

Detecting Disjoint Shortest Paths in Linear Time and More

Shyan Akmal*
MIT

Virginia Vassilevska Williams†
MIT

Nicole Wein‡
University of Michigan

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 $s_1, \dots, s_k$, and target vertices $t_1, \dots, t_k$, 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 $k \geq 6$.

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. 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.

*naysh@mit.edu. Partially supported by Virginia Vassilevska Williams' Simons Investigator Award.

†virgi@mit.edu. Supported by NSF Grant CCF-2330048, BSF Grant 2020356 and a Simons Investigator Award.

‡nswein@umich.edu.

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 $s_1, \dots, s_k$ and target nodes $t_1, \dots, t_k$, 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 [RS95].

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 [FW80, 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 $k = 2$ [Tho05], and in $O(n^2)$ time or $m^{1+o(1)}$ time for any constant $k \geq 3$ [KKR12, KPS24].

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 [BK17], 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 [Loc21]. 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 [BNRZ21].

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 weighted¹ undirected graphs [ET98]. The first improvement for this problem came over twenty years later in [Akh20], which presented an algorithm solving 2-DSP in weighted undirected graphs in $O(n^7)$ time, and in unweighted undirected graphs in $O(n^6)$ time. Soon after, [BNRZ21, 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. The 2-DSP problem 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 [ET98].

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

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

²It seems plausible that the method of [BNRZ21] could be adapted to handle weighted undirected graphs as well, but such a generalization does not appear to currently be known.

on DAGs [BK17]. Similarly, the fastest known algorithm for k -DSP on undirected graphs works by reducing to several instances of disjoint paths on DAGs [BNRZ21].

It is known that 2-DP in DAGs can be solved in linear time [Tho12]. 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 $k \geq 3$ [FWW80, Theorem 3].

As observed in [BK17, Proposition 10], the algorithm of [FWW80] 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. The 2-DSP problem 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 [FWW80] 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 [Tho12], 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.

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 (see Appendix A for the details). 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. We can solve 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 (see [Appendix A](#) for details), [Theorem 5](#) also implies faster algorithms for this latter problem. Our algorithm is simple and employs the same general approach used by previous routines [[FW80](#), [BK17](#)] 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 $V_1 \sqcup \dots \sqcup V_k$, where each part V_i has n vertices, and are tasked with determining if G contains a k -clique.

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 [[EG04](#)]. The current fastest matrix multiplication algorithms yield $\omega < 2.37156$ [[WXXZ23](#)]. A popular fine-grained hardness hypothesis posits (e.g., in [[Wil18](#), [DW22](#)]) that current algorithms for k -Clique are optimal.

Hypothesis 6 (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, [[BNRZ21](#)] 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 7. 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.

³This 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 [[CFK⁺15](#), Proof of Theorem 13.7].

Corollary 8. 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 [BNRZ21] 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 8 is nontrivial for $k \geq 5$, and if $\omega = 2$, Corollary 8 is still nontrivial for $k \geq 7$. See Table 1 for the concrete lower bounds we achieve for small k .

As mentioned before (and as shown in detail in Appendix A), 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 [Chi21]. In particular, [Chi21] together with standard reductions in parameterized complexity [CFK⁺15, 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 [BNRZ20, 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, [Sli10] 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 9. 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.

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 9 to obtain conditional lower bounds for p -DP and p -DSP on DAGs.

Corollary 10. Assuming the k -Clique Hypothesis, the p -DSP and p -DP problems require

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

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

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

and

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

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 10 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

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

Table 1: A list of lower bounds implied by [Corollary 8](#) 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 [BNRZ21] gave no nontrivial lower bound for k -DSP for any value of k in this table**, and the reduction of [Sli10] matches our lower bound for $k = 6$, but is worse everywhere else. Table entry values are based off rectangular matrix multiplication exponents from [WXXZ23, Table 1].

implied by [Corollary 8](#) (see [Table 1](#)). If $\omega = 2$ however, [Corollary 10](#) does not yield better lower bounds than [Corollary 8](#) 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 10](#) 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 [Sli10] yielded the best lower bounds for p -DP for $p \leq 2016$, and otherwise the reduction of [Chi21] yielded better lower bounds. In comparison, [Corollary 10](#) yields lower bounds matching the reduction from [Sli10] 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 10](#) yields the best lower bounds for p -DP for all $p \leq 4031$ (with [Chi21] yielding better lower bounds only for larger p).

To see quantitatively how [Corollary 10](#) improves the best conditional lower bounds for p -DP from previous work at various concrete values of p , see [Table 2](#).

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, [BHK22] 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

p	p -DP Exponent Lower Bound (if $\omega = 2$)		
	From Corollary 10	Reduction of [Sli10]	Reduction of [Chi21]
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

Table 2: A list of lower bounds implied by [Corollary 10](#) (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 10](#), the reduction of [\[Sli10\]](#), and the reduction of [\[Chi21\]](#) 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 Conjecture.

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, [\[BH19\]](#) showed that MinSum 2-DP can be solved in polynomial time, with [\[BHK22, Section 6\]](#) 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 [\[BHK22\]](#) 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, [\[FGK⁺22, Section 4\]](#) presents an algorithm solving this problem in $2^{k+p} \text{poly}(n)$ 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, [EKW23, Section 6] presented an algorithm solving the linkage problem discussed above in $2^k \text{poly}(n)$ 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, [BHT12] presents an efficient algorithm for finding shortest cycles through specified subsets of vertices, [CGS15] presents algorithms for finding shortest cycles and perfect matchings in essentially matrix multiplication time, and [BHK22] 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 [EKW23, Table 1].

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 intuition for the proofs of our results. In Section 4 we present our linear time algorithms for 2-DSP in weighted undirected graphs and DAGs. In Section 5 we present our algorithm for k -EDSP. In Section 6 we present our lower bounds. Finally, we conclude in Section 7 by highlighting some open problems motivated by this work.

See Appendix A for proofs of some simple reductions between variants of disjoint path problems which were mentioned previously.

2 Preliminaries

General Notation

Given a positive integer a , we let $[a] = \{1, \dots, a\}$ denote the set of the first a positive integers. Given positive integers a and b , we let $[a, b] = \{a, \dots, b\}$ 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 $s_1, \dots, s_k$ denote the source vertices of G , and $t_1, \dots, t_k$ 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 $P = \langle v_0, \dots, v_r \rangle$ 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 $P = \langle v_0, \dots, v_r \rangle$ 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 11 (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

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

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 Eq. (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 [Tho97] 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 12 (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 $1 - d/|\mathbb{F}|$.

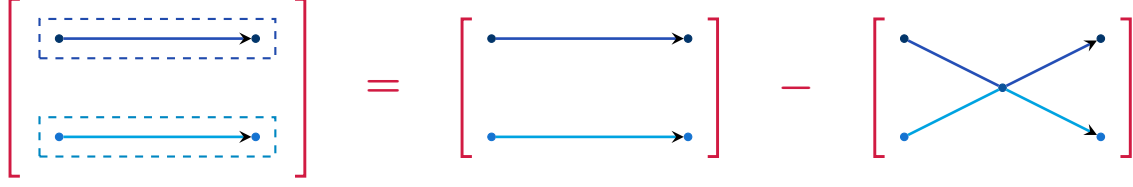


Figure 1: To enumerate the family of disjoint pairs of paths on the left (the dashed borders around the paths indicate that the paths do not intersect), it suffices to enumerate all pairs of paths and subtract out those pairs in the family which intersect at some point.

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 12](#)).

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

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

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 2](#).

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.

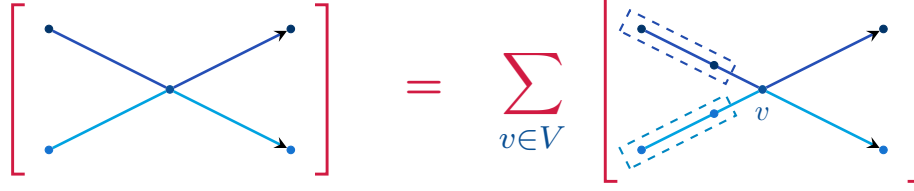


Figure 2: To enumerate the family of intersecting pairs of paths on the left, we can perform casework on the earliest intersection point v for the paths (the dashed border on the subpaths on the right indicates that the paths do not intersect before v).

We 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 4, for $u = a$ and $v = b$. Then modulo two, the contributions of the pairs $\langle P_1, P_2 \rangle$ 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 3,

Given this observation, we can use our evaluations of $\tilde{F}_v$ in Eq. (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 13 (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

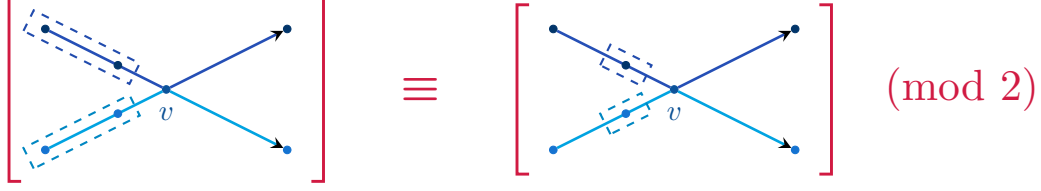


Figure 3: If we work modulo two, then we can enumerate pairs of paths which have common first intersection at node v by enumerating pairs of paths which intersect at v and have the property that the vertices appearing immediately before v on each path are distinct.

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.

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 3, 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 [BK17, 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 $(e_1, \dots, e_k)$ 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 $(v_1, \dots, v_k)$, 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.

3.3 Lower Bounds

Disjoint Shortest Paths

Our proof of Theorem 7 is based on the reduction of [BNRZ20, 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 [BNRZ20] 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 9 is the work of [Sli10], 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 [Sli10] 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 2-DSP

In this section, we present our new algorithms for 2-DSP.

We begin in Section 4.1 by introducing notation and terminology used in our algorithms. We also present a general “subpath swapping lemma,” which we repeatedly invoke to efficiently evaluate polynomials modulo 2. Then in Section 4.2 we define and prove results about various polynomials used in our algorithms. All the definitions and results in Section 4.1 and Section 4.2 apply to both DAGs and undirected graphs.

Following this setup, in Section 4.3 we present an algorithm for 2-DSP in DAGs proving Theorem 2, and in Section 4.4 we present an algorithm for 2-DSP in undirected graphs proving Theorem 1. We note that Section 4.4 can be read independently of Section 4.3 (but both Sections 4.3 and 4.4 rely on the definitions and results from Sections 4.1 and 4.2).

4.1 Graph and Polynomial Preliminaries

Graph Notation

Let G denote the input graph on n vertices and m edges. Let V be the vertex set of G .

For each $i \in [2]$, we define G_i to be the s_i -shortest paths DAG of G .

Given $i \in [2]$ and vertex v , we let $V_{\text{in}}^i(v)$ denote the in-neighbors of v in G_i , and $V_{\text{out}}^i(v)$ denote the out-neighbors of v in G_i . We further let $V_{\text{in}}(v) = V_{\text{in}}^1(v) \cap V_{\text{in}}^2(v)$ and $V_{\text{out}}(v) = V_{\text{out}}^1(v) \cap V_{\text{out}}^2(v)$ be the sets of in-neighbors and out-neighbors respectively of node v common to both G_1 and G_2 . Furthermore, we define $V_{\text{mix}}(v) = V_{\text{in}}^1(v) \cap V_{\text{out}}^2(v)$ to be the “mixed neighborhood” of v .

We call a pair of paths $\langle P_1, P_2 \rangle$ *standard* if each P_i is an (s_i, t_i) -path in G_i .

Algebraic Preliminaries

For each edge $e = (u, v)$, we introduce an indeterminate x_{uv} .

If G is undirected, we set $x_{uv} = x_{vu} = x_{\{u,v\}}$ to reflect that edges are unordered pairs of vertices. Given a path $P = \langle v_0, \dots, v_r \rangle$, we assign it the monomial

$$f(P) = \prod_{j=0}^{r-1} x_{v_j v_{j+1}}.$$

Given a pair of paths $\mathcal{P} = \langle P_1, P_2 \rangle$, we assign it monomial

$$f(\mathcal{P}) = f(P_1, P_2) = f(P_1)f(P_2).$$

Given a collection $\mathcal{S}$ of paths or pairs of paths, we say that a “polynomial F enumerates $\mathcal{S}$,” or equivalently “ F is the *enumerating polynomial* for $\mathcal{S}$,” if

$$F = \sum_{S \in \mathcal{S}} f(S).$$

Subpath Swapping

Lemma 14 (Shortest Path Swapping). Let P_1 and P_2 be shortest paths passing through vertices a and b in that order. Then the walks obtained by swapping the a to b subpaths in P_1 and P_2 are also shortest paths.

Proof. Since P_1 and P_2 are shortest paths, each of their a to b subpaths have length $\text{dist}(a, b)$. Since these subpaths have the same length, swapping the a to b subpaths of P_1 and P_2 produce walks with the same endpoints and lengths as P_1 and P_2 respectively. Since all edge weights are positive, these walks have no repeat vertices (since if one of the walks did have repeat vertices, then it would contain a cycle which we could remove to produce a walk of shorter total length between its endpoints, which would then contradict the assumption that P_1 and P_2 are shortest paths). Thus these walks must be shortest paths as claimed. ■

We will repeatedly make use of the following result, which gives a way of simplifying enumerating polynomials for certain families of pairs of paths. Although the statement may appear technical, the lemma simply formalizes the idea that for certain enumerating polynomials, we can pair up terms of equal value and get cancellation modulo two.

Lemma 15 (Vanishing Modulo 2). Let $\mathcal{F}$ be a family of pairs of paths in G , and let $\mathcal{S} \subseteq \mathcal{F}$. Suppose there exists maps $\alpha, \beta : \mathcal{S} \rightarrow V$ and $\Phi : \mathcal{S} \rightarrow \mathcal{S}$ such that for all $\mathcal{P} = \langle P_1, P_2 \rangle \in \mathcal{S}$,

1. the vertices $a = \alpha(\mathcal{P})$ and $b = \beta(\mathcal{P})$ lie in $P_1 \cap P_2$, a appears before b in P_1 and P_2 , and the subpaths $P_1[a, b]$ and $P_2[a, b]$ are distinct;
2. we have $\Phi(\mathcal{P}) = \langle Q_1, Q_2 \rangle$, where Q_1 is obtained by replacing the a to b subpath in P_1 with $P_2[a, b]$, and Q_2 is obtained by replacing the a to b subpath in P_2 with $P_1[a, b]$; and
3. we have $\Phi(\Phi(\mathcal{P})) = \mathcal{P}$.

Then the enumerating polynomial for $\mathcal{F}$ is the same as the enumerating polynomial for $\mathcal{F} \setminus \mathcal{S}$.

Proof. Let F be the enumerating polynomial for $\mathcal{F}$. By definition, we have

$$F = \sum_{\mathcal{P} \in \mathcal{F}} f(\mathcal{P}) = \sum_{\mathcal{P} \in \mathcal{F} \setminus \mathcal{S}} f(\mathcal{P}) + \sum_{\mathcal{P} \in \mathcal{S}} f(\mathcal{P}). \quad (3)$$

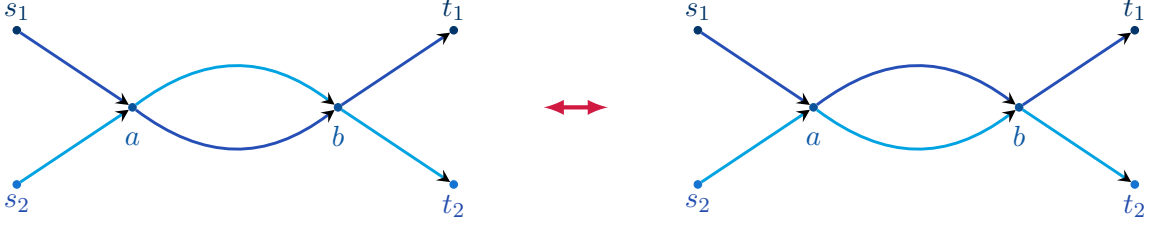


Figure 4: Given paths P_1 and P_2 which intersect at nodes $a = \alpha(P_1, P_2)$ and $b = \beta(P_1, P_2)$, such that a appears before b on both paths, if we swap the a to b subpaths of P_1 and P_2 to produce new paths Q_1 and Q_2 respectively, then these pairs $f(P_1, P_2) = f(Q_1, Q_2)$ have the same monomials. Moreover, swapping the a to b subpaths of Q_1 and Q_2 recovers P_1 and P_2 .

Take any $\mathcal{P} = \langle P_1, P_2 \rangle \in \mathcal{S}$. By property 1 from the lemma statement, the subpaths from $\alpha(\mathcal{P})$ to $\beta(\mathcal{P})$ in P_1 and P_2 are distinct. Then by property 2, $\Phi(\mathcal{P}) \neq \mathcal{P}$. Consequently, by property 3, we can partition $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ into two equally sized pieces such that Φ is a bijection from $\mathcal{S}_1$ to $\mathcal{S}_2$. So we can write

$$\sum_{\mathcal{P} \in \mathcal{S}} f(\mathcal{P}) = \sum_{\mathcal{P} \in \mathcal{S}_1} f(\mathcal{P}) + \sum_{\mathcal{P} \in \mathcal{S}_2} f(\mathcal{P}) = \sum_{\mathcal{P} \in \mathcal{S}_1} (f(\mathcal{P}) + f(\Phi(\mathcal{P}))). \quad (4)$$

By property 2, the multiset of edges traversed by the pair $\mathcal{P}$ is the same as the multiset of edges traversed by $\Phi(\mathcal{P})$, for all $\mathcal{P} \in \mathcal{S}$. Consequently, $f(\mathcal{P}) = f(\Phi(\mathcal{P}))$ for all $\mathcal{P} \in \mathcal{S}$.

The subpath swapping procedure determined by Φ is depicted in Figure 4.

Since we work over a field of characteristic two, this implies that

$$\sum_{\mathcal{P} \in \mathcal{S}_1} (f(\mathcal{P}) + f(\Phi(\mathcal{P}))) = 0.$$

Substituting the above equation into Eq. (4) implies that

$$\sum_{\mathcal{P} \in \mathcal{S}} f(\mathcal{P}) = 0.$$

Then substituting the above equation into Eq. (3) yields

$$F = \sum_{\mathcal{P} \in \mathcal{F} \setminus \mathcal{S}} f(\mathcal{P}).$$

This proves that F is the enumerating polynomial for $\mathcal{F} \setminus \mathcal{S}$ as desired. ■

4.2 Helper Polynomials

In this section we define various enumerating polynomials which will help us solve 2-DSP. All of the results in this section hold if the input graph G is a DAG or is undirected.

Paths to and from Terminals

For each $i \in [2]$ and vertex v , let $L_i(v)$ be the enumerating polynomial for the set of (s_i, v) -paths in G_i , and let $R_i(v)$ be the enumerating polynomial for the set of (v, t_i) -paths in G_i .

Lemma 16 (Path Polynomials). For each $i \in [2]$ and vertex v , we have

$$L_i(v) = \sum_{u \in V_{\text{in}}^i(v)} L_i(u) x_{uv} \quad (5)$$

and

$$R_i(v) = \sum_{w \in V_{\text{out}}^i(v)} x_{vw} R_i(w). \quad (6)$$

Proof. Since $L_i(u)$ enumerates (s_i, u) -paths in G_i , the polynomial

$$L_i(u) x_{uv}$$

enumerates (s_i, v) -paths in G_i , whose penultimate vertex is u . Every (s_i, v) -path in G_i has some unique penultimate vertex $u \in V_{\text{in}}^i(v)$. Consequently

$$\sum_{u \in V_{\text{in}}^i(v)} L_i(u) x_{uv}$$

enumerates all (s_i, v) -paths in G_i , which proves Eq. (5) as desired.

Symmetric reasoning proves Eq. (6). ■

Intersecting Paths

Let F_{disj} be the enumerating polynomial for the collection of vertex-disjoint, standard pairs of paths. Let $F_{\cap}$ be the enumerating polynomial for the collection of intersecting, standard pairs of paths. Our algorithms for 2-DSP work by evaluating F_{disj} at a random point. The following observation shows that we can compute an evaluation of F_{disj} by evaluating $F_{\cap}$ instead.

Lemma 17 (Disjoint Paths $\leq$ Intersecting Paths). We have

$$F_{\text{disj}} = L_1(t_1)L_2(t_2) - F_{\cap}.$$

Proof. By expanding out the product, we see that $L_1(t_1)L_2(t_2)$ enumerates all standard pairs of paths $\langle P_1, P_2 \rangle$. Each such pair is either vertex-disjoint or consists of paths intersecting at a common node, so we have

$$L_1(t_1)L_2(t_2) = F_{\text{disj}} + F_{\cap}$$

which implies the desired result. ■

Linkages from Two Sources to a Common Vertex

Our algorithms will use the following “linkage” polynomials, which enumerate pairs of internally vertex-disjoint paths beginning at the source nodes s_1 and s_2 , and ending at a common vertex.

Definition 18 (Source Linkage). Given a vertex v , let $\mathcal{D}(v)$ be the set of pairs of paths $\langle P_1, P_2 \rangle$ where each P_i is an (s_i, v) -path in G_i and the paths intersect only at v . Let $D(v)$ be the enumerating polynomial for $\mathcal{D}(v)$.

We now observe that for the purpose of enumeration modulo two, one can replace the “global constraint” that a pairs of paths in $\mathcal{D}(v)$ has no intersection anywhere before v with the easier to check “local constraint” that a pair of paths has no intersection immediately before v .

Definition 19 (Linkage Relaxation). Given a vertex v , let $\tilde{\mathcal{D}}(v)$ be the set of pairs of paths $\langle P_1, P_2 \rangle$, where each P_i is an (s_i, v) -path in G_i , such that the penultimate vertices of P_1 and P_2 are distinct.

Lemma 20 (Relaxing Source Linkages). For each vertex v , the polynomial $D(v)$ enumerates $\tilde{\mathcal{D}}(v)$.

Proof. For convenience, let $\mathcal{F} = \tilde{\mathcal{D}}(v)$. Let $\mathcal{S} = \mathcal{F} \setminus \mathcal{D}(v)$ be the family of pairs of paths $\langle P_1, P_2 \rangle$ where each P_i is an (s_i, v) -path in G_i , such that

1. the paths P_1 and P_2 intersect at some node other than v , and
2. the nodes immediately before v on P_1 and P_2 are distinct.

Take arbitrary $\langle P_1, P_2 \rangle \in \mathcal{S}$. Let u be the vertex in $P_1 \cap P_2$ maximizing the value of $\text{dist}(u, v)$. By condition 1 above, $u \neq v$. By condition 2 above, $P_1[u, v]$ and $P_2[u, v]$ are distinct.

Now define paths

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

By [Lemma 14](#) each Q_i is a shortest path, and thus an (s_i, v) -path in G_i .

The pair $\langle Q_1, Q_2 \rangle$ satisfies condition 1 above, since Q_1 and Q_2 intersect at u . This pair also satisfies condition 2 above, since the penultimate vertices of Q_1 and Q_2 are the same as the penultimate vertices of P_2 and P_1 respectively. Also, u is the node in $Q_1 \cap Q_2$ maximizing $\text{dist}(u, v)$, so we can perform the same subpath swapping operation as above to go from $\langle Q_1, Q_2 \rangle$ to $\langle P_1, P_2 \rangle$.

Then by the discussion in the previous paragraph, [Lemma 15](#) implies that the enumerating polynomial for $\mathcal{F}$ is the same as the enumerating polynomial for $\mathcal{F} \setminus \mathcal{S} = \mathcal{D}(v)$. The enumerating polynomial for $\mathcal{D}(v)$ is $D(v)$, so this proves the desired result. $\blacksquare$

Lemma 21 (Enumerating Relaxed Linkages). For each vertex v , the polynomial

$$L_1(v)L_2(v) - \sum_{u \in V_{\text{in}}(v)} L_1(u)L_2(u)x_{uv}^2$$

enumerates $\tilde{\mathcal{D}}(v)$.

Proof. By expanding out the product, we see that the polynomial $L_1(v)L_2(v)$ enumerates all pairs of paths $\langle P_1, P_2 \rangle$ where each P_i is an (s_i, v) -path in G_i . We claim that

$$\sum_{u \in V_{\text{in}}(v)} L_1(u)L_2(u)x_{uv}^2 \tag{7}$$

enumerates all pairs of paths $\langle P_1, P_2 \rangle$ such that each P_i is a path in G_i beginning at s_i and ending at an edge (u, v) for some node $u \in V_{\text{in}}(v)$.

Indeed, paths P_1 and P_2 from any such pair can be split along their final edges into

$$P_1 = P_1[s_1, u] \diamond (u, v) \quad \text{and} \quad P_2 = P_2[s_2, u] \diamond (u, v).$$

The paths $P_i[s_i, u]$ are enumerated by the $L_i(u)$ factors in [Eq. \(7\)](#), and the two copies of (u, v) are encoded by the x_{uv}^2 factor in [Eq. \(7\)](#). Conversely, any monomial in the expansion of

$$L_1(u)L_2(u)x_{uv}^2 \tag{8}$$

is the product of monomials for some (s_i, u) -paths Q_i in G_i and two occurrences of the edge (u, v) , so that if we define

$$P_1 = Q_1 \diamond (u, v) \quad \text{and} \quad P_2 = Q_2 \diamond (u, v)$$

then the monomial we were considering is precisely the monomial for the pair $\langle P_1, P_2 \rangle$. Summing over all vertices $u \in V_{\text{in}}$ proves that [Eq. \(8\)](#) enumerates the pairs of paths we described above.

From the discussion above, it follows that

$$L_1(v)L_2(v) - \sum_{u \in V_{\text{in}}(v)} L_1(u)L_2(u)x_{uv}^2$$

enumerates all pairs of paths $\langle P_1, P_2 \rangle$, where P_i is an (s_i, v) -path in G_i , such that P_1 and P_2 have distinct penultimate vertices. Thus this polynomial enumerates $\tilde{D}(v)$ as desired. $\blacksquare$

Lemma 22 (Enumerating Source Linkages). For each vertex v , we have

$$D(v) = L_1(v)L_2(v) - \sum_{u \in V_{\text{in}}(v)} L_1(u)L_2(u)x_{uv}^2.$$

Proof. This result follows by combining [Lemmas 20](#) and [21](#). $\blacksquare$

4.3 2-DSP in DAGs

In this subsection, assume that G is a weighted DAG, and recall the definitions from [Section 4.1](#). We fix an arbitrary topological order on the vertices of G .

Our main goal is to efficiently evaluate a polynomial encoding pairs of disjoint shortest paths in G . By [Lemma 17](#), it suffices to evaluate a polynomial enumerating pairs of intersecting shortest paths in G . We do this by casework on the first intersection point of these paths.

Lemma 23 (Enumerating Intersecting Paths). We have

$$F_{\cap} = \sum_{v \in V} D(v)R_1(v)R_2(v).$$

Proof. For any fixed vertex v , we claim that

$$D(v)R_1(v)R_2(v) \tag{9}$$

enumerates all standard pairs of paths $\langle P_1, P_2 \rangle$ such that P_1 and P_2 have first intersection at v . Indeed, given any such pair of paths $\langle P_1, P_2 \rangle$, we can decompose

$$P_i = P_i[s_i, v] \diamond P_i[v, t_i]$$

and observe that the pair $\langle P_1[s_1, v], P_2[s_2, v] \rangle$ is enumerated by the $D(v)$ factor in [Eq. \(9\)](#), and the $P_i[v, t_i]$ paths are enumerated by the respective $R_i(v)$ factors from [Eq. \(9\)](#). Conversely, any monomial in the expansion of [Eq. \(9\)](#) is the product of monomials encoding the edges of a pair of paths $\langle A_1, A_2 \rangle$, where A_i is an (s_i, v) -path in G_i such that A_1 and A_2 only intersect at v , and paths B_i from v to t_i in G_i . Then if we define

$$P_i = A_i \diamond B_i$$

we see that the P_i are (s_i, t_i) -paths in G_i with the property that P_1 and P_2 first intersect at v . Here, we are using the fact that G is a DAG—this ensures that every node in A_1 or A_2 appears at or before v in the topological order, and that every node in B_1 or B_2 appears at or after v in the topological order, so that $A_1 \cap B_2 = A_2 \cap B_1 = \{v\}$.

Since every pair of intersecting paths intersects at a unique earliest vertex, it follows that

$$\sum_{v \in V} D(v)R_1(v)R_2(v)$$

enumerates all intersecting, standard pairs of paths $\langle P_1, P_2 \rangle$. This proves the desired result. $\blacksquare$

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

Proof. Each pair of internally vertex-disjoint paths produces a distinct monomial in F_{disj} . It follows that disjoint (s_i, t_i) -shortest paths exist in G if and only if F_{disj} is nonzero as a polynomial.

So we can solve 2-DSP as follows. We assign each x_{uv} variable an independent, uniform random element of $\mathbb{F}_{2^q}$, and then evaluate F_{disj} on this assignment. If the evaluation is nonzero we return YES (two disjoint shortest paths exist), and otherwise we return NO.

If two disjoint shortest paths do not exist, then F_{disj} is the zero polynomial, and our algorithm correctly returns NO. If two disjoint shortest paths do exist, then F_{disj} is a nonzero polynomial of degree strictly less than $2n$, so by Schwartz-Zippel (Proposition 12) our algorithm correctly returns YES with high probability for large enough $q = O(\log n)$.

It remains to show that we can compute F_{disj} in linear time.

First, we can compute G_1 and G_2 in linear time by Proposition 11.

Then, by dynamic programming forwards and backwards over the topological order of G , we can evaluate the polynomials $L_i(v)$ and $R_i(v)$ for each $i \in [2]$ and vertex v at our given assignment in linear time, using the recurrences from Lemma 16.

Having computed these values, Lemma 22 shows that for any vertex v we can compute $D(v)$ at the given assignment in $O(\deg_{\text{in}}(v))$ time. So we can evaluate $D(v)$ for all v in $O(m)$ time.

Given the above evaluations, we can compute $F_{\cap}$ at the given point in $O(n)$ time by Lemma 23.

Finally, given the value of $F_{\cap}$, we can evaluate F_{disj} in $O(1)$ additional time by Lemma 17.

Thus we can solve 2-DSP in linear time. ■

4.4 2-DSP in Undirected Graphs

In this subsection, we assume that G is a weighted undirected graph.

To solve 2-DSP in G , we will show how to efficiently evaluate a polynomial enumerating pairs of disjoint shortest paths in G . To help construct this polynomial, it will first be helpful to characterize the ways in which two shortest paths can intersect in an undirected graph.

Structure of Shortest Paths

We begin with the following simple observation about shortest paths in undirected graphs.

Proposition 24 (Shortest Path Orderings). Let G be an undirected graph. Suppose vertices a, b, c appear in that order on some shortest path of G . Then on any shortest path in G passing through these three nodes, b appears between a and c .

Proof. Since some shortest path in G passes through vertices a, b, c in that order, we know that $\text{dist}(a, b)$ and $\text{dist}(b, c)$ are both less than $\text{dist}(a, c)$.

Now, consider any shortest path in G which passes through these three vertices. If a appears between b and c in this shortest path, then

$$\text{dist}(a, c) < \text{dist}(b, c)$$

which contradicts the observation from the first sentence.

Similarly, if c appears between a and b in this shortest path, then

$$\text{dist}(a, c) < \text{dist}(a, b)$$

which again contradicts the observation from the first sentence of this proof.

Thus b must appear between a and c on the shortest path as claimed. ■

Definition 25 (Intersection Types). Let P_1 and P_2 be intersecting shortest paths in an undirected graph. Let v be the first vertex in P_1 appearing in $P_1 \cap P_2$.

- If $P_1 \cap P_2 = \{v\}$, we say P_1 and P_2 have *single intersection*.
- If $|P_1 \cap P_2| \geq 2$ and v is also the first vertex of P_2 in $P_1 \cap P_2$, we say P_1 and P_2 *agree*.
- If $|P_1 \cap P_2| \geq 2$ and v is the last vertex of P_2 in $P_1 \cap P_2$, we say P_1 and P_2 are *reversing*.

If P_1 and P_2 do not agree, we say they *disagree*.

In other words, P_1 and P_2 disagree if the first vertex v of P_1 appearing in $P_1 \cap P_2$ is also the last vertex of P_2 appearing in $P_1 \cap P_2$. Equivalently, paths P_1 and P_2 disagree if they have single intersection or are reversing.

Lemma 26 (Intersection Types are Exhaustive). Let P_1 and P_2 be intersecting shortest paths in an undirected graph. Then either P_1 and P_2 agree, are reversing, or have single intersection.

Proof. If $|P_1 \cap P_2| = 1$, then the paths have single intersection.

Otherwise, $|P_1 \cap P_2| \geq 2$. Let v be the first vertex in P_1 appearing in $P_1 \cap P_2$. It suffices to show v is either the first or last vertex in P_2 appearing in $P_1 \cap P_2$.

Suppose to the contrary this is not the case. Then let u and w be the first and last vertices respectively in path P_2 appearing in $P_1 \cap P_2$. By assumption, u, v, w are all distinct. By definition, u, v, w appear in that order on P_2 . Since P_2 is a shortest path, by [Proposition 24](#), v must appear between u and w on P_1 as well. This contradicts the definition of v as the first vertex on P_1 lying in the intersection $P_1 \cap P_2$.

Thus our original supposition was false, and the paths P_1 and P_2 must agree or be reversing whenever $|P_1 \cap P_2| \geq 2$, which proves the desired result. $\blacksquare$

Agreeing and Disagreeing Polynomials

Let F_{agree} be the enumerating polynomial for the collection of standard pairs of paths $\langle P_1, P_2 \rangle$ which agree. Let F_{dis} be the enumerating polynomial for the collection of standard pairs of paths $\langle P_1, P_2 \rangle$ which disagree. Any intersecting paths either agree or disagree, so

$$F_{\cap} = F_{\text{agree}} + F_{\text{dis}}. \quad (10)$$

Thus, to compute $F_{\cap}$, it suffices to compute F_{agree} and F_{dis} separately.

Agreeing Paths

We first show how to evaluate F_{agree} efficiently.

Lemma 27 (Common Final Intersection). If paths P_1 and P_2 agree, then they have a common last intersection point, distinct from their common first intersection point.

Proof. Since P_1 and P_2 agree, they have a common first intersection point at some vertex v . Then the vertex $w \in P_1 \cap P_2$ which maximizes $\text{dist}(v, w)$ must be the last node on both paths in $P_1 \cap P_2$. Since the paths agree, we have $|P_1 \cap P_2| \geq 2$, so $w \neq v$. $\blacksquare$

We say a pair of paths $\langle P_1, P_2 \rangle$ is *edge-agreeing* if P_i is an (s_i, t_i) -path in G_i , and P_1 and P_2 traverse a common edge in the same direction.

Lemma 28 (Edge-Agreeing $\subseteq$ Agreeing). Any edge-agreeing pair is agreeing.

Proof. Suppose pair $\langle P_1, P_2 \rangle$ is edge-agreeing.

Let $e = (a, b)$ be a common edge traversed by both P_1 and P_2 .

Since $\{a, b\} \subseteq P_1 \cap P_2$, we have $|P_1 \cap P_2| \geq 2$.

So by Lemma 26, P_1 and P_2 are either agreeing or reversing.

Suppose to the contrary that P_1 and P_2 are reversing. Let v be the first vertex on P_1 in $P_1 \cap P_2$. Then $v \neq b$, since a appears before b on P_1 .

Since the paths are reversing, v is the final vertex on P_2 in $P_1 \cap P_2$. Then $v \neq a$, since b appears after a on P_2 . Path P_1 passes through v, a, b in that order. Then by Proposition 24, a must appear between v and b on any shortest path containing these three vertices. However, this contradicts the fact that b is between a and v on P_2 .

Thus P_1 and P_2 are not reversing, and so must agree as claimed. $\blacksquare$

Lemma 29. The polynomial F_{agree} enumerates the collection of edge-agreeing paths in G .

Proof. By Lemma 28, the set of pairs F_{agree} enumerates includes all edge-agreeing pairs.

Let $\mathcal{F}$ be the family of standard pairs of paths $\langle P_1, P_2 \rangle$ such that P_1 and P_2 agree.

By definition, F_{agree} enumerates $\mathcal{F}$.

By Lemma 28, every edge-agreeing standard pair of paths is in $\mathcal{F}$.

Let $\mathcal{S}$ be the collection of pairs which are agreeing but not edge-agreeing. To prove the desired result, it suffices to show that the monomials corresponding to pairs in $\mathcal{S}$ have net zero contribution to F_{agree} over a field of characteristic two.

To that end, take any pair $\langle P_1, P_2 \rangle \in \mathcal{S}$. Since $\langle P_1, P_2 \rangle$ is agreeing, P_1 and P_2 have a unique first intersection point v . By Lemma 27, these paths also have a unique last intersection point $w \neq v$.

Hence, we can decompose the paths into

$$P_1 = P_1[s_1, v] \diamond P_1[v, w] \diamond P_1[w, t_1] \quad \text{and} \quad P_2 = P_2[s_2, v] \diamond P_2[v, w] \diamond P_2[w, t_2].$$

Define the paths

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

by swapping the v to w subpaths of P_1 and P_2 .

By Lemma 14 the Q_i are (s_i, t_i) -shortest paths.

Since P_1 and P_2 are not edge-agreeing, the subpaths $P_1[v, w]$ and $P_2[v, w]$ are distinct.

The paths Q_1 and Q_2 have a common first intersection point of v , and so the pair $\langle Q_1, Q_2 \rangle$ is agreeing. Since $\langle P_1, P_2 \rangle$ is not edge-agreeing, neither is $\langle Q_1, Q_2 \rangle$. Thus $\langle Q_1, Q_2 \rangle \in \mathcal{S}$. Finally, if we swap the v to w subpaths of Q_1 and Q_2 , we recover P_1 and P_2 .

The above discussion implies that the map Φ from $\langle P_1, P_2 \rangle$ to $\langle Q_1, Q_2 \rangle$ described above satisfies all conditions of Lemma 15, and so F_{agree} is the enumerating polynomial for $\mathcal{F} \setminus \mathcal{S}$, which is precisely the set of edge-agreeing pairs. $\blacksquare$

Lemma 30 (Enumerating Agreeing Pairs). We have

$$F_{\text{agree}} = \sum_{v \in V} \sum_{w \in V_{\text{out}}(v)} (D(v)x_{vw}^2) R_1(w)R_2(w).$$

Proof. Let $\mathcal{F}$ be the family of edge-agreeing pairs. By Lemma 29, it suffices to show that

$$\sum_{v \in V} \sum_{w \in V_{\text{out}}(v)} (D(v)x_{vw}^2) R_1(w)R_2(w)$$

is the enumerating polynomial for $\mathcal{F}$. To that end, the following claim about the individual terms of the above sum will be useful.

Claim 31. For any choice of vertices v and w with $w \in V_{\text{out}}(v)$, the polynomial

$$(D(v)x_{vw}^2) R_1(w)R_2(w) \quad (11)$$

enumerates all standard pairs of paths $\langle P_1, P_2 \rangle$ such that P_1 and P_2 overlap at edge $e = (v, w)$, and have common first intersection at vertex v .

Proof. Take any pair $\langle P_1, P_2 \rangle$ satisfying the conditions from the statement of the claim. Then we can decompose these paths in the form

$$P_i = P_i[s_i, v] \diamond (v, w) \diamond P_i[w, t_i]$$

such that the $P_i[s_i, v]$ subpaths intersect only at v .

This means that the pair $\langle P_1[s_1, v], P_2[s_2, v] \rangle$ is enumerated by $D(v)$, the two edges (v, w) are enumerated by x_{vw}^2 , and each path $P_i[w, t_i]$ is enumerated by $R_i(w)$, so that the expansion of the polynomial Eq. (11) includes some monomial corresponding to the given pair.

Conversely, any monomial in the expansion of Eq. (11) is the product of monomials recording the edges traversed by a pair of paths $\langle A_1, A_2 \rangle$ only intersecting at node v , where A_i is an (s_i, v) -path in G_i , two copies of the edge (v, w) , and some (w, t_i) -paths B_i in G_i . This product is equal to the monomial given by

$$f(A_1 \diamond (v, w) \diamond B_1, A_2 \diamond (v, w) \diamond B_2).$$

Define the paths $P_i = A_i \diamond (v, w) \diamond B_i$ for each $i \in [2]$. Since A_i and B_i are paths in G_i , and (v, w) is an edge in both G_1 and G_2 , we know that each P_i is an (s_i, t_i) -shortest path.

We claim that A_1 does not intersect B_2 . Indeed, suppose to the contrary that A_1 and B_2 intersect at some node u . Then P_1 is a shortest path which passes through nodes u, v, w in that order, yet P_2 is a shortest path which passes through v, w, u in that order, which contradicts Proposition 24.

Hence A_1 does not intersect B_2 . Symmetric reasoning shows that A_2 does not intersect B_1 .

Thus the paths P_1 and P_2 have common first intersection at v . Then the pair $\langle P_1, P_2 \rangle$ satisfies the conditions from the claim statement, so the polynomial from Eq. (11) enumerates all pairs of paths described in the claim. $\square$

By Claim 31, the sum

$$\sum_{v \in V} \sum_{w \in V_{\text{out}}(v)} (D(v)x_{vw}^2) R_1(w)R_2(w) \quad (12)$$

enumerates all edge-agreeing pairs whose first intersection point v is the beginning of an edge traversed by both paths in the pair. Let $\mathcal{H}$ be the set of such pairs, and let $\mathcal{S} = \mathcal{F} \setminus \mathcal{H}$.

We claim that the set of pairs in $\mathcal{S}$ has net zero contribution to F_{agree} .

Indeed, take any pair $\langle P_1, P_2 \rangle \in \mathcal{S}$. Since the pair is edge-agreeing, by Lemma 28 the pair is agreeing. Hence P_1 and P_2 have a common first intersection at some node v . By Lemma 27, these paths also have a common last intersection point at some node $w \neq v$. Since the pair is not in $\mathcal{S}$, the subpaths $P_1[v, w]$ and $P_2[v, w]$ are distinct. If we swap these subpaths to produce paths

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

then by Lemma 14 each Q_i is still an (s_i, t_i) -path in G_i . Since the P_i are edge-agreeing, and this edge-overlap must occur in their v to w subpaths, the Q_i are also edge-agreeing. By assumption,

v is not the beginning of a common edge traversed by the P_i , so it is also not such a beginning for Q_i . These observations combined show that $\langle Q_1, Q_2 \rangle \in \mathcal{S}$. Also, swapping v to w subpaths in the Q_i recover the P_i paths. It follows that the map Φ sending $\langle P_1, P_2 \rangle$ to $\langle Q_1, Q_2 \rangle$ satisfies all the conditions of [Lemma 15](#), so in fact F_{agree} enumerates $\mathcal{F} \setminus \mathcal{S} = \mathcal{H}$.

Since [Eq. \(12\)](#) also enumerates $\mathcal{H}$, this proves the desired result. $\blacksquare$

Disagreeing Paths

Let $\mathcal{F}_{\text{dis}}$ be the family of disagreeing pairs of paths. Recall that F_{dis} enumerates $\mathcal{F}_{\text{dis}}$. Our goal is to show how to efficiently evaluate F_{dis} . To do this, we describe a subfamily of pairs in $\mathcal{F}_{\text{dis}}$, and argue that F_{dis} enumerates this subfamily.

Definition 32. For each node v , let $\tilde{\mathcal{B}}(v)$ be the set of standard pairs of paths $\langle P_1, P_2 \rangle$ intersecting at v , such that if we let a_i and b_i denote the nodes appearing immediately before and after v on P_i respectively, then

1. $a_1 \neq a_2$,
2. $b_1 \neq b_2$, and
3. $a_1 \neq b_2$.

Let $\mathcal{B}(v) \subseteq \tilde{\mathcal{B}}(v)$ be the subfamily of such pairs such that v is the first vertex in P_1 lying in $P_1 \cap P_2$. Then define the collection

$$\mathcal{B} = \bigcup_{v \in V} \mathcal{B}(v).$$

Next, we prove some lemmas, which will help us prove that to enumerate $\mathcal{F}_{\text{dis}}$, it suffices to design enumerating polynomials for $\tilde{\mathcal{B}}(v)$ for each vertex v .

Lemma 33. For each vertex v , the enumerating polynomial for $\tilde{\mathcal{B}}(v)$ enumerates $\mathcal{B}(v)$.

Proof. Fix a vertex v . Let $\mathcal{S} = \tilde{\mathcal{B}}(v) \setminus \mathcal{B}(v)$.

Take any pair $\langle P_1, P_2 \rangle \in \mathcal{S}$. Let u be the first vertex in P_1 lying in $P_1 \cap P_2$. Since the pair is not in $\mathcal{B}(v)$, we know that $u \neq v$.

There are two cases to consider, based off the relative positions of u and v on P_2 .

Case 1: u before v Suppose that u appears before v in P_2 . In this case, we form paths

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

by swapping the u to v subpaths in P_1 and P_2 . By [Lemma 14](#), each Q_i is still an (s_i, t_i) -path in G_i . After this subpath swap, the vertices immediately before v in the Q_i are still distinct, and the vertices immediately after v in the Q_i are still distinct. Moreover, the vertex immediately before v in Q_1 is not equal to the vertex immediately after v in Q_2 , since these are distinct nodes on P_2 . Then by [Definition 32](#), $\langle Q_1, Q_2 \rangle \in \tilde{\mathcal{B}}(v)$. Also, since Q_1 and Q_2 intersect at u , $\langle Q_1, Q_2 \rangle \notin \mathcal{B}(v)$. Finally, since u is the first vertex in Q_1 lying in $Q_1 \cap Q_2$, applying the same subpath swap procedure as above to $\langle Q_1, Q_2 \rangle$ recovers $\langle P_1, P_2 \rangle$.

So the map Φ sending $\langle P_1, P_2 \rangle$ to $\langle Q_1, Q_2 \rangle$ as above satisfies all conditions of [Lemma 15](#).

Case 2: u after v Suppose that u appears after v on P_2 . In this case, we form paths

$$Q_1 = P_1[s_1, u] \diamond \overleftarrow{P_2}[u, v] \diamond P_1[v, t_i] \quad \text{and} \quad Q_2 = P_2[s_1, v] \diamond \overleftarrow{P_1}[v, u] \diamond P_2[u, t_i]$$

by swapping the u to v subpaths in P_1 and P_2 (and using the fact that G is undirected, so we can traverse the edges in these subpaths backwards). Since the u to v subpaths of P_1 and P_2 both have length $\text{dist}(u, v)$, each path Q_i has the same length as P_i , and is thus still an (s_i, t_i) -path in G_i .

Let a_i and b_i denote the vertices on P_i immediately before and after v respectively.

Then the nodes immediately before and after v on Q_1 are b_2 and b_1 respectively, and the nodes immediately before and after v on Q_2 are a_2 and a_1 .

We know that $b_i \neq a_i$, since a_i and b_i are distinct vertices on the path P_i . Thus $\langle Q_1, Q_2 \rangle$ satisfies the first two conditions from [Definition 32](#).

Since $\langle P_1, P_2 \rangle \in \mathcal{S}$, by condition three of [Definition 32](#), we know that $b_2 \neq a_1$. Since b_2 is the node immediately before v in Q_1 and a_1 is the node immediately after v in Q_2 , we see that $\langle Q_1, Q_2 \rangle$ satisfies condition three of [Definition 32](#) as well.

Thus $\langle Q_1, Q_2 \rangle \in \tilde{\mathcal{B}}(v)$.

Furthermore, since $u \in Q_1 \cap Q_2$ appears before v in Q_1 , we have $\langle Q_1, Q_2 \rangle \notin \mathcal{B}(v)$.

Thus $\langle Q_1, Q_2 \rangle \in \mathcal{S}$.

This shows that the swapping procedure described above, sending $\langle P_1, P_2 \rangle$ to $\langle Q_1, Q_2 \rangle$, is a map from $\mathcal{S}$ to itself. Moreover, because u is the first vertex on P_1 lying in $P_1 \cap P_2$ and the first vertex on Q_1 lying in $Q_1 \cap Q_2$, performing the above swapping procedure on $\langle Q_1, Q_2 \rangle$ recovers $\langle P_1, P_2 \rangle$. Finally, since the multisets of edges traversed by $\langle P_1, P_2 \rangle$ and $\langle Q_1, Q_2 \rangle$ are the same, we have

$$f(P_1, P_2) = f(Q_1, Q_2).$$

The above discussion shows that $\mathcal{S}$ can be partitioned into groups of size two, with each group consisting of two pairs with the same monomial. Then the monomials from pairs in $\mathcal{S}$ have net zero contribution to the enumerating polynomial for $\tilde{\mathcal{B}}(v)$ modulo two.

Hence the enumerating polynomial for $\tilde{\mathcal{B}}(v)$ actually enumerates $\tilde{\mathcal{B}}(v) \setminus \mathcal{S} = \mathcal{B}(v)$, as claimed. $\blacksquare$

Lemma 34. The enumerating polynomial for $\mathcal{B}$ is F_{dis} .

Proof. First, observe that $\mathcal{F}_{\text{dis}} \subseteq \mathcal{B}$. Indeed, suppose $\langle P_1, P_2 \rangle \in \mathcal{F}_{\text{dis}}$ is a standard pair of paths which disagree. Let v be the first vertex of P_1 in $P_1 \cap P_2$. Let a_i and b_i be the vertices immediately before and after v in P_i . Then $a_1 \notin \{a_2, b_2\}$ by the definition of v , and $b_1 \neq b_2$ because P_1 and P_2 disagree. Thus $\langle P_1, P_2 \rangle$ satisfies all the conditions from [Definition 32](#), so this pair is in $\mathcal{B}$. Since we chose an arbitrary pair $\langle P_1, P_2 \rangle$ from $\mathcal{F}_{\text{dis}}$, we have $\mathcal{F}_{\text{dis}} \subseteq \mathcal{B}$ as claimed.

Let $\mathcal{S} = \mathcal{B} \setminus \mathcal{F}_{\text{dis}}$ be the set of agreeing pairs in $\mathcal{B}$.

Take any pair $\langle P_1, P_2 \rangle \in \mathcal{S}$. Let v be the first common intersection point of the pair. Let w be the last common intersection point of the pair. By [Lemma 27](#), $w \neq v$. For each $i \in [2]$, let

$$Q_i = P_i[s_i, v] \diamond P_i[v, w] \diamond P_i[w, t_i]$$

be the paths formed by swapping the v to w subpaths in P_1 and P_2 .

We claim that $\langle Q_1, Q_2 \rangle \in \mathcal{S}$ as well.

By [Lemma 14](#) each Q_i is still an (s_i, t_i) -path in G_i . The first and last intersection points of these paths are still v and w , so the new pair is still agreeing.

Let vertices a_i and b_i be vertices immediately before and after v on each P_i respectively. Since the a_i occur before v , the nodes before v on Q_1 and Q_2 are distinct. Since the nodes after v on Q_1 and Q_2 are $\{b_1, b_2\}$, these nodes are also distinct. Finally, the node before v on Q_1 is a_1 and the

node after v on Q_2 is b_1 , and $a_1 \neq b_1$ since a_1 and b_1 are distinct vertices on path P_1 , the node before v on Q_1 and after v on Q_2 are distinct.

The discussion in the previous paragraph shows that $\langle Q_1, Q_2 \rangle \in \mathcal{S}$.

Finally, if we apply the subpath swapping procedure above to $\langle Q_1, Q_2 \rangle$, we recover $\langle P_1, P_2 \rangle$. Then the map Φ on $\mathcal{S}$ sending $\langle P_1, P_2 \rangle$ to $\langle Q_1, Q_2 \rangle$ satisfies the conditions from the statement of [Lemma 15](#), so the enumerating polynomial for $\mathcal{B}$ in fact enumerates $\mathcal{B} \setminus \mathcal{S} = \mathcal{F}_{\text{dis}}$ as claimed. $\blacksquare$

By [Lemmas 33](#) and [34](#), enumerating $\mathcal{F}_{\text{dis}}$ reduces to enumerating $\tilde{\mathcal{B}}(v)$ for each vertex v . Our next goal is to perform this enumeration efficiently. To do this, we start by defining and establishing formulas for some additional helper polynomials.

Definition 35 (Relaxed Target Linkages). For each vertex v , let $T(v)$ be the enumerating polynomial for the set of pairs of paths $\langle P_1, P_2 \rangle$ where

1. each P_i is a (v, t_i) -path in G_i , and
2. the second nodes of P_1 and P_2 are distinct.

Lemma 36. For each vertex v , we have

$$T(v) = R_1(v)R_2(v) - \sum_{w \in V_{\text{out}}(v)} x_{vw}^2 R_1(w)R_2(w).$$

Proof. This follows from symmetric reasoning to the proof of [Lemma 21](#). $\blacksquare$

Definition 37. For each vertex v , let $H(v)$ be the enumerating polynomial for the set of standard pairs of paths $\langle P_1, P_2 \rangle$ intersecting at v , such that the vertex immediately before v on P_1 is the same as the vertex immediately after v on P_2 .

Intuitively, $H(v)$ is a polynomial we use to enforce the $a_1 \neq b_2$ condition from [Definition 32](#).

Lemma 38. For each vertex v , we have

$$H(v) = \sum_{u \in V_{\text{mix}}(v)} L_1(u)x_{uv}R_1(v)L_2(v)x_{vu}R_2(u).$$

Proof. For any pair of paths $\langle P_1, P_2 \rangle$ satisfying the conditions from [Definition 37](#), there exists a unique vertex u such that u appears immediately before v on P_1 and immediately after v on P_2 . Any such u must lie in $V_{\text{in}}^1(v) \cap V_{\text{out}}^2(v) = V_{\text{mix}}(v)$ by definition.

Given $v \in V$ and $u \in V_{\text{mix}}(v)$, let $\mathcal{F}(u, v)$ be the set of all standard pairs of paths $\langle P_1, P_2 \rangle$ such that P_1 traverses edge (u, v) and P_2 traverses edge (v, u) . The discussion in the previous paragraph implies that to prove the lemma, it suffices to show that the polynomial

$$L_1(u)x_{uv}R_1(v) \cdot L_2(v)x_{vu}R_2(u) \tag{13}$$

enumerates $\mathcal{F}(u, v)$ for all v and u .

To that end, let $\langle P_1, P_2 \rangle \in \mathcal{F}(u, v)$. We claim the monomial $f(P_1, P_2)$ appears in the expansion of [Eq. \(13\)](#). Indeed, by definition, we can split

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

Then path $P_1[s, u]$ is enumerated by $L_1(u)$, edge (u, v) is enumerated by x_{uv} , path $P_1[v, t_1]$ is enumerated by $R_1(v)$, path $P_2[s, v]$ is enumerated by $L_2(v)$, edge (v, u) is enumerated by x_{vu} , and path $P_2[u, t_2]$ is enumerated by $R_2(u)$, so the expansion of [Eq. \(13\)](#) has the term $f(P_1, P_2)$.

Conversely, we claim that any monomial in the expansion of Eq. (13) corresponds to a pair of paths in $\mathcal{F}(u, v)$. Indeed, any such monomial is the product of monomials from each of the factors in Eq. (13). By definition, $L_1(u)$ enumerates (s_1, u) -paths A_1 in G_1 , $R_1(v)$ enumerates (v, t_1) -paths B_1 in G_1 , $L_2(v)$ enumerates (s_2, v) -paths A_2 in G_2 , and $R_2(u)$ enumerates (u, t_2) -paths B_2 in G_2 . So an arbitrary monomial in Eq. (13) is of the form

$$f(A_1)x_{uv}f(B_1) \cdot f(A_2)x_{vu}f(B_2)$$

for A_i and B_i satisfying the conditions above.

Since $u \in V_{\text{mix}}(v)$, if we define paths

$$P_1 = A_1 \diamond (u, v) \diamond B_1 \quad \text{and} \quad P_2 = A_2 \diamond (v, u) \diamond B_2$$

then we see that P_i is an (s_i, t_i) -path in G_i . Moreover, P_1 traverses (u, v) and P_2 traverses (v, u) . So $\langle P_1, P_2 \rangle \in \mathcal{F}(u, v)$, and since

$$f(P_1, P_2) = f(P_1) \cdot f(P_2) = f(A_1)x_{uv}f(B_1) \cdot f(A_2)x_{vu}f(B_2)$$

we get that monomials of Eq. (13) correspond to pairs in $\mathcal{F}(u, v)$. Thus Eq. (13) enumerates $\mathcal{F}(u, v)$.

By the discussion at the beginning of the proof, this proves the claim. $\blacksquare$

Lemma 39 (Enumerating Disagreeing Pairs). We have

$$F_{\text{dis}} = \sum_{v \in V} (D(v)T(v) - H(v)).$$

Proof. By Lemma 34, $\mathcal{F}_{\text{dis}}$ is the enumerating polynomial for $\mathcal{B}$. By Definition 32, the enumerating polynomial for $\mathcal{B}$ is the sum of the enumerating polynomials for $\mathcal{B}(v)$ over all vertices v (here, we are using the fact that the $\mathcal{B}(v)$ are disjoint for each v by definition). Applying Lemma 33, we see this sum is equal to the sum of the enumerating polynomials for $\tilde{\mathcal{B}}(v)$ over all vertices v . So, it suffices to show that for each vertex v , the polynomial

$$D(v)T(v) - H(v) \tag{14}$$

enumerates $\tilde{\mathcal{B}}(v)$.

By Lemmas 20 and 21, the polynomial $D(v)$ enumerates all pairs $\langle A_1, A_2 \rangle$ where each A_i is an (s_i, v) -path in G_i , and the vertices a_i immediately before v in A_i are distinct. By Definition 35, the polynomial $T(v)$ enumerates all pairs $\langle B_1, B_2 \rangle$ where each B_i is a (v, t_i) -path in G_i , and the vertices b_i after v in B_i are distinct.

Then the product $D(v)T(v)$ enumerates all pairs of paths

$$\langle A_1 \diamond B_1, A_2 \diamond B_2 \rangle$$

for A_i and B_i satisfying the conditions from the previous paragraph. Note that this pair satisfies conditions 1 and 2 from Definition 32. Moreover, any standard pair of paths $\langle P_1, P_2 \rangle$ satisfying conditions 1 and 2 from Definition 32 can be decomposed into

$$P_i = A_i \diamond B_i$$

where A_i is an (s_i, v) -path in G_i , B_i is a (v, t_i) -path in G_i , the penultimate vertices of the A_i are distinct, and the second vertices of the B_i are distinct.

Thus $D(v)T(v)$ enumerates precisely all standard pairs of paths satisfying the first two conditions from [Definition 32](#).

By [Definition 37](#), $H(v)$ enumerates the standard pairs of paths $\langle P_1, P_2 \rangle$ intersecting at v , such that the node immediately before v on P_1 is the same as the node immediately after v on P_2 . If we let a_i and b_i again denote the nodes immediately before and after v on P_i respectively, this means we are enumerating all standard pairs of paths intersecting at v such that $a_1 = b_2$. Note that all such paths also have $a_1 \neq a_2$ (since a_2 and b_2 are distinct nodes in P_2) and $b_1 \neq b_2$ (since a_1 and b_1 are distinct nodes in P_1).

Thus $H(v)$ enumerates precisely the standard pairs of paths which satisfy the first two conditions but fail the third condition of [Definition 32](#).

Consequently, the difference [Eq. \(14\)](#) enumerates the standard pairs of paths which satisfy all conditions from [Definition 32](#). Thus the polynomial from [Eq. \(14\)](#) enumerates $\tilde{B}(v)$ as claimed, which proves the desired result. ■

We are finally ready to prove our main theorem. Note that the first part of the proof is completely identical to the proof of [Theorem 2](#), since our algorithms for 2-DSP in DAGs and undirected graphs have the same overall structure.

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

Proof. Each pair of internally vertex-disjoint paths produces a distinct monomial in F_{disj} . It follows that disjoint (s_i, t_i) -shortest paths exist in G if and only if F_{disj} is nonzero as a polynomial.

So we can solve 2-DSP as follows. We assign each x_{uv} variable an independent, uniform random element of $\mathbb{F}_{2^q}$, and then evaluate F_{disj} on this assignment. If the evaluation is nonzero we return YES (two disjoint shortest paths exist), and otherwise we return NO.

If two disjoint shortest paths do not exist, then F_{disj} is the zero polynomial, and our algorithm correctly returns NO. If two disjoint shortest paths do exist, then F_{disj} is a nonzero polynomial of degree strictly less than $2n$, so by Schwartz-Zippel ([Proposition 12](#)) our algorithm correctly returns YES with high probability for large enough $q = O(\log n)$.

It remains to show that we can compute F_{disj} in linear time.

First, we can compute G_1 and G_2 in linear time by [Proposition 11](#).

Then, by dynamic programming forwards and backwards over the topological order of G , we can evaluate the polynomials $L_i(v)$ and $R_i(v)$ for each $i \in \{1, 2\}$ and vertex v at our given assignment in linear time, using the recurrences from [Lemma 16](#).

Having computed these values, [Lemmas 22](#) and [36](#) shows that for any vertex v we can compute $D(v)$ and $T(v)$ at the given assignment in $O(\deg_{\text{in}}(v))$ and $O(\deg_{\text{out}}(v))$ time respectively. So we can evaluate $D(v)$ and $T(v)$ for all v in $O(m)$ time.

Then by [Lemma 30](#), we can evaluate F_{agree} in $O(m)$ time.

By [Lemma 38](#), we can compute $H(v)$ at any given vertex v in $O(\deg_{\text{in}}(v))$ time. So we can evaluate $H(v)$ for all vertices v in $O(m)$ time.

From the values for $D(v)$, $T(v)$, and $H(v)$, by [Lemma 39](#) we can compute F_{dis} in $O(n)$ time.

Having computed F_{agree} and F_{dis} , by [Eq. \(10\)](#) we can compute $F_{\cap}$ as well in $O(1)$ time.

Finally, given the value of $F_{\cap}$, we can evaluate F_{disj} in $O(1)$ additional time by [Lemma 17](#).

Thus we can solve 2-DSP in linear time. ■

4.5 Search to Decision Reduction

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

Proof. The proofs of [Theorems 1](#) and [2](#) construct arithmetic circuits of size $O(m)$ for the two disjoint shortest paths polynomial F_{disj} for DAGs and undirected graphs. By the Baur-Strassen theorem (see [\[BS83\]](#) for the original proof, and [\[SY09, Theorem 2.5\]](#) for a more recent exposition), we can construct arithmetic circuits of size $O(m)$ which simultaneously compute all single-order partial derivatives of F_{disj} .

Given an edge (u, v) , the polynomial $(\partial/\partial x_{uv}) F_{\text{disj}}$ is nonzero if and only if edge (u, v) appears in some solution to the 2-DSP problem. So by Schwartz-Zippel ([Proposition 12](#)), with high probability, $(\partial/\partial x_{uv}) F_{\text{disj}}$ has nonzero evaluation at a uniform random assignment over $\mathbb{F}_{2^q}$ if and only if (u, v) is an edge occurring in some pair of disjoint shortest paths, for $q = O(\log n)$ sufficiently large.

We can compute all partial derivatives at some random evaluation point in $O(m)$ time using the arithmetic circuit for these polynomials. We pick an edge (s_1, v) such that $(\partial/\partial x_{s_1 v}) F_{\text{disj}}$ has nonzero evaluation. Then we delete vertex s_1 from G , and consider a smaller instance of 2-DSP on the graph, where source s_1 is replaced with v . We can repeat this process on the new instance, to find the first edge on a (v, t_1) -shortest path which is disjoint from some (s_2, t_2) -shortest path. Repeating this process at most n times, we can recover an (s_1, t_1) -shortest path P_1 , which is disjoint from some (s_2, t_2) -shortest path.

At this point, we just delete all vertices of P_1 from the original graph G , find an (s_2, t_2) -shortest path P_2 in the resulting graph in linear time, and then return $\langle P_1, P_2 \rangle$ as our answer.

Overall, we compute at most n evaluations of arithmetic circuits of size $O(m)$, so the algorithm runs in $O(mn)$ time. ■

5 Edge-Disjoint Paths Algorithm

In this section, we present our algorithm for k -EDSP in weighted DAGs and prove [Theorem 5](#). The algorithm works by constructing a large graph on n^k nodes, whose paths correspond to collections of edge-disjoint paths in the original graph. As we previously mentioned in [Sections 1](#) and [3](#), our algorithm uses the same framework as that of [\[FW80\]](#).

Throughout this section, we let G be the input DAG on n nodes and m edges. For each $i \in [k]$, we let G_i denote the s_i -shortest paths DAG of G .

Fix a topological order $(\prec)$ of G .

We construct a graph G' on n^k nodes, whose paths encode k -tuples of paths in G .

The graph G' has a node for each k -tuple of vertices from G .

Given a node $\vec{v} = (v_1, \dots, v_k)$ in G' (where each v_i is a vertex in G), let $v = \text{early}(\vec{v})$ be the unique vertex v such that

- $v = v_i$ for some index $i \in [k]$, and
- $v \preceq v_j$ for all $j \in [k]$.

In other words, $\text{early}(\vec{v})$ is the earliest coordinate of $\vec{v}$ with respect to the topological order of G . Let $I(\vec{v})$ denote the set of indices $i \in [k]$ such that $v_i = \text{early}(\vec{v})$. Then we include an edge from node $\vec{v} = (v_1, \dots, v_k)$ to node $\vec{w} = (w_1, \dots, w_k)$ in G' precisely when

1. for all $i \in I(\vec{v})$, the pair (v_i, w_i) is an edge in G_i ,
2. the vertices w_i are pairwise distinct over all $i \in I(\vec{v})$, and
3. for all $j \notin I(\vec{v})$, we have $w_j = v_j$.

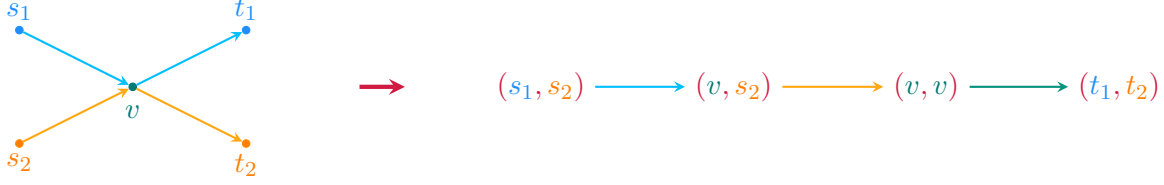


Figure 5: For $k = 2$, the paths $\langle s_1, v, t_1 \rangle$ and $\langle s_2, v, t_2 \rangle$ in G map to a single path in G' , pictured on the right, with respect to the topological order $s_1 \prec s_2 \prec v \prec t_1 \prec t_2$. Since both coordinates of the node (v, v) are equal, we can step from (v, v) to (t_1, t_2) with a single edge in G' .

We refer to the rules above as the *conditions for edges of G'* . Intuitively, the conditions say that we can move from a node $\vec{v}$ consisting of k vertices v_i by stepping from all of the earliest vertices to new distinct vertices. An example of these edge transitions is depicted in Figure 5.

Let $\vec{s} = (s_1, \dots, s_k)$ and $\vec{t} = (t_1, \dots, t_k)$ be nodes in G' containing all of the source and target vertices in G respectively. As the following lemmas show, our conditions for including edges in G' allow us to relate paths from $\vec{s}$ to $\vec{t}$ in G' to solutions to the k -EDSP problem in G .

Lemma 40. If G contains edge-disjoint (s_i, t_i) -shortest paths, then G' contains a path from $\vec{s}$ to $\vec{t}$.

Proof. For $i \in [k]$, let P_i be (s_i, t_i) -shortest paths in G which are all edge-disjoint. We show how to simultaneously traverse these paths P_i to recover a path in G' from $\vec{s}$ to $\vec{t}$.

Initialize $P \leftarrow \langle \vec{s} \rangle$ and $\vec{v} \leftarrow \vec{s}$.

Suppose $\vec{v} = (v_1, \dots, v_k)$. Define the new node $\vec{w} = (w_1, \dots, w_k)$ in G' by setting w_i to be the node after v_i on P_i for each $i \in I(\vec{v})$, and setting $w_j = v_j$ for all $j \notin I(\vec{v})$. Then append $\vec{w}$ to P , and update $\vec{v} \leftarrow \vec{w}$.

Repeat the process in the above paragraph until $\vec{v} = \vec{t}$. We claim this procedure halts and produces a path P from $\vec{s}$ to $\vec{t}$ in G' .

Indeed, an easy induction argument shows that the value of $\vec{v} = (v_1, \dots, v_k)$ always has the property that v_i is a vertex in P_i , and each step strictly decreases the value of

$$\sum_{i=1}^k \text{dist}(v_i, t_i).$$

The above quantity is a nonnegative integer, and so stops decreasing once it reaches zero, at which point we must have $\vec{v} = \vec{t}$. Moreover, our procedure is designed so that each step from a node $\vec{v}$ to a node $\vec{w}$ on P satisfies conditions 1 and 3 for edges in G' . Condition 2 for edges in G' is also satisfied, because the P_i are edge-disjoint, so P is a valid path in G' as desired. ■

Lemma 41. If G' contains a path from $\vec{s}$ to $\vec{t}$, then G contains edge-disjoint (s_i, t_i) -shortest paths.

Proof. Let P be a path from $\vec{s}$ to $\vec{t}$ in G' . We show how to recover edge-disjoint paths in G by reading off the edges traversed in P .

Initialize $\vec{v} \leftarrow \vec{s}$. For each $i \in [k]$, initialize $P_i \leftarrow \langle s_i \rangle$.

Suppose $\vec{v} = (v_1, \dots, v_k)$. Let $\vec{w} = (w_1, \dots, w_k)$ be the node after $\vec{v}$ on P . For each index i with $w_i \neq v_i$, append w_i to P_i . Then update $\vec{v} \leftarrow \vec{w}$.

Repeat the process in the above paragraph until $\vec{v} = \vec{t}$. We claim this procedure produces edge-disjoint shortest paths P_i from s_i to t_i .

Indeed, an easy induction argument shows that at any time in the procedure, the i^{th} coordinate of $\vec{v}$ is the last vertex in path P_i . Since $\vec{v}$ begins at $\vec{s}$ and ends at $\vec{t}$, and G is a DAG, each P_i is

an (s_i, t_i) -path. By condition 1 of edges in G' , any edge $(\vec{v}, \vec{w})$ in G' which changes the value of the i^{th} coordinate must change that coordinate by stepping along an edge lying in a shortest path from s_i . It follows that each P_i is an (s_i, t_i) -shortest path.

It remains to prove that the P_i are edge-disjoint. Suppose to the contrary that some of these paths overlap at some edge. Without loss of generality, suppose paths P_1 and P_2 overlap at some edge (a, b) .

Let $\vec{v} = (v_1, \dots, v_k)$ be the last node in P which has $v_1 = a$ as its first coordinate. Such a node exists since P_1 passes through a . By condition 3 for edges in G' , we must have $a = \text{early}(\vec{v})$. We claim that $v_2 = a$ as well.

Suppose $v_2 \neq a$. Then since $a = \text{early}(\vec{v})$, we must have $v_2 \succ a$.

In this case, define $\vec{u}$ to be the last node in P which has $u_2 = a$ as its second coordinate. Such a node exists since P_2 passes through a . By condition 3 for edges in G' , we have $a = \text{early}(\vec{u})$.

By the assumption that $v_2 \succ a$ and condition 1 for edges of G' , node $\vec{u}$ occurs before $\vec{v}$ in P . Then since $v_1 = a$, by condition 1 for edges of G' , the first coordinate u_1 of $\vec{u}$ must satisfy $u_1 \preceq a$. Since $a = \text{early}(\vec{u})$, this forces $u_1 = a = \text{early}(\vec{u})$.

Since $u_1 = \text{early}(\vec{u})$, by conditions 1 and 2 for edges in G' , all nodes in P after $\vec{u}$ cannot have first coordinate equal to a . This contradicts the fact that $\vec{u}$ occurs before $\vec{v}$ in P . So our initial assumption was false, and in fact $v_2 = a$ as claimed.

Let $\vec{w} = (w_1, \dots, w_k)$ be the node after $\vec{v}$ on P . Since $v_1 = v_2 = \text{early}(\vec{v})$, by condition 2 for edges of G' we have $w_1 \neq w_2$. Our procedure for constructing the paths P_i from P has P_1 traverse edge (a, w_1) and P_2 traverse edge (a, w_2) . Since w_1 and w_2 distinct, this contradicts the assumption that paths P_1 and P_2 both traverse edge (a, b) .

It follows that our original assumption was false, and the paths P_i are edge-disjoint as claimed. This completes the proof. $\blacksquare$

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

Proof. First, we compute a topological order of G in linear time. Then we compute the s_i -shortest paths DAGs G_i of G for all $i \in [k]$, which takes linear time by [Proposition 11](#).

By [Lemmas 40](#) and [41](#), we can solve k -EDSP on G by constructing the graph G' described in this section, and checking whether this graph contains a path from $\vec{s}$ to $\vec{t}$. Checking whether there is a path from $\vec{s}$ to $\vec{t}$ takes time linear in the size of G' , so to prove the theorem, it suffices to show that we can construct G' in $O(mn^{k-1})$ time.

We can construct all nodes of G' in $O(n^k)$ time. We then go through each node $\vec{v}$ in G , and add in edges from $\vec{v}$ to $\vec{w}$ in the out-neighborhood of $\vec{v}$ according to the three conditions. Each addition of an edge takes $O(1)$ time, so it suffices to show that G' has $O(mn^{k-1})$ edges.

Consider a node $\vec{v}$ in G . Let $v = \text{early}(\vec{v})$ be the earliest coordinate of $\vec{v}$, and let $\ell = |I(\vec{v})|$ denote the number of coordinates in $\vec{v}$ equal to this earliest vertex. There are at most $n^{k-\ell}$ choices for the coordinates of $\vec{v}$ not equal to v . By conditions 1 and 3 for edges in G' , the node $\vec{v}$ can have edges to at most $(\deg_{\text{out}}(v))^\ell$ nodes in G' .

Summing over all possible values for $\ell = |I(\vec{v})|$ and $v = \text{early}(\vec{v})$, we see that the number of edges in G' is bounded above by

$$\sum_{\ell=1}^k \sum_{v \in V} n^{k-\ell} (\deg_{\text{out}}(v))^\ell = \sum_{\ell=1}^k \sum_{v \in V} \left(n^{k-\ell} (\deg_{\text{out}}(v))^{\ell-1} \cdot \deg_{\text{out}}(v) \right).$$

Substituting the inequality $\deg_{\text{out}}(v) \leq n$ in the right hand side above, we see that the number of

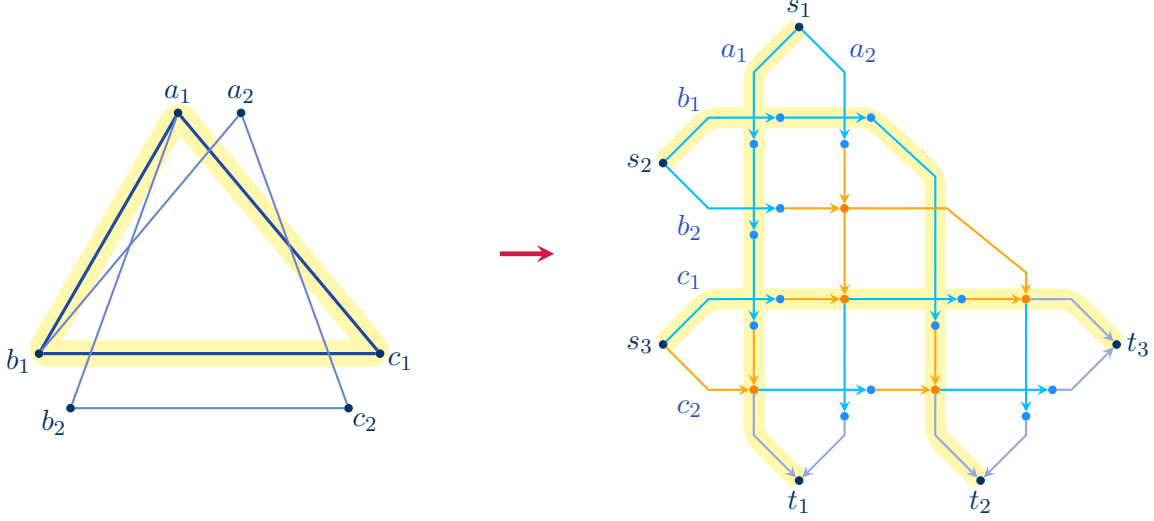


Figure 6: An example of the reduction from k -Clique to k -DSP for $k = 3$ and $n = 2$. The k -Clique instance G has vertex parts $V_1 = \{a_1, a_2\}$, $V_2 = \{b_1, b_2\}$, and $V_3 = \{c_1, c_2\}$. The unique triangle (a_1, b_1, c_1) in G on the left has highlighted bold edges. The vertices in this triangle are mapped to the disjoint highlighted paths in G' on the right. For vertices v, w in different parts of G , if (v, w) is an edge, the paths for v and w in G' pass by each other with blue edges (since $v(w) \neq w(v)$ in G'), and if (v, w) is not an edge, the paths for v and w intersect at the orange node $v(w) = w(v)$.

edges in G' is at most

$$\sum_{\ell=1}^k \sum_{v \in V} \left(n^{k-\ell} \cdot n^{\ell-1} \cdot \deg_{\text{out}}(v) \right) = kn^{k-1} \left(\sum_{v \in V} \deg_{\text{out}}(v) \right) = kmn^{k-1}.$$

For constant k , the above expression is $O(mn^{k-1})$, which proves the desired result. $\blacksquare$

6 Lower Bounds

6.1 Disjoint Shortest Paths

Our goal in this section is to prove the following theorem.

Theorem 7. 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.

Construction Let $G = V_1 \sqcup \dots \sqcup V_k$ be the input instance of k -Clique.

Order each of the vertices in each V_i . Order the set of all vertices V in the graph, by putting all vertices of V_i before vertices of V_j for $i < j$ (and within a part V_i using the order of V_i). For each index i , let $\overline{V}_i = V \setminus V_i$. The order on V induces an order on each $\overline{V}_i$ as well.

We now produce an instance G' of k -DSP, depicted in Figure 6.

For each $i \in [k]$, graph G' has a source node s_i and a target node t_i .

For each choice of vertices v, w from different parts, we introduce node $v(w)$ in G' .

For every vertex $v \in V_i$, the graph G' includes a path $P(v)$ from s_i to t_i , whose internal nodes are all the nodes of the form $v(w)$ where $w \in \overline{V}_i$, traversed according to the order of $\overline{V}_i$.

Finally, for each pair (v, w) which is not an edge in G , we identify the nodes $v(w) = w(v)$ in G' . This completes the construction of G' .

Correctness To prove our reduction is correct, we characterize shortest paths in G' .

Lemma 42. The (s_i, t_i) -shortest paths in G' are precisely the $P(v)$ paths for vertices $v \in V_i$.

Proof. Fix $i \in [k]$.

Given a vertex $w \in \overline{V}_i$, let $\text{pos}(w)$ denote the position of w in $\overline{V}$ from the end of this list. For example, if w is the final vertex in $\overline{V}_i$ we set $\text{pos}(w) = 1$, and if w is the first vertex in $\overline{V}_i$, we set $\text{pos}(w) = |\overline{V}_i| = (k-1)n$.

Given nodes a and b in G' , in this proof we let $\text{dist}(a, b)$ denote the distance from a to b in G' .

Claim 43. For every $v \in V_i$ and $w \in \overline{V}_i$, we have

$$\text{dist}(v(w), t_i) \geq \text{pos}(w) \quad \text{and} \quad \text{dist}(w(v), t_i) \geq \text{pos}(w). \quad (15)$$

Proof. We prove the inequalities by induction on the value of $\text{pos}(w)$.

For the base case, suppose $\text{pos}(w) = 1$. Then $v(w) \neq t_i$ and $w(v) \neq t_i$, so the distance from these nodes to t_i is at least 1 as claimed.

For the inductive step, suppose now that $\text{pos}(w) \geq 2$, and we have already proven the claim for all vertices occurring after w in $\overline{V}_i$.

The only edges exiting $v(w)$ and $w(v)$ come from the paths $P(v)$ and $P(w)$. We consider these two cases separately below.

Suppose we take a path from $v(w)$ or $w(v)$ which begins with an edge from $P(v)$. Let w' be the node after w in $\overline{V}_i$. Then $v(w')$ is the second vertex in this path. By the induction hypothesis,

$$\text{dist}(v(w'), t_i) \geq \text{pos}(w') = \text{pos}(w) - 1.$$

By the above equation, any path from $v(w)$ or $w(v)$ to t_i in this case has length at least

$$\text{dist}(v(w'), t_i) + 1 \geq \text{pos}(w).$$

Suppose now instead that we take a path from $v(w)$ or $w(v)$ to t_i which begins with an edge from $P(w)$. No such path exists if v is the final vertex in V_i , so we may assume that v is not the final vertex in V_i . Let v' be the vertex after v in V_i . Then $w(v')$ is the second vertex in our path. By the induction hypothesis,

$$\text{dist}(w(v'), t_i) \geq \text{pos}(w)$$

which implies that the path in this case has length at least $\text{pos}(w)$ as well.

So in either case, [Eq. \(15\)](#) holds. This completes the induction, and proves the desired result. $\square$

By [Claim 43](#), for any $v \in V_i$ and $w \in \overline{V}_i$ we have

$$\text{dist}(v(w), t_i) \geq \text{pos}(w).$$

The $v(w)$ to t_i subpath of $P(v)$ shows that $\text{dist}(v(w), t_i) \leq \text{pos}(w)$, so in fact we have

$$\text{dist}(v(w), t_i) = \text{pos}(w).$$

Now, every out-neighbor of s_i is of the form $v(w^*)$, where $v \in V_i$ and w^* is the first node in $\overline{V}_i$. Then by the above discussion,

$$\text{dist}(s_i, t_i) = |\overline{V}_i| + 1. \quad (16)$$

This immediately implies that for all $v \in V_i$, each $P(v)$ is an (s_i, t_i) -shortest path.

It remains to show that these are the only (s_i, t_i) -shortest paths in G' .

To that end, let Q be an (s_i, t_i) -shortest path in G' . Then the second vertex in Q is of the form $v(w^*)$, for $v \in V_i$ and w^* defined as above. Let w be the last vertex in $\overline{V}_i$ with the property that the s_i to $v(w)$ subpaths of Q and $P(v)$ agree.

Suppose w is not the final vertex in $\overline{V}_i$. In case, immediately after $v(w)$, path Q must traverse an edge of $P(w)$.

If v is the final vertex in V_i , then this edge brings Q to a node of the form $w(v')$, where $v' \in V_j$ for some $j > i$. However, there is no path from such a node $w(v')$ to t_i . Since we assumed Q is a path to t_i , this case cannot occur.

So v is not the final vertex in V_i . Let v' be the vertex after v in V_i .

Then using the edge of $P(w)$, Q goes from node $v(w)$ to $w(v')$. By [Claim 43](#), we know that

$$\text{dist}(w(v'), t_i) \geq \text{pos}(w).$$

So the $w(v')$ to t_i subpath of Q has length at least $\text{pos}(w)$.

However, we know that $P(v)$ and Q agree up to $v(w)$, so the s_i to $v(w)$ subpath of Q has length

$$|\overline{V}_i| - \text{pos}(w) + 1.$$

The path Q consists of these two subpaths and the edge from $v(w)$ to $w(v')$.

Thus, Q has length at least

$$(|\overline{V}_i| - \text{pos}(w) + 1) + 1 + \text{pos}(w) = |\overline{V}_i| + 2$$

which is greater than $\text{dist}(s_i, t_i)$ by [Eq. \(16\)](#).

This violates the definition of Q as a shortest path. So w must be the final vertex in $\overline{V}_i$, which forces $Q = P(v)$. So all (s_i, t_i) -shortest paths in G' are of the desired form. $\blacksquare$

We are now ready to prove our lower bound for k -DSP.

Theorem 7. 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.

Proof. Let $G = V_1 \sqcup \dots \sqcup V_k$ instance of k -Clique. Construct the graph G' defined in this section in $O((kn)^2)$ time. By definition, G' has $O((kn)^2)$ nodes.

We claim that G contains a k -clique if and only if G' contains disjoint (s_i, t_i) -shortest paths.

Indeed, suppose G contains a clique of the form $(v_1, \dots, v_k) \in V_1 \times \dots \times V_k$.

Take the $P(v_i)$ paths. By [Lemma 42](#) these are (s_i, t_i) -shortest paths.

The internal nodes of $P(v_i)$ are of the form $v_i(w)$, for $w \in \overline{V}_i$.

Thus the only way paths $P(v_i)$ and $P(v_j)$ could intersect for $i \neq j$ is if $v_i(v_j) = v_j(v_i)$ in G' . However, vertices v_i and v_j belong to a clique, which means (v_i, v_j) is an edge in G , so $v_i(v_j) \neq v_j(v_i)$.

Hence the $P(v_i)$ are vertex-disjoint paths.

Conversely, suppose G' contains disjoint (s_i, t_i) -shortest paths. By [Lemma 42](#), these paths are of the form $P(v_i)$ for some vertices $v_i \in V_i$. Since these paths are vertex-disjoint, we must have $v_i(v_j) \neq v_j(v_i)$ for all $i \neq j$. This means that (v_i, v_j) are edges in G for all $i \neq j$.

Thus $(v_1, \dots, v_k)$ forms a clique in G , which proves the desired result. $\blacksquare$

6.2 Disjoint Paths

Our goal in this section is to prove the following theorem.

Theorem 9. 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.

To establish these reductions, we first introduce the notion of a *covering family* of increasing lists, and then apply the construction of such a family to the reduction framework of [Sli10].

Covering Pairs with Lists

Definition 44 (Covering by Increasing Lists). Given a positive integer k , we say a collection of lists $\mathcal{L}$ of increasing integers is a *k -covering family* if for all integers i, j with $1 \leq i < j \leq k$, the integers i and j appear as consecutive members of some list in $\mathcal{L}$. We let $\lambda(k)$ denote the minimum possible number of lists in a k -covering family.

The following lemma lower bounds the minimum possible size of a k -covering family.

Lemma 45. We have $\lambda(k) \geq \lfloor k^2/4 \rfloor$.

Proof. Partition the set of the first k positive integers into sets

$$[k] = A \sqcup B$$

of consecutive positive integers

$$A = [1, \lfloor k/2 \rfloor] \quad \text{and} \quad B = [\lfloor k/2 \rfloor + 1, k]$$

of size $\lfloor k/2 \rfloor$ and $\lceil k/2 \rceil$ respectively. We claim that it cannot be the case that for $a, a' \in A$ and $b, b' \in B$ both (a, b) and (a', b') appear consecutively in the same list L . This is because L must be increasing, so both a and a' must appear before both b and b' in L . This means that whichever of a or a' appears first, cannot appear consecutively with b or b' .

It follows that the number of lists needed to cover all pairs $(i, j) \in [k]^2$ with $i < j$ is at least

$$|A||B| = \lfloor k/2 \rfloor \lceil k/2 \rceil = \lfloor k^2/4 \rfloor$$

as claimed. ■

In light of Lemma 45, the following lemma shows that we can efficiently construct a minimum size k -covering family (and that $\lambda(k) = \lfloor k^2/4 \rfloor$). This construction will be helpful in designing a reduction from k -Clique. The proof of this lemma is due to an anonymous reviewer.

Lemma 46. There is an algorithm which, given a positive integer k , runs in $O(k^3)$ time and constructs a k -covering family $\mathcal{L}$ of $\lfloor k^2/4 \rfloor$ lists with elements from $[k]$, with the property that each integer in $[k]$ appears in fewer than k lists of $\mathcal{L}$.

Proof. For each pair (a, d) of positive integers with $d \leq k-1$ and $a \leq \min(d, k-d)$, define $L_{a,d}$ to be the list of positive integers in $[k]$ whose $(j+1)^{\text{st}}$ term is $(a+jd)$ for each $j \geq 0$. Let $\mathcal{L}$ be the collection of all the $L_{a,d}$ lists, for the pairs (a, d) satisfying the aforementioned conditions.

We claim that any integer $x \in [k]$ shows up in fewer than k lists of $\mathcal{L}$. This is because for any $d \in [k-1]$, we have $x \in L_{a,d}$ only if $a \equiv x \pmod{d}$. Since $a \in [d]$, this means that for each d , there

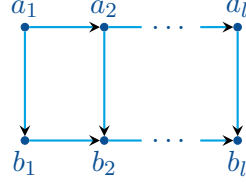


Figure 7: Given a vertex $v \in V_i$ from G , the gadget graph G_v has top nodes $a_j = a_j(v)$ and bottom nodes $b_j = b_j(v)$ for each $j \in [l]$, where $l = l(i)$ is the number of lists in $\mathcal{L}$ containing i .

is at most one a for which $x \in L_{a,d}$. Since $d \in [k-1]$ takes on fewer than k values, x appears in fewer than k lists of $\mathcal{L}$.

We now show that $\mathcal{L}$ is a k -covering family. Take arbitrary integers $i, j \in [k]$ with $i < j$.

Set $d = j - i$, and take a to be the smallest positive integer with $a \equiv i \pmod{d}$.

By construction, $a \leq d$. Moreover, we have $a + d \leq i + d = j \leq k$, so $a \leq k - d$ as well. So the pair (a, d) corresponds to a list $L_{a,d}$. Then i and j are both in $L_{a,d}$ because $j \equiv i \equiv a \pmod{d}$. Finally, because $d = j - i$, we know that i and j are consecutive elements in $L_{a,d}$. So $\mathcal{L}$ is a k -covering family as claimed.

The number of lists in $\mathcal{L}$ is

$$\sum_{d=1}^{k-1} \min(d, k-d) = \sum_{d=1}^{\lfloor k/2 \rfloor} d + \sum_{d=\lfloor k/2 \rfloor + 1}^{k-1} (k-d). \quad (17)$$

We now perform casework on the parity of k .

Case 1: k is even

If k is even, we can write $k = 2\ell$ for some positive integer ℓ . Then Eq. (17) simplifies to

$$(1 + 2 + \dots + \ell) + (1 + 2 + \dots + \ell - 1) = \frac{\ell(\ell+1)}{2} + \frac{\ell(\ell-1)}{2} = \ell^2 = \lfloor k^2/4 \rfloor.$$

Case 2: k is odd

If k is odd, we can write $k = 2\ell + 1$ for some positive integer ℓ . Then Eq. (17) simplifies to

$$(1 + 2 + \dots + \ell) + (1 + 2 + \dots + \ell) = \ell^2 + \ell = \lfloor k^2/4 \rfloor.$$

In either case, we get that the number of lists in $\mathcal{L}$ is $\lfloor k^2/4 \rfloor$ as claimed.

We can easily construct $\mathcal{L}$ in $O(k^3)$ time by going through each of the $O(k^2)$ pairs (a, d) defining a list $L_{a,d}$ and then listing its at most $O(k)$ elements by starting at a , and incrementing by d until we reach an integer greater than k . This completes the proof of the lemma. $\blacksquare$

The Reduction

Let $G = V_1 \sqcup \dots \sqcup V_k$ be an instance of k -Clique on kn vertices.

We assume $n \geq 2$, since otherwise the problem is trivial.

Set $\lambda = \lfloor k^2/4 \rfloor$. By Lemma 46, there exist a collection $\mathcal{L}$ of λ increasing lists of integers from $[k]$, such that for every choice of $i, j \in [k]$ with $i < j$, the integers i and j appear as consecutive elements in some list of $\mathcal{L}$. Moreover, we can construct $\mathcal{L}$ in $O(1)$ time for constant k . We give this collection of lists some arbitrary order.

Given $i \in [k]$, let $l(i)$ denote the number of lists in $\mathcal{L}$ containing i .

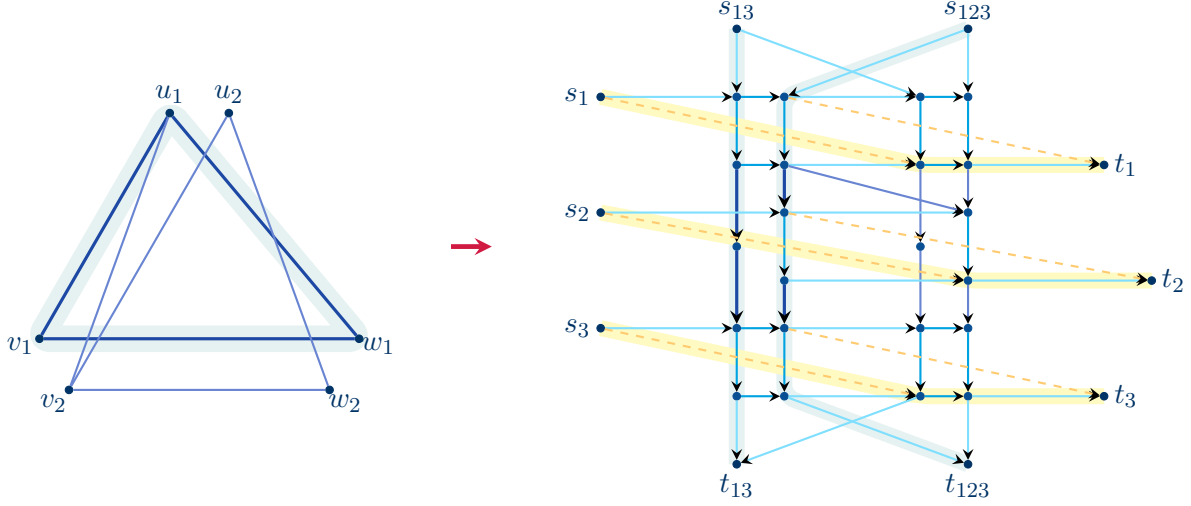


Figure 8: Given an instance G of k -Clique for $k = 3$ and $n = 2$ on the left, with parts $V_1 = \{u_1, u_2\}$, $V_2 = \{v_1, v_2\}$, and $V_3 = \{w_1, w_2\}$, and lists $\langle 1, 3 \rangle$ and $\langle 1, 2, 3 \rangle$ forming a k -covering family for $k = 3$, we produce the instance G' of p -DP on the right for $p = 5$. There are three rows of gadgets in G' , corresponding to the parts of G . The gadgets in rows one and three have two columns because the integers 1 and 3 are covered by two lists, while the gadgets in row two have a single column because the integer 2 is covered by one list. Note that the paths $\Pi_{13}(u_1, w_1)$ and $\Pi_{13}(u_2, w_2)$ from gadgets in row 1 to row 3 have length two. The triangle (u_1, v_1, w_1) in G , highlighted on the left, is mapped to the p -DP solution in G' which takes the yellow highlighted paths $P(u_1), P(v_1), P(w_1)$ and the blue highlighted paths from s_{13} to t_{13} and s_{123} to t_{123} . These paths are disjoint because the first three paths skip the gadgets for vertices u_1, v_1, w_1 respectively, which leaves room for the (s_{13}, t_{13}) -path checking for edge (u_1, w_1) in G , and the (s_{123}, t_{123}) -path checking for edges (u_1, v_1) and (v_1, w_1) in G .

Vertex Gadgets

For each $i \in [k]$ and vertex $v \in V_i$, we construct a gadget graph G_v as depicted in Figure 7.

For every $r \in [l(i)]$, we include nodes $a_r(v)$ and $b_r(v)$.

We call the $a_r(v)$ and $b_r(v)$ the *top* and *bottom* nodes of G_v respectively.

For every $r \in [l(i)]$, we include edges $e_r(v) = (a_r(v), b_r(v))$. For each $r \in [l(i) - 1]$, we include edges from $a_r(v)$ to $a_{r+1}(v)$ and from $b_r(v)$ to $b_{r+1}(v)$.

This completes the description of G_v .

For convenience, if L is the r^{th} list containing v , we write $a_L(v) = a_r(v)$ and $b_L(v) = b_r(v)$.

Arranging Gadgets in Rows

For each $i \in [k]$, we list the vertices of V_i in some arbitrary order

$$v_{i,1}, \dots, v_{i,n}.$$

For every $j \in [n - 1]$, we connect $G_{v_{i,j}}$ to $G_{v_{i,j+1}}$ by including edges from

$$a_{l(i)}(v_{i,j}) \text{ to } a_1(v_{i,j+1}) \quad \text{and} \quad b_{l(i)}(v_{i,j}) \text{ to } b_1(v_{i,j+1}).$$

For every $j \in [n - 2]$, we additionally include the edge from $a_{l(i)}(v_{i,j})$ to $b_1(v_{i,j+2})$, which we call the *skip edge* for $v_{i,j+1}$, because traversing this edge corresponds to skipping the gadget for $v_{i,j+1}$.

Skip edges are depicted with dashed lines in Figure 8. We view this construction as arranging the G_v gadgets for all $v \in V_i$ in the “ i^{th} row of G' .”

Terminal Vertices Checking Nodes

We now introduce source and target nodes and describe how they connect to the G_v gadgets.

For each $i \in [k]$, we introduce new vertices s_i and t_i .

We include edges from s_i to $a_1(v_{i,1})$ and from $b_{l(i)}(v_{i,n})$ to t_i .

Additionally, we include skip edges from s_i to $b_1(v_{i,2})$ and from $a_{l(i)}(v_{i,n-1})$ to t_i (traversing these edges corresponds to skipping $G_{v_{i,1}}$ and $G_{v_{i,n}}$ respectively).

Paths Between Rows of Gadgets

For every choice of indices $x, y \in [k]$ with $x < y$, list $L \in \mathcal{L}$ covering (x, y) , and vertices $u \in V_x$ and $w \in V_y$ such that (u, w) is an edge in G , we include a path $\Pi_L(u, w)$ of length $2(y - x) - 1$ from $b_L(u)$ to $a_L(w)$ in G' . These paths encode adjacency information about G . We say the $\Pi_L(u, w)$ are the paths between rows corresponding to list L .

Terminal Vertices Checking Edges

For each list L in our collection, we introduce source node s_L and target node t_L .

Suppose i is the first element in L . Then for all $v \in V_i$, we have edges from s_L to $a_L(v)$.

Suppose j is the final element in L . Then for all $v \in V_j$, we have edges from $a_L(v)$ to t_L .

We observe that any (s_L, t_L) -path in G' must pass through nodes in row i of G' for all $i \in L$.

This completes our construction of the graph G' , an example of which is depicted in Figure 8.

We claim that G' has disjoint paths from its sources to its targets if and only if G has a k -clique.

To prove this result, it will be helpful to first identify some special paths in G' .

For any $i \in [k]$ and $v \in V_i$, we let $P(v)$ be the path which begins at s_i , passes through the top nodes of G_u for all $u \in V_i$ before v , then takes the skip edge skipping over G_v , passes through the bottom nodes of G_w for all $w \in V_i$ after v , and then finally ends at t_i .

Lemma 47. For every $v \in V$, $P(v)$ is a shortest path in G' .

Proof. Fix $v \in V$. Let $i \in [k]$ be the index such that $v \in V_i$.

Recall that for any $j \in [k]$, row j of G' consists of the all gadgets G_u for vertices $u \in V_i$. The graph G' is structured so that there is no edge from row q to p whenever $q > p$. Then since s_i only has edges to nodes in row i and t_i only has edges from nodes in row i , any (s_i, t_i) -path in G' must lie completely in row i . Such a path can use at most one skip edge, since a skip edge goes from a top node to a bottom node, and there are no edges from bottom nodes to top nodes within row i . If an (s_i, t_i) -path uses no skip edges, it has exactly $n \cdot l(i) + 1$ internal nodes. If instead the path uses one skip edge, it has exactly $(n - 1) \cdot l(i) + 1$ internal nodes, because it skips over one gadget.

This implies that any (s_i, t_i) -path in G' which uses a skip edge is a shortest path.

Since $P(v)$ uses a skip edge, it is an (s_i, t_i) -shortest path as claimed. ■

Lemma 48 (Clique $\Rightarrow$ Disjoint Shortest Paths). If G has a k -clique, then G' has node-disjoint shortest paths from its sources to its targets.

Proof. Let $(v_1, \dots, v_k) \in V_1 \times \dots \times V_k$ be a k -clique in G .

For each list

$$L = \langle i_1, \dots, i_\ell \rangle$$

in $\mathcal{L}$, let $Q(L)$ be the path which begins at s_L , passes through nodes

$$a_L(v_{i_1}), b_L(v_{i_1}), \dots, a_L(v_{i_\ell}), b_L(v_{i_\ell})$$

in that order using the paths corresponding to list L , and then ends at t_L . More precisely, $Q(L)$ goes from $a_L(v_{i_j})$ to $b_L(v_{i_j})$ for each $j \in [\ell]$ by taking the edge between these vertices in G' , and goes from $b_L(v_{i_j})$ to $a_L(v_{i_{j+1}})$ for each $j \in [\ell - 1]$ by traversing the path $\Pi(v_{i_j}, v_{i_{j+1}})$.

Claim 49. For every list $L \in \mathcal{L}$, $Q(L)$ is a shortest path.

Proof. The path $\Pi(v_{i_j}, v_{i_{j+1}})$ has length $2(i_{j+1} - i_j) - 1$. By telescoping, this means the sum of the lengths of the paths of $\Pi(v_{i_j}, v_{i_{j+1}})$ over all $j \in [\ell - 1]$ is $2(i_\ell - i_1) - \ell$. The only edges of $Q(L)$ not in the $\Pi(v_{i_j}, v_{i_{j+1}})$ paths are the edge from s_L to $a_L(v_{i_1})$, the edges from $a_L(v_{i_j})$ to $b_L(v_{i_j})$ for each $j \in [\ell]$, and the edge from $b_L(v_{i_\ell})$ to t_L . It follows that the length of $Q(L)$ is

$$2 + \ell + (2(i_\ell - i_1) - \ell) = 2(i_\ell - i_1 + 1).$$

To prove the claim, it suffices to show that any (s_L, t_L) -path in G' has length at least

$$2(i_\ell - i_1 + 1).$$

To that end, take an arbitrary (s_L, t_L) -path Q in G' . Let $j_1, \dots, j_r \in [k]$ be the sequence of distinct rows Q visits in order. Since G' only includes edges going from row x to row y for $x < y$, we must have $j_1 < j_2 < \dots < j_r$.

By construction the only way to go from row x to row y in G' is to traverse a path $\Pi_{L'}(u, w)$ for some nodes $u \in V_x$ and $w \in V_y$, and list $L' \in \mathcal{L}$. Such a path begins at the bottom of row x at $b'_L(u)$, ends at the top of row y at $a'_L(w)$, and has length $2(y - x) - 1$. So the sum of the lengths of all such paths Q traverses to hit rows $j_1, \dots, j_r$ is at least $2(j_r - j_1) - r$ by telescoping. Moreover, Q must traverse at least one edge within each row it visits, to get from the top of that row to the bottom of that row. This accounts for at least r additional edges, so Q has length at least

$$(2(j_r - j_1) - r) + r = 2(j_r - j_1).$$

Finally, Q also must contain an edge leaving s_L , and an edge entering t_L . This accounts for two more edges, so Q has length at least $2(j_r - j_1 + 1)$.

By definition, i_1 and i_ℓ are the first and final elements in list L respectively. Consequently, edges exiting s_L must go to row i_1 , and edges entering t_L must depart from i_ℓ . This forces $j_1 = i_1$ to be the first row Q enters, and $j_r = i_\ell$ to be the last row Q enters. Thus Q has length at least

$$2(j_r - j_1 + 1) = 2(i_\ell - i_1 + 1).$$

This lower bound for the length of an arbitrary (s_L, t_L) -path in G' is equal to the length of $Q(L)$. Thus $Q(L)$ is a shortest path in G' , as claimed. $\square$

We claim that the collection of paths obtained by taking $P(v_i)$ for each $i \in [k]$ and $Q(L)$ for each $L \in \mathcal{L}$ is a collection of node-disjoint paths in G' .

Indeed, for each $i \in [k]$, $P(v_i)$ is in the i^{th} row of G' , so $P(v_i)$ and $P(v_j)$ are disjoint for $i \neq j$. Similarly, for each choice of lists $L \neq K$ in $\mathcal{L}$, $Q(L)$ and $Q(K)$ pass through distinct vertices.

Take $i \in [k]$ and $L \in \mathcal{L}$. The only way $Q(L)$ can intersect $P(v_i)$ is if $i \in L$. If $i \in L$, the only nodes at which $Q(L)$ can intersect $P(v_i)$ are $a_L(v_i)$ and $b_L(v_i)$. However, $P(v_i)$ skips G_{v_i} , so neither of these nodes occur in $P(v_i)$. Thus $P(v_i)$ and $Q(L)$ are disjoint.

Since by [Lemma 47](#) the $P(v_i)$ are shortest paths, and by [Claim 49](#) the $Q(L)$ are shortest paths, we have proven the desired result. $\blacksquare$

Lemma 50 (Disjoint Paths $\Rightarrow$ Clique). If G' has node-disjoint paths from its sources to its targets, then G has a k -clique.

Proof. Let $\mathcal{F}$ be a family of node-disjoint paths from the sources to the targets of G' .

For each $i \in [k]$, let P_i denote the (s_i, t_i) -path in $\mathcal{F}$.

For each $L \in \mathcal{L}$, let Q_L denote the (s_L, t_L) -path in $\mathcal{F}$.

By construction of G' , each P_i must stay in row i . Since skip edges take us from the top of a row to the bottom of a row, each P_i uses at most one skip edge.

We say a path Q “uses the edges corresponding to a list $L \in \mathcal{L}$ ” if Q uses an edge of $\Pi_L(u, w)$, for some vertices u and w .

Claim 51. For each list $L \in \mathcal{L}$, the path Q_L does not use edges corresponding to list $L' \in \mathcal{L}$, for any $L' \neq L$.

Proof. Suppose to the contrary that there does exist a list $L \in \mathcal{L}$ such that Q_L uses edges corresponding to lists distinct from $\mathcal{L}$. Pick such an L which occurs latest in the ordering of $\mathcal{L}$.

Consider the first edge Q_L uses which corresponds to another list.

Suppose this edge belongs to $\Pi_{L'}(u, w)$ for some list $L' \neq L$, and vertices u and w . Let $x \in [k]$ be the index such that $u \in V_x$. Since the edge we are considering is the first edge Q_L uses which corresponds to a list other than L , before this edge Q_L can only have entered rows with indices contained in L . Thus $x \in L$, and Q_L must have entered row x at a vertex $a_L(v)$, for some $v \in V_x$. We also know that $x \in L'$, since otherwise $\Pi_{L'}(u, w)$ would not exist. Since Q_L uses an edge of $\Pi_{L'}(u, w)$, we know that the starting vertex of this path $b_{L'}(u)$ must occur after $a_L(v)$ in the topological order of G' .

Previously, we observed that each P_i uses at most one skip edge. We claim that path P_x uses the skip edge to skip gadget G_v in row x . Suppose to the contrary that P_x does not use this skip edge. Since P_x and Q_L are disjoint, P_x must pass through $b_L(v)$ (otherwise it would hit $a_L(v)$). Since $b_{L'}(u)$ occurs after $a_L(v)$ in the topological order, this means that P_x passes through $b_{L'}(u)$. This contradicts our assumption that P_x and Q_L are disjoint.

Thus P_x uses the skip edge corresponding to G_v in row x , as claimed.

Since P_x uses this skip edge, we know that P_x passes through $b_K(v')$ for all lists K containing x and all $v' \in V_x$ after v in the order on V_x . Since $b_{L'}(u)$ is not in P_x (because P_x and Q_L are disjoint), we must have $u = v$.

Then for $a_L(v)$ to occur before $b_{L'}(v)$ in the topological order, we must have L occur before L' in the ordering for $\mathcal{L}$.

Now, observe that $Q_{L'}$ cannot cross between rows while only using edges corresponding to L' . Indeed, if $Q_{L'}$ only uses edges corresponding to L' to go between rows, it would pass through a column of row x , since $x \in L'$. To be disjoint from P_x , this is only possible if $Q_{L'}$ uses a column of G_v , since P_x hits the columns of every other gadget in row x . But Q_L and $Q_{L'}$ are disjoint, and we already said that Q_L hits the column of G_v corresponding to L' .

Thus, $Q_{L'}$ must use an edge corresponding to a list distinct from L' . We also observed earlier that L' occurs after L in the ordering on $\mathcal{L}$.

This contradicts our choice of L as the final list in $\mathcal{L}$ with the property that Q_L uses an edge corresponding to a list distinct from L .

Thus our initial assumption was false, and the claim holds. $\square$

Claim 52. For every $i \in [k]$, P_i uses a skip edge.

Proof. Suppose to the contrary that there exists an index $i \in [k]$ such that P_i does not use a skip edge. Then for every $v \in V_i$ and $r \in [s(i)]$, the path P_i passes through at least one of $a_r(v)$ and

$b_r(v)$. Let $L \in \mathcal{L}$ be a list containing i (such a list exists because $\mathcal{L}$ is a k -covering family). By [Claim 51](#) and the fact that each P_i uses at most one skip edge, the path Q_L must pass through vertices $a_L(v)$ and $b_L(v)$ for some $v \in V_i$. Consequently, Q_L intersects P_i , which contradicts the assumption that paths in $\mathcal{F}$ are disjoint. So each P_i traverses a skip edge as claimed. $\square$

Since a skip edge moves a path from the top nodes to the bottom nodes of a row, by [Claim 52](#) we know that each P_i traverses exactly one skip edge.

For each $i \in [k]$, let $v_i \in V_i$ be the unique vertex such that P_i traverses the edge skipping G_{v_i} . Then we claim that $(v_1, \dots, v_k)$ is a clique in G .

Indeed, take any pair $(i, j) \in [k]^2$ with $i < j$. Let $L \in \mathcal{L}$ be a list covering (i, j) .

By [Claim 51](#), the path Q_L traverses a path $\Pi_L(u, w)$ for some $u \in V_i$ and $w \in V_j$.

Since the only gadgets skipped by P_i and P_j are G_{v_i} and G_{v_j} respectively, and Q_L is disjoint from P_i and P_j , we see that in fact Q_L traverses the path $\Pi_L(v_i, v_j)$.

But this path exists in G' only if (v_i, v_j) is an edge in G .

So (v_i, v_j) is an edge in G for all $i < j$, and thus G contains a k -clique as claimed. $\blacksquare$

Theorem 9. 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.

Proof. Construct a collection $\mathcal{L}$ of $\lambda = \lfloor k^2/4 \rfloor$ lists which form a k -covering family using [Lemma 46](#). For constant k , this takes $O(1)$ time. This collection has the property that each element of $[k]$ shows up in fewer than k list of $\mathcal{L}$.

Let G be the input instance of k -Clique.

Using $\mathcal{L}$ and G , construct the graph G' described previously in this section.

By [Lemmas 48](#) and [50](#), solving p -DP or p -DSP on G' solves k -Clique on G , where

$$p = k + \lambda = k + \lfloor k^2/4 \rfloor.$$

Since G' consists of $2(k + \lambda)$ terminals and kn gadget graphs G_v , each on fewer than $2k$ nodes, the graph G' has at most $O(kn)$ nodes. Since we can construct G' in time linear in its size, this proves the desired result. $\blacksquare$

7 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 Problem 1. Is there a *deterministic* linear time algorithm solving 2-DSP?

Open Problem 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 Problem 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 Problem 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 be nice to resolve this discrepancy, either by designing faster algorithms for the latter problem, or showing better lower bounds for the former problem.

Open Problem 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 Problem 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

- [Akh20] Maxim Akhmedov. Faster 2-disjoint-shortest-paths algorithm. In *Computer Science – Theory and Applications*, pages 103–116. Springer International Publishing, 2020. [1](#)
- [BH19] Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. *SIAM Journal on Computing*, 48(6):1698–1710, January 2019. [6](#)
- [BHK22] 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. [5](#), [6](#), [7](#)
- [BHT12] 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. [7](#)
- [BK17] Kristof Berce 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 fuer Informatik. [1](#), [2](#), [3](#), [12](#)
- [BNRZ20] Matthias Bentert, André Nichterlein, Malte Renken, and Philipp Zschoche. Using a geometric lens to find k disjoint shortest paths, 2020. [4](#), [12](#)
- [BNRZ21] 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. [1](#), [2](#), [3](#), [4](#), [5](#)
- [BS83] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22(3):317–330, February 1983. [28](#)
- [CFK⁺15] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015. [3](#), [4](#)
- [CGS15] 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. [7](#)
- [Chi21] Rajesh Chitnis. A tight lower bound for edge-disjoint paths on planar dags, 2021. [4](#), [5](#), [6](#)
- [DW22] 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. [3](#)
- [EG04] 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. [3](#)

- [EKW23] Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving, 2023. [7](#)
- [ET98] Tali Eilam-Tzoref. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, June 1998. [1](#)
- [FGK⁺22] Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, Kirill Simonov, and Giannos Stamoulis. Fixed-parameter tractability of maximum colored path and beyond, 2022. [6](#)
- [FHW80] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, February 1980. [1](#), [2](#), [3](#), [28](#), [44](#)
- [KKR12] 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. [1](#)
- [KPS24] Tuukka Korhonen, Michał Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time, 2024. [1](#)
- [Loc21] 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. [1](#)
- [RS95] 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. [1](#)
- [Sli10] Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 24(1):146–157, January 2010. [4](#), [5](#), [6](#), [12](#), [13](#), [34](#)
- [SY09] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3-4):207–388, 2009. [28](#)
- [Tho97] M. Thorup. Undirected single source shortest paths in linear time. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc, 1997. [8](#)
- [Tho05] Torsten Tholey. Solving the 2-disjoint paths problem in nearly linear time. *Theory of Computing Systems*, 39(1):51–78, November 2005. [1](#)
- [Tho12] Torsten Tholey. Linear time algorithms for two disjoint paths problems on directed acyclic graphs. *Theoretical Computer Science*, 465:35–48, December 2012. [2](#)
- [Wil18] 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. [3](#)
- [WXXZ23] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega, 2023. [3](#), [5](#)

A Standard Reductions

In this section, we collect proofs of some standard reductions that allow us to move between different variants of disjoint path problems.

Proposition 53 (Edge-Disjoint $\leq$ Vertex-Disjoint). There is a reduction from k -EDSP on n vertices and m edges to k -DSP on $m + k(n + 2)$ nodes and $2k(m + 1)$ edges.

Proof. Let G be the instance of k -EDSP on n vertices and m edges, with sources $s_1, \dots, s_k$ and targets $t_1, \dots, t_k$. Let V and E be the vertex and edge sets of G respectively.

We construct a graph G' as follows. For every vertex $v \in V$, G' has k nodes $v_1, \dots, v_k$ (we call these the *copies* of v in G'). For every edge $e \in E$, G' has a node e . For every edge $e = (v, w) \in E$, we include edges in G' from v_i to e and from e to w_i for all $i \in [k]$. If $e = (v, w)$ had weight $\ell(v, w)$ in G , then the (v_i, e) and (e, w_i) edges in G' each have weight $\ell(v, w)$.

Finally, we introduce new sources $s'_1, \dots, s'_k$ and targets $t'_1, \dots, t'_k$ in G' . Then for every $i, j \in [k]$, we add edges from s'_i to $(s_i)_j$ and from $(t_i)_j$ to t'_i of weight 1.

From this construction, G' has $m + kn + 2k$ nodes and $2km + 2k$ edges as claimed.

Suppose we have vertex-disjoint (s'_i, t'_i) -shortest paths P'_i in G' . We can assume each P'_i never traverses two copies of the same vertex $v \in V$ (otherwise, we could remove the subpath between two copies of v and obtain a shorter path than P'_i). We map each P'_i to an (s_i, t_i) -shortest path P_i in G , by having P_i pass through the vertices $v \in V$ for which P'_i contains a copy of v , in the order the copies appear in P'_i .

By construction, if P_i has length ℓ , then P'_i has length $2\ell + 2$. So since the P'_i are shortest paths, the P_i are also shortest paths. The P_i are also edge-disjoint, since if some P_i and P_j overlap at an edge e , the paths P'_i and P'_j would overlap at node e in G' , which would contradict the assumption that the P'_i are vertex-disjoint.

So any solution to k -DSP on G' pulls back to a solution to k -EDSP on G .

Conversely, given edge-disjoint (s_i, t_i) -shortest paths P_i in G of the form

$$P_i = \langle v_{i,1}, \dots, v_{i,\ell_i} \rangle$$

we can produce (s'_i, t'_i) -paths P'_i in G' of the form

$$P'_i = \langle s'_i, (v_{i,1})_i, ((v_{i,1})_i, (v_{i,2})_i), (v_{i,2})_i, \dots, (v_{i,\ell_i})_i, t'_i \rangle.$$

Similar reasoning to the above shows that the P'_i are shortest paths in G' . These paths are vertex-disjoint because each P'_i only uses the i^{th} copies of $v \in V$, and the P_i were edge-disjoint, so the P'_i cannot overlap at any nodes of the form $e \in E$.

So any solution to k -EDSP in G has a corresponding solution to k -DSP in G' .

This proves the desired result. ■

As mentioned in [Section 1](#), [Proposition 53](#) combined with [Theorems 1](#) and [2](#) implies [Corollary 4](#), and [Proposition 53](#) combined with the $O(mn^{k-1})$ time algorithm for k -DSP in DAGs from [\[FHW80, Theorem 3\]](#) implies that k -EDSP in DAGs can be solved in $O(m^k)$ time for constant k .

Proposition 54 (Disjoint Paths $\leq$ Disjoint Shortest Paths). There are reductions from k -DP on DAGs to k -DSP on DAGs and from k -EDP on DAGs to k -EDSP on DAGs with the same number of nodes and edges respectively.

Proof. Let G be the DAG which is an instance of k -DP or k -EDP. Take a topological order $v_1, \dots, v_n$ of the vertices in G . Let G' be the DAG with the same vertex and edge sets as G , but where each edge (v_i, v_j) has weight $(j - i)$. Then by telescoping, for any vertices v_i and v_j in G' , every path from v_i to v_j in G' has total length $(j - i)$.

Consequently, every path in G becomes a shortest path in G' .

So solving k -DSP and k -EDSP in G' corresponds to solving k -DP and k -EDP respectively in G , which proves the desired result. ■

As mentioned in [Section 1](#), [Proposition 54](#) shows that k -EDSP in weighted DAGs generalizes the k -EDP problem in DAGs.