Detecting Disjoint Shortest Paths in Linear Time and More

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

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

Nicole Wein
University of Michigan, Ann Arbor, MI, USA

Abstract

In the k-Disjoint Shortest Paths (k-DSP) problem, we are given a graph G (with positive edge weights) on n nodes and m edges with specified source vertices $s_1, \ldots, s_k$, and target vertices $t_1, \ldots, t_k$, and are tasked with determining if G contains vertex-disjoint (s_i, t_i)-shortest paths. For any constant k, it is known that k-DSP can be solved in polynomial time over undirected graphs and directed acyclic graphs (DAGs). However, the *exact* time complexity of k-DSP remains mysterious, with large gaps between the fastest known algorithms and best conditional lower bounds. In this paper, we obtain faster algorithms for important cases of k-DSP, and present better conditional lower bounds for k-DSP and its variants.

Previous work solved 2-DSP over weighted undirected graphs in $O(n^7)$ time, and weighted DAGs in $O(mn)$ time. For the main result of this paper, we present optimal *linear time* algorithms for solving 2-DSP on weighted undirected graphs and DAGs. Our linear time algorithms are algebraic however, and so only solve the detection rather than search version of 2-DSP (we show how to solve the search version in $O(mn)$ time, which is faster than the previous best runtime in weighted undirected graphs, but only matches the previous best runtime for DAGs).

We also obtain a faster algorithm for k-Edge Disjoint Shortest Paths (k-EDSP) in DAGs, the variant of k-DSP where one seeks edge-disjoint instead of vertex-disjoint paths between sources and their corresponding targets. Algorithms for k-EDSP on DAGs from previous work take $\Omega(m^k)$ time. We show that k-EDSP can be solved over DAGs in $O(mn^{k-1})$ time, matching the fastest known runtime for solving k-DSP over DAGs.

Previous work established conditional lower bounds for solving k-DSP and its variants via reductions from detecting cliques in graphs. Prior work implied that k-Clique can be reduced to $2k$-DSP in DAGs and undirected graphs with $O((kn)^2)$ nodes. We improve this reduction, by showing how to reduce from k-Clique to k-DSP in DAGs and undirected graphs with $O((kn)^2)$ nodes (halving the number of paths needed in the reduced instance). A variant of k-DSP is the k-Disjoint Paths (k-DP) problem, where the solution paths no longer need to be shortest paths. Previous work reduced from k-Clique to p-DP in DAGs with $O(kn)$ nodes, for $p = k + k(k-1)/2$. We improve this by showing a reduction from k-Clique to p-DP, for $p = k + \lfloor k^2/4 \rfloor$.

Under the k-Clique Hypothesis from fine-grained complexity, our results establish better conditional lower bounds for k-DSP for all $k \geq 4$, and better conditional lower bounds for p-DP for all $p \leq 4031$. Notably, our work gives the first nontrivial conditional lower bounds 4-DP in DAGs and 4-DSP in undirected graphs and DAGs. Before our work, nontrivial conditional lower bounds were only known for k-DP and k-DSP on such graphs when $k \geq 6$.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases disjoint shortest paths, algebraic graph algorithms, disjoint paths, fine-grained complexity, clique

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.9

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.15916 [2]

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 9; pp. 9:1–9:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding *Shyan Akmal*: Supported by Virginia Vassilevska Williams' Simons Investigator Award. *Virginia Vassilevska Williams*: Supported by NSF Grant CCF-2330048, BSF Grant 2020356, and a Simons Investigator Award.

Acknowledgements We thank an anonymous reviewer for pointing out issues with a proof in a previous version of this work, and another anonymous reviewer for simplifying our construction of covering families. We thank Malte Renken and André Nichterlein for answering questions about previous work, as well as Matthias Bentert for telling us about his and his coauthors' work [3], which independently proves Theorem 6 from this paper. The first author additionally thanks Ryan Williams and Zixuan Xu for conversations about early versions of ideas presented here, and Yinzhan Xu for nice discussions about the arguments in this paper.

1 Introduction

Routing disjoint paths in graphs is a classical problem in computer science. For positive integer k, in the k-Disjoint Paths (k-DP) problem, we are given a graph G with n vertices and m edges, with specified source nodes $s_1, \ldots, s_k$ and target nodes $t_1, \ldots, t_k$, and are tasked with determining if G contains (s_i, t_i)-paths which are internally vertex-disjoint. Beyond being a natural graph theoretic problem to study, k-DP is important because of its deep connections with various computational tasks from the Graph Minors project [22].

Following a long line of research, the polynomial-time complexity of k-DP has essentially been settled: in directed graphs the 2-DP problem is NP-hard [18, Lemma 3], and so is unlikely to admit a polynomial time algorithm, while in undirected graphs k-DP can be solved in $\tilde{O}(m+n)$ time for $k = 2$ [24], and in $O(n^2)$ time or $m^{1+o(1)}$ time for any constant $k \geq 3$ [19, 20].

In this work we study an optimization variant of k-DP, the k-Disjoint Shortest Paths (k-DSP) problem. In k-DSP we are given the same input as in k-DP, but are now tasked with determining if the input contains (s_i, t_i)-*shortest* paths which are internally vertex-disjoint. This problem is interesting both because it is a natural graph algorithms question to investigate from the perspective of combinatorial optimization, and because understanding the complexity of k-DSP could lead to a deeper understanding of the interaction between shortest paths structures in graphs (analogous to how studying k-DP helped develop the rich theory surrounding forbidden minors in graphs).

Compared to k-DP, not much is known about the exact time complexity of k-DSP. In directed graphs, 2-DSP can be solved in polynomial time [6], but no polynomial-time algorithm (or NP-hardness proof) is known for k-DSP for *any* constant $k \geq 3$. In undirected graphs, it was recently shown that for any constant k, k-DSP can be solved in polynomial time [21]. However, the current best algorithms for k-DSP in undirected graphs run in $n^{O(k \cdot k!)}$ time, so in general this polynomial runtime is quite large for $k \geq 3$. For example, the current fastest algorithm for 3-DSP in undirected graphs takes $O(n^{292})$ time [4].

Significantly faster algorithms are known for detecting $k = 2$ disjoint shortest paths. The paper which first introduced the k-DSP problem in 1998 also presented an $O(n^8)$ time algorithm for solving 2-DSP in weighted[1] undirected graphs [15]. The first improvement for this problem came over twenty years later in [1], which presented an algorithm solving 2-DSP in weighted undirected graphs in $O(n^7)$ time, and in unweighted undirected graphs in $O(n^6)$

[1] Throughout, we assume that weighted graphs have positive edge weights.

time. Soon after, [4, Theorem 1] presented an even faster $O(mn)$ time algorithm for solving 2-DSP in the special case of unweighted undirected graphs.[2] The main result of our work is an optimal algorithm for 2-DSP in weighted undirected graphs.

▶ **Theorem 1.** *2-DSP can be solved in weighted undirected graphs in $O(m+n)$ time.*

This result pins down the true time complexity of 2-DSP in undirected graphs, and (up to certain limitations of our algorithm, which we discuss later) closes the line of research for this specific problem, initiated twenty-five years ago in [15].

As discussed previously, over directed graphs the 2-DP problem is NP-hard, and the complexity of k-DSP is open even for $k = 3$. This lack of algorithmic progress in general directed graphs has motivated researchers to characterize the complexity of disjoint path problems in restricted classes of directed graphs. In this context, studying algorithms for routing disjoint paths in directed acyclic graphs (DAGs) has proved to be particularly fruitful. For example, the only known polynomial time algorithm for 2-DSP on general directed graphs works by reducing to several instances of 2-DP on DAGs [6]. Similarly, the fastest known algorithm for k-DSP on undirected graphs works by reducing to several instances of disjoint paths on DAGs [4].

It is known that 2-DP in DAGs can be solved in linear time [25]. More generally, since 1980 it has been known that k-DP in DAGs can be solved in $O(mn^{k-1})$ time, and this remains the fastest known algorithm for these problems for all $k \geq 3$ [18, Theorem 3].

As observed in [6, Proposition 10], the algorithm of [18] for k-DP on DAGs can be modified to solve k-DSP in weighted DAGs in the same $O(mn^{k-1})$ runtime. This is the fastest known runtime for k-DSP in DAGs. In particular, the fastest algorithm for 2-DSP from previous work runs in $O(mn)$ time.

The second result of our work is an optimal algorithm for 2-DSP in weighted DAGs.

▶ **Theorem 2.** *2-DSP can be solved in weighted DAGs in $O(m+n)$ time.*

This settles the time complexity of 2-DSP in DAGs, and marks the first improvement over the $O(mn)$ time algorithm implied by [18] from over thirty years ago. The 2-DSP problem in weighted DAGs generalizes the 2-DP problem in DAGs, and so Theorem 2 also offers an alternate linear time algorithm for 2-DP in DAGs, which is arguably simpler than the previous approaches leading up to [25], many of which involved tricky case analyses and carefully constructed data structures.

Our algorithms for solving 2-DSP in undirected graphs and DAGs are algebraic, and work by testing whether certain polynomials, whose terms encode pairs of disjoint shortest paths in the input graph, are nonzero. As a consequence, the algorithms establishing Theorems 1 and 2 are randomized, and solve 2-DSP with high probability. Moreover, these algorithms determine whether the input graph has a solution, but do not explicitly return solution paths. So while our algorithms solve the decision problem 2-DSP, they do not solve the search problem of returning two disjoint shortest paths if they exist. This is a common limitation for algebraic graph algorithms.

The 2-DSP problem does admit a search to decision reduction – with $O(m)$ calls to an algorithm which detects whether a graph contains two disjoint shortest paths, we can actually find two disjoint shortest paths if they exist. Thanks to the algebraic nature of our algorithms, we can get a slightly better reduction, and find two disjoint shortest paths when they exist with only $O(n)$ calls to the algorithms from Theorems 1 and 2.

[2] It seems plausible that the method of [4] could be adapted to handle weighted undirected graphs as well, but such a generalization does not appear to currently be known.

▶ **Theorem 3.** *We can solve 2-DSP over weighted DAGs and undirected graphs, and find a solution if it exists, in $O(mn)$ time.*

So we can *find* two disjoint shortest paths in weighted undirected graphs in $O(mn)$ time (which still beats the previous fastest $O(n^7)$ time algorithm for weighted undirected graphs, and matches the previous fastest algorithm for *unweighted* undirected graphs), and in weighted DAGs in $O(mn)$ time (which only matches, rather than beats, the previous fastest runtime for 2-DSP in DAGs).

Finally, one can also consider *edge-disjoint* versions of all the problems discussed thus far. The k-Edge Disjoint Paths (k-EDP) and k-Edge Disjoint Shortest Paths (k-EDSP) problems are the same as the respective k-DP and k-DSP problems, except the solutions paths merely need to be edge-disjoint instead of internally vertex-disjoint.

For any constant k, there is a simple reduction from k-EDSP on n nodes and m edges to k-DSP on $O(m+n)$ nodes and $O(m)$ edges. Combining this reduction with Theorems 1 and 2, we get that we can solve 2-EDSP over weighted DAGs and undirected graphs in linear time as well.

▶ **Corollary 4.** *There is an algorithm solving 2-EDSP over weighted DAGs and undirected graphs in $O(m+n)$ time.*

More generally, for all $k \geq 3$ the fastest known algorithms for k-EDSP on DAGs in the literature work by reducing this problem to k-DSP using the reduction mentioned in the previous paragraph. Consequently, the current fastest algorithm for k-EDSP in DAGs runs in $O(m^k)$ time, which in dense graphs is much slower than the $O(mn^{k-1})$ time algorithm known for k-DSP. For the same reason, the fastest known algorithm for k-EDP in DAGs for $k \geq 3$ runs in $O(m^k)$ time.

Our final algorithmic result is that we can solve k-EDSP in weighted DAGs as quickly as the fastest known algorithms for k-DSP.

▶ **Theorem 5.** *The k-EDSP problem can be solved in weighted DAGs in $O(mn^{k-1})$ time.*

Since k-EDSP in weighted DAGs generalizes the k-EDP problem in DAGs, Theorem 5 also implies faster algorithms for this latter problem. Our algorithm is simple and employs the same general approach used by previous routines [18, 6] for this problem.

Lower Bounds

For $k \geq 3$, the known $O(mn^{k-1})$ algorithms for k-DP and k-DSP in DAGs have resisted any improvements over the past three decades. Thus, it is natural to wonder whether there is complexity theoretic evidence that solving these problems significantly faster would be difficult. Researchers have presented some evidence in this vein, in the form of reductions from the conjectured hard problem of detecting cliques in graphs.

Let $k = \Theta(1)$ be a positive integer. A *k-clique* is a collection of k mutually adjacent vertices in a graph. In the k-Clique problem,[3] we are given a k-partite graph G with vertex set $V_1 \sqcup \cdots \sqcup V_k$, where each part V_i has n vertices, and are tasked with determining if G contains a k-clique.

[3] This problem is sometimes referred to in the literature as k-Multicolored Clique. A folklore argument reduces from detecting a k-clique in an arbitrary n-node graph to the k-Clique problem as defined here, by making k copies of the input graph, and only including edges between different copies, e.g. as in [11, Proof of Theorem 13.7].

We can of course solve k-Clique in $O(n^k)$ time, just by trying out all possible k-tuples of vertices. Better algorithms for k-Clique are known, which employ fast matrix multiplication. Let ω denote the exponent of matrix multiplication (i.e., ω is the smallest real such that two $n \times n$ matrices can be multiplied in $n^{\omega+o(1)}$ time). Given positive reals a, b, c, we more generally write $\omega(a, b, c)$ to denote the smallest real such that we can compute the product of an $n^a \times n^b$ matrix and an $n^b \times n^c$ matrix in $n^{\omega(a,b,c)+o(1)}$ time. Then it is known that k-Clique can be solved in

$$C(n, k) = \Theta(n^{\omega(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)})$$

time [16]. The current fastest matrix multiplication algorithms yield $\omega < 2.37156$ [28]. A popular fine-grained hardness hypothesis posits (e.g., in [27, 13]) that current algorithms for k-Clique are optimal.

▶ Hypothesis 1 (k-Clique Hypothesis). For any integer constant $k \geq 3$, solving k-Clique requires at least $C(n, k)^{1-o(1)}$ time.

In this context, previous work provided reductions from k-Clique to disjoint path problems. For example, [4] presents a reduction from k-Clique to $2k$-DSP on undirected graphs with $O((kn)^2)$ vertices (and this reduction easily extends to DAGs). Our first conditional lower bound improves this result for DAGs, by halving the number of paths needed in the reduction.

▶ **Theorem 6.** *There is a reduction from k-Clique to k-DSP on unweighted DAGs with $O((kn)^2)$ vertices, that runs in $O((kn)^2)$ time.*

▶ **Corollary 7.** *Assuming the k-Clique Hypothesis, k-DSP requires $C(n^{1/2}, k)^{1-o(1)}$ time to solve on unweighted DAGs.*

The previous reduction of [4] yields a weaker bound of $C(n^{1/2}, \lfloor k/2 \rfloor)^{1-o(1)}$ for the time needed to solve k-DSP, assuming the k-Clique Hypothesis. If $\omega > 2$, this earlier result only gives nontrivial (that is, superquadratic) lower bounds for $k \geq 10$, and if $\omega = 2$ is only nontrivial for $k \geq 14$. In comparison, if $\omega > 2$, Corollary 7 is nontrivial for $k \geq 5$, and if $\omega = 2$, Corollary 7 is still nontrivial for $k \geq 7$. See Table 1 for the concrete lower bounds we achieve for small k.

As mentioned before, the k-DSP problem in weighted DAGs generalizes k-DP in DAGs. However, the current fastest algorithms for k-DP have the same time complexity as the current best algorithms for the more general k-DSP problem. To explain this behavior, it is desirable to show conditional lower bounds for k-DP in DAGs, which are similar in quality to the known lower bounds for k-DSP in DAGs.

Such lower bounds have been shown by [10]. In particular, [10] together with standard reductions in parameterized complexity [11, Proofs of Theorems 14.28 and 14.30] implies that there is a reduction from k-Clique to $8k$-EDSP on graphs with $O((kn)^4)$ nodes. One can easily modify this reduction, using the idea in the construction from [5, Section 6], to instead reduce from k-Clique to $8k$-DSP on graphs with $O((kn)^4)$ nodes.

This reduction implies that k-DSP requires $C(n^{1/4}, \lfloor k/8 \rfloor)^{1-o(1)}$ time to solve on DAGs, assuming the k-Clique Hypothesis. For large k, this is the current best conditional lower bound for k-DP in DAGs. However, the blow-up in the graph size and parameter value in this reduction makes this lower bound irrelevant for small k (in fact, the reduction only yields nontrivial lower bounds under the k-Clique Hypothesis for $k \geq 96$).

For small values of k, better conditional lower bounds are known for k-DP. In particular, [23] presents reductions from k-Clique to p-DP and p-DSP on DAGs with $O(kn)$ vertices, for parameter value $p = k + \binom{k}{2}$. For our final conditional lower bound, we improve this reduction, by reducing the number of paths needed to $k + \lfloor k^2/4 \rfloor$.

▶ **Theorem 8.** *Let $k \geq 3$ be a constant integer, and set $p = k + \lfloor k^2/4 \rfloor$. There are $O((kn)^2)$ time reductions from k-Clique to p-DP and p-DSP on unweighted DAGs with $O(kn)$ vertices.*

Table 1 A list of lower bounds implied by Corollary 7 for k-DSP when $5 \leq k \leq 9$. Each row corresponds to a value of k. An entry of α in the left column of the row for a given k value indicates that solving k-DSP in $O(n^{\alpha-\delta})$ time for any constant $\delta > 0$ would require refuting the k-Clique Hypothesis or designing faster matrix multiplication algorithms. An entry of β in the right column in the row for a given k value indicates that assuming the k-Clique Hypothesis, k-DSP requires $n^{\beta-o(1)}$ time to solve. **The previous reduction of [4] gave no nontrivial lower bound for k-DSP for any value of k in this table**, and the reduction of [23] matches our lower bound for $k = 6$, but is worse everywhere else. Table entry values are based off rectangular matrix multiplication exponents from [28, Table 1].

k	k-DSP Exponent Lower Bound	
	(for current ω)	(if $\omega = 2$)
5	2.042	Trivial
6	2.371	Trivial
7	2.794	2.5
8	3.198	3
9	3.557	3

For each integer $p \geq 5$, we can find the largest integer $k \geq 3$ such that $k + \lfloor k^2/4 \rfloor \leq p$, and then apply Theorem 8 to obtain conditional lower bounds for p-DP and p-DSP on DAGs.

▶ **Corollary 9.** *Assuming the k-Clique Hypothesis, the p-DSP and p-DP problems require*

$$\max\left(C(n, k_{even}(p)), C(n, k_{odd}(p))\right)^{1-o(1)}$$

time to solve on unweighted DAGs for all integers $p \geq 5$, where

$$k_{even}(p) = 2\lfloor\sqrt{p+1}\rfloor - 2$$

and

$$k_{odd}(p) = 2\left\lfloor\frac{\sqrt{p+5}-1}{2}\right\rfloor - 1$$

are the largest even and odd integers k such that $k + \lfloor k^2/4 \rfloor \leq p$ respectively.

Assuming the k-Clique Hypothesis, Corollary 9 shows that 5-DSP requires at least $n^{\omega-o(1)}$ time and 8-DSP requires at least $C(n,4)^{1-o(1)}$ time to solve. For the current value of ω, these yield lower bounds of $n^{2.371-o(1)}$ for 5-DSP and $n^{3.198-o(1)}$ for 8-DSP, which are better than the lower bounds implied by Corollary 7 (see Table 1). If $\omega = 2$ however, Corollary 9 does not yield better lower bounds than Corollary 7 for k-DSP.

Previous reductions give nontrivial lower bounds for p-DP only when $p \geq 6$ if $\omega > 2$, and $p \geq 10$ if $\omega = 2$. In comparison, Corollary 9 yields nontrivial lower bounds for p-DP under the k-Clique Hypothesis for $p \geq 5$ if $\omega > 2$, and $p \geq 8$ if $\omega = 2$.

Previously, the reduction of [23] yielded the best lower bounds for p-DP for $p \leq 2016$, and otherwise the reduction of [10] yielded better lower bounds. In comparison, Corollary 9 yields lower bounds matching the reduction from [23] for $p \in \{6, 7, 10\}$, and otherwise, for $\omega > 2$, yields strictly better lower bounds for p-DP for all $p \geq 5$. Moreover, for $\omega = 2$, Corollary 9 yields the best lower bounds for p-DP for all $p \leq 4031$ (with [10] yielding better lower bounds only for larger p). To see quantitatively how Corollary 9 improves the best conditional lower bounds for p-DP from previous work at various concrete values of p, see Table 2.

Table 2 A list of lower bounds implied by Corollary 9 (and previous work) for p-DP at various values of p. Rows correspond to values of p. For a given such row, the entries α, β, γ in the three columns collected under the heading of "p-DP Exponent Lower Bound," read from left to right, indicate that Corollary 9, the reduction of [23], and the reduction of [10] imply that p-DP requires $n^{\alpha-o(1)}$, $n^{\beta-o(1)}$, and $n^{\gamma-o(1)}$ time to solve respectively, assuming the k-Clique Conecture.

p	p-DP Exponent Lower Bound (if $\omega = 2$)		
	From Corollary 9	Reduction of [23]	Reduction of [10]
9	3	Trivial	Trivial
24	6	4	Trivial
89	12	8	Trivial
239	20	14	5
929	40	28	19.5
2016	58	42	42
2969	72	51	62
4031	84	60	84

1.1 Comparison with Previous Algorithms

Previous algorithms for 2-DSP and 2-DP in DAGs are combinatorial in nature: they observe certain structural properties of candidate solutions, and then leverage these observations to build up pairs of disjoint paths. In the special case of *unweighted* undirected graphs, [8] presented an algebraic algorithm for solving a generalization of 2-DSP, but all other prior algorithms for 2-DSP and 2-DP in undirected graphs are combinatorial. Our work is the first to employ algebraic methods to tackle the general weighted 2-DSP problem: our algorithms for 2-DSP on undirected graphs and DAGs work by checking that a certain polynomial, whose monomials correspond uniquely to pairs of disjoint shortest paths in the input graph, is nonzero. To obtain the fast runtimes in Theorems 1 and 2, we evaluate this polynomial over a field of characteristic two, and crucially exploit certain symmetries which make efficient evaluation possible when working modulo two.

Such "mod 2 vanishing" methods have appeared previously in the literature for algebraic graph algorithms, but the symmetries we exploit in our algorithms for 2-DSP differ in interesting ways from those of previous approaches. For example, previous methods tend to work exclusively in undirected graphs (relying on the ability to traverse cycles in both the forwards and backwards directions to produce terms in polynomials which cancel modulo 2), while our approach is able to handle 2-DSP in both undirected graphs and DAGs. It is also interesting that our algorithms solve 2-DSP in *weighted* graphs without any issue, since the previous algebraic graph algorithms we are aware of are efficient in unweighted graphs, but in weighted graphs have a runtime which depends polynomially on the value of the maximum edge weight.

Below, we compare our techniques to previous algebraic algorithms in the literature.

Two Disjoint Paths with Minimum Total Length

The most relevant examples of algebraic graph algorithms in the literature to our work are previous algorithms for the MinSum 2-DP problem: in this problem, we are given a graph G on n vertices, with specified sources s_1, s_2 and targets t_1, t_2, and are tasked with finding internally vertex-disjoint paths P_i from s_i to t_i, such that the sum of the lengths of P_1 and P_2 is minimized, or reporting that no such paths exists.

In *unweighted* undirected graphs, [7] showed that MinSum 2-DP can be solved in polynomial time, with [8, Section 6] providing a faster implementation of this approach running in $\tilde{O}(n^{4+\omega})$ time. Similar to our work, these algorithms check if a certain polynomial enumerating disjoint pairs of paths in G is nonzero or not. These methods rely on G being undirected, and are based off computing determinants of $n \times n$ matrices.

Our approach for 2-DSP differs from these arguments because we seek linear time algorithms, and so **avoid computing determinants** (which would yield $\Omega(n^\omega)$ runtimes). We instead directly enumerate pairs of intersecting paths and subtract them out. This alternate approach also allows us to obtain algorithms which apply to both undirected graphs and DAGs, whereas the cycle-reversing arguments of [8] do not appear to extend to DAGs.

Paths and Linkages with Satisfying Length Conditions

Given sets S and T of p source and target vertices respectively, an (S, T)-linkage is a set of p vertex-disjoint paths, beginning at different nodes in S and ending at different nodes in T. The length of such a linkage is the sum of the lengths of the paths it contains. Recent work has presented algorithms for the problem of finding (S, T)-linkages in undirected graphs of length at least k, fixed-parameter tractable in k. In particular, [17, Section 4] presents an algorithm solving this problem in $2^{k+p}\,\mathrm{poly}(n)$ time. Their algorithm enumerates collections of p walks beginning at different nodes in S and ending at different nodes in T. They then argue that all terms in this enumeration with intersecting walks cancel modulo 2, leaving only the (S, T)-linkages. One idea used in the above cancellation argument is that if two paths P and Q in a collection intersect at a vertex v, then we can pair this collection with a new collection obtained by swapping the suffixes of P and Q after vertex v.

In the 2-DSP problem, solution paths must connect sources s_i to corresponding targets t_i instead of to arbitrary targets, and so we cannot use the above suffix-swapping argument to get cancellation. So to enumerate disjoint shortest paths in our algorithms, we employ somewhat trickier cancellation arguments than what was previously used.

More recently, [14, Section 6] presented an algorithm solving the linkage problem discussed above in $2^k\,\mathrm{poly}(n)$ time (with runtime independent of p). Their approach uses determinants to enumerate (S, T)-linkages. As mentioned previously, we explicitly avoid using determinants so that we can obtain linear time algorithms.

Additional Related Work

There are many additional examples of algebraic graph algorithms in the literature. For example, [9] presents an efficient algorithm for finding shortest cycles through specified subsets of vertices, [12] presents algorithms for finding shortest cycles and perfect matchings in essentially matrix multiplication time, and [8] presents a polynomial time algorithm for finding a shortest cycle of even length in a directed graph. Even more examples of algebraic methods in parameterized algorithms are listed in [14, Table 1].

Bibliographic Remark

While the current paper was under submission, the work [3] of Bentert, Fomin, and Golovach was posted online. The reduction they use to establish [3, Theorem 1] is essentially the same as the reduction we use to prove Theorem 6, so this result was independently shown by [3].

Organization

In Section 2 we introduce notation and recall useful facts about graphs and polynomials used in our results. In Section 3 we provide some informal overviews for the proofs of our results. Full proofs of the results claimed in this paper can be found in the full version of this work [2]. We conclude in Section 4 by highlighting some open problems motivated by this work.

2 Preliminaries

General Notation

Given a positive integer a, we let $[a] = \{1, \ldots, a\}$ denote the set of the first a positive integers. Given positive integers a and b, we let $[a, b] = \{a, \ldots, b\}$ denote the set of consecutive integers from a to b inclusive (if $a > b$, then $[a, b]$ is the empty set).

Throughout, we let k denote a constant positive integer parameter.

Graph Notation and Assumptions

Throughout, we let G denote the input graph on n vertices and m edges. We let $s_1, \ldots, s_k$ denote the source vertices of G, and $t_1, \ldots, t_k$ denote the target vertices of G. A *terminal* is a source or target node. We assume without loss of generality that G is weakly connected (we can do this because we only care about solving disjoint path problems on G, and if terminals of G are in separate weakly connected components, we can solve smaller disjoint path problems on each component separately).

Given an edge $e = (u, v)$, we let $\ell(u, v)$ denote the weight of e in G. We assume all edge weights are positive. We let $\text{dist}(u, v)$ denote the distance of a shortest path (i.e., the sum of the weights of the edges used in a shortest path) from u to v. When we write "path P traverses edge (u, v)" we mean that P first enters u, then immediately goes to v.

We represent paths $P = \langle v_0, \ldots, v_r \rangle$ as sequences of vertices. If the path P passes through vertices u and v in that order, we let $P[u, v]$ denote the subpath of P which begins at u and ends at v. We let $\overleftarrow{P}$ denote the *reverse path* of P, which traverses the vertices of P in reverse order. Given two paths P and Q such that the final vertex of P is the same as the first vertex of Q, we let $P \diamond Q$ denote the concatenation of P and Q.

Shortest Path DAGs

Given a graph G and specified vertex s, the *s-shortest paths DAG of G* is the graph with the same vertex set as G, which includes edge (u, v) if and only if (u, v) is an edge traversed by an (s, v)-shortest path in G. From this definition, it is easy to see that a sequence of vertices is an (s, v)-shortest path of G if and only if it is an (s, v)-path in the s-shortest paths DAG of G. Indeed, every edge of an (s, v)-shortest path in G is contained in the s-shortest paths DAG by definition, and so forms a path in this graph. Conversely, if the sequence of vertices $P = \langle v_0, \ldots, v_r \rangle$ is an (s, v)-path in the s-shortest paths DAG of G, then we can inductively show that $P[s, v_i]$ is a shortest path in G for each index i.

We observe that shortest paths DAGs can be constructed in linear time.

▶ **Proposition 10** (Shortest Path DAGs). *Let G be a weighted DAG or undirected graph with distinguished vertex s. Then we can construct the s-shortest paths DAG of G in linear time.*

Proof. By definition, an edge (u, v) is in the s-shortest paths DAG of G if and only if (u, v) is the last edge of some (s, v)-shortest path in G. This is equivalent to the condition that (u, v) is an edge in G, and

$$\text{dist}(s, v) = \text{dist}(s, u) + \ell(u, v). \tag{1}$$

So, we can construct the s-shortest paths DAG of G by computing the values of $\text{dist}(s, v)$ for all vertices v, and then going through each edge (u, v) in G (if G is undirected, we try out both ordered pairs (u, v) and (v, u) of an edge $\{u, v\}$) and checking if Equation (1) holds.

So to prove the claim, it suffices to compute $\text{dist}(s, v)$ for all vertices v in linear time.

When G is a weighted DAG, we can compute a topological order of G in linear time, and then perform dynamic programming over the vertices in this order to compute $\text{dist}(s, v)$ for all vertices v in linear time (this procedure is just a modified breadth-first search routine).

When G is an undirected graph, we instead use Thorup's linear-time algorithm for single-source shortest paths in weighted undirected graphs [26] to compute $\text{dist}(s, v)$ for all vertices v. ◀

Finite Fields

Our algorithms for 2-DSP in undirected graphs and DAGs involve working over a finite field $\mathbb{F}_{2^q}$ of characteristic two, where $q = O(\log n)$. We work in the Word-RAM model with words of size $O(\log n)$, so that addition and multiplication over this field take constant time.

We make use of the following classical result, which shows that we can test if a polynomial is nonzero by evaluating it at a random point of a sufficiently large finite field.

▶ **Proposition 11** (Schwartz-Zippel Lemma)**.** *Let f be a nonzero polynomial of degree at most d. Then a uniform random evaluation of f over $\mathbb{F}$ is nonzero with probability at least $1 - d/|\mathbb{F}|$.*

3 Technical Overview

3.1 2-DSP Algorithms

We first outline a linear time algorithm solving 2-DP in DAGs. We then discuss the changes needed to solve the 2-DSP problem in weighted DAGs, and then the additional ideas used to solve 2-DSP in weighted undirected graphs.

Let G be the input DAG. For each edge (u, v) in G, we introduce an indeterminate x_{uv}. We assign each pair of paths in G a certain monomial over the x_{uv} variables, which records the pairs of consecutive vertices traversed by the paths. These monomials are constructed so that any pair of disjoint paths has a unique monomial.

Let F be the sum of monomials corresponding to all pairs of paths $\langle P_1, P_2 \rangle$ such that P_i is an (s_i, t_i)-path in G. Let F_{disj} and $F_{\cap}$ be the sums of monomials corresponding to all such pairs of paths which are disjoint and intersecting respectively. Since each disjoint pair of paths produces a distinct monomial, we can solve 2-DP by testing whether F_{disj} is a nonzero polynomial. We can perform this test by evaluating F_{disj} at a random point, by the Schwartz-Zippel lemma (Proposition 11).

Since every pair of paths is either disjoint or intersecting, we have

$$F = F_{\text{disj}} + F_{\cap}$$

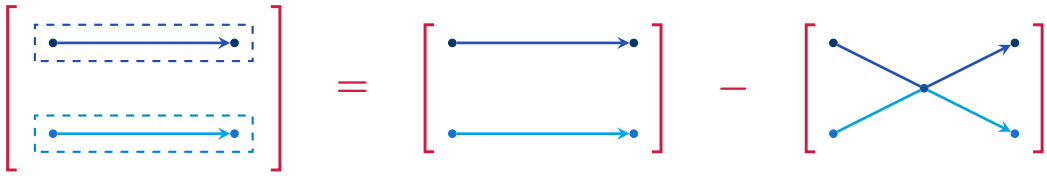

Figure 1 To enumerate the family of disjoint pairs of paths on the left (the dashed borders around the paths indicate that the paths do not intersect), it suffices to enumerate all pairs of paths and subtract out those pairs in the family which intersect at some point.

which implies that

$$F_{\text{disj}} = F - F_{\cap}.$$

This relationship is pictured in Figure 1.

Thus, in order to evaluate F_{disj}, it suffices to evaluate F and $F_{\cap}$. Since F enumerates pairs of paths from the sources to their corresponding targets with no constraints, it turns out that F is easy to evaluate. So solving 2-DP amounts to evaluating $F_{\cap}$ efficiently.

To evaluate $F_{\cap}$, we need a way of enumerating over all pairs of intersecting paths. Each pair of intersecting paths overlaps at a unique earliest vertex v (with respect to the topological order of G). Consequently, if we let F_v be the sum of monomials of pairs of intersecting paths with first intersection at v, we have

$$F_{\cap} = \sum_{v \in V} F_v \tag{2}$$

as depicted in Figure 3.

We evaluate each F_v by relating it to a seemingly simpler polynomial. Let $\tilde{F}_v$ be the polynomial enumerating pairs of paths $\langle P_1, P_2 \rangle$ where P_i is an (s_i, t_i)-path in G such that

1. P_1 and P_2 intersect at vertex v, and
2. the vertices appearing immediately before v on P_1 and P_2 are distinct.

We can think of property 2 as a relaxation of the condition that P_1 and P_2 have v as their earliest intersection point: instead of requiring that $P_1[s_1, v]$ and $P_2[s_2, v]$ never overlap before v, we merely require that these subpaths do not overlap at the position *immediately* before v. It turns out evaluating $\tilde{F}_v$ is easy, because we can enforce property 2 above by enumerating over all pairs of paths which intersect at v, and then subtracting out all such pairs which overlap at some edge ending at v. Simultaneously evaluating all $\tilde{F}_v$ can then be done in $O(m)$ time, roughly because we perform one subtraction for each possible edge the paths could overlap at.

So far, we have explained how to compute all $\tilde{F}_v$ values in linear time. Now comes the key idea behind our algorithm: over fields of characteristic two, the polynomials $\tilde{F}_v$ and F_v are actually identical! Indeed, consider a pair of paths $\langle P_1, P_2 \rangle$ enumerated by $\tilde{F}_v$, which intersects before v. Let the first intersection point of these paths be some vertex u. Then by condition 2 above, the subpaths $P_1[u, v]$ and $P_2[u, v]$ are distinct, because their penultimate vertices are distinct. So if we define new paths

$$Q_1 = P_1[s_1, u] \diamond P_2[u, v] \diamond P_1[v, t_1] \quad \text{and} \quad Q_2 = P_2[s_2, u] \diamond P_1[u, v] \diamond P_2[v, t_2]$$

obtained by swapping the u to v subpaths in P_1 and P_2, we get a new pair of paths $\langle Q_1, Q_2 \rangle$ satisfying conditions 1 and 2 from before, such that each Q_i is an (s_i, t_i)-path in G, which produces the same monomial as $\langle P_1, P_2 \rangle$. This subpath swapping operation is depicted in Figure 2, for $u = a$ and $v = b$. Then modulo two, the contributions of the pairs $\langle P_1, P_2 \rangle$

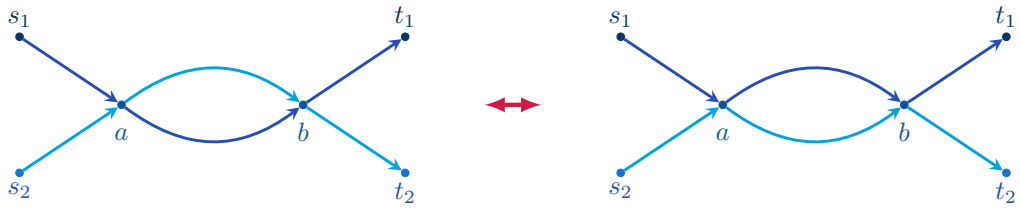

Figure 2 Given paths P_1 and P_2 which intersect at nodes $a = \alpha(P_1, P_2)$ and $b = \beta(P_1, P_2)$, such that a appears before b on both paths, if we swap the a to b subpaths of of P_1 and P_2 to produce new paths Q_1 and Q_2 respectively, then these pairs $f(P_1, P_2) = f(Q_1, Q_2)$ have the same monomials. Moreover, swapping the a to b subpaths of Q_1 and Q_2 recovers P_1 and P_2.

and $\langle Q_1, Q_2 \rangle$ to $\tilde{F}_v$ will cancel out. It follows that all pairs of paths which intersect before v have net zero contribution to $\tilde{F}_v$, and so $\tilde{F}_v = F_v$ as claimed. This congruence is depicted in Figure 4.

Given this observation, we can use our evaluations of $\tilde{F}_v$ in Equation (2) to evaluate $F_\cap$ and thus, by the previous discussion, solve the 2-DP problem.

From Disjoint Paths to Disjoint Shortest Paths

To solve 2-DSP in weighted DAGS, we can modify the 2-DP algorithm sketched above as follows. First, for $i \in [2]$, we compute G_i, the s_i-shortest paths DAG of G. We then construct polynomials as above, but with the additional constraint that they only enumerate pairs of paths $\langle P_1, P_2 \rangle$ with the property that every edge in path P_i lies in G_i. This ensures that we only enumerate pairs of paths which are shortest paths between their terminals.

With this change, the above algorithm for 2-DP generalizes to solving 2-DSP.

▶ Remark 12 (Enumeration Makes Generalization Easy). Previous near-linear time algorithms for 2-DP in DAGs and undirected graphs do not easily generalize to solving 2-DSP. In contrast, as outlined above, in our approach moving from 2-DP to 2-DSP is simple. Why is this?

Intuitively, this happens because our algorithms take an *enumerative* perspective on 2-DSP, rather than the detection-based strategy of previous algorithms. Older algorithms iteratively build up solutions to 2-DP or 2-DSP. Depending on the problem, this involves enforcing different sorts of constraints, since partial solutions to these problems may look quite different. In our approach, we just need to enumerate paths to solve 2-DP and enumerate shortest paths to solve 2-DSP. Enumerating paths and shortest paths are both easy in DAGs by dynamic programming. Hence algorithms for these two problems end up being essentially the same in our framework.

From DAGs to Undirected Graphs

When solving 2-DSP in DAGs, we used the fact that DAGs have a topological order, so that any pair of paths intersects at a unique earliest vertex v in this order. This simple decomposition does not apply to solving 2-DSP in undirected graphs, since we cannot rely on a fixed topological order.

Instead, we perform casework on the first vertex v in P_1 lying in $P_1 \cap P_2$. We observe that in undirected graphs, there are two possibilities: v is either the first vertex in P_2 lying in $P_1 \cap P_2$, or it is the final vertex in P_2 lying in $P_1 \cap P_2$. Intuitively, the paths either "agree" and go in the same direction, or "disagree" and go in opposite directions.

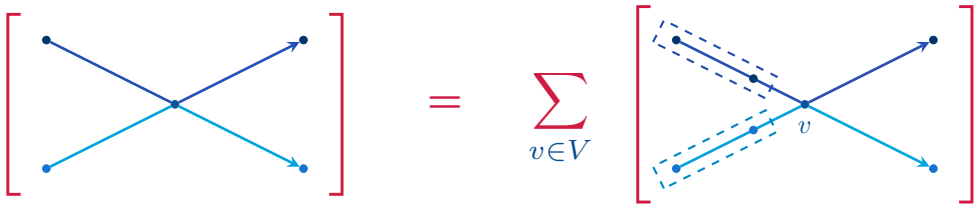

Figure 3 To enumerate the family of intersecting pairs of paths on the left, we can perform casework on the earliest intersection point v for the paths (the dashed border on the subpaths on the right indicates that the paths do not intersect before v).

We then argue that over a field of characteristic two, we can efficiently enumerate over pairs of paths in each of these cases. As with DAGs, we make this enumeration efficient by arguing that modulo 2 we can relax the (a priori difficult to check) condition of v being the first intersection point on P_1 to some simpler "local" condition. When the paths agree, this argument is similar to the reasoning used for solving 2-DSP in DAGs.

For the case where the paths disagree, this enumeration is more complicated, because there is no consistent linear ordering of the vertices neighboring v on the two shortest paths, but can still be implemented in linear time using a more sophisticated local condition. Specifically, if we let a_i and b_i denote the nodes appearing immediately before and after v on path P_i, then to enumerate the "disagreeing paths" modulo two, we prove that it suffices to enumerate paths P_1 and P_2 which intersect at v and have the properties that $a_1 \neq a_2$, $b_1 \neq b_2$, and $a_1 \neq b_2$. Intuitively, using the subpath swapping idea depicted in Figure 4, the conditions that $a_1 \neq a_2$ and $a_1 \neq b_2$ ensure that v is the first vertex of P_1 lying in $P_1 \cap P_2$, and the condition that $b_1 \neq b_2$ ensures that the paths disagree. To implement this idea, we need a slightly more complicated subpath swapping argument, which can also handle the case where two paths P_1 and P_2 intersect at vertices u and v, with u appearing before v on P_1 but u appearing after v on P_2 (this situation does not occur in DAGs, but can occur in undirected graphs). We do this by combining the previous subpath swapping idea with the observation that in undirected graphs we can also traverse subpaths in the reverse direction (so it is possible to swap the subpaths $P_1[u, v]$ and $P_2[v, u]$ in P_1 and P_2, even though u and v appear in different orders on P_1 and P_2).

By combining the enumerations for both cases, we can evaluate F_{disj}, and thus solve 2-DSP over undirected graphs.

3.2 k-EDSP Algorithm

The previous algorithm of [6, Proposition 10] for k-EDSP works by constructing a graph G' encoding information about k-tuples of edge-disjoint shortest paths in the original graph G. This new graph G' has special nodes $\vec{s}$ and $\vec{t}$, such that there is a path from $\vec{s}$ to $\vec{t}$ in G' if and only if G contains k edge-disjoint shortest paths connecting its terminals. The nodes of the new graph G' are k-tuples of edges $(e_1, \ldots, e_k)$ where each e_i is an edge in G. So constructing G' already takes $\Omega(m^k)$ time.

Our algorithm for k-EDSP uses the same general idea. We construct an alternate graph G' which still has the property that finding a single path between two specified vertices of G' solves the k-EDSP problem in G. However, we design G' to have nodes of the form $(v_1, \ldots, v_k)$, where each v_i is a *vertex* in G. Our construction produces a graph with n^k nodes and $O(mn^{k-1})$ edges, which yields the speed-up. We avoid the $\Omega(m^k)$ bottleneck of the previous algorithm by showing how to encode edge-disjointness information simply through the k-tuples of vertices, rather than edges, that the k potential solution paths in G traverse.

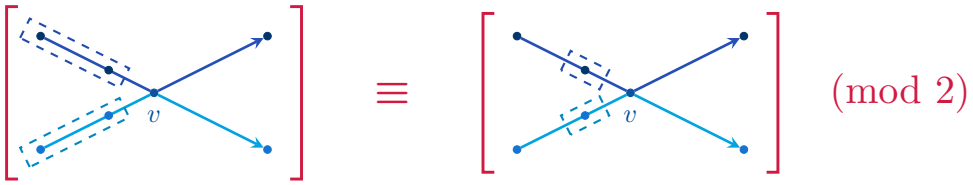

Figure 4 If we work modulo two, then we can enumerate pairs of paths which have common first intersection at node v by enumerating pairs of paths which intersect at v and have the property that the vertices appearing immediately before v on each path are distinct.

3.3 Lower Bounds

Disjoint Shortest Paths

Our proof of Theorem 6 is based on the reduction of [5, Proposition 1] from k-Clique to $2k$-DSP on undirected graphs, which also easily extends to DAGs. Our contribution is a transformation that reduces the number of paths in their reduction from $2k$ to k by exploiting the symmetry of the construction.

The reduction of [5] maps each vertex v in the k-Clique instance to a horizontal path P_v and a vertical path Q_v, each of length n. These paths are arranged so that for each pair of vertices (v, w) in the input graph, the paths P_v and Q_w intersect if and only if (v, w) is not an edge in the input graph. To achieve this, the paths are placed along a grid, and at the intersection point in the grid between paths P_v and Q_w, these two paths are modified to bypass each other to avoid intersection if (v, w) is an edge in the input graph.

The main idea of our transformation is the following. Since the known reduction is symmetric along the diagonal of the grid, it contains some redundancy. We remove this redundancy by only keeping the portion of the grid below the diagonal. To do this, we only have one path P_v for each vertex v in the input graph, and each such path has both a horizontal component and a vertical component. Each path turns from horizontal to vertical when it hits the diagonal. As a result, each pair of paths (P_v, P_w) has exactly one intersection point in the grid (which we bypass if (v, w) is an edge in the input graph). Since we produce only a single path P_v for each vertex v, we obtain a reduction to k-DSP instead of $2k$-DSP.

Disjoint Paths

The starting point for Theorem 8 is the work of [23], which reduces from k-Clique to p-EDP in a DAG with $O(kn)$ nodes, for $p = k + \binom{k}{2}$. The parameter blows up from k to p in this way because the reduction uses k solution paths to pick k vertices in the original graph, and then for each of the $\binom{k}{2}$ pairs of vertices chosen, uses an additional solution path to verify that the vertices in that pair are adjacent in the original graph.

We improve upon this by modifying the reduction graph to allow some solution paths to check multiple edges simultaneously. This lets us avoid using $\binom{k}{2}$ solution paths to separately check for edges between each pair of nodes in a candidate k-clique. Instead, we employ just $\lfloor k^2/4 \rfloor$ solution paths in the reduction, roughly halving the number of paths needed.

To do this, we need to precisely identify which paths can check for multiple edges without compromising the correctness of the reduction. To this end, we examine the structure of the reduction and define a notion of a *covering family* which characterizes which paths can safely check for multiple edges at once. Formally, a k-covering family is a collection $\mathcal{L}$ of increasing lists of positive integers, with the property that for all integers i, j with $1 \leq i < j \leq k$, some list in $\mathcal{L}$ contains i and j as consecutive members.

We show that for any k, the smallest number of lists in a k-covering family is $\lambda(k) = \lfloor k^2/4 \rfloor$ (note that merely obtaining asymptotically tight bounds would not suffice for designing interesting conditional lower bounds). We then insert this construction of a minimum size covering family into the framework of the reduction and prove that the reduction remains correct. Intuitively, given lists in a covering family, we can map each list L to a path which checks edges between vertex parts V_i and V_j for each (i, j) pair appearing as consecutive members of L.

The original reduction of [23] corresponds to implementing this strategy with the trivial k-covering family using $\binom{k}{2}$ lists, achieved by taking a single increasing list of two elements for each unordered pair of integers from $[k]$. Our improved reduction comes from implementing this framework with the optimal bound of $\lfloor k^2/4 \rfloor$ lists.

This yields reductions from k-Clique to p-DP and p-DSP for $p = k + \lambda(k) = k + \lfloor k^2/4 \rfloor$.

4 Conclusion

In this work, we obtained linear time algorithms for 2-DSP in undirected graphs and DAGs. These algorithms are based off algebraic methods, and as a consequence are *randomized* and only solve the *decision*, rather than search, version of 2-DSP. This motivates the following questions:

▶ Open 1. Is there a *deterministic* linear time algorithm solving 2-DSP?

▶ Open 2. Given a DAG or undirected graph G with sources s_1, s_2 and targets t_1, t_2, is there a linear time algorithm *finding* disjoint (s_i, t_i)-shortest paths in G for $i \in \{1, 2\}$?

It is also an interesting research direction to see if algebraic methods can help design faster algorithms for k-DSP in undirected graphs and DAGs when $k \geq 3$, or help tackle this problem in the case of general directed graphs.

In this work, we also established tighter reductions from finding cliques to disjoint path and shortest path problems. There still remain large gaps however, between the current best conditional lower bounds and current fastest algorithms for these problems.

▶ Open 3. Is there a fixed integer $k \geq 3$ and constant $\delta > 0$ such that k-DSP in DAGs can be solved in $O(n^{k+1-\delta})$ time? Or does some popular hypothesis rule out such an algorithm?

Since k-Clique admits nontrivial algorithms by reduction to matrix multiplication, it is possible that k-DSP can be solved faster using fast matrix multiplication algorithms. On the other hand, if we want to rule out this possibility and obtain better conditional lower bounds for k-DSP, we should design reductions from problems which are harder than k-Clique. In this context, a natural strategy would be to reduce from Negative k-Clique and 3-Uniform k-Hyperclique instead, since these problems are conjectured to require $n^{k-o(k)}$ time to solve (and it is not known how to leverage matrix multiplication to solve these problems faster than exhaustive search).

For all $k \geq 3$, the current fastest algorithm for k-DSP in undirected graphs takes $n^{O(k \cdot k!)}$ time, much slower than the $O(mn^{k-1})$ time algorithm known for the problem in DAGs. Despite this, every conditional lower bound that has been established for k-DSP in undirected graphs so far also extends to showing the same lower bound for the problem in DAGs. This is bizarre behavior, and suggests we should try establishing a lower bound which separates the complexities of k-DSP in undirected graphs and DAGs. If designing such a lower bound proves difficult, that would offer circumstantial evidence that far faster algorithms for k-DSP in undirected graphs exist.

▶ Open 4. Can we show a conditional lower bound for k-DSP in undirected graphs, which is stronger than any conditional lower bound known for k-DSP in DAGs?

Finally, for large k, the best conditional time lower bounds we have for k-DP in DAGs are far weaker than the analogous lower bounds we have for k-DSP in DAGs. This is despite the fact that the fastest algorithms we have for both problems run in the same time. It would nice to resolve this discrepancy, either by designing faster algorithms for the latter problem, or showing better lower bounds for the former problem.

▶ Open 5. Is there a fixed integer $k \geq 3$ such that we can solve k-DP in DAGs faster than we can solve k-DSP in weighted DAGs?

▶ Open 6. Can we show a conditional lower bound for k-DP in DAGs matching the best known conditional lower bound for k-DSP in DAGs?

References

1 Maxim Akhmedov. Faster 2-disjoint-shortest-paths algorithm. In *Computer Science – Theory and Applications*, pages 103–116. Springer International Publishing, 2020. doi:10.1007/978-3-030-50026-9_7.

2 Shyan Akmal, Virginia Vassilevska Williams, and Nicole Wein. Detecting Disjoint Shortest Paths in Linear Time and More, 2024. arXiv:2404.15916.

3 Matthias Bentert, Fedor V. Fomin, and Petr A. Golovach. Tight approximation and kernelization bounds for vertex-disjoint shortest paths, 2024. arXiv:2402.15348.

4 Matthias Bentert, André Nichterlein, Malte Renken, and Philipp Zschoche. Using a Geometric Lens to Find k Disjoint Shortest Paths. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.26.

5 Matthias Bentert, André Nichterlein, Malte Renken, and Philipp Zschoche. Using a geometric lens to find k disjoint shortest paths, 2020. doi:10.48550/arXiv.2007.12502.

6 Kristof Berczi and Yusuke Kobayashi. The Directed Disjoint Shortest Paths Problem. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2017.13.

7 Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. *SIAM Journal on Computing*, 48(6):1698–1710, January 2019. doi:10.1137/18m1223034.

8 Andreas Björklund, Thore Husfeldt, and Petteri Kaski. The shortest even cycle problem is tractable. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, June 2022. doi:10.1145/3519935.3520030.

9 Andreas Björklund, Thore Husfeldt, and Nina Taslaman. Shortest cycle through specified elements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, January 2012. doi:10.1137/1.9781611973099.139.

10 Rajesh Chitnis. A tight lower bound for edge-disjoint paths on planar dags, 2021. doi:10.48550/arXiv.2101.10742.

11 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.

12 Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of baur-strassen's theorem. *Journal of the ACM*, 62(4):1–30, September 2015. doi:10.1145/2736283.

13 Mina Dalirrooyfard and Virginia Vassilevska Williams. Induced cycles and paths are harder than you think. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 531–542. IEEE, 2022.

14 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving, 2023. arXiv:2304.02091.

15 Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, June 1998. doi:10.1016/s0166-218x(97)00121-2.

16 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, October 2004. doi:10.1016/j.tcs.2004.05.009.

17 Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, Kirill Simonov, and Giannos Stamoulis. Fixed-parameter tractability of maximum colored path and beyond, 2022. arXiv:2207.07449.

18 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, February 1980. doi:10.1016/0304-3975(80)90009-2.

19 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, March 2012. doi:10.1016/j.jctb.2011.07.004.

20 Tuukka Korhonen, Michał Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time, 2024. arXiv:2404.03958.

21 Willian Lochet. A polynomial time algorithm for the k-disjoint shortest paths problem. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 169–178. Society for Industrial and Applied Mathematics, January 2021. doi:10.1137/1.9781611976465.12.

22 N. Robertson and P.D. Seymour. Graph minors .XIII. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, January 1995. doi:10.1006/jctb.1995.1006.

23 Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 24(1):146–157, January 2010. doi:10.1137/070697781.

24 Torsten Tholey. Solving the 2-disjoint paths problem in nearly linear time. *Theory of Computing Systems*, 39(1):51–78, November 2005. doi:10.1007/s00224-005-1256-9.

25 Torsten Tholey. Linear time algorithms for two disjoint paths problems on directed acyclic graphs. *Theoretical Computer Science*, 465:35–48, December 2012. doi:10.1016/j.tcs.2012.09.025.

26 M. Thorup. Undirected single source shortest paths in linear time. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc, 1997. doi:10.1109/sfcs.1997.646088.

27 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*, pages 3447–3487. World Scientific, 2018.

28 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega, 2023. arXiv:2307.07970.