) \leq d(v)\}$. Let $G_{(u,\infty)}$ denote the induced subgraph of $G$ on the set $\{u\} \cup \{w \in V \mid d(u) < d(w)\}$. Note that for every $u$ and $v$, the subgraphs $G_{(u,v]}$ and $G_{(v,\infty)}$ overlap at vertex $v$, but are disjoint otherwise.

# 3 Technical Overview

In this section, we provide an overview of how our $k$-Detour algorithm works. Our starting point is the algorithm for this problem presented in [2, Section 4], which we review in Section 3.1. Then in Section 3.2 we review how the algorithm from Lemma 3 for $(\ell, k_1, \ell_2)$-Bipartitioned Path has previously been used to obtain the fastest known algorithm for $k$-Path in undirected graphs. With this context established, in Section 3.3 we outline how we combine the techniques from Sections 3.1 and 3.2 with new ideas to prove Theorem 1.

## 3.1 Previous Detour Algorithm

The previous algorithm for $k$-Detour from [2, Section 4] performs dynamic programming over nodes in the graph, starting from $t$ and moving to vertices closer to $s$. In the dynamic program, for each vertex $x$ with $d(x) \leq d(t)$, we compute all offsets $r \leq k$ such that there is a path of length $d(t) - d(x) + r$ from $x$ to $t$ in the subgraph $G_{(x,\infty)}$. Determining this set of offsets for $x = s$ solves the $k$-Detour problem, since $G_{(s,\infty)} = G$.

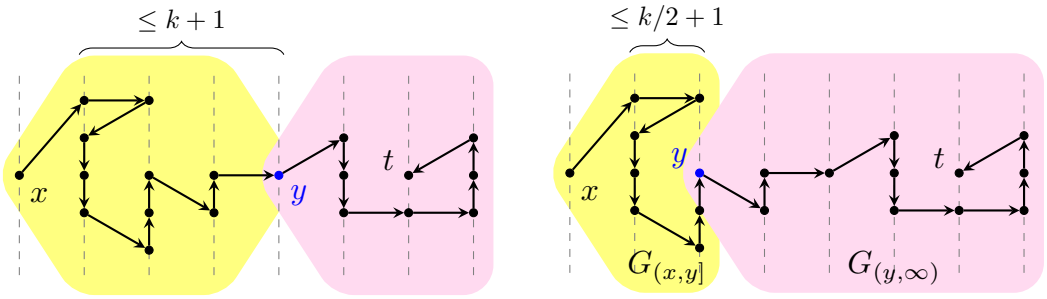
If $d(t) - d(x) \leq k$, we can find all such offsets just by solving $\ell$-Path for $\ell \leq 2k$.

So, suppose we are given a vertex $x$ with $d(t) - d(x) \geq k + 1$ and an offset $r \leq k$, and wish to determine if there is a path of length $d(t) - d(x) + r$ from $x$ to $t$ in $G_{(x,\infty)}$. If there is such a path $P$, then [2] argues that $P$ can always be split in as depicted in Figure 1a: for some vertex $y$ with $d(y) > d(x)$, we can decompose $P$ into two subpaths:

1. a subpath $A$ from $x$ to $y$ of length at most $2k + 1$, such that all internal vertices $v$ in $A$ satisfy $d(x) < d(v) < d(y)$, and
2. a subpath $B$ from $y$ to $t$ in $G_{(y,\infty)}$ of length at most $d(t) - d(y) + k$.

We can always split a path $P$ in this manner because $P$ has length at most $d(t) - d(x) + k$, so at most $k$ edges in $P$ are not forward edges. Intuitively, as we follow the vertices along the path $P$, the distance of the current vertex from $s$ can decrease or stay the same at most $k$ times, and so $P$ cannot contain too many vertices which are the same distance from $s$. This allows one to argue that there is a vertex $y$ with $d(y) \leq d(x) + k + 1$ such that all internal vertices $v$ of the subpath $P[x, y]$ have $d(x) < d(v) < d(y)$. Since $d(y) \leq d(x) + k + 1$ and $P$ has length at most $d(t) - d(x) + k$, it turns out that $P[x, y]$ has length at most $2k + 1$.

[2] If $G$ were not connected, we could replace it with the connected component containing $s$, and solve the detour problems on this smaller graph instead.




**(a) Previous Algorithms**: A subpath $P$ from $x$ to $t$ in $G_{(x,\infty)}$ of a solution path can always be split at a vertex $y$ with $d(y) \leq d(x) + k + 1$, such that $d(u) \neq d(y)$ for all vertices $u \neq y$ in $P$.

**(b) Our Algorithm**: In undirected graphs, a subpath $P$ from $x$ to $t$ in $G_{(x,\infty)}$ of a solution path can be split at a vertex $y$ with $d(y) \leq d(x)+k/2+1$, such that $P[x, y]$ is in $G_{(x,y]}$ and $P[y, t]$ is in $G_{(y,\infty)}$.

**Figure 1** To solve $k$-Detour, we split subpaths $P$ of candidate solutions at a vertex $y$ satisfying certain nice properties. We obtain a speed-up by getting better upper bounds on $d(y)$ in Figure 1b than previous work did in Figure 1a, by allowing $P[x, y]$ to have internal vertices $u$ with $d(u) = d(y)$.

Note that since $G_{(y,\infty)}$ only contains vertices $v$ with $d(v) \geq y$, the paths $A$ and $B$ must be disjoint. We can find the length of $A$ using an algorithm for $(2k+1)$-Path, and the length of $B$ will have already been computed in our dynamic program (since $y$ is further from $s$ than $x$). So, by trying out all possible $y$, finding the possible lengths for subpaths $A$ and $B$, and then adding up these lengths, we can get all possible lengths for $P$ in the dynamic program, and solve $k$-Detour.

## 3.2 Previous Path Algorithm

The fastest known algorithm for $k$-Path in undirected graphs goes through the $(k, k_1, \ell_2)$-Bipartitioned Path problem. Recall that in this problem, we are given a bipartition $V_1 \sqcup V_2$ of the vertices in the graph, and want to find a path of length $k$ from $s$ to $t$, which uses $k_1$ vertices in $V_1$ and $\ell_2$ edges with both endpoints in $V_2$. The authors of [3] showed that $(k, k_1, \ell_2)$-Bipartitioned Path can be solved in $2^{k_1+\ell_2}\,\mathrm{poly}(n)$ time over undirected graphs.

Why does this imply a faster algorithm for $k$-Path in undirected graphs? Well, suppose the input graph contains a path $P$ of length $k$ from $s$ to $t$. Consider a uniform random bipartition of the vertices of the graph into parts $V_1$ and $V_2$. We expect $(k+1)/2$ vertices of $P$ to be in $V_1$, and $k/4$ edges of $P$ to have both endpoints in $V_2$. In fact, this holds with constant probability, so we can solve $k$-Path by solving $(k, (k+1)/2, k/4)$-Bipartitioned Path in the randomly partitioned graph. By Lemma 3 this yields a $2^{3k/4}\,\mathrm{poly}(n) \approx 1.682^k\,\mathrm{poly}(n)$ time algorithm for $k$-Path. We can obtain a faster algorithm using the following modification: take several uniform random bipartitions of the graph, and solve $(k, k_1, \ell_2)$-Bipartitioned Path separately for each bipartition, for $k_1 + \ell_2 \leq 3(1-\varepsilon)k/4$, where $\varepsilon > 0$ is some constant. The number of bipartitions used is some function of $k$ and $\varepsilon$, set so that with high probability at least one of the partitions $V_1 \sqcup V_2$ has the property that the total number of vertices of $P$ in $V_1$ and number of edges of $P$ with both endpoints in $V_2$ is at most $3(1-\varepsilon)k/4$. Setting the parameter $\varepsilon$ optimally yields a $1.657^k\,\mathrm{poly}(n)$ time algorithm for $k$-Path [3].

## 3.3 Our Improvement

As in the previous approach outlined in Section 3.1, our algorithm for $k$-Detour performs dynamic programming over vertices in the graph, starting at $t$, and then moving to vertices closer to $s$. For each vertex $x$ with $d(x) \leq d(t)$, we compute all offsets $r \leq k$ such that there is a path of length $d(t) - d(x) + r$ from $x$ to $t$ in the subgraph $G_{(x,\infty)}$. Obtaining this information for $x = s$ and $r = k$ solves the $k$-Detour problem.

Given a vertex $x$ and offset $r \leq k$, we wish to determine if $G$ contains a path of length $d(t) - d(x) + r$ from $x$ to $t$ in $G_{(x,\infty)}$. Suppose there is such a path $P$. If $d(t) - d(x)$ is small enough, it turns out we can find $P$ by solving $p$-Path for small values of $p$. So, for the purpose of this overview, suppose that $d(t) - d(x)$ is sufficiently large. In this case, as outlined in Section 3.1, previous work showed that $P$ can be split into two subpaths $A$ and $B$ contained in disjoint subgraphs, such that $A$ has length at most $2k + 1$. This splitting argument holds even for directed graphs. Our first improvement comes from the observation that in undirected graphs, we can decompose the path $P$ with a smaller prefix: as depicted in Figure 1b, there must exist a vertex $y$ with $d(y) > d(x)$, such that $P$ splits into a subpath $A$ from $x$ to $y$ in $G_{(x,y]}$ of length at most $3k/2+1$, and a path $B$ from $y$ to $t$ in $G_{(y,\infty)}$ of length at most $d(t) - d(y) + k$. We can find the length of $A$ by solving $(3k/2 + 1)$-Path, and the length of $B$ will already have been computed by dynamic programming, since $d(y) > d(x)$.

This split is possible because any consecutive vertices $u$ and $v$ in $P$ have $|d(u) - d(v)| \leq 1$ (this is true for undirected graphs, but is not true in general for directed graphs). Since $P$ has length at most $d(t) - d(x) + k$, it turns out that $P$ has at most $k/2$ backwards edges. This lets us argue that there exists a vertex $y$ with $d(y) \leq d(x) + k/2 + 1$ such that $P[x, y]$ is contained in $G_{(x,y]}$ and $P[y, t]$ is contained in $G_{(y,\infty)}$. Finally, $A = P[x, y]$ should have length at most $k$ more than $d(y) - d(x)$, which means it has length at most $3k/2 + 1$.

This simple modification already yields a faster algorithm[3] for $k$-Detour. We get further improvements by performing casework on the number of stable edges in $P$ (recall that an edge $(u, v)$ is stable if both its endpoints have the same distance $d(u) = d(v)$ from $s$).

First, suppose $P$ has at least $m$ stable edges, for some parameter $m$. Since $P$ has length at most $d(t) - d(x) + k$, we can argue that $P$ has at most $(k - m)/2$ backwards edges. With this better upper bound on the number of backwards edges, we can improve the splitting argument and show that $P$ decomposes into subpaths $A$ and $B$, such that the length of $A$ is at most $(3k - m)/2$, and the length of $B$ was already computed by our dynamic program. It then suffices to solve $(3k - m)/2$-Path, which yields a speed-up whenever $m \geq \Omega(k)$.

Otherwise, $P$ has at most $m$ stable edges. In this case, we consider the bipartition $V_1 \sqcup V_2$ of the vertex set, where $V_1$ has all vertices at an odd distance from $s$, and $V_2$ has all vertices with even distance from $s$. Since $G$ is undirected, consecutive vertices on the path $P$ differ in their distance from $s$ by at most one. In particular, all forward and backward edges in $P$ cross between the parts $V_1$ and $V_2$. Only the stable edges can contribute to edges with both endpoints in $V_2$. Since we assumed that the number of stable edges is small, it turns out we can find the length of the subpath $A$ of $P$ by solving $(\ell, k_1, \ell_2)$-Bipartitioned Path with respect to the given bipartition, for some $\ell_2$ which is very small. In particular, this approach computes the length of $A$ faster than naively solving $(3k - m)/2$-Path. By setting an appropriate threshold for $m$, we can minimize the runtimes of the algorithm in both of the above cases, and establish Theorem 1.

So in summary, our faster algorithms come from two main sources of improvement: using the structure of shortest paths in undirected graphs to get a better "path-splitting" argument in the dynamic program from $k$-Detour, and cleverly applying the fast algorithm from Lemma 3 for $(\ell, k_1, \ell_2)$-Bipartitioned Path with carefully chosen bipartitions.

We note that our application of $(\ell, k_1, \ell_2)$-Bipartitioned Path is qualitatively different from its uses in previous work. As discussed in Section 3.1, previous algorithms for $k$-Detour work by solving instances of $k$-Path, and as described in Section 3.2, the fastest algorithms for $k$-Path on undirected graphs work by reduction to various instances of $(\ell, k_1, \ell_2)$-Bipartitioned Path. Thus, previous algorithms for $k$-Detour on undirected graphs *implicitly* rely on the

[3] In fact, this observation already yields the fast deterministic algorithms for Theorems 4 and 5.

fast algorithm for $(\ell, k_1, \ell_2)$-Bipartitioned Path, applied to random bipartitions of the input graph. We obtain a faster algorithm for $k$-Detour arguing that in certain cases, we can "beat randomness," by constructing bipartitions which leverage structural information about the graph (namely, whether the shortest path distance from $s$ to a given vertex is even or odd).

## 4 Detour Algorithm

In this section, we present Algorithm 1, our new algorithm for the $k$-Detour problem. As mentioned in the previous section, our algorithm behaves differently depending on the number of stable edges that a potential solution path contains. In particular, the algorithm depends on a parameter $\alpha \in (0, 1)$, which determines the threshold for what counts as "many" stable edges. Later, we will set $\alpha$ to optimize the runtime of Algorithm 1. Certain lines of Algorithm 1 have comments indicating case numbers, which are explained in Section 4.1.

Our algorithm computes a set $L(x)$ for each vertex $x$ in the graph, corresponding to the possible lengths of potential subpaths from $x$ to $t$ of a solution path from $s$ to $t$.

In step 3 of Algorithm 1, we compute $L(x)$ for all $x$ that are "far" from $s$, by solving instances of $\ell$-Path for $\ell \leq (3-\alpha)k/2$. Starting in step 4, we compute $L(x)$ for vertices $x$ closer to $s$, in terms of the previously computed sets $L(y)$ for vertices $y$ further from $s$. In steps 5 through 7, we compute some lengths in $L(x)$ by solving instances of $(a, k_1, \ell_2)$-Bipartitioned Path for appropriate $a, k_1, \ell_2$ values, and in 8 and 9 we compute the remaining lengths in $L(x)$ by solving $a$-Path for $a \leq (3 - \alpha)k/2 + 1$.

### 4.1 Correctness

In this section, we show that Algorithm 1 correctly solves the $k$-Detour problem for any choice of $\alpha \in (0, 1)$. The main technical part of the proof lies in inductively showing that every possible solution path from $s$ to $t$ will be considered by the algorithm and its length will be included in the set $L(s)$. In Algorithm 1, we try out values of the variable $m$ from 0 to $k$, and execute differently depending on how $m$ compares to $\alpha k$. This is interpreted as follows: suppose there is a solution path $P$ from $x$ to $t$, then $m$ corresponds to a guess of the number of stable edges in $P$.

In **Case 1**, we guess that $P$ has few stable edges $m < \alpha k$ which corresponds to steps 5 to 7. Under **Case 1**, there are two possible structures a potential solution path might take on depending on how $d(x)$ compares to $d(t)$. We refer to the case where $d(x) - d(t)$ is small as **Case 1(a)** considered by step 6, and the case where $d(x) - d(t)$ is large as **Case 1(b)** considered by step 7. In **Case 2**, we guess that $m \geq \alpha k$, so $P$ has many stable edges, which corresponds to steps 8 to 9. These cases are also formally defined in our proof of correctness.

▶ **Theorem 6.** *For any fixed $\alpha \in (0, 1)$, Algorithm 1 correctly solves the $k$-Detour problem.*

**Proof.** We prove that upon halting, each set $L(x)$ computed by Algorithm 1 has the property that for all integers $\ell \in [d(t) - d(x), d(t) - d(x) + k]$, we have

$$\ell \in L(x) \text{ if and only if there is a path of length } \ell \text{ from } x \text{ to } t \text{ in } G_{(x,\infty)}. \tag{1}$$

If this property holds, then step 10 of Algorithm 1 returns the correct answer to the $k$-Detour problem, since $\mathsf{dist}(s, t) + k$ is in $L(s)$ if and only if there is a path from $s$ to $t$ of length $\mathsf{dist}(s, t) + k$ in $G_{(s,\infty)} = G$.

So, it suffices to show that Equation (1) holds for all vertices $x$. We prove this result by induction on the distance of $x$ from $s$ in the graph.

**Algorithm 1** Our algorithm for solving $k$-Detour in undirected graphs.

**Input:** An undirected graph $G$ with distinguished vertices $s$ and $t$, and a parameter $\alpha \in (0,1)$.

1: Initialize $V_1 \leftarrow \{x \in V \mid d(x) \equiv 1 \mod 2\}$, $V_2 \leftarrow \{x \in V \mid d(x) \equiv 0 \mod 2\}$.
2: For each vertex $x$ in the graph with $d(x) \leq d(t)$, initialize $L(x) \leftarrow \emptyset$.
▷ *$L(x)$ will be the set of lengths $\ell \in [d(t) - d(x), d(t) - d(x) + k]$ such that there is a path of length $\ell$ from $x$ to $t$ in $G_{(x,\infty)}$.*

3: For each vertex $x$ with $d(x) \in [d(t) - (1-\alpha)k/2, d(t)]$, set $L(x)$ to be the set of all positive integers $\ell \leq (3-\alpha)k/2$ such that there is a path of length $\ell$ from $x$ to $t$ in $G_{(x,\infty)}$.
▷ *Base case: we compute $L(x)$ for the vertices $x$ which are furthest from $s$.*

4: For each $d$ from $d(t) - (1-\alpha)k/2 - 1$ down to 0, for each vertex $x$ with $d(x) = d$, complete steps 5 through 9.
▷ *Inductive Case: compute $L(x)$ layer by layer towards $s$.*

5: For each integer $m$ with $0 \leq m < \alpha k$, and for each choice of integers $k_1, \ell_2 \geq 0$ satisfying $k_1 + \ell_2 \leq (3k + m + 2)/4$, complete steps 6 and 7.
▷ *This step handles **Case 1:** the solution path has few stable edges.*

6: If there is a path of length $\ell \leq 2k_1 + \ell_2$ from $x$ to $t$ in $G_{(x,\infty)}$, update $L(x) \leftarrow L(x) \cup \{\ell\}$.
▷ *This step handles **Case 1(a):** $d(t) - d(x) \leq (k-m)/2$.*

7: Try out all vertices $y$ with $d(y) \in [d(x)+1, \min(d(t), d(x) + (3k-m)/2 + 1)]$. If for some such $y$, there is a path from $x$ to $y$ in $G_{(x,y]}$ of length $a \leq 2k_1 + \ell_2$ with exactly $k_1$ vertices in $V_1$, and $\ell_2$ edges with both endpoints in $V_2$, update $L(x) \leftarrow L(x) \cup (a + L(y))$.
▷ *This step handles **Case 1(b):** $d(t) - d(x) > (k-m)/2$.*

8: For each integer $m$ with $\alpha k < m \leq k$, complete step 9.
▷ *This step handles **Case 2:** the solution path has many stable edges.*

9: Try out all vertices $y$ with $d(y) \in [d(x)+1, d(x) + (1-\alpha)k/2 + 1]$. If for some such $y$, there is a path from $x$ to $y$ in $G_{(x,y]}$ of length $a \leq (3-\alpha)k/2 + 1$, update $L(x) \leftarrow L(x) \cup (a + L(y))$

10: Return **yes** if and only if $(\mathsf{dist}(s,t) + k) \in L(s)$.

**Base Case.** For the base case, suppose $x$ is a vertex with

$$d(x) \in [d(t) - (1-\alpha)k/2, d(t)]. \tag{2}$$

Then $L(x)$ is computed in step 3 of Algorithm 1. We now verify that Equation (1) holds.
First, suppose $\ell \in L(x)$.
Then, $\ell$ must be the length of some path from $x$ to $t$ in $G_{(x,\infty)}$ by design.
Conversely, suppose we have a path $P$ from $x$ to $t$ in $G_{(x,\infty)}$ of some length

$$\ell \leq d(t) - d(x) + k.$$

Then by the assumption on $x$ from Equation (2) in this case, we have

$$\ell \leq d(t) - d(x) + k \leq (1-\alpha)k/2 + k = (3-\alpha)k/2$$

so step 3 of Algorithm 1 correctly includes $\ell$ in $L(x)$.

Thus Equation (1) holds for all vertices $x$ satisfying Equation (2).

**Inductive Case.** For the inductive step, suppose $x$ is a vertex with

$$d(x) \leq d(t) - (1-\alpha)k/2 - 1. \tag{3}$$

We may inductively assume that we have computed sets $L(y)$ satisfying Equation (1), for all vertices $y$ with $d(y) > d(x)$.

Suppose $\ell \in L(x)$ at the end of Algorithm 1. Then either $\ell$ was added to $L(x)$ in step 6, or $\ell$ was added to $L(x)$ in steps 7 or 9 of Algorithm 1. In the former case, $\ell$ is the length of a path from $x$ to $t$ in $G_{(x,\infty)}$ by design. In the latter cases, we have $\ell = a + b$, where $a$ is the length of some path from $x$ to $y$ (for some vertex $y$ with $d(y) > d(x)$) in $G_{(x,y]}$, and (by the inductive hypothesis) $b$ is the length of some path from $y$ to $t$ in $G_{(y,\infty)}$. Since $G_{(x,y]}$ and $G_{(y,\infty)}$ intersect only at $y$, the union of these paths is a path from $x$ to $t$ in $G_{(t,\infty)}$. So, every integer in $L(x)$ is a valid length of a path from $x$ to $t$ in $G_{(x,\infty)}$ as desired.

Conversely, suppose there is a path $P$ from $x$ to $t$ in $G_{(x,\infty)}$ of length

$$\ell \in [d(t) - d(x), d(t) - d(x) + k]. \tag{4}$$

We prove that $\ell$ appears in $L(x)$.

To do this, we will analyze the number of forward, backward, and stable edges appearing in $P$. Note that $P$ has at least $d(t) - d(x)$ forward edges, since $P$ begins at a vertex at distance $d(x)$ from $s$, ends at a vertex at distance $d(t)$ from $s$, and only the forward edges allow us to move to vertices further from $s$.

Let $m$ denote the number of stable edges in $P$. We have $m \leq k$, since the length of $P$ is at most $d(t) - d(x) + k$, and $P$ has at least $d(t) - d(x)$ forward edges.

▷ Claim 7. Suppose $d(x) \leq d(t) - (k-m)/2 - 1$. Then $P$ contains a vertex $y$ such that

1. $d(y) \in [d(x) + 1, d(x) + (k-m)/2 + 1]$,
2. every vertex $u \in P[y,t]$ with $u \neq y$ has $d(u) > d(y)$, and
3. every vertex $v \in P[x,y]$ has $d(v) \leq d(y)$.

Proof. For each $i \in [(k-m)/2 + 1]$, let $z_i$ denote the last vertex on $P$ satisfying

$$d(z_i) = d(x) + i.$$

These vertices exist because we are assuming that $d(x) \leq d(t) - (k-m)/2 - 1$, and $P$ must contain vertices $v$ with $d(v) = d$ for every $d \in [d(x), d(t)]$.

By definition, each $z_i$ satisfies conditions **1** and **2** from the claim. If some $z_i$ satisfies condition **3** as well, then the claim is true.

So, suppose that none of the $z_i$ satisfy condition **3**. This means that for each index $i$, the subpath $P[x, z_i]$ contains a vertex $u$ with $d(u) > d(z_i)$. Consecutive vertices in $P$ differ in their distance from $s$ by at most one, so $P[x, z_i]$ must contain an edge $e = (v, w)$ such that $d(v) = d(w) + 1$ and $d(w) = d(z_i) = d(x) + i$. That is, $P$ contains a backwards edge from a vertex at distance $i + 1$ from $s$ to a vertex at distance $i$ from $s$, as depicted in Figure 2.

Note that $z_1, z_2, \ldots, z_{(k-m)/2+1}$ occur on $P$ in the listed order. This is because

$$d(z_1) < d(z_2) < \cdots < d(z_{(k-m)/2+1})$$

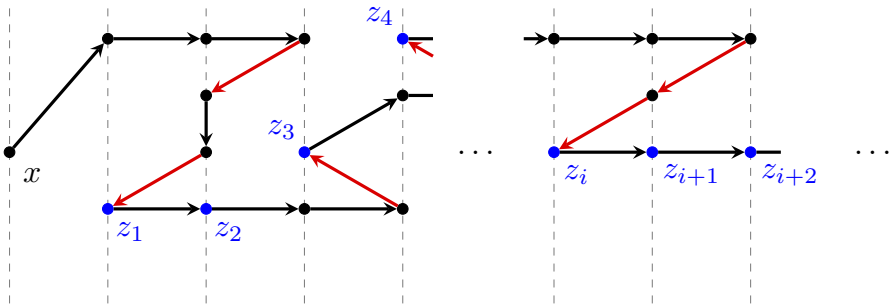



**Figure 2** If node $z_i$ does not satisfy condition **3** of Claim 7, it means that before hitting $z_i$, the path visited a node further from $s$ than $z_i$. Thus, we can associate $z_i$ with some backwards edge. The presence of too many of these backwards edges would violate the length condition on $P$, so it turns out that one such node (in the figure, $z_{i+2}$) does have to satisfy condition **3**.

and each $z_i$ satisfies condition **1** from the claim. Combined with the discussion from the previous paragraph, this means that $P$ contains at least $(k-m)/2+1$ backwards edges. We now argue that this violates the assumption on the length of $P$.

Let $f$ and $b$ denote the number of forward and backwards edges in $P$ respectively.

Since $P$ starts at $x$ and ends at $t$, we have $f - b = d(t) - d(x)$, which implies that

$$f = d(t) - d(x) + b. \tag{5}$$

Then the total length of $P$ is $f + b + m = d(t) - d(x) + m + 2b$ by Equation (5). However, since $P$ has at least $(k-m)/2+1$ backwards edges, this length satisfies

$$d(t) - d(x) + m + 2b \geq d(t) - d(x) + m + 2\left((k-m)/2 + 1\right) > d(t) - d(x) + k$$

which contradicts the fact that the length $\ell$ of $P$ satisfies Equation (4). Thus our assumption was incorrect, and one of the $z_i$ satisfies all three conditions from the claim, as desired. ◁

We now perform casework on the number of stable edges $m$ in $P$. We start with **Case 2** from step 8 of Algorithm 1, since this is the easiest case to analyze.

**Case 2: Many Stable Edges ($m \geq \alpha k$).** Suppose $m \geq \alpha k$. In this case, by Equation (3) we have

$$d(x) \leq d(t) - (1-\alpha)k/2 - 1 \leq d(t) - (k-m)/2 - 1$$

So, by Claim 7, there exists a vertex $y$ in $P$ satisfying the three conditions of Claim 7.

By condition **3** from Claim 7, the subpath $A = P[x, y]$ is contained in $G_{(x,y]}$. By condition **2** from Claim 7, the subpath $B = P[y, t]$ is contained in $G_{(y,\infty)}$.

Let $a$ denote the length of $A$, and $b$ denote the length of $B$.

Since $A$ has length at least $d(y) - d(x)$, and $P$ has length at most $d(t) - d(x) + k$ by Equation (4), we know that the length $B$ satisfies

$$b \leq d(t) - d(y) + k. \tag{6}$$

By the inductive hypothesis, $L(y)$ satisfies Equation (1), so $b \in L(y)$.

Similar to the reasoning that established Equation (6), we can prove that

$$a \leq d(y) - d(x) + k. \tag{7}$$

By condition **1** of Claim 7, we know that $d(y) \leq d(x) + (k-m)/2 + 1$. Since $m \geq \alpha k$, this implies that $d(y) \leq d(x) + (1-\alpha)k/2 + 1$.

Substituting this into Equation (7) yields

$$a \leq (1-\alpha)k/2 + 1 + k = (3-\alpha)k/2 + 1.$$

Thus, the length $a$ of $A$ will be found in step 9 of Algorithm 1. As mentioned before, $b \in L(y)$. Thus, $\ell = a + b \in (a + L(y))$ is correctly added to the set $L(x)$ in step 9 of Algorithm 1, which proves the desired result in this case.

**Case 1: Few Stable Edges ($m < \alpha k$).** If we do not fall into **Case 2**, we must have $m < \alpha k$. Recall that in step 1 of Algorithm 1, we defined $V_1 = \{u \mid d(u) \text{ is odd}\}$ and $V_2 = \{u \mid d(u) \text{ is even}\}$.

We want to argue that most edges in path $P$ cross the bipartition $V_1 \sqcup V_2$. To that end, the following claim will be helpful.

▷ Claim 8. Let $Q$ be a path of length $q$, with at most $m$ stable edges. Let $k_1$ denote the number of vertices of $Q$ in $V_1$, and let $\ell_2$ denote the number of edges in $Q$ with both endpoints in $V_2$. Then we have

$$k_1 + \ell_2 \leq (q + m + 1)/2.$$

Proof. Let $k_2$ denote the number of vertices of $Q$ in $V_2$.

Consider the cycle $C$ formed by taking $Q$ together with an additional edge between its endpoints (this new edge is imagined for the purpose of argument, and does not change the definition of $V_1$ and $V_2$).

Let $q_1$, $q_2$, and $q_{\text{cross}}$ denote the number of edges of $C$ with both endpoints in $V_1$, both endpoints in $V_2$, and endpoints in both $V_1$ and $V_2$ respectively. We have

$$2k_1 = 2q_1 + q_{\text{cross}} \tag{8}$$

because both sides of the above equation count the number of pairs $(u, e)$ such that $u$ is a vertex in $C \cap V_1$, and $e$ is an edge in $C$ incident to $u$. A symmetric argument implies that

$$2q_2 + q_{\text{cross}} = 2k_2. \tag{9}$$

Adding Equation (8) and Equation (9) together and simplifying yields

$$k_1 + q_2 = k_2 + q_1.$$

This implies that

$$k_1 + q_2 = (k_1 + k_2 + q_1 + q_2)/2.$$

Since $C$ is $Q$ with one additional edge, we have $\ell_2 \leq q_2$. So the above equation implies that

$$k_1 + \ell_2 \leq (k_1 + k_2 + q_1 + q_2)/2. \tag{10}$$

We have

$$k_1 + k_2 = q + 1 \tag{11}$$

since the total number of vertices in $Q$ must be one more than its length. By assumption on the number of stable edges in $Q$, we have

$$q_1 + q_2 \leq m. \tag{12}$$

Substituting Equation (11) and Equation (12) into the right hand side of Equation (10) yields

$$k_1 + \ell_2 \leq (q + m + 1)/2$$

which proves the desired result. ◁

With Claim 8 established, we are now ready to analyze the two subcases under **Case 1**, based on the relative distances of $x$ and $t$ from $s$.

**Case 1(a): $d(t) - d(x)$ is small.** Suppose that $d(x) \in [d(t) - (k - m)/2, d(t)]$.

In this case, Equation (4) implies that $P$ has length

$$\ell \leq d(t) - d(x) + k \leq (3k - m)/2.$$

Let $k_1$ denote the number of vertices of $P$ in $V_1$, and $k_2$ denote the number of edges in $P$ with both endpoints in $V_2$. Then by setting $Q = P$ and $q = \ell$ in Claim 8, we have

$$k_1 + \ell_2 \leq (\ell + m + 1)/2 \leq (3k + m + 2)/4. \tag{13}$$

Also, note that $P$ has length $\ell \leq 2k_1 + \ell_2$, since $2k_1$ is greater than or equal to the number of edges in $P$ incident to a vertex in $V_1$. This observation, together with Equation (13), shows that in this case, the length $\ell$ is correctly included in $L(x)$ in step 6 of Algorithm 1.

**Case 1(b): $d(t) - d(x)$ is large.** If we do not fall into **Case 1(a)**, it means that

$$d(x) \leq d(t) - (k - m)/2 - 1. \tag{14}$$

Thus, by Claim 7, there exists a vertex $y$ in $P$ satisfying the three conditions of Claim 7. The proof that $\ell \in L(x)$ in this case is essentially a combination of the proofs from **Case 2** and **Case 1(a)**.

As in **Case 2**, by condition **3** from Claim 7, the subpath $A = P[x, y]$ is contained in $G_{(x,y]}$. By condition **2** from Claim 7, the subpath $B = P[y, t]$ is contained in $G_{(y,\infty)}$.

Let $a$ and $b$ denote the lengths of paths $A$ and $B$ respectively. Reasoning identical to the arguments which established Equations (6) and (7) prove that in this case we also have

$$b \leq d(t) - d(y) + k \tag{15}$$

and

$$a \leq d(y) - d(x) + k. \tag{16}$$

Condition **1** of Claim 7 implies that $d(y) \leq d(x) + (k - m)/2 + 1$. Substituting this into Equation (16) implies that

$$a \leq (3k - m)/2 + 1.$$

Let $k_1$ denote the number of vertices of $A$ in $V_1$, and let $\ell_2$ denote the number of edges in $A$ with both endpoints in $V_2$. Then by setting $Q = A$ and $q = a$ in Claim 8, we have

$$k_1 + \ell_2 \leq (a + m + 1)/2 \leq (3k + m + 2)/4. \tag{17}$$

Also, we know that $a \leq 2k_1 + \ell_2$, because $2k_1$ is greater than or equal to the number of edges in $A$ incident to a vertex in $V_1$. Combining this observation with Equation (17), we see that the length $a$ is indeed computed in step 7 of Algorithm 1.

By the inductive hypothesis (Equation (1)) and Equation (15), we know that $b \in L(y)$. Thus we have $\ell = a + b \in (a + L(y))$, so in this case, $\ell$ is correctly included in $L(x)$ in step 7 of Algorithm 1.

This completes the induction, and proves that Equation (1) holds for all vertices $x$ in the graph. In particular, Equation (1) holds for $x$ equal to $s$. This implies that step 10 of Algorithm 1 returns the correct answer to the $k$-Detour algorithm. ◀

## 5 Applications

In this section, we present consequences of our new algorithm for $k$-Detour from Section 4.

▶ **Theorem 1.** *In undirected graphs, $k$-Detour can be solved in $1.853^k \operatorname{poly}(n)$ time.*

**Proof.** By Theorem 6, Algorithm 1 correctly solves $k$-Detour, for any value $\alpha \in (0, 1)$,

What is the runtime of Algorithm 1? Well, steps 3 and 9 of Algorithm 1 involve solving polynomially many instances of $\ell$-Path, for $\ell \leq (3 - \alpha)k/2 + 1$. Using the fastest known algorithm for $k$-Path in undirected graphs [3], these steps take

$$1.657^{(3-\alpha)k/2} \operatorname{poly}(n)$$

time. The remaining computationally intensive steps of Algorithm 1 occur in steps 6 and 7, which can be implemented by solving $\operatorname{poly}(n)$ instances of $(\ell, k_1, \ell_2)$-Bipartitioned Path, for $k_1 + \ell_2 < (3k + \alpha k + 2)/4$. By Lemma 3, these steps then take

$$2^{(3+\alpha)k/4} \operatorname{poly}(n)$$

time overall. Then by setting $\alpha = 0.55814$ to balance the above runtimes, we see that we can solve $k$-Detour over undirected graphs in

$$\left(1.657^{(3-\alpha)k/2} + 2^{(3+\alpha)k/4}\right) \operatorname{poly}(n) \leq 1.8526^k \operatorname{poly}(n)$$

time, as desired. ◀

▶ **Theorem 2.** *In undirected graphs, $k$-Longest Detour can be solved in $3.432^k \operatorname{poly}(n)$ time.*

**Proof.** The proof of [8, Corollary 2] shows that $k$-Longest Detour in undirected graphs reduces, in polynomial time, to solving $p$-Detour for all $p \leq 2k$ and $\operatorname{poly}(n)$ instances of $(3k/2)$-Longest Path on graphs with at most $n$ nodes.

The proof of Theorem 1 implies that $k$-Detour can be solved over undirected graphs in $1.8526^k \operatorname{poly}(n)$ time. Previous work in [7, Section 6.3] shows that $k$-Longest Path can be solved over undirected graphs in $1.657^k \operatorname{poly}(n)$ time. Combining these results together with the above discussion shows that $k$-Longest Detour can be solved over undirected graphs in

$$\left(1.8526^{2k} + 1.657^{3k/2}\right) \operatorname{poly}(n) \leq 3.432^k \operatorname{poly}(n)$$

time, as desired. ◀

▶ **Theorem 4.** *The $k$-Detour problem can be solved over undirected graphs by a deterministic algorithm in $4.082^k \operatorname{poly}(n)$ time.*

**Proof.** By Theorem 6, we can solve $k$-Detour over an undirected graph by running Algorithm 1 with parameter $\alpha = 0$. When $\alpha = 0$ in Algorithm 1, steps 5, 6, 7 never occur. In this case, the algorithm only needs to solve $\text{poly}(n)$ instances of $\ell$-Path, for $\ell \leq 3k/2 + 1$, in steps 3 and 9. Since $k$-Path can be solved deterministically in $2.554^k \text{poly}(n)$ time [15], this means that we can solve $k$-Detour deterministically in

$$2.554^{3k/2} \text{poly}(n) \leq 4.0817^k \text{poly}(n)$$

time, as desired. ◀

▶ **Theorem 5.** *The $k$-Longest Detour problem can be solved over undirected graphs by a deterministic algorithm in $16.661^k \text{poly}(n)$ time.*

**Proof.** The proof of [8, Corollary 2] shows that $k$-Longest Detour in undirected graphs reduces, in deterministic polynomial time, to solving $p$-Detour for $p \leq 2k$, and $\text{poly}(n)$ instances of $(3k/2)$-Longest Path on graphs with at most $n$ nodes.

The proof of Theorem 4 implies that $k$-Detour can be solved over undirected graphs deterministically in $4.0817^k \text{poly}(n)$ time. Previous work [9] shows that $k$-Longest Path can be solved deterministically in $4.884^k \text{poly}(n)$ time. Combining these results together with the above discussion shows that $k$-Longest Detour can be solved over undirected graphs deterministically in

$$\left(4.0817^{2k} + 4.884^{3k/2}\right) \text{poly}(n) \leq 16.661^k \text{poly}(n)$$

time, as desired. ◀

## 6 Conclusion

In this paper, we obtained faster algorithms for $k$-Detour and $k$-Longest Detour over undirected graphs. However, many mysteries remain surrounding the true time complexity of these problems. We highlight some open problems of interest, relevant to our work.

1. The most pertinent question: what is the true parameterized time complexity of $k$-Detour and $k$-Longest Detour? In particular, could it be the case that $k$-Detour can be solved as quickly as $k$-Path, and $k$-Longest Detour can be solved as quickly as $k$-Longest Path? No known conditional lower bounds rule out these possibilities.
2. The current fastest algorithm for $k$-Longest Path in directed graphs has a bottleneck of solving $2k$-Path. The current fastest algorithm for $k$-Detour in directed graphs has a bottleneck of solving $2k$-Path. Similarly, the fastest known algorithm[4] for $k$-Longest Detour in undirected graphs requires first solving $2k$-Detour. Is this parameter blow-up necessary? Could it be possible to solve these harder problems with parameter $k$ faster than solving these easier problems with parameter $2k$?
3. The speed-up in our results crucially uses a fast algorithm for the $(\ell, k_1, \ell_2)$-Bipartitioned Path problem in undirected graphs. In directed graphs no $(2-\varepsilon)^\ell \text{poly}(n)$ time algorithm appears to be known for this problem, for any constant $\varepsilon > 0$ and interesting range of parameters $k_1$ and $\ell_2$. Such improvements could yield faster algorithms for $k$-Detour in directed graphs. Can we get such an improvement? Also of interest: can we get a faster deterministic algorithm for $(\ell, k_1, \ell_2)$-Bipartitioned Path?

[4] In fact, even the recent alternate algorithm of [10] for $k$-Longest Detour requires solving $2k$-Detour first.

4. An easier version of the previous question, also raised in [7, Section 9.1]: can we solve $k$-Path in directed bipartite graphs in $(2-\varepsilon)^k \operatorname{poly}(n)$ time, for some constant $\varepsilon > 0$? In the unparameterized setting, the Hamiltonian Path ($k$-Path for $k = n$) problem admits several distinct algorithms running in $(2-\varepsilon)^n \operatorname{poly}(n)$ time in directed bipartite graphs. Specifically, [6] shows Hamiltonian Path in directed bipartite graphs can be solved in $1.888^n \operatorname{poly}(n)$ time, and [4] uses very different methods to solve this problem even faster in $3^{n/2} \operatorname{poly}(n)$ time.[5] We conjecture that the same speed-up is possible for $k$-Path, so that this problem can be solved over directed bipartite graphs in $3^{k/2} \operatorname{poly}(n)$ time.

## References


1 Shyan Akmal, Virginia Vassilevska Williams, Ryan Williams, and Zixuan Xu. Faster detours in undirected graphs, 2023. arXiv:2307.01781.

2 Ivona Bezáková, Radu Curticapean, Holger Dell, and Fedor V. Fomin. Finding Detours is Fixed-Parameter Tractable. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.54.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, August 2017. doi:10.1016/j.jcss.2017.03.003.

4 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed Hamiltonicity and Out-Branchings via Generalized Laplacians. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 91:1–91:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.91.

5 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015. doi:10.1007/978-3-319-21275-3.

6 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *Journal of the ACM*, 65(3):1–46, March 2018. doi:10.1145/3148227.

7 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving, 2023. doi:10.48550/arXiv.2304.02091.

8 Fedor V. Fomin, Petr A. Golovach, William Lochet, Danil Sagunov, Kirill Simonov, and Saket Saurabh. Detours in Directed Graphs. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2022.29.

9 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Long directed (s, t)-path: FPT algorithm. *Information Processing Letters*, 140:8–12, December 2018. doi:10.1016/j.ipl.2018.04.018.

10 Ashwin Jacob, Michał Włodarczyk, and Meirav Zehavi. Long directed detours: Reduction to 2-disjoint paths, 2023. doi:10.48550/arXiv.2301.06105.


[5] It is also known that sufficient improvements to algorithms for multiplying two $n \times n$ matrices together would imply that even the weighted version of Hamiltonian Path in directed bipartite graphs can be solved in $(2-\varepsilon)^n \operatorname{poly}(n)$ time, for some constant $\varepsilon > 0$ [14].

**11** Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming, 35th International Colloquium, ICALP*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.

**12** Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. *ACM Transactions on Algorithms*, 12(3):1–18, May 2016. doi:10.1145/2885499.

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

**14** Jesper Nederlof. Bipartite TSP in $o(1.9999^n)$ time, assuming quadratic time matrix multiplication. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. ACM, June 2020. doi:10.1145/3357713.3384264.

**15** Dekel Tsur. Faster deterministic parameterized algorithm for k-path. *Theoretical Computer Science*, 790:96–104, October 2019. doi:10.1016/j.tcs.2019.04.024.

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

**17** Meirav Zehavi. A randomized algorithm for long directed cycle. *Information Processing Letters*, 116(6):419–422, June 2016. doi:10.1016/j.ipl.2016.02.005.