Using a Geometric Lens to Find k Disjoint Shortest Paths

Matthias Bentert
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

André Nichterlein
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Malte Renken
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Philipp Zschoche
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

Given an undirected n-vertex graph and k pairs of terminal vertices $(s_1, t_1), \ldots, (s_k, t_k)$, the k-Disjoint Shortest Paths (k-DSP) problem asks whether there are k pairwise vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i is a shortest s_i-t_i-path for each $i \in [k]$. Recently, Lochet [SODA '21] provided an algorithm that solves k-DSP in $n^{O\left(k^{5^k}\right)}$ time, answering a 20-year old question about the computational complexity of k-DSP for constant k. On the one hand, we present an improved $O(kn^{16k \cdot k! + k + 1})$-time algorithm based on a novel geometric view on this problem. For the special case $k = 2$, we show that the running time can be further reduced to $O(nm)$ by small modifications of the algorithm and a further refined analysis. On the other hand, we show that k-DSP is W[1]-hard with respect to k, showing that the dependency of the degree of the polynomial running time on the parameter k is presumably unavoidable.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases graph algorithms, dynamic programming, W[1]-hardness, geometry

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.26

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2007.12502`

Funding This work was started at the research retreat of the group *Algorithmics and Computational Complexity* in September 2019, held in Schloss Neuhausen (Prignitz, Germany).
Malte Renken: Supported by the DFG project MATE (NI 369/17).

1 Introduction

The k-Disjoint Paths problem is a fundamental and well-studied combinatorial problem. Given an undirected graph G and k terminal pairs $(s_i, t_i)_{i \in [k]}$, the question is whether there are pairwise disjoint [1] s_i-t_i-paths P_i for each $i \in [k]$. The problem was shown to be NP-hard by Karp [9] when k is part of the input. On the positive side, Robertson and Seymour [13] provided an algorithm running in $O(n^3)$ time for any constant k. Later, Kawarabayashi et al. [10] improved the running time to $O(n^2)$, again for fixed k. On directed graphs, in contrast, the problem is NP-hard even for $k = 2$ [7]. However, on directed acyclic graphs, the problem becomes again polynomial-time solvable for constant k [7] and linear-time solvable for $k = 2$ [14].

[1] We only consider the vertex-disjoint setting to which the edge-disjoint version can be reduced.

Focusing on the undirected case, we study the problem variant where all paths in the solution have to be shortest paths. This variant was introduced by Eilam-Tzoreff [5].

k Disjoint Shortest Paths (k-DSP)

Input: A graph $G = (V, E)$ and $k \in \mathbb{N}$ pairs $(s_i, t_i)_{i \in [k]}$ of vertices.

Task: Find k disjoint paths P_i such that P_i is a shortest s_i-t_i-path for each $i \in [k]$.

Throughout the whole paper, we assume that the input graph is connected. Eilam-Tzoreff [5] showed the NP-hardness of k-DSP when k is part of the input. Moreover, Eilam-Tzoreff provided a dynamic-programming based $O(n^8)$-time algorithm for 2-DSP. This algorithm for 2-DSP works also for positive edge lengths. Recently, Gottschau et al. [8] and Kobayashi and Sako [11] independently extended this result by providing polynomial-time algorithms for the case that the edge lengths are non-negative. As for directed graphs, Berczi and Kobayashi [2] provided a polynomial-time algorithm for strictly positive edge length. Very recently, Akhmedov [1] presented an algorithm solving 2-DSP in $O(n^6)$ time for unweighted graphs and in $O(n^7)$ for strictly positive edge lengths. Note that allowing zero-length edges generalizes 2-Disjoint Path on directed graphs, which is NP-hard [7]. Extending the problem to finding two disjoint s_i-t_i-paths of minimal total length (in undirected graphs), Björklund and Husfeldt [4] provided a randomized algorithm with running time $O(n^{11})$.

The existing algorithms for 2-DSP are based on dynamic programming with tedious case distinctions. We provide a new algorithm using a simple and elegant geometric perspective:

▶ **Theorem 1.** *2-DSP can be solved in $O(nm)$ time.*

Whether or not k-DSP for constant $k \geq 3$ is polynomial-time solvable was posed as a research challenge [6, open problem 4.6]. Recently, Lochet [12] settled this long standing open question by showing that k-DSP can be solved in $n^{O(k^{5^k})}$ time. Using our geometric approach, we provide an improved $O(k \cdot n^{16k \cdot k! + k + 1})$-time algorithm.

▶ **Theorem 2.** *k-DSP can be solved in $O(k \cdot n^{16k \cdot k! + k + 1})$ time.*

We describe the basic idea of our algorithms and the new geometric tools in Section 2. In Section 3, we formalize these geometric tools for two paths and prove Theorem 1. In Section 4, we lift these arguments to $k > 2$ paths. In Section 5, we present our algorithm for Theorem 2 and prove its correctness.

Finally, we show that k-DSP is W[1]-hard with respect to k. Hence, under standard assumptions from parameterized complexity, there is no algorithm with running time $f(k)n^{O(1)}$ for any function f. Thus, polynomial-time algorithms where k does not appear in the exponent (as the $O(n^2)$-time algorithm for k-Disjoint Path for any constant k [13, 10]) are unlikely to exist for k-DSP. Furthermore, under the Exponential-Time Hypothesis (ETH), we show that there is no algorithm with running time $f(k) \cdot n^{o(k)}$ for any computable function f.

▶ **Proposition 3** ($\star$[2])**.** *k-DSP is W[1]-hard with respect to k. Moreover, assuming ETH, there is no $f(k) \cdot n^{o(k)}$-time algorithm for k-DSP.*

Preliminaries. We set $\mathbb{N} := \{0, 1, 2, \dots, \}$ and $[n] := \{1, 2, \dots, n\}$. We always denote by $G = (V, E)$ a graph (undirected and connected unless said otherwise) and by n and m the number of vertices and edges in G, respectively. A *path* of length $\ell \geq 0$ in a graph G is a

[2] Some proofs (marked with a $\star$) are deferred to a full version.

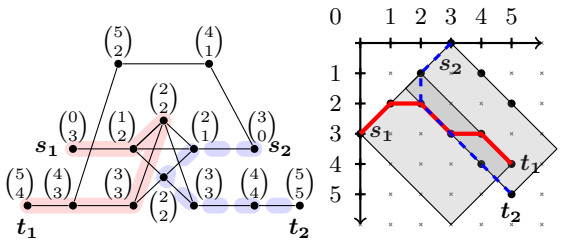

Figure 1 *Left side:* A graph with distinguished vertices s_1, s_2, t_1, t_2; vertex coordinates written next to the vertices. Two shortest paths are highlighted. *Right side:* The 2D-arrangement of the vertices. The two gray rectangles spanned by s_1 and t_1 and by s_2 and t_2 contain the two shortest paths s_1-t_1 (solid red path) and s_2-t_2 (dashed blue path).

sequence of distinct vertices $v_0v_1 \dots v_\ell$ such that each pair v_{i-1}, v_i is connected by an edge in G. The first and last vertex v_0 and v_ℓ are called the *end vertices* or *ends* of P and are denoted by s_P and t_P. We also say that P is a path *from* v_0 *to* v_ℓ, a path *between* v_0 and v_ℓ, or a v_0-v_ℓ-path. When no ambiguity arises, we do not distinguish between a path and its set of vertices. For $v, w \in P$, we denote by $P[v, w]$ the subpath of P with end vertices v and w. For two vertices v, w, we denote the length of a shortest v-w-path in G by $\mathrm{dist}_G(v, w)$ or $\mathrm{dist}(v, w)$ if the graph G is clear from the context. If for all $i \in [k]$ there is a path P_i that is a shortest s_i-t_i-path and disjoint with P_j for all $j \in [k] \setminus \{i\}$, then we say that the paths $(P_i)_{i\in[k]}$ are a solution for an instance $(G, (s_i, t_i)_{i\in[k]})$ of k-DSP.

2 The Key Concepts behind our Polynomial-Time Algorithm

In this section, we describe our approach to solve k-DSP in polynomial-time for any fixed k. As a warm-up, we start with sketching an algorithm for 2-DSP based on the same approach.

Solving 2-DSP in the plane. Before describing the algorithm, we show the central geometric idea behind it. Recall that we want to find two shortest paths P_1 and P_2 from s_1 to t_1 and from s_2 to t_2, respectively. We now arrange the vertices on a 2-dimensional grid where the first coordinate of each vertex is the distance to s_1 and the second coordinate the distance to s_2; see left side of Figure 1 for an example graph with the corresponding coordinates and the right side for an arrangement of the vertices in a grid with a continuous drawing of the paths (drawing straight lines between points occurring in the paths). Clearly, with two breadth-first searches from s_1 and s_2 we can compute the coordinates of all vertices in linear time. Note that there might be multiple vertices with the same coordinates. However, at most one vertex per coordinate can be part of a shortest s_1-t_1- or s_2-t_2-path.

This arrangement of the vertices allows the following simple geometric observation: The drawing of each shortest s_1-t_1-path has to be within a rectangle with angles of 45° to the coordinate axes and with corner points s_1 and t_1 (see gray rectangles in right side of Figure 1). As a consequence, shortest paths that have to stay within two disjoint such rectangles cannot intersect. We use this argument extensively in the subsequently sketched algorithm.

We assume that the drawings of P_1 and P_2 cross as displayed in the right side of Figure 1 (the non-crossing case is easier to deal with). Our algorithm solving 2-DSP in this case is as follows: We distinguish whether the intersection of the drawings of P_1 and P_2 contain a point with integer coordinates (that is, our algorithm tries both possibilities).

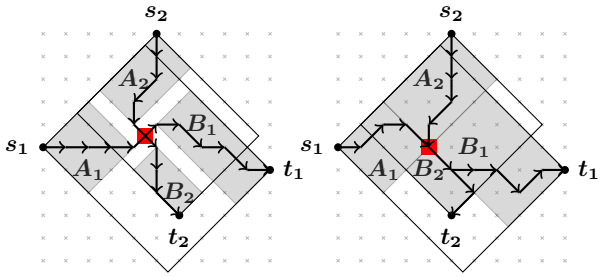

Figure 2 An illustration of the directed acyclic graph used for Theorem 1. The first point p in the intersection of the drawing of P_1 and P_2 is marked by a red square. *Left side:* Straigh-line drawings of P_1 and P_2 intersect, but not in points with integer coordinates. *Right side:* Both P_1 and P_2 use vertices with same coordinates.

If the intersection does not have a point with integer coordinates, then it is easy to see that the intersection of the drawing of P_1 and P_2 has to be a single point p (with non-integer coordinates). We guess[3] the four coordinate pairs $(x, y), (x, y+1), (x+1, y)$, and $(x+1, y+1)$, $x, y \in \mathbb{N}$, surrounding the intersection point of the drawings of P_1 and P_2. Note that this can be done in $O(n)$ time by guessing the vertex on the coordinate pair (x, y). The goal is now to turn the input graph G into a directed acyclic graph D such that each shortest s_i-t_i-path in G corresponds to an s_i-t_i-path in D. To this end, we partition the grid into four areas A_1, B_1, A_2, B_2 (each area is defined by one of the guessed points and the closer endpoint of the path going through the point) and orient each edge according to the area it lies in (see left side of Figure 2 for an illustration). An edge $\{v, w\}$ in the area A_i or B_i, $i \in [2]$, is oriented towards the vertex w with the larger i coordinate. Edges between A_i and B_i are oriented towards the vertex in B_i. All remaining (unoriented) edges are removed. Note that this results in a directed acyclic graph. Furthermore, a shortest s_i-t_i-path in G induces an s_i-t_i-path in D and each s_i-t_i-path in D is a shortest path in G because it is strictly monotone increasing in the i-coordinate and all strictly monotone increasing paths have the same length as each path contains one vertex for each integer i-coordinate between the i-coordinates of s_i and t_i. Observe that in this case the two paths cannot intersect as P_1 can only reach vertices with coordinates in A_1 and B_1 and P_2 can only use vertices with coordinates in A_2 and B_2. Hence, one can find P_1 and P_2 in linear time. Altogether, this gives a running time of $O(nm)$ in this case.

Assume now that there is a point with integer coordinates in this intersection; this case requires more work. We assume that if there are two points (x_1, y_1) and (x_2, y_2) in the intersection of the drawings of P_1 and P_2, then we have $x_1 < x_2 \Leftrightarrow y_1 < y_2$. If this is not the case, then repeat the algorithm below with swapped s_2 and t_2. We guess the first point p in the intersection (note that it has integer coordinates). This can be done in $O(n)$ time by guessing a vertex on p. Now we arrange the areas slightly different. The areas are defined by p and one coordinate of s_1, t_1, s_2, t_2; see the right side of Figure 2 for an illustration. Edges in the area $A_i \setminus B_j$ or $B_i \setminus A_j$ are oriented towards the vertex with the larger i-coordinate. Note that edges on the line in $A_i \cap B_j, i \neq j$, could either be used by P_1 or P_2 (but not both), meaning that we have to direct the edges towards the vertex with either the larger 1- or 2-coordinate. Since there are only two possibilities for orienting the edges in $A_1 \cap B_2$, there are only four different possibilities to orient the edges on $A_1 \cap B_2$ and $A_2 \cap B_1$. We try

[3] Whenever we pretend to guess something, the algorithm actually exhaustively tests all possible choices.

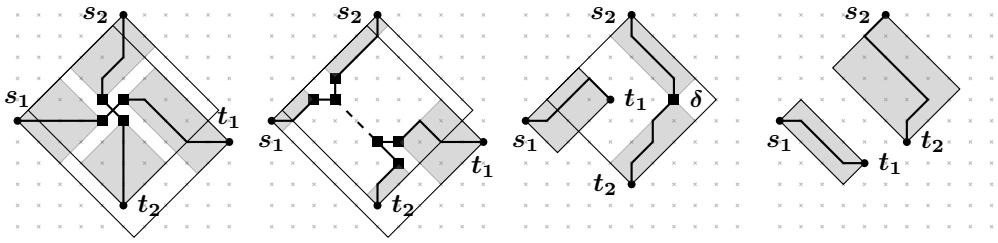

Figure 3 The four cases for the projection of two paths P_1 and P_2 in the two-dimensional grid. From left to right: (1) The projection of the paths cross in one point with non-integer coordinates. (2) The projection of the paths cross in at least one point with integer coordinates. (3) The rectangles defined by the endpoints of P_1 and P_2 intersect, but the projections of P_1 and P_2 do not intersect. (4) The rectangles defined by the endpoints of P_1 and P_2 do not intersect. For each of the two paths our algorithm guesses the vertices on the positions marked by squares.

all four possible orientations and if at least one of them yields a solution, then we know that there is a solution. All other (unorientated) edge are removed – a shortest s_i-t_i-path cannot use it. Note that this results for each of the four described cases in a directed acyclic graph. Furthermore, again a shortest s_i-t_i-path in G induces a s_i-t_i-path in D and each s_i-t_i-path in D is an shortest path in G.

Finally, we use a $O(n+m)$-time algorithm of Tholey [14] for 2-Disjoint Paths on a DAG to find P_1 and P_2. Since there are $O(n)$ possibilities for the point p and 4 possibilities for directing the edges between A_i and B_j, $i \neq j$, we call $O(n)$ instances of the algorithm of Tholey [14]. Thus, we obtain Theorem 1 which is formally proven in Section 3.

Generalizing to k-DSP. We now discuss how to generalize the ideas from above to k-DSP, where $k > 2$. One central idea for $k = 2$ is that the subpaths within the areas A_1, A_2, B_1, B_2 (see Figure 2) can hardly overlap. The only overlap is possible along the borders. In our approach for $k > 2$, we simplify this even further by guessing the vertices on each path before and after the intersection (thus incurring a higher running time). This results in four cases; see Figure 3 for an overview of the cases and the guessed vertices (marked by black squares). It is easy to see in Figure 3 that in each case no subpath within one gray area can possibly intersect with a subpath within another gray area. As can be seen in Figure 3, there is only one case where subpaths of P_1 and P_2 have to be computed carefully due to possible intersections: Both paths use the dashed line in the second case from the left. However, this is the part where both paths are strictly monotone in both coordinates. This is what allowed us for $k = 2$ to transform the graph into a DAG while preserving the solutions (both paths being strictly monotone in at least one coordinate is actually sufficient for this transformation).

Considering k paths, we associate with each vertex $v \in V$ a position in the k-dimensional Euclidean vector space. For brevity, we say that a path has color i if it is strictly monotone in its i-coordinate. Thus, each path P_j has color j. The problem k-Disjoint Paths on a DAG is solvable in polynomial time for constant k [7]. Thus, if we want to find k subpaths from u_i to v_i, $i \in [k]$, that all have the same color (i. e. for each $i \in [k]$ we have that the difference of the i-th coordinate of u_i and v_i is $\mathrm{dist}(u_i, v_i)$), then we can use the algorithm of Fortune et al. [7]. For completeness, we provide a dynamic program with a precise running time analysis as Fortune et al. [7] only state "polynomial time". The general approach to solve the given k-DSP instance is thus as follows: Split the paths $P_1, \ldots, P_k$ into $f(k)$ subpaths (i. e. guess the endpoints of the subpaths) and find a partition of the subpaths such that

(i) subpaths in the same part of the partition share a common color and hence can be computed by the algorithm of Fortune et al. [7] or our dynamic program, and

(ii) subpaths in different parts of the partition cannot intersect.

We remark that this is essentially the same general approach used by Lochet [12]. However, he does not use the geometric view of the paths (as we do). As a result, even for $k = 2$ he only bounds the number of created subpaths by 9^{91} (cf. Lochet [12, Lemma 4.2]). While this constant is certainly not optimized, one can easily see in Figure 3 that our approach splits the two paths in at most five parts (in the second case). Moreover, our geometric view allows us to use a more efficient way of splitting the paths for general k, which we describe below.

Recall that for $k = 2$ the two paths P_1 and P_2 have at most one intersection (point or straight line); see Figure 3. However, in three dimensions $k > 2$ this is no longer true as neither P_1 nor P_2 needs to be monotone in a third dimension. Thus, to exploit the properties shown in Figure 3 for two paths P_i and P_j, we need to project into two dimensions using the i and j coordinate. Hence, we need to be careful with using proper projections to 2D; see Section 3 for details on the geometric arguments. However, whenever two paths P_i and P_j intersect, then we know that the two subpaths in the intersection have both colors i and j. Thus, we can use for these subpaths the two-dimensional observations behind Figure 3 with new projections. We store for each subpath P' of P_i the set Φ of all indices of paths P' intersects, that is, Φ is a subset of all colors that P' has. Now, if P' and P_j intersect, then there is a subpath of P_j that has colors $\Phi \cup \{i\}$. Hence this set Φ of colors can be seen as a "tower of colors" that is transferred to other paths. Our algorithm transfers these towers from one path to another as long as possible. These towers will be defined over permutations of subsets of $[k]$ that encode how these color-towers are produced; see Section 4. As there are at most $k \cdot k!$ such permutations, this explains the exponent of our algorithm. In the end, we arrive at Theorem 2, see Section 5 for details.

3 The Geometry of Two Shortest Paths

In this section, we formalize and generalize the idea behind the geometric view (visualized in Figures 1–3). We start by introducing our notation for projections. For any $\emptyset \subset I \subseteq [k]$ and any vector $x \in \mathbb{R}^k$ we denote with $x^I \in \mathbb{R}^{|I|}$ the orthogonal projection of x to the coordinates in I. That is, x^I is the $|I|$-dimensional vector obtained by deleting all dimensions in x that are not in I. We usually drop the brackets in the exponent, thus writing e.g., $(5,6,7,8,9)^{1,3,4} = (5,7,8)$ or $(5,6,7)^2 = 6$. Similarly, for $R \subseteq \mathbb{R}^k$ we define $R^I := \{x^I \mid x \in R\} \subseteq \mathbb{R}^{|I|}$.

We associate with each vertex $v \in V$ a position in the k-dimensional Euclidean vector space. Formally, $\vec{v} := (\vec{v}^i)_{i\in[k]} := (\mathrm{dist}(s_i, v))_{i\in[k]} \in \mathbb{N}^k$ and for $U \subseteq V$ we use $\vec{U} := \{\vec{u} \mid u \in U\}$. For the k-DSP-instance at hand, one can compute the positions of each vertex in $O(km)$ using simple breadth-first-search from each vertex s_i.

In the following, we will use the following notations for any index set $\emptyset \subset I \subseteq [k]$:
$v \circ^I w \Leftrightarrow \forall a \in I\colon \vec{v}^a \circ \vec{w}^a$, for any $\circ \in \{<, \leq, =, \geq, >\}$ and vertices v, w.
$V \circ^I W \Leftrightarrow \{\vec{v}^I \mid v \in V\} \circ \{\vec{w}^I \mid w \in W\}$, for any $\circ \in \{\subset, \subseteq, =, \supseteq, \supset\}$ and vertex sets V, W.
We further write $x \in^I X$ if there is a vertex $x' \in X$ with $x' =^I x$ and $x \notin^I X$ otherwise.

▶ **Lemma 4.** *For any pair of vertices $v, w \in V$, we have $\|\vec{v} - \vec{w}\|_\infty \leq \mathrm{dist}(v, w)$.*

Proof. Let P be a shortest v-w-path with edge set E_P. Each edge $\{p, q\}$ of P has $\|\vec{p} - \vec{q}\|_\infty = 1$ and thus by the triangle inequality $\|\vec{v} - \vec{w}\|_\infty \leq \sum_{e\in E_P} 1 = \mathrm{dist}(v, w)$. ◀

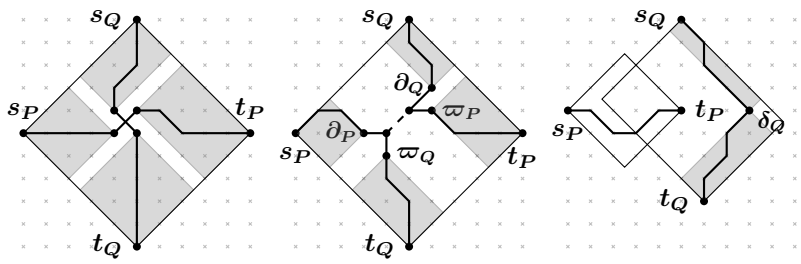

Figure 4 Picture of the a,b-projection of an a-colored path P and a b-colored path Q. The labels are abbreviated as $\partial_Q = \partial_Q^{a,b}(P)$, $\Delta = \Delta^{a,b}(P,Q)$, etc. *Left side:* The case that the two paths a,b-cross but do not share vertices with common a,b-coordinates (thus $\alpha_P(Q) = \omega_P(Q) = \bot$, see Definition 8). *Middle:* Illustration of Lemma 10. The dashed black edge is the a,b-crossing of P and Q. The rectangle areas $\overrightarrow{s_P} \diamond \overrightarrow{\partial_P}$, $\overrightarrow{\varpi_P} \diamond \overrightarrow{t_P}$, $\overrightarrow{s_Q} \diamond \overrightarrow{\partial_Q}$, and $\overrightarrow{\varpi_Q} \diamond \overrightarrow{t_Q}$ are highlighted in gray. These areas are pairwise disjoint. *Right side:* Illustration of Lemma 14. If P and Q are a,b-noncrossing and $\delta_Q \neq \bot$, then the two shaded areas are disjoint from $\overrightarrow{s_P} \diamond \overrightarrow{t_P}$.

For two vertices $u, w \in V$, define $u \diamond w := \{v \in V \mid \mathrm{dist}(u,v) + \mathrm{dist}(v,w) = \mathrm{dist}(u,w)\}$ to be the set of all vertices that lie on a shortest u-w-path. Similarly, for any $x, y \in \mathbb{N}^k$ define $x \diamond y := \{z \in \mathbb{R}^k \mid \|x - z\|_\infty + \|z - y\|_\infty = \|x - y\|_\infty\}$ which is a rectangle whose sides form an angle of 45° with the coordinate axes (see right side of Figure 1).

▶ **Definition 5.** *Let $s, t \in V$ with $\mathrm{dist}(s,t) = \|\vec{s} - \vec{t}\|_\infty$ and P be any shortest s-t-path. We then call the pair (s,t) and the path P* colored*. Furthermore, we define $C(P) := C(s,t) := \{a \in [k] \mid |\vec{s}^a - \vec{t}^a| = \|\vec{s} - \vec{t}\|_\infty\}$ and call (s,t) and P a-*colored *if $a \in C(s,t)$.*

Note that, if P is an a-colored path, then P is strictly monotone in its a-coordinate. Note that for arbitrary $u, w \in V$ we do *not* always have $\overrightarrow{u \diamond w} \subseteq \vec{u} \diamond \vec{w}$, that is the coordinates of all vertices on shortest u-w-paths are not necessarily contained in the set of coordinates "spanned" by $\vec{u}$ and $\vec{w}$. However, this inclusion holds for colored vertex pairs as shown next:

▶ **Lemma 6** (⋆). *Let $v, w \in V$ be an a-colored pair for any color a. Then, $\overrightarrow{v \diamond w} \subseteq \vec{v} \diamond \vec{w}$.*

We will usually be concerned with the projection of $\vec{v} \diamond \vec{w}$ to some set of coordinates $I \subseteq [k]$. Note in this context that $(\vec{v} \diamond \vec{w})^I = \vec{v}^I \diamond \vec{w}^I$.

For some $a \neq b$, consider the projections of an a-colored path P_a and a b-colored path P_b to the $\{a,b\}$-plane, that is, the coordinates of the vertices of the paths are projected to their a- and b-coordinate and edges are drawn as straight lines between the projected vertices. To this end, for any path P we define $\zeta(P) \subset \mathbb{R}^k$ as the piecewise linear curve connecting the points of $\vec{P}$ in the order given by P. It is not hard to see (e.g. in Figure 1 (right) and Figure 3 case (2)), that the intersection of P_a and P_b in the a,b-projection is also a straight line segment with an angle of 45° to the coordinate axes.

▶ **Lemma 7** (⋆). *Let P be an a-colored path and Q be a b-colored path. Then $\zeta(P)^{a,b} \cap \zeta(Q)^{a,b}$ is a (possibly empty) straight line segment.*

Note that even if $\zeta(P)^{a,b} \cap \zeta(Q)^{a,b}$ is non-empty, it needs not contain points from $\mathbb{N}^2$ as can be seen in the left side of Figure 4. In the following, we define the first and last vertices of P and Q on their crossing as well as the coordinates of the vertices before and after their crossing.

▶ **Definition 8.** *Let P be an a-colored path and Q be a b-colored path. We say P and Q are a,b-*crossing *if the intersection $X := \zeta(P)^{a,b} \cap \zeta(Q)^{a,b}$ is non-empty. If $X = \emptyset$, they are called a,b-noncrossing.*

If $\overrightarrow{P}^{a,b} \cap X \neq \emptyset$, then we define $\alpha_P^{a,b}(Q)$ (resp. $\omega_P^{a,b}(Q)$) as the first (resp. last) vertex v of P with $v^{a,b} \in X$. In all other cases set $\alpha_P^{a,b}(Q) := \omega_P^{a,b}(Q) := \perp$.

If P and Q are a,b-crossing, we further define $\partial_P^{a,b}(Q)$ (resp. $\varpi_P^{a,b}(Q)$) as the last (resp. first) vertex of P before (resp. after) that intersection. If no such vertex exists, we set $\partial_P^{a,b}(Q) := \perp$ (resp. $\varpi_P^{a,b}(Q) := \perp$). In all these notations we will omit a, b, and Q if it is clear from context.

▶ **Observation 9.** *If P, Q are two paths with $\alpha_P^{a,b}(Q) \neq \perp$, then $P[\alpha_P^{a,b}(Q), \omega_P^{a,b}(Q)] =^{a,b} Q[\alpha_Q^{a,b}(P), \omega_Q^{a,b}(P)]$. In particular, both of these subpaths are a,b-colored.*

If P and Q are a,b-crossing, then Observation 9 characterizes the behavior of the "crossing subpaths". Let us now consider the remaining path segments before and after the crossing. By Lemma 6 these segments have to lie in the rectangle areas $\overrightarrow{s_P} \diamond \overrightarrow{\partial_P^{a,b}(Q)}$, $\overrightarrow{\varpi_P^{a,b}(Q)} \diamond \overrightarrow{t_P}$, $\overrightarrow{s_Q} \diamond \overrightarrow{\partial_Q^{a,b}(P)}$, and $\overrightarrow{\varpi_Q^{a,b}(P)} \diamond \overrightarrow{t_Q}$ which are displayed in Figure 4 (left and middle). We can show that these areas are indeed pairwise disjoint.

▶ **Lemma 10** (⋆). *Let P and Q be two a,b-crossing paths. Then the sets $\left(\overrightarrow{s_P} \diamond \overrightarrow{\partial_P^{a,b}(Q)}\right)^{a,b}$, $\left(\overrightarrow{\varpi_P^{a,b}(Q)} \diamond \overrightarrow{t_P}\right)^{a,b}$, $\left(\overrightarrow{s_Q} \diamond \overrightarrow{\partial_Q^{a,b}(P)}\right)^{a,b}$, and $\left(\overrightarrow{\varpi_Q^{a,b}(P)} \diamond \overrightarrow{t_Q}\right)^{a,b}$ are pairwise disjoint (or undefined).*

Unfortunately, when P and Q are a,b-noncrossing, then $s_P \diamond t_P$ and $s_Q \diamond t_Q$ are not disjoint in general, see for example Figure 4 (right). To deal with this case, we show that when splitting one path into two subpaths at the vertex δ_Q (see Figure 4 (right)), then we get the desired properties that the respective rectangles do not intersect.

▶ **Definition 11.** *Let P, Q be two colored paths and $a, b \in [k]$. The* common a,b-area *of P and Q is $\Delta^{a,b}(P,Q) := (\overrightarrow{s_P} \diamond \overrightarrow{t_P})^{a,b} \cap (\overrightarrow{s_Q} \diamond \overrightarrow{t_Q})^{a,b}$.*

▶ **Definition 12.** *Let P be an a-colored path and Q a b-colored path where (without loss of generality) $s_P <^a t_P$. Define $B := \{v \in V \mid v =^b s_P \wedge v <^a s_P\} \cup \{v \in V \mid v =^b t_P \wedge v >^a t_P\}$. Define $\delta_Q^{b,a}(P)$ as the unique vertex in $Q \cap B$ or as $\perp$ if that intersection is empty.*

▶ **Lemma 13** (⋆). *If P, Q are a,b-noncrossing paths with $\Delta^{a,b}(P,Q) \neq \emptyset$, then $\delta_P^{a,b}(Q) \neq \perp$ or $\delta_Q^{a,b}(P) \neq \perp$.*

The next lemma shows that if P and Q are not crossing but have a common area $\Delta_{a,b}$, then the path whose end vertex lies not in the common area $\Delta_{a,b}$ does not enter $\Delta_{a,b}$ at all.

▶ **Lemma 14** (⋆). *If P is an a-colored path and Q a b-colored path with $\delta_P^{a,b}(Q) \neq \perp$, then $\left(\overrightarrow{s_Q} \diamond \overrightarrow{t_Q}\right)^{a,b}$ is disjoint from $\left(\overrightarrow{s_P} \diamond \overrightarrow{\delta_P^{a,b}(Q)}\right)^{a,b} \cup \left(\overrightarrow{\delta_P^{a,b}(Q)} \diamond \overrightarrow{t_P}\right)^{a,b}$.*

▶ **Definition 15.** *Let P be an a-colored s_P-t_P-path and Q a b-colored s_Q-t_Q-path. We then define $\mathcal{C}_P^{a,b}(Q) := \{s_P, t_P, \mu_P^{a,b}(Q) \mid \mu \in \{\alpha, \omega, \partial, \varpi, \delta\}\} \setminus \{\perp\}$.*

The next proposition shows the sets $\mathcal{C}_P^{a,b}(Q)$ and $\mathcal{C}_Q^{a,b}(P)$ "characterize" the crossing of P and Q in the sense that any two shortest paths using these vertices have exactly the same vertex coordinates in the crossing.

▶ **Proposition 16** (⋆). *Let P and P' be a-colored s_P-t_P-paths, and let Q and Q' be b-colored s_Q-t_Q-paths. If $\mathcal{C}_P^{a,b}(Q) \subseteq P'$ and $\mathcal{C}_Q^{a,b}(P) \subseteq Q'$, then $\{v \in P' \mid v \in^{a,b} Q'\} =^{a,b} \{v \in P \mid v \in^{a,b} Q\}$.*

We now have all ingredients for the proof of Theorem 1.

Proof of Theorem 1. Let $I := (G = (V, E), k, ((s_1, t_1), (s_2, t_2))$ be an instance of 2-DSP. Compute $\vec{v}$ for all $v \in V$ via two breadth-first searches in $O(n + m)$ time. We assume without loss of generality that G is connected. To ensure that we report I being a *yes*-instance only if I is indeed a *yes*-instance, we perform a sanity-check in the very end to verify that our guesses were correct. Hence, we only need to show that we find in $O(nm)$ time a solution to I if there is one. To this end, assume there are disjoint shortest s_i-t_i-paths P_i for $i \in [2]$. By Lemma 7 we have three cases.

(Case 1): $\zeta(P_1) \cap \zeta(P_2)$ is empty. If $\Delta^{1,2}(P_1, P_2) = \emptyset$, then, by Lemma 6, a solution can easily be found by two independent breadth-first-searches. Otherwise, we guess in $O(n)$ time the vertex $\delta_{P_1}^{1,2}(P_2)$ on P_1 or $\delta_{P_2}^{2,1}(P_1)$ on P_2 from Definition 12. By Lemma 13, at least one of them exists (assume without loss of generality that $\delta_{P_1}^{1,2}(P_2)$ exists). By Lemma 14, $\delta_{P_1}^{1,2}(P_2) \neq \perp$ and any shortest s_1-$\delta_{P_1}^{1,2}(P_2)$-path ($\delta_{P_1}^{1,2}(P_2)$-t_1-path) is vertex disjoint from any shortest s_2-t_2-path. Hence, we now can check in $O(m)$ time whether I is a *yes*-instance.

(Case 2): $\zeta(P_1) \cap \zeta(P_2)$ has no point with integer coordinates. Then, we guess the four points surrounding $\zeta(P_1) \cap \zeta(P_2)$ in $O(n)$ time. This can be done in $O(n)$ time by guessing the vertex $\partial_{P_1}^{1,2}(P_2)$ and branch into two cases. Let p_{s_i} and p_{t_i} be the guessed points used by P_i such that $p_{s_i}^i + 1 = p_{t_i}^i$, for $i \in [2]$. Now we construct a directed graph D on the vertices V such that there is an arc (v, w) if $\{v, w\} \in E$, $\vec{v}^i + 1 = \vec{w}^i$, and $\overrightarrow{\{v, w\}} \subseteq \vec{s_i} \diamond p_{s_i} \cup p_{t_i} \diamond \vec{t_i}$ for some $i \in [2]$. Note that D is acyclic and that each s_i-t_i-path in D corresponds (same set of vertices) to a shortest s_i-t_i-path in G, and has an arc (v, w) such that $\vec{v} = p_{s_i}$ and $\vec{w} = p_{t_i}$. Hence, by Lemma 10 we can simply use two breadth-first-searches from s_1 and s_2 to find a solution.

(Case 3): $\zeta(P_1) \cap \zeta(P_2)$ has at least one point with integer coordinates. Without loss of generality, we assume that $\overrightarrow{\alpha_{P_1}^{1,2}(P_2)} = \overrightarrow{\alpha_{P_2}^{1,2}(P_1)}$, otherwise we swap the terminal pairs in the input instance. We guess in $O(n)$ time the discrete point $p \in \zeta(P_1)^{1,2} \cap \zeta(P_2)^{1,2}$ such that $\vec{p}^1$ is minimized. Let $A_i := \vec{s_i} \diamond \vec{p}$ and $B_i := \vec{p} \diamond \vec{t_i}$, for all $i \in [2]$. Now we construct a directed acyclic graph D on the vertices V such that there is an arc (v, w) if for some $i \in [2]$ we have (1) $\{v, w\} \in E$, (2) $\vec{v}^i + 1 = \vec{w}^i$, and (3) (a) $\vec{v} \in A_i \setminus B_i$ and $\vec{w} \in A_i$ or (b) $\vec{v} \in B_i$ and $\vec{w} \in B_i \setminus A_i$.

To add the edges with coordinates in $A_1 \cap B_2, A_2 \cap B_1$ to D, we observe that our assumption implies that $\zeta(P_1) \cap \zeta(P_2) \subseteq B_1 \cap B_2$. Hence, all edges where the vertices have coordinates in $A_i \cap B_j$ can only be used by either P_1 or P_2, for each $\{i, j\} = [2]$. Thus, we branch in four cases and add the edges accordingly. Note that D is acyclic and that each s_i-t_i-path in D corresponds to a shortest s_i-t_i-path in G going through point p. Hence, by Lemma 10 I is *yes*-instance if and only if there are disjoint s_i-t_i-path in D, for all $i \in [2]$. Thus, we apply an $O(n+m)$-time algorithm of Tholey [14] for 2-Disjoint Paths on a DAG. This yields a total running time of $O(nm)$. ◀

4 The Geometry of Many Shortest Paths

In the previous section, we looked at two shortest paths P and Q from s_P to t_P and s_Q and t_Q respectively. We showed that selecting at most six vertices from P and Q (three per path; see Definition 15) is sufficient to ensure that each pair of shortest s_P-t_P- and s_Q-t_Q-paths

that also contain the vertices $\mathcal{C}_P^{a,b}(Q)$ and $\mathcal{C}_Q^{a,b}(P)$, respectively, "behave" like P and Q in the sense of using the same coordinates or not (see Proposition 16). In this section, we define a set $\mathcal{C}$, $|\mathcal{C}| \in O(k \cdot k!)$, that basically ensures the same properties for k paths. To formalize our goal in this section, we introduce the concept of *avoiding* paths.

▶ **Definition 17** (I-avoiding). *Let $\emptyset \subset I \subseteq [k]$. We say that two paths P and Q are* I-avoiding *if $p \notin^I Q$ holds for every internal vertex p of P and $q \notin^I P$ for every internal vertex q of Q.*

We further call two vertex pairs (s_p, t_p) and (s_q, t_q) I-avoiding, if $(\vec{s_p}^I \diamond \vec{t_p}^I) \cap (\vec{s_q}^I \diamond \vec{t_q}^I) \subseteq \{\vec{s_p}^I, \vec{t_p}^I\} \cap \{\vec{s_q}^I, \vec{t_q}^I\}$.

Note that being I-avoiding implies being I'-avoiding for all $I' \supseteq I$. We use *avoiding* as a shorthand for $[k]$-avoiding. The reason for defining *avoiding* in such a way that the endpoints of the paths play a special role is as follows: When partitioning a colored path $P = v_1 \dots v_\ell$ into two subpaths $P' = v_1 \dots v_j$ and $P'' = v_j v_{j+1} \dots v_n$, then these two subpaths obviously share exactly one vertex, namely v_j. However, we still want to call the pairs $(\vec{v_1}, \vec{v_j})$ and $(\vec{v_j}, \vec{v_\ell})$ avoiding since these two subpaths cannot have any other intersection besides $\vec{v_j}$.

Two paths P_1 and P_2 are internally disjoint if neither of them contains an internal vertex of the other path. Avoiding paths are clearly internally disjoint.

▶ **Observation 18.** *Let P, Q be two avoiding paths. Then P is internally disjoint from Q.*

Moreover, avoiding vertex pairs (s, t) and (u, w) ensure that the corresponding shortest s-t- and u-v-paths are internally disjoint.

▶ **Lemma 19** ($\star$). *Let (s, t) and (u, w) be two colored pairs of vertices. If (s, t) and (u, w) are avoiding, then each shortest s-t-path is internally disjoint from each shortest u-w-path.*

With the notation of avoiding pairs, we can formulate our goal for this section. To this end, fix a solution $\mathcal{P} = (P_i)_{i \in [k]}$ for the k-DSP instance $(G, (s_i, t_i)_{i \in [k]})$, that is, P_i is the s_i-t_i-path in this solution. Essentially, we want to partition the paths in $\mathcal{P}$ into subpaths and assign labels (subsets of $[k]$) to each subpath such that the following holds:

(1.) Let P be a subpath with labels $\Phi \subseteq [k]$. For each $a \in \Phi$, P is a-colored.

(2.) Let P and Q be subpaths from s_P to t_P and s_Q and t_Q with labels $\Phi_P, \Phi_Q \subseteq [k]$ respectively. If $\Phi_P \neq \Phi_Q$, then (s_P, t_P) and (s_Q, t_Q) are avoiding.

Note that (2.) will be the central argument in our algorithm for k-DSP. The algorithm guesses the endpoints of these subpaths and based on (2.), the algorithm can compute the interior points of subpaths with different label sets independently of each other.

Note that for $k = 2$ the partition of P_1 and P_2 along the sets $\mathcal{C}_{P_1}^{1,2}(P_2)$ and $\mathcal{C}_{P_2}^{1,2}(P_1)$, respectively, satisfies the above two points: Each subpath of P_i, $i \in [2]$, has label i. Moreover, the subpaths between the α and ω-vertices have both labels 1 and 2. Hence, (1.) above is satisfied. Furthermore, (2.) essentially follows from Proposition 16.

We now give some intuition on how to lift this to arbitrary fixed k leading to Definition 20, a generalization of Definition 15. Initially, each path P_i has label i. Whenever two paths P_i and P_j in the solution intersect in the (i, j)-projection (that is, we have α and ω vertices), then the subpaths in the intersection gets both labels i and j. If a third path P' also intersects with the subpath of P_j that intersects with P_i, then we try to use the intersections to move the label i via path P_j to some subpath of P'. Generalizing this, we consider for each sequence $\sigma = (\ell_1, \dots, \ell_h)$ whether label ℓ_1 could be "transported" from P_{ℓ_1} to P_{ℓ_2}, from P_{ℓ_2} to P_{ℓ_3}, ..., and from $P_{\ell_{h-1}}$ to P_{ℓ_h}. The reason for doing it this way is that we will have for each such sequence $\sigma = (\ell_1, \dots, \ell_h)$ at most one consecutive subpath on P_{ℓ_h} that has label ℓ_1 transported via σ. While the idea of transporting labels would also work with triplets (transport label a via path P_b to path P_c), we do not have any bound on the number of resulting subpaths (as for each triplets there might be many such subpaths).

In order to formalize this idea and define the *crossing set* $\mathcal{C}$ of these paths (Definition 20), we need some notation. Let $\sigma = (\ell_1, \dots, \ell_h)$ be a sequence. We define $\text{set}(\sigma) := \{\ell_1, \dots, \ell_h\}$ to be the set with all entries of σ. For a path $P = v_0 \dots v_h$ and $1 \leq i < j \leq h$ let $P[v_i, v_j] := v_i \dots v_j$ be the subpath of P with endpoints v_i and v_j. The set $\mathcal{C}$ for each permutation σ of each $\Phi \subseteq [k]$ is recursively defined as follows. Therein, $\mathcal{T}$ describes the first and last vertex on the respective subpath having exactly the set of labels in Φ.

▶ **Definition 20.** *For each $\Phi \subseteq [k]$ and each permutation $\sigma = (\ell_1, \dots, \ell_h)$ of Φ set:*

- *If $h = 1$ with $\sigma = (i)$, then set $\mathcal{C}^\sigma := \mathcal{T}(\sigma) := \{s_i, t_i\}$.*
- *If $h = 2$ with $\sigma = (i, j)$, then set $\mathcal{T}(\sigma) := \{\alpha^{i,j}_{P_j}(P_i), \omega^{i,j}_{P_j}(P_i)\}$, and $\mathcal{C}^\sigma := \mathcal{C}^{i,j}_{P_j}(P_i)$.*
- *If $h \geq 3$, then let $\sigma_{start} := (\ell_1, \dots, \ell_{h-1})$, $\sigma_{end} := (\ell_2, \dots, \ell_h)$. If $\mathcal{T}(\sigma_{start}) = \{\perp\}$ or $\mathcal{T}(\sigma_{end}) = \{\perp\}$ or $Q := P_{\ell_{h-1}}[\mathcal{T}(\sigma_{start})] \cap P_{\ell_{h-1}}[\mathcal{T}((\ell_h, \ell_{h-1}))] = \emptyset$, then set $\mathcal{T}(\sigma) := \mathcal{C}^\sigma := \{\perp\}$. Otherwise, let $P := P_{\ell_h}[\mathcal{T}(\sigma_{end})]$. Then set $\mathcal{T}(\sigma) := \{\alpha^{\ell_1,\ell_h}_P(Q), \omega^{\ell_1,\ell_h}_P(Q)\}$. Moreover, set $\mathcal{C}^\sigma := \mathcal{C}^{\ell_1,\ell_h}_P(Q) \cup \mathcal{C}^{\ell_1,\ell_h}_Q(P)$.*

The set $\mathcal{C} := \bigcup_\sigma \mathcal{C}^\sigma$ is the crossing set *of $\mathcal{P}$.*

As subsequently shown, when transporting the labels via a sequence $\sigma = (\ell_1, \dots, \ell_h)$, the intersecting subpath in the target path P_{ℓ_h} agrees in all coordinates in $\text{set}(\sigma)$ with the subpath of $P_{\ell_{h-1}}$ where the label is transported from.

▶ **Lemma 21 (⋆).** *Let $\sigma := (\ell_1, \dots, \ell_h)$ be any permutation of any $\Phi \subseteq [k]$ with $|\Phi| = h \geq 2$. If $\mathcal{T}(\sigma) \neq \{\perp\}$, then $P_{\ell_h}[\mathcal{T}(\sigma)] =^\Phi Q'$ for some subpath Q' of $Q := P_{\ell_{h-1}}[\mathcal{T}(\sigma_{start})] \cap P_{\ell_{h-1}}[\mathcal{T}((\ell_h, \ell_{h-1}))]$ where $\sigma_{start} := (\ell_1, \dots, \ell_{h-1})$.*

We next formalize the notions used in the context of the intersection of $\mathcal{C}$ with the paths $\mathcal{P}$.

▶ **Definition 22.** *An i-*marble path *T is a set of vertices such that $\{s_i, t_i\} \subseteq T$ and for each $u, v \in T$ the pair (u, v) is i-colored. A* segment *S of a i-marble path T is a subset of T containing two vertices denoted* $\text{start}(S)$ *and* $\text{end}(S)$ *and all vertices $v \in T$ with* $\text{start}(S) <^i v <^i \text{end}(S)$*. A segment is minimal if it contains exactly two vertices.*

We say a segment is i-colored, if $(\text{start}(S), \text{end}(S))$ is i-colored. We say two segments S and S' are avoiding if the minimal subsegments of S and S' are pairwise avoiding.

We say a path P follows *S if it is i-colored, has end vertices* $\text{start}(S)$ *and* $\text{end}(S)$*, and $S \subseteq V(P)$.*

To prove the central statement of this section, we need to formalize the labels of a segment. To this end, let S_i be a segment of P_i. Then set

$$\mathit{labels}[S_i] := \{a \mid \exists \sigma = (a = \ell_1, \dots, \ell_h = i), h \geq 1 \colon S_i \subseteq P_i[\mathcal{T}(\sigma)]\}.$$

▶ **Proposition 23 (⋆).** *For $i, j \in [k]$ let $S_i \subseteq V(P_i) \cap \mathcal{C}$ and $S_j \subseteq V(P_j) \cap \mathcal{C}$ be two minimal segments. If $\mathit{labels}[S_i] \neq \mathit{labels}[S_j]$, then S_i and S_j are avoiding.*

5 The Algorithm: Utilizing the Geometry

In this section, we finally present the algorithm behind Theorem 2. Pseudo-code for this algorithm is listed in Algorithm 1. In a nutshell, we first guess all marble paths T_i and the map $\mathcal{T}$ corresponding to the crossing set $\mathcal{C}$ of some solution (if one exists). Then, we find all minimal segments of each marble path T_i and partition them such that (1) all minimal segments in the same part of the partition are strictly monotone in the same set of

Algorithm 1 Our algorithm for k-DSP.

```
1  function solve(G, (s_i, t_i)_{i∈[k]})
2      foreach guess (T_i)_{i∈[k]}, Ends of the crossing set do
           /* We assume now that the guess is correct, that is, for a
              solution P = (P_i)_{i∈[k]} we have T_i = C ∩ P_i, i ∈ [k], & Ends = T   */
3          P_i = ∅, for all i ∈ [k]
4          foreach minimal segment S of some T_i, i ∈ [k] do
5              marks[S] ← ∅
6              foreach sequence σ = (ℓ_1, ℓ_2, ..., i) with S ⊆ Ends(σ) do
7                  marks[S] ← marks[S] ∪ set(σ)
8              j ← min marks[S]
9              x ← arg min{ v⃗^j | v ∈ {start(S), end(S)} }
10             y ← arg max{ v⃗^j | v ∈ {start(S), end(S)} }
11             P_j = P_j ∪ {(x, y)}
12         foreach j ∈ [k] do Order P_j = ((x_1, y_1), (x_2, y_2), ...) so that x⃗_1^j ≤ x⃗_2^j ≤ ...
13         if all instances (D(G, i), P_i), i ∈ [k] of |P_i|-Disjoint Paths on i-layered
            DAGs are yes-instances and the combined solutions form a solution of
            k-DSP then return yes
14     return no
```

coordinates, and (2) two minimal segments in distinct parts of the partition are avoiding. The crucial improvement over the algorithm of Lochet [12] is that our partition is much smaller. Afterwards, we find shortest disjoint paths for each part of our partition separately via dynamic programming.

To this end, we introduce *c-layered DAGs* and the p-Disjoint Paths on DAGs problem. For a graph G with coordinates $\vec{v}$ (as defined in Section 3) for all $v \in V$, the c-layered DAG $D(G, c)$ of G is the directed graph $(V(G), A)$, where $(x, y) \in A$ if and only if $\{x, y\} \in E(G)$ and $\vec{y}^c - \vec{x}^c = 1$. Crucial here is the following simple observation.

▶ **Observation 24.** *A path P in G is c-colored if and only if $(V(P), \{(u, v) \mid \{u, v\} \in E(P), \vec{v}^c - \vec{u}^c = 1\})$ is a path in the c-layered DAG of G.*

In p-Disjoint Paths on DAGs we are given a directed acyclic graph D and a list $(s_i, t_i)_{i\in[p]}$ of (possibly intersecting) terminal pairs, and ask whether there are pairwise internally disjoint s_i-t_i-path in D, for each $i \in [p]$. Fortune et al. [7] showed a $n^{O(p)}$-time algorithm for p-Disjoint Path on DAGs. A straight-forward dynamic program yields a more specific running time of $O(n^{p+1})$.

▶ **Lemma 25** (⋆). *An instance of p-Disjoint Paths on DAGs on a graph with n vertices can be solved in $O(n^{p+1})$ time.*

▶ **Lemma 26** (⋆). *Algorithm 1 runs in $O(k \cdot n^{16k \cdot k! + k + 1})$ time.*

For the correctness of Algorithm 1, we need to show that each part of the partition of minimal segments can be solved independently. This follows from Proposition 23 together with the fact that Algorithm 1 exhaustively tries all possibilities for the crossing set $\mathcal{C}$. Together with Lemma 26, this implies Theorem 2.

6 Conclusion

We provided an improved polynomial-time for k-DSP. However, while the running time of our algorithm can certainly be slightly improved by some case distinctions and a more careful analysis, the algorithm is still far from being practical. Reducing the factor in the exponent to a polynomial in k is a clear challenge for future work. Considering the fine-grained complexity of 2-DSP, it would be interesting to know whether there are running time barriers based on e. g. the Strong Exponential Time Hypothesis.

Concerning generalizations of k-DSP, we believe that we can modify our algorithm in a straight-forward way to work with positive edge-lengths. However, the case of non-negative edge-lengths seems much more difficult. Our basic geometric observations made in Section 3 crucially depend on the fact that we are looking for shortest paths. Thus, if there are no k disjoint shortest paths, then computing k disjoint paths minimizing their total length in polynomial time is still an open problem for $k \geq 3$ (for $k = 2$ Björklund and Husfeldt [4] provided a randomized $O(n^{11})$ time algorithm and moreover proved that this problem is contained in NC if the graph is planar and cubic [3]).

References

1 Maxim Akhmedov. Faster 2-disjoint-shortest-paths algorithm. In *Proceedings of the 15th International Computer Science Symposium in Russia (CSR '20)*, volume 12159 of *Lecture Notes in Computer Science*, pages 103–116. Springer, 2020. doi:10.1007/978-3-030-50026-9_7.

2 Kristof Berczi and Yusuke Kobayashi. The Directed Disjoint Shortest Paths Problem. In *Proceedings of the 25th Annual European Symposium on Algorithms (ESA '17)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.13.

3 Andreas Björklund and Thore Husfeldt. Counting shortest two disjoint paths in cubic planar graphs with an NC algorithm. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC '18)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.19.

4 Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. *SIAM Journal on Computing*, 48(6):1698–1710, 2019. doi:10.1137/18M1223034.

5 Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, 1998. doi:10.1016/S0166-218X(97)00121-2.

6 Fedor V. Fomin, Dániel Marx, Saket Saurabh, and Meirav Zehavi. New Horizons in Parameterized Complexity (Dagstuhl Seminar 19041). *Dagstuhl Reports*, 9(1):67–87, 2019. doi:10.4230/DagRep.9.1.67.

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

8 Marinus Gottschau, Marcus Kaiser, and Clara Waldmann. The undirected two disjoint shortest paths problem. *Operations Research Letters*, 47(1):70–75, 2019. doi:10.1016/j.orl.2018.11.011.

9 Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975. doi:10.1002/net.1975.5.1.45.

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

11 Yusuke Kobayashi and Ryo Sako. Two disjoint shortest paths problem with non-negative edge length. *Operations Research Letters*, 47(1):66–69, 2019. doi:10.1016/j.orl.2018.11.012.

12 William Lochet. A polynomial time algorithm for the k-disjoint shortest paths problem. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA '21)*, pages 169–178. SIAM, 2021. doi:10.1137/1.9781611976465.12.

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

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