Algorithmica (2024) 86:1512–1544
https://doi.org/10.1007/s00453-023-01200-5

Connectivity with Uncertainty Regions Given as Line Segments

Sergio Cabello[1,2] · **David Gajser**[2,3]

Received: 17 March 2023 / Accepted: 11 December 2023 / Published online: 9 January 2024

Abstract
For a set $\mathcal{Q}$ of points in the plane and a real number $\delta \geq 0$, let $\mathbb{G}_\delta(\mathcal{Q})$ be the graph defined on $\mathcal{Q}$ by connecting each pair of points at distance at most δ.We consider the connectivity of $\mathbb{G}_\delta(\mathcal{Q})$ in the best scenario when the location of a few of the points is uncertain, but we know for each uncertain point a line segment that contains it. More precisely, we consider the following optimization problem: given a set $\mathcal{P}$ of $n-k$ points in the plane and a set $\mathcal{S}$ of k line segments in the plane, find the minimum $\delta \geq 0$ with the property that we can select one point $p_s \in s$ for each segment $s \in \mathcal{S}$ and the corresponding graph $\mathbb{G}_\delta(\mathcal{P} \cup \{p_s \mid s \in \mathcal{S}\})$ is connected. It is known that the problem is NP-hard. We provide an algorithm to exactly compute an optimal solution in $\mathcal{O}(f(k)n \log n)$ time, for a computable function $f(\cdot)$. This implies that the problem is FPT when parameterized by k. The best previous algorithm uses $\mathcal{O}((k!)^k k^{k+1} \cdot n^{2k})$ time and computes the solution up to fixed precision.

Keywords Computational geometry · Uncertainty · Geometric optimization · Fixed parameter tractability · Parametric search

1 Introduction

For a set $\mathcal{Q}$ of points in the plane and a real value $\delta \geq 0$, let $\mathbb{G}_\delta(\mathcal{Q})$ be the graph with vertex set $\mathcal{Q}$ and edges connecting each pair of points p, q at Euclidean distance at most δ. Connectivity of the graph $\mathbb{G}_\delta(\mathcal{Q})$ is one of the basic properties associated to the point set $\mathcal{Q}$. For example, if the points represent devices that can communicate

✉ Sergio Cabello
sergio.cabello@fmf.uni-lj.si

David Gajser
david.gajser@um.si

1 Faculty of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia

2 Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

3 Faculty of Natural Sciences and Mathematics, University of Maribor, Maribor, Slovenia

and δ is the broadcasting range of each device, then the connectivity of $\mathbb{G}_\delta(\mathcal{Q})$ reflects whether all the devices form a connected network and they can exchange information, possibly through intermediary devices.

In this work we consider the problem of finding the smallest δ such that $\mathbb{G}_\delta(\mathcal{Q})$ is connected, when some of the points from $\mathcal{Q}$ are to be chosen from prescribed regions. More precisely, we consider the following optimization problem.

CONNECTIVITY
Given a set $\mathcal{U} = \{U_1, \ldots, U_k\}$ of regions in the plane and a set $\mathcal{P} = \{p_{k+1}, p_{k+2}, \ldots, p_n\}$ of points in the plane, find

$$\begin{aligned} \delta^* = \min \ & \delta \\ \text{s.t. } & p_i \in U_i, \quad \text{for } i = 1, \ldots, k \\ & \mathbb{G}_\delta(\{p_1, \ldots, p_n\}) \text{ is connected.} \end{aligned}$$

In this work we will provide efficient algorithms for the CONNECTIVITY problem when the regions are line segments and k is small. See Fig. 1 for an example.

There are other ways to characterize the connectivity of the graph $\mathbb{G}_\delta(\mathcal{Q})$. Let $D(p, r)$ denote the closed disk of radius r centered at p. Then, the graph $\mathbb{G}_\delta(\mathcal{Q})$ is connected if and only if $\bigcup_{p\in\mathcal{Q}} D(p, \delta/2)$ is a connected set. Another characterization is provided by the Euclidean Minimum Bottleneck Spanning Tree of $\mathcal{Q}$, denoted by MBST($\mathcal{Q}$), a spanning tree of $\mathcal{Q}$ where the length of the longest edge, called bottleneck edge, is minimized; a formal definition is given below. The graph $\mathbb{G}_\delta(\mathcal{Q})$ is connected if and only if MBST(Q) uses only edges of length at most δ.

It follows that the problem CONNECTIVITY is equivalent to the following problems:

- Choose a point p_i per region U_i, where $i = 1, \ldots, k$, in such a way that $\bigcup_{i=1,\ldots,n} D(p_i, r)$ is connected and r is the smallest possible; here the minimum r is $\frac{\delta^*}{2}$.
- Choose a point p_i per region U_i, where $i = 1, \ldots, k$, in such a way that the $MBST$ on points $p_1, p_2, \ldots, p_n$ has shortest bottleneck edge.

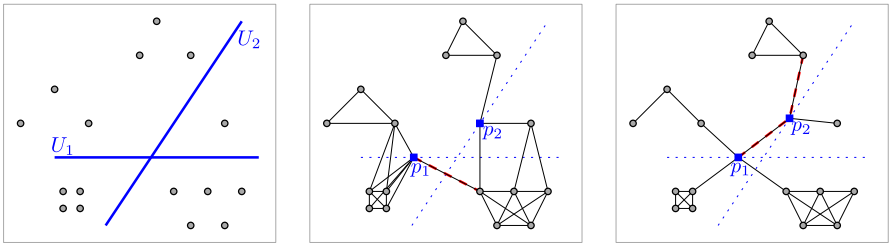

Fig. 1 Example of the instances we consider, where the regions are line segments. Left: input data with two segments. Center and right: two possible choices of points $(p_1, p_2) \in U_1 \times U_2$ and the resulting graph G_δ for the minimum δ that makes G_δ connected. The edges of length δ are marked in dashed red; δ is different in each case

1.1 Related Work

The problem we consider, CONNECTIVITY, was introduced by Chambers et al. [9] under the name of *Best-Case Connectivity with Uncertainty*. In this setting each region U_i is the uncertainty region for the point p_i. They also considered the worst-case connectivity scenario, where one seeks for the minimum δ such that $\mathbb{G}_\delta(\{p_1, \ldots, p_n\})$ is connected for *all* choices $p_i \in U_i$, where $i = 1, \ldots, k$. Thus, while in the best case we want to select points to achieve connectivity, in the worst case we want to guarantee connectivity for all possible choices.

Chambers et al. [9] showed that CONNECTIVITY is NP-hard even in the very restricted case when the uncertainty regions are vertical line segments of unit length or when the uncertainty regions are axis-parallel unit squares. For the case when the regions are line segments, they provide an algorithm that in $\mathcal{O}((k!)^k k^{k+1} \cdot (n+k)^{2k})$ time computes an optimal solution up to fixed precision. The precision appears because of rounding the intermediary computations.

The case when the uncertainty regions are the whole plane, thus $U_1 = \cdots = U_k = \mathbb{R}^2$ has been studied earlier under names like bottleneck Steiner tree problem or bottleneck k-Steiner tree problem. The results by Sarrafzadeh and Wong [24] imply that the problem is NP-hard. Ganley and Salowe [13] provided an approximation algorithm; they also considered the rectilinear metric. Wang and Li [29] provided approximation algorithms, while Wang and Du [28] provided inapproximability results, assuming P≠NP. Bae et al. [3] showed that the case of $k = 1$ can be solved exactly in $\mathcal{O}(n \log n)$ time, and the case $k = 2$ can be solved in $\mathcal{O}(n^2)$ time. Bae et al. [2] showed that the problem can be solved in $f(k) \cdot (n^k + n \log n)$ time, for some function $f(\cdot)$. This last paper provides algorithms for the L_1 and L_∞ metrics with a better running time, $f(k) \cdot (n \log^2 n)$. Very recently, Bandyapadhyay et al. [4] have shown that the problem can be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time for some function $f(\cdot)$, which means that the bottleneck k-Steiner tree problem in the plane is fixed-parameter tractable with respect to k. The techniques can also be used in other L_p metrics.

Instead of minimizing the longest edge of the spanning tree (bottleneck version), one could minimize the total length of the tree. In the k-Steiner tree problem, we are given a set $\mathcal{P}$ of points, and we want compute a shortest Steiner tree for $\mathcal{P}$ with at most k Steiner points. This is similar to the CONNECTIVITY problem because we can take $U_1 = \cdots = U_k = \mathbb{R}^2$, but the optimization criteria is the sum of the lengths of the edges. The 1-Steiner tree problem was solved in $\mathcal{O}(n^2)$ time with the algorithm of Georgakopoulos and Papadimitriou [14]. Brazil et al. [7] solved the k-Steiner tree problem in $\mathcal{O}(n^{2k})$ time. Their technique can be adapted to the problem CONNECTIVITY and, assuming that the uncertainty regions are convex of constant description size, the problem reduces to $\mathcal{O}(n^{2k})$ instances of quasiconvex programming [12], each with $\mathcal{O}(k)$ variables and constraints.

Closer to our setting is the work of Bose et al. [6], who considered the version of the k-Steiner tree problem where the points have to lie on given lines, and showed how to solve it in $\mathcal{O}(n^k + n \log n)$ time. Like the work of Brazil et al. [7], their technique can be adapted to the problem CONNECTIVITY with uncertain segment regions. With

this, the CONNECTIVITY problem with uncertain segments reduces to $\mathcal{O}(n^k)$ instances of quasiconvex programming [12], each with $\mathcal{O}(k)$ variables and constraints.

The problem CONNECTIVITY is an instance of the paradigm of computing optimal geometric structures for points sets with uncertainty or imprecision: each point is specified by a region in which the point may lie. In this setting, we can model that the position of some points is certain by taking the region to be a single point. Usually one considers the maximum and the minimum value that can be attained for some geometric structure, such as the diameter, the area of the convex hull, or the minimum-area rectangle that contains the points. This trend was started by Löffler and van Kreveld; see for example [18, 19, 27] for some of the earlier works in this paradigm.

1.2 Our Results

We consider the CONNECTIVITY problem when each of the regions are line segments. To emphasize this, we use $\mathcal{S}$ instead of $\mathcal{U}$ and s_i instead of U_i.

First, we consider the decision version of the problem: Given a set $\mathcal{S} = \{s_1, \ldots, s_k\}$ of k segments in the plane, a set $\mathcal{P} = \{p_{k+1}, p_{k+2}, \ldots, p_n\}$ of $n-k$ points in the plane, and a value $\delta \geq 0$, decide whether there exist points $p_i \in S_i$ for $i = 1, \ldots, k$ such that $\mathbb{G}_\delta(\{p_1, \ldots, p_n\})$ is connected. We call this version of the problem DCONNECTIVITY.

Our first main result is showing that DCONNECTIVITY for segments can be solved using $f(k)n \log n$ operations, for some computable function $f(\cdot)$. In fact, after a preprocessing of the instance taking $\mathcal{O}(k^2 n \log n)$ time, we can solve DCONNECTIVITY for any $\delta \geq 0$ using $f(k)n$ operations. For this we use the following main ideas:

- Instead of searching for connectivity, we search for a MBST.
- It suffices to restrict our attention to MBST of maximum degree 5.
- The MBST will have at most $\mathcal{O}(k)$ edges that are not part of the minimum spanning tree of $\mathcal{P}$.
- We can iterate over all the possible combinatorial ways how the uncertain points interact with the rest of the instance. Such interaction is encoded by a so-called topology tree with $\mathcal{O}(k)$ nodes, and there are $k^{\mathcal{O}(k)}$ different options.
- For each topology tree τ, we can employ a bottom-up dynamic programming across τ to describe all the possible placements of points on the segment that are compatible with the subtree.

Our strongest result is showing that CONNECTIVITY for segments can be solved using $f(k)n \log n$ operations, for some other computable function $f(\cdot)$. For this we use parametric search [20, 21], a generic tool to transform an algorithm for the decision version of the problem into an algorithm for the optimization problem. We provide a careful description of the challenges that appear when using parametric search in our setting. While we can use Cole's technique [10] in one of the steps, we provide an alternative that is simpler, self-contained and uses properties of the problem. Eventually, we manage to solve the optimization problem without increasing in the time complexity of the algorithm the dependency on n; the dependency on k increases slightly.

Our result shows that CONNECTIVITY for line segments is fixed-parameter tractable when parameterized by the number k of segments. The running time of our algorithms

are a large improvement over the best previous time bound of $\mathcal{O}(n^{2k})$ by Chambers et al. [9] and the bound of $\mathcal{O}(n^k + n \log n)$ that could be obtained adapting the approach of Bose et al. [6].

Compared to the work of Chambers et al. [9], we note that they did not consider the decision problem, but instead guessed a *critical* path that defines the optimal value. Then they show that there are $\mathcal{O}(n^{2k})$ critical paths. Compared to the work of Bose et al. [6] and Brazil et al. [7], we first note that they are minimizing the sum of the length of the edges. Optimizing the sum is usually harder than optimizing the bottleneck value. Also, considering the decision problem is often useful for the bottleneck version because it reduces the number of combinatorial options to consider, but this benefit is rarely present when minimizing the sum. To be more precise, in the decision version of our bottleneck problem, to extend a partial solution we only need to know whether it can be connected to a connected subgraph, but we do not care to which vertex of the connected subgraph we are connecting. In contrast, when minimizing the sum of the lengths one has to carry the information of whom do you connect to and how much it costs. This means that we have to carry a description of the cost function, which has a larger combinatorial description complexity.

When $k = O(1)$, our algorithms take $O(n \log n)$ time, which is asymptotically optimal: since finding a maximum-gap in a set of n unsorted numbers has a lower bound of $\Omega(n \log n)$ in the algebraic decision tree model, the problem of finding a MBST also takes $\Omega(n \log n)$, even without uncertainty regions.

In our problem, we have to be careful about the computability of the numbers appearing through the computation. It is easy to note that we get a cascading effect of square roots. A straightforward approach is to round the numbers that appear through the computation to a certain precision and bound the propagation of the errors. This is easy and practical, but then we do not get exact results.

The numbers computed through our algorithm have algebraic degree over the input numbers that depends on k, and thus can be manipulated exactly if we assume that the input numbers are rational or of bounded algebraic degree. The actual running time to manipulate these numbers exactly depends on k and the assumptions on the input numbers, and this is hidden in the function $f(\cdot)$. Below we provide background on algebraic numbers and computation trees, the tool we use to manipulate numbers. Our running times are stated assuming exact computation.

To summarize, there are two sources that make the dependency on k at least exponential: the number of topology trees considered in the algorithm is $k^{\mathcal{O}(k)}$, and we are manipulating $\Theta(n)$ numbers of algebraic degree at least $2^{\Omega(k)}$.

Organization The rest of the paper is organized as follows. In Sect. 2 we explain the notation and some of the concepts used in the paper. We also provide some basic geometric observations. In Sect. 3 we provide a careful description of geometric operations that will be used in the algorithm. We pay attention to the details to carry out the algebraic degree of the operations. In Sect. 4 we provide the algorithm for the decision problem. In Sect. 5 we analyze a function that will appear when analyzing parametric search; we need to bound the algebraic degree of certain equations that appear in the algorithm. In Sect. 6 we provide the optimization algorithm using the paradigm of parametric search. We conclude in Sect. 7.

2 Notation, Numbers and Preliminary Results

2.1 Notation

For each positive integer n, we use $[n] = \{1, \ldots, n\}$. For each set A and $t \in \mathbb{N}$, we use $\binom{A}{t}$ to denote the set of all subsets of A with t elements.

All graphs in this paper will be undirected. Hence, each graph will be given as an ordered pair $G = (V, E)$, where V is the set of its vertices and $E \subseteq \binom{V}{2}$ is the set of its edges. We also write $V(G)$ and $E(G)$ for the sets of vertices and edges of G, respectively. We use the notation uv for the edge with vertices u and v. The graph G is *edge-weighted*, if it is accompanied by a weight function $w : E \to \mathbb{R}_{\geq 0}$. The *weight* of an edge-weighted subgraph H, denoted $w(H)$, is the sum of weights of all of its edges.

If $\mathcal{Q}$ is a set of points in the plane, then we denote by $K_{\mathcal{Q}}$ the complete graph on vertices $\mathcal{Q}$. Such a graph is naturally accompanied by a weight function on edges that we call *edge length*. In this setting, the edge length of the edge uv, where $u, v \in \mathcal{Q}$, is the Euclidean distance between u and v, denoted by $d(u, v)$. We also write $|uv| = d(u, v)$.

Let $G = (V, E)$ be an edge-weighted graph. If T is a spanning tree of G, a *bottleneck edge* of T is an edge in T with largest weight. We call T a *minimum bottleneck spanning tree* or *MBST* of G, if its bottleneck edge has the smallest weight among all bottleneck edges of spanning trees of G. A *minimum spanning tree* or *MST* of G is a spanning tree of G that has minimum weight, over all spanning trees of G.

We will not distinguish between points and their position vectors. For two sets A and B in the plane, their *Minkowski sum* is $A \oplus B = \{a + b \mid a \in A, b \in B\}$. Recall that $D(p, r)$ is the closed disk with center p and radius r. Then $A \oplus D(0, r)$ is precisely the set of points at distance at most r from some point of A. We call this set the r-neighborhood of A. Any segment in this paper will be a line segment. A segment between two points a and b will be denoted as $\overline{ab}$. For a point a, we will use $||a||$ to denote the distance from a to the origin.

2.2 Representation of Numbers and Algebraic Operations

Each number that will be computed through our algorithm will be obtained from previous numbers by either one of the usual arithmetic operations (addition, subtraction, multiplication or division), by computing a square root or by solving a polynomial equation of degree at most $2^{\mathcal{O}(k)}$. Each number that is computed inside the algorithm has its *computation tree*. This is a rooted tree that has the just described operations as internal vertices and input values as leaves. Each vertex in the tree represents a number that is computed by the operation described in this vertex applied to its descendants. See Fig. 2, left, for an example.

Computer algebra provides tools to manipulate and compare these numbers exactly; see for example [5, Section 10.4] or [30]. Exact computation is a paradigm promoted within Computational Geometry [8, 16, 17, 26].

We will show that the depth of computation trees for numbers inside our algorithms will depend only on k. It follows that in our algorithms the time complexity of each of

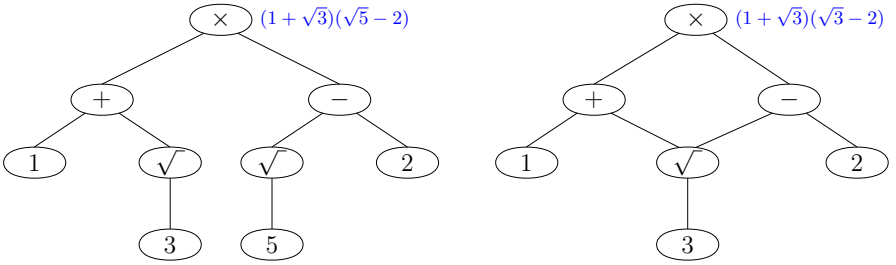

Fig. 2 Left: Example of computation tree for $(1+\sqrt{3})(\sqrt{5}-2)$. Right: Example of a directed acyclic graph for computing $(1+\sqrt{3})(\sqrt{3}-2)$

the numerical operations will always be a function only of k, independent of n. To avoid a cumbersome description, **we will assume in the description that the manipulation of numbers takes $\mathcal{O}(1)$ time**. Let us note here that our algorithms will not compute numbers directly by their computation trees, rather by related rooted directed acyclic graphs which are essentially computation trees, but with joint nodes that produce the same number using the same operations. This is a known concept, used for example in [22]. To give an example, if we consider a slightly modified example from Fig. 2, left, that computes $(1+\sqrt{3})(\sqrt{3}-2)$, we see that we do not need a node for number 5 and that we only need to compute $\sqrt{3}$ once, hence we need 2 fewer nodes. See Fig. 2, right. Note that such a transformation does not change the depth of the computation.

The following discussion is about bounding the algebraic degree of the numbers appearing in the algorithm and it is aimed to readers familiar with algebraic computations. In our final results we do not talk explicitly about the algebraic degree of the numbers and it suffices to know that the problem of performing ℓ algebraic operations on at most ℓ numbers is decidable for any ℓ.

A possible way to represent such a number α is as a univariate polynomial $P(x)$ with integer coefficients together with an isolating interval I; this is an interval with the property that α is the unique root of $P(x)$ inside I. The minimum degree of the polynomial representing α is the *algebraic* degree of α (over the integers).

It is known that if α and $\beta \neq 0$ are numbers of algebraic degree k, then $\alpha \pm \beta$, $\alpha \cdot \beta$, α/β and $\sqrt{\alpha}$ have algebraic degree at most k^2. Moreover, if α is the root of a polynomial of degree d with coefficients of algebraic degree k, then α has algebraic degree $\mathcal{O}(dk^d)$. To see this, we first construct a common field extension for all the coefficients, which will have degree $\mathcal{O}(k^d)$, and then use the relation of the degree between towers of field extensions.

In our decision algorithm, the numbers used in our computations have a computation tree of depth $\mathcal{O}(k)$ on the input numbers, with internal nodes containing only arithmetic operations and square roots. Therefore, we employ numbers of algebraic degree $2^{\mathcal{O}(k)}$. For the optimization problem, at the leaves of the computation tree of some numbers we will also have a root of a polynomial of degree $2^{\mathcal{O}(k)}$. For a computation tree, it is always the same root of a polynomial that is being used. Therefore, for the computation, it suffices to work with an extension field of all the input numbers, which has degree $2^{\mathcal{O}(k)}$ because there is a single number of algebraic degree $2^{\mathcal{O}(k)}$. Therefore, also in

the optimization problem, the numbers involved in the computation have algebraic degree $2^{\mathcal{O}(k)}$.

2.3 Properties of Minimum Trees

In this section we present some well known claims that will be used later. The first property is a standard consequence of Kruskal's algorithm to compute the MST.

Claim 1 *In any edge-weighted connected graph, all MSTs have the same weight of the bottleneck edge which is the same as the weight of a bottleneck edge in any MBST. In particular, each MST is also a MBST.*

The following result is also well known; see for example [23].

Claim 2 *For each non-empty set $\mathcal{Q}$ of points in the plane, there exists a MST and a MBST of the complete graph $K_{\mathcal{Q}}$ with maximum degree at most 5.*

3 Geometric Computations

In this section we describe representations of geometric objects that we will use and we present some basic geometric computations that will be needed in the algorithms.

Each line segment s is determined by a quadruple (p_s, e_s, a_s, b_s), where p_s is an arbitrary point on the line supporting s, e_s is a unit direction vector of s, and real numbers $a_s \leq b_s$ determine the endpoints of s, which are $p_s + a_s e_s$ and $p_s + b_s e_s$. When $a_s = b_s$, the segment degenerates to a single point; we keep calling it a segment to avoid case distinction. We will write $s = (p_s, e_s, a_s, b_s)$. We could easily allow for non-unit vectors e_s, but then some of the equations below become a bit more cumbersome. We will use such representations of line segments because we will often consider their subsegments, hence p_s and e_s will remain constant and only a_s and b_s will change.

A *segmentation* on a line L denotes a union of pairwise-disjoint line segments on L. Some of the segments may be a single point. If the line is given by a position vector of some point p on the line and a unit direction vector e, then we will represent a segmentation of this line with N disjoint subsegments and points as a $(2N+2)$-tuple

$$X = (p, e, a_1, b_1, a_2, b_2, \ldots, a_N, b_N),$$

where $a_i \leq b_i$, for each $i \in [N]$, and $b_i < a_{i+1}$, for each $i \in [N-1]$. For each $i \in [N]$, the line segment (p, e, a_i, b_i) is part of the segmentation X. We call N the size of the segmentation X. See Fig. 3 for an example.

I) Intersection of two lines Suppose we are given two lines $L_1 \equiv p_1 + t_1 e_1$ and $L_2 \equiv p_2 + t_2 e_2$, where p_1 and p_2 are points on the first and second line, respectively, e_1 and e_2 are their unit direction vectors, respectively, and $t_1, t_2 \in \mathbb{R}$ are parameters. We would like to compute $L_1 \cap L_2$.

If the lines are parallel, which is equivalent to $e_1 = \pm e_2$, then we have two options. If the system of two linear equations in one unknown $p_2 = p_1 + t e_1$ has some solution, then the lines are equal and $L_1 \cap L_2 = L_1$. Otherwise, $L_1 \cap L_2 = \emptyset$.

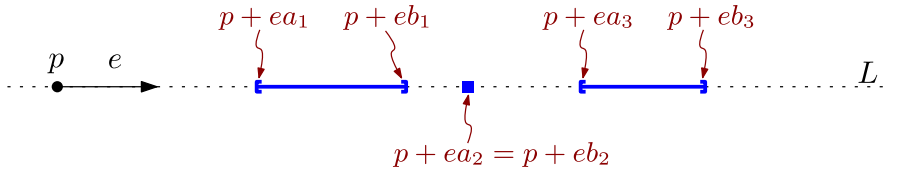

Fig. 3 A segmentation of size 3 where the second segment degenerates to a point

If the lines are not parallel, then the system of two linear equations with two unknowns $p_1 + t_1 e_1 = p_2 + t_2 e_2$ has a unique solution (t_1^*, t_2^*). In this case $L_1 \cap L_2$ contains exactly one point, namely $p_1 + t_1^* e_1 = p_2 + t_2^* e_2$.

In all cases, we can compute $L_1 \cap L_2$ with $\mathcal{O}(1)$ arithmetic operations.

II) **Intersection of a circle with a line** Suppose we are given a circle (curve) C with center at c and radius $\delta > 0$, and a line $L \equiv p + te$, where p is a point on the line, e is its unit direction vector and $t \in \mathbb{R}$ is a parameter. We would like to compute the intersection $C \cap L$.

This means that we need to solve the equation $||p+te-c|| = \delta$, which is quadratic in the unknown $t \in \mathbb{R}$. This is equivalent to solving

$$||p + te - c||^2 = \delta^2,$$

which can be rewritten as

$$t^2 + 2te \cdot (p - c) + ||p - c||^2 - \delta^2 = 0.$$

Let $\Delta = 4\,(e \cdot (p - c))^2 - 4||p - c||^2 + 4\delta^2$ be the discriminant of this equation in t. We consider the following 3 cases.

(i) $\Delta < 0$. In this case, $C \cap L = \emptyset$.
(ii) $\Delta = 0$. In this case, there is one solution of our quadratic equation, which is $t_0 = e \cdot (c - p)$. Hence, $C \cap L = \{p + t_0 e\}$. The line L is tangent to C.
(iii) $\Delta > 0$. In this case, there are two solutions of our quadratic equation, which are

$$t_1 = e \cdot (c - p) - \frac{1}{2}\sqrt{\Delta}, \quad t_2 = e \cdot (c - p) + \frac{1}{2}\sqrt{\Delta},$$

hence $C \cap L = \{p + t_1 e, p + t_2 e\}$.

III) **Intersection of a disk with a line segment** Suppose we are given a disk D with center at c and radius $\delta > 0$, and a line segment $s = (p_s, e_s, a_s, b_s)$. We would like to compute the intersection $D \cap s$.

First we compute the intersection between the boundary C of D and the line L that contains s, as described in II).

(i) If $C \cap L = \emptyset$, then $D \cap s = \emptyset$.
(ii) If L is tangent to C at the point $p_s + t_0 e_s$, for some t_0, then we verify whether t_0 is between a_s and b_s. In this case $D \cap s$ contains exactly the point $p_s + t_0 e_s$, otherwise $D \cap s = \emptyset$.

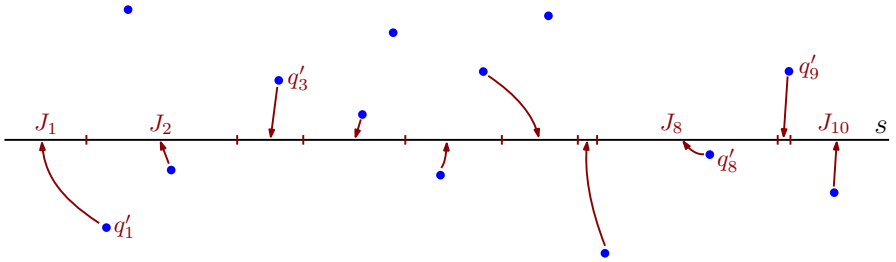

Fig. 4 Example of Voronoi diagram on a segment s defined by points. For each Voronoi cell we mark its closest site

(iii) If L intersects C in two points $p_s + t_1 e_s$ and $p_s + t_2 e_s$, where $t_1 < t_2$, this means that $D \cap L$ is the line segment $s' = (p_s, e_s, t_1, t_2)$. Hence, $D \cap s = s' \cap s$ and can be computed with $\mathcal{O}(1)$ additional comparisons between the values t_1, t_2, a_s, b_s.

Note that to compute $D \cap s$ all arithmetic computations were performed when computing $C \cap L$. The rest of the operations are only $\mathcal{O}(1)$ comparisons.

IV) Intersection of two segmentations of a line Suppose we have segmentations X_1 and X_2 on some line of sizes N_1 and N_2, respectively. We would like to compute the segmentation $X_1 \cap X_2$. The segmentation $X_1 \cap X_2$ has size at most $N_1 + N_2$ because the points of the segmentation $X_1 \cap X_2$ are points of X_1 or X_2. Moreover, because each segmentation already has its segments and points ordered, we can compute $X_1 \cap X_2$ with $\mathcal{O}(N_1 + N_2)$ comparisons. The idea is similar to the merging of two sorted lists. For example, we can use 3 pointers, one for X_1, one for X_2 and one for the merged list, that are traversed simultaneously once and, for each two consecutive points in the merged list, we verify whether the corresponding line segment is a subset of $X_1 \cap X_2$ or not in $\mathcal{O}(1)$ time.

V) **Computing a Voronoi diagram on a line segment for some set of points** Suppose we are given a set $\mathcal{Q} = \{q_1, q_2, \ldots, q_N\}$ of N points in the plane and a line segment $s = (p_s, e_s, a_s, b_s)$. We would like to compute the *Voronoi diagram on the line segment s for points in $\mathcal{Q}$*. See Fig. 4. For our purpose we can say that such a Voronoi diagram is a sequence of pairs $(q'_1, J_1), (q'_2, J_2), \ldots, (q'_{N'}, J_{N'})$ with the following properties:

- For each $i \in [N']$, the point q'_i belongs to $\mathcal{Q}$ and J_i is a segment contained in s,
- For each $i \in [N']$ and for each point p in J_i, the smallest distance from p to any point in $\mathcal{Q}$ is $d(p, q'_i)$,
- The union of the segments $J_1, \ldots, J_{N'}$ is the segment s,
- For each $i \in [N' - 1]$, J_i and J_{i+1} have exactly a boundary point in common,
- For each distinct and non-consecutive $i, j \in [N']$, the segments J_i and J_j are disjoint, and
- For each $q \in \mathcal{Q}$ there is at most one $i \in [N']$ with $q'_i = q$.

It is well known that this can be done in $\mathcal{O}(N \log N)$ time by computing the Voronoi diagram of $\mathcal{Q}$ in $\mathbb{R}^2$ and intersecting it with the segment s. See for example the textbook by de Berg et al. [11, Chapters 7 and 2].

Because we are only interested in the segment s, this can be done also easily using a simple divide-and-conquer approach, as follows. When $\mathcal{Q}$ contains a single point, its Voronoi diagram is the whole segment s. When $\mathcal{Q}$ has at least two points, we split $\mathcal{Q}$ arbitrarily into two sets, $\mathcal{Q}_1$ and $\mathcal{Q}_2$, of roughly the same size, and recursively compute the Voronoi diagram on s for $\mathcal{Q}_1$ and for $\mathcal{Q}_2$. We get those two Voronoi diagrams as sequences $(q'_1, J'_1), (q'_2, J'_2), \ldots, (q'_{N'}, J'_{N'})$ for $\mathcal{Q}_1$ and $(q''_1, J''_1), (q''_2, J''_2), \ldots, (q''_{N''}, J''_{N''})$ for $\mathcal{Q}_2$. To merge them, we first compute all the non-empty intervals $J'_i \cap J''_j$ for $i \in [N']$ and $j \in [N'']$. This takes $O(N' + N'')$ time because the two sequences are sorted along s. We also obtain the output sorted along s. We know that, for each such non-empty $J'_i \cap J''_j$, the closest point of $\mathcal{Q}$ is either q'_i or q''_j. For each such non-empty $J'_i \cap J''_j$, we compute the intersection $p_{i,j}$ of the bisector for q'_i and q''_j with the line supporting s. If the intersection point $p_{i,j}$ lies on $J'_i \cap J''_j$, we split $J'_i \cap J''_j$ into two intervals at $p_{i,j}$. We have obtained a sequence of intervals along s with the property that each point in an interval has the same closest point of $\mathcal{Q}$. With a final walk along the intervals, we merge adjacent intervals with the same closest point in $\mathcal{Q}$ into a single interval. This merging step takes $O(N' + N'')$ time.

If $T(N)$ denotes the running time of the divide-and-conquer algorithm for N points, we have the recurrence $T(N) = \mathcal{O}(N) + T(|\mathcal{Q}_1|) + T(|\mathcal{Q}_2|)$, with base case $T(1) = \mathcal{O}(1)$. Because $|\mathcal{Q}_1|$ and $|\mathcal{Q}_2|$ are approximately $N/2$, this solves to $T(N) = \mathcal{O}(N \log N)$.

Each number computed in the procedure is obtained from the input data by $\mathcal{O}(1)$ additions, subtractions, multiplications and divisions. This is because each value is obtained by computing the intersection of a bisector of two points of $\mathcal{Q}$ with the line supporting s.

VI) **Intersection of a union of disks with a line segment, equipped with a Voronoi diagram** Suppose we are given N disks $D(q_1, \delta), D(q_2, \delta), \ldots, D(q_N, \delta)$ with radius $\delta > 0$, a line segment $s = (p_s, e_s, a_s, b_s)$ and a Voronoi diagram $(q'_1, J_1), (q'_2, J_2), \ldots, (q'_{N'}, J_{N'})$ on s for the points $q_1, q_2, \ldots, q_N$. We would like to compute the intersection

$$X = \Big(\bigcup_{j \in [N]} D(q_j, \delta) \Big) \cap s.$$

Although we do not need the Voronoi diagram to compute X, we will use it to compute it in a linear number of steps. We first observe that (see Fig. 5)

$$X = \bigcup_{j \in [N']} \big(D(q'_j, \delta) \cap J_j \big).$$

This implies that X can be computed in time $\mathcal{O}(N)$ by applying N' times the procedure in III) and then joining each two consecutive line segments $D(q'_j, \delta) \cap J_j$ and $D(q'_{j+1}, \delta) \cap J_{j+1}$, if they have a common endpoint $J_j \cap J_{j+1}$. Note that the output is a segmentation contained in s.

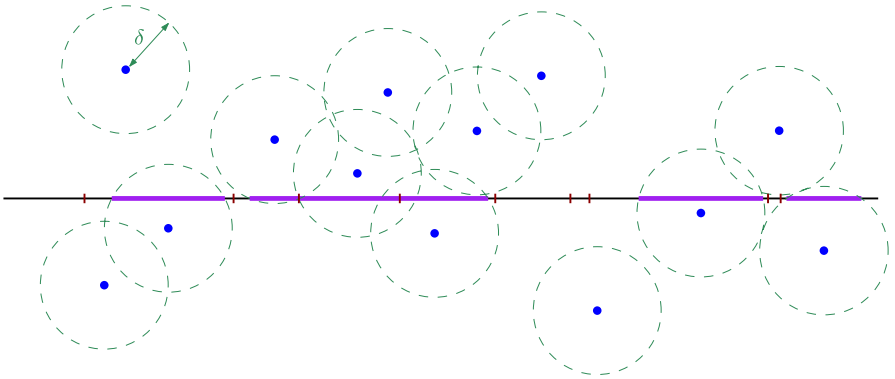

Fig. 5 Computing the intersection of a segment and the union of congruent disks using the Voronoi diagram of the centers

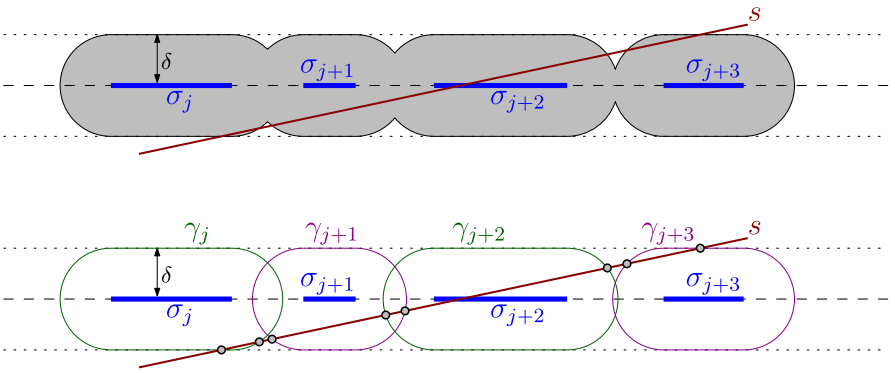

Fig. 6 Top: an example showing the input in VII). Bottom: the curves γ_i and their intersection with the segment s

VII) **Intersection of a δ-neighborhood of a segmentation with a line segment** Suppose we are given some $\delta > 0$, a line segmentation $X = (p_X, e_X, a_1, b_1, \ldots, a_N, b_N)$ and a line segment $s = (p_s, e_s, a_s, b_s)$. We would like to compute the intersection $X' = (X \oplus D(0, \delta)) \cap s$ between s and the δ-neighborhood of X. This is a segmentation in the line supporting the segment s. See the top of Fig. 6.

For each $j \in [N]$, let σ_j be the j-th segment of the segmentation X; thus $\sigma_j = (p_X, e_X, a_j, b_j)$. For each $j \in [N]$, let γ_j be the boundary of $\sigma_j \oplus D(0, \delta)$; see the bottom of Fig. 6. The boundary of γ_j consists of two semicircles of radius δ, one centered at $p_X + a_j e_X$ and another centered at $p_X + b_j e_X$, and two copies of the segment σ_j translated perpendicularly to σ_j by δ, one in each direction.

For each $j \in [N]$, we compute the (possibly empty, possibly degenerate) segment $\eta_j = \big(\sigma_j \oplus D(0, \delta)\big) \cap s$. See the top of Fig. 7. To do this we compute $\gamma_j \cap s$ using I) and II) for the lines and circles supporting pieces of γ_j and, for each intersection point we find, we test whether it indeed belongs to γ_j. We also test whether the endpoints of s belong to $\sigma_j \oplus D(0, \delta)$, as the segment s may start inside multiple regions $\sigma_j \oplus D(0, \delta)$. This takes $\mathcal{O}(N)$ time and each number we computed requires

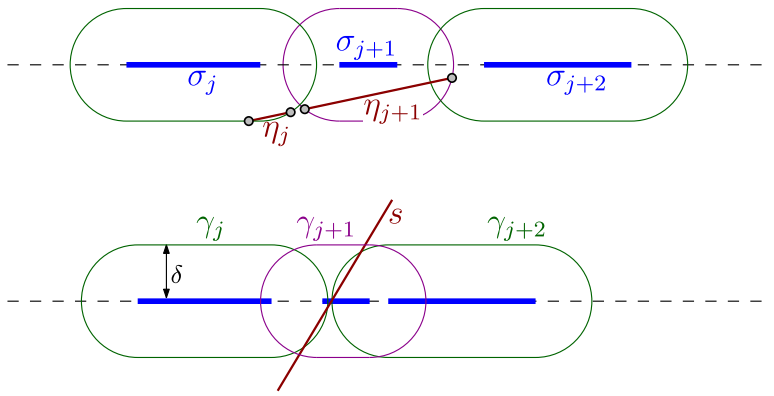

Fig. 7 Top: an example showing η_j and η_{j+1}. Note that η_{j+2} is not shown, but it would overlap with η_{j+1}. Bottom: example showing that s may not enter $\gamma_j, \gamma_{j+1}, \gamma_{j+2}$ in that order

$\mathcal{O}(1)$ arithmetic operations, square roots, and comparisons. (The explicit computation of η_j may require the square root.) Each such non-empty segment η_j is represented as $\eta_j = (p_s, e_s, a'_j, b'_j)$, using the same point p_s and direction unit vector e_s for all $j \in [N]$.

If we found no intersections, meaning that $\eta_j = \emptyset$ for all $j \in [N]$, we return $X' = \emptyset$. Otherwise, the segments $\eta_1, \ldots, \eta_N$ may overlap and we have to merge them. We must be careful because the line segment s does not need to enter the regions γ_j, γ_{j+1} and γ_{j+2}, for some $j \in [N-2]$ in this order. See for example the bottom of Fig. 7. However, we can be sure about two things:

- If $\eta_i \neq \emptyset$ and $\eta_j \neq \emptyset$ for some $1 \leq i < j \leq N$, then $\eta_k \neq \emptyset$ for all $k \in \{i, i+1, \ldots, j\}$.
- If $\eta_i \cap \eta_j \neq \emptyset$ for some $1 \leq i < j \leq N$, then $\eta_i \cap \eta_k \neq \emptyset$ for all $k \in \{i, i+1, \ldots, j\}$.

This means that we can merge the segments $\eta_1, \ldots, \eta_N$ by considering only segments with adjacent indices. One way to do it is to compute

$$m = \min\{j \in [N] \mid \eta_j \neq \emptyset\} \quad \text{and} \quad M = \max\{j \in [N] \mid \eta_j \neq \emptyset\},$$

and make a linked list with the collinear segments

$$\eta_m = (p_s, e_s, a'_m, b'_m), \ldots, \eta_M = (p_s, e_s, a'_M, b'_M),$$

in that order. Walking along the list, whenever two consecutive segments $\eta = (p_s, e_s, a, b)$ and $\eta' = (p_s, e_s, a', b')$ in the list intersect, which can be checked by sorting numbers a, a', b and b', we merge them into the single segment η'', which replaces η and η' in the list. If the final list is $\tilde{\eta}_1 = (p_s, e_s, \tilde{a}_1, \tilde{b}_1), \ldots, \tilde{\eta}_j = (p_s, e_s, \tilde{a}_{N'}, \tilde{b}_{N'})$, we then have

$$(X \oplus D(0, \delta)) \cap L_s = (p_s, e_s, \tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_{N'}, \tilde{b}_{N'}).$$

The whole computation takes $\mathcal{O}(N)$ time and each number in the output is obtained from the input data by performing $\mathcal{O}(1)$ arithmetic operations and square roots.

4 Solving the Decision Version

In this section we show how to solve the decision of the problem, DCONNECTIVITY, when the uncertain regions are segments. Let us recall the problem. Given a set $\mathcal{S} = \{s_1, \ldots, s_k\}$ of k segments in the plane, a set $\mathcal{P} = \{p_{k+1}, p_{k+2}, \ldots, p_n\}$ of $n - k$ points in the plane, and a value $\delta \geq 0$, decide whether there exist points $p_i \in s_i$ for $i = 1, \ldots, k$ such that $\mathbb{G}_\delta(\{p_1, \ldots, p_n\})$ is connected. Denoting by δ^* the optimal value in the optimization problem, we want to decide whether $\delta \geq \delta^*$.

We will consider the case $\delta = 0$ separately. Then we will present an algorithm that solves DCONNECTIVITY for $\delta > 0$ in 4 parts, which are in bold in the next sentence. **DStep** 1 will be executed first, followed by **DStep** 2. Then we will use a **DLoop** inside of which **DStep** 3 will be performed $k^{\mathcal{O}(k)}$ times. The letter **D** in **DStep** and **DLoop** specifies that the steps are of the algorithm for the decision variant.

Case $\delta = 0$ Solving the problem DCONNECTIVITY for $\delta = 0$ is equivalent to deciding whether all line segments from $\mathcal{S}$ have a common point which is the same as each point in $\mathcal{P}$. This can clearly be done in time $\mathcal{O}(n)$. For the rest of the description, we assume $\delta > 0$.

DStep 1. We compute a MST T for points in $\mathcal{P}$. It is well known that the tree T can be computed in $\mathcal{O}(n \log n)$ time [25]. The most usual way is noting that a MST of any Delaunay triangulation of the point set $\mathcal{P}$ is a MST for $\mathcal{P}$ [11, Exercise 9.11].

We remove all edges from T that are longer than δ. Let the remaining connected components of the tree T be $\mathcal{C} = \{C_1, C_2, \ldots, C_\ell\}$. Note that we removed exactly $\ell - 1$ edges. If $\ell > 4k + 1$, return *FALSE*. We first show that this decision based on $\ell > 4k + 1$ is correct.

Lemma 3 *Any two points in $\mathcal{P}$ from distinct components $C_1, C_2, \ldots, C_\ell$ are more than δ apart.*

Proof If the lemma was false, then there would exist points $p_a, p_b \in \mathcal{P}$ such that $d(p_a, p_b) \leq \delta$ and $p_a \in C_{a'}$, $p_b \in C_{b'}$, where $a' \neq b'$. Clearly, the edge $p_a p_b$ is not part of the tree T. If we add it to the tree, we get a cycle that connects the points p_a and p_b either via the edge $p_a p_b$ or via a path that contains an edge e that was longer than δ, because the components $C_{a'}$ and $C_{b'}$ are distinct. It follows that if we add the edge $p_a p_b$ and remove the edge e from T, we get a spanning tree on points $\mathcal{P}$ with weight less than the weight of T, which is a contradiction. Hence, the lemma must be true. □

Lemma 4 *If $\ell > 4k + 1$, then $\delta < \delta^*$.*

Proof We show the contrapositive statement. Hence, we assume $\delta \geq \delta^*$. Let $p_i \in s_i$, for $i \in [k]$, be such points that any MBST on points $p_1, p_2, \ldots, p_n$ has a bottleneck edge of length δ^*. Such points exist by definition of δ^*. By Claim 1, any MST on

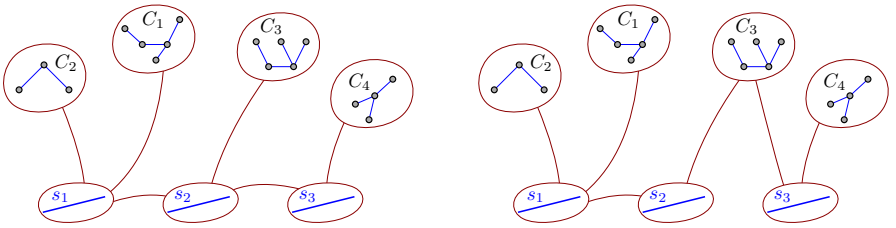

Fig. 8 Two examples of topology trees. The left example has one significant topology subtree (a concept we will introduce later), while the right one has two significant topology subtrees, one spanned by the nodes $\{s_1, s_2, C_1, C_2, C_3\}$ and one spanned by $\{s_3, C_3, C_4\}$

these points also has a bottleneck edge of length δ^*. Let T' be a MST tree on points $p_1, p_2, \ldots, p_n$ with the degree of each vertex at most 5. Such a tree exists by Claim 2. Hence, there are at most $5k$ neighbors of points $p_1, p_2, \ldots, p_k$. By Lemma 3, any two points from distinct components $C_1, C_2, \ldots, C_\ell$ are not connected directly with edges of T', hence for each component there is an edge in T' from a point in this component to a point in $\{p_1, p_2, \ldots, p_k\}$. Let E_1 be the set of edges in T' that have exactly one vertex in $\{p_1, p_2, \ldots, p_k\}$ and let E_2 be the set of edges in T' that have both vertices from the set $\{p_1, p_2, \ldots, p_k\}$. We have $|E_1|+2|E_2| \leq 5k$. Because there are no edges between components C_i and C_j, for $i \neq j$, we have $|E_1| + |E_2| \geq k + \ell - 1$. This is because using edges from $E_1 \cup E_2$ we have to connect at least $k+\ell$ distinct "clusters" of points: each point in $\{p_1, p_2, \ldots, p_k\}$ is one "cluster" and each component $C_1, C_2, \ldots, C_\ell$ contributes at least one cluster. This implies $k + \ell - 1 \leq 5k$, which gives $\ell \leq 4k + 1$. □

DStep 2. For each component $C_i \in \mathcal{C}$ and for each line segment $s_j \in \mathcal{S}$, we compute the Voronoi diagram on s_j of the points in C_i. This can be done with $\mathcal{O}(kn \log n)$ steps as explained in V).

DLoop. We treat each line segment from $\mathcal{S}$ and each component from $\mathcal{C}$ as an abstract node and we iterate over all possible trees on these $k + \ell$ nodes such that

a) each node from $\mathcal{S}$ has degree at most 5 and
b) no two nodes from $\mathcal{C}$ are adjacent.

We call each such a tree a *topology tree*, because it describes a potential way to connect the components in $\mathcal{C}$ via points from the line segments in $\mathcal{S}$. See Fig. 8 for an example. Note that we are reserving the term *node* for each connected component of $\mathcal{C}$ and each segment of $\mathcal{S}$. In this way we distinguish nodes from a topology tree from vertices of other graphs.

We say that a topology tree τ is *δ-realizable*, if there exist points $p_j \in s_j$, for each $j \in [k]$, such that

(a) for each edge $s_i s_j$ in τ, where $s_i, s_j \in \mathcal{S}$, it holds $d(p_i, p_j) \leq \delta$, and
(b) for each edge $C_i s_j$ in τ, where $C_i \in \mathcal{C}$ and $s_j \in \mathcal{S}$, there exists a point $p \in C_i$ such that $d(p, p_j) \leq \delta$. (Note that we may use different points $p \in C_i$ for different edges $C_i s_j$, $C_i s'_j$ of τ.)

Lemma 5 *There exists a δ-realizable topology tree if and only if $\delta \geq \delta^*$.*

Proof Let τ be a δ-realizable topology tree. Let us fix points $p_j \in s_j$, for each $j \in [k]$, such that

(a) for each edge $s_i s_j$ in τ, where $s_i, s_j \in \mathcal{S}$, it holds $d(p_i, p_j) \leq \delta$ and
(b) for each edge $C_i s_j$ in τ, where $C_i \in \mathcal{C}$ and $s_j \in \mathcal{S}$, there exists a point $p \in C_i$ such that $d(p, p_j) \leq \delta$.

Let $G = \mathbb{G}_\delta(\{p_1, p_2, \ldots p_n\})$. We will prove that G is connected, which by definition of δ^* implies $\delta \geq \delta^*$. Note that any two vertices in the same component C of $\mathcal{C}$ are connected in G. Indeed, since there exists a path connecting two vertices of C in the MST T that uses only edges of length at most δ, such a path is also present in G.

Consider two arbitrary points p_i and p_j, where $i, j \in [n]$. Let v_i be the node of τ that contains p_i; it may be that $v_i = C$ for some $C \in \mathcal{C}$ or that $v_i = s$ for some $s \in \mathcal{S}$. Similarly, let v_j be the node of τ that contains p_j. If $v_i = v_j$ and v_i is a segment of $\mathcal{S}$, then $p_i = p_j$ and they are connected in G. If $v_i = v_j$ and v_i is a connected component $C \in \mathcal{C}$, then they are also connected in G.

It remains to handle the case when $v_i \neq v_j$. Because τ is a tree, then there exist nodes $u_1, u_2, \ldots, u_m$ of τ such that $u_1 u_2 \cdots u_m$ is a path in τ with $u_1 = v_i$ and $u_m = v_j$ We will construct a corresponding walk in G that connects p_i and p_j by transforming the path $u_1 u_2 u_3 \cdots u_m$ in the following way.

1. If $u_1 = s_i \in \mathcal{S}$, then we replace u_1 by p_i. Otherwise $u_1 = C$ for some $C \in \mathcal{C}$. From the definition of τ it follows that the node $u_2 = s_y$ for some segment $s_y \in \mathcal{S}$, and there exists a point p from C such that $d(p, p_y) \leq \delta$. We replace the node u_1 with a path from p_i to p in C.
2. If $u_m = s_j \in \mathcal{S}$, then we replace u_m by p_j. Otherwise $u_m = C$ for some $C \in \mathcal{C}$. From the definition of τ it follows that the node $u_{m-1} = s_y$ for some segment $s_y \in \mathcal{S}$, and there exists a point p from C such that $d(p, p_y) \leq \delta$. We replace the node u_m with a path from p to p_j in C.
3. For each vertex $u_x \in \{u_2, \ldots, u_{m-1}\}$ from $\mathcal{C}$, we know from the definition of τ that $u_{x-1} = s_y \in \mathcal{S}$ and $u_{x+1} = s_z \in \mathcal{S}$, for some $y, z \in [k]$. By definition of τ there exist points q_x and q'_x from the component $u_x \in \mathcal{C}$ such that $d(q_x, p_y) \leq \delta$ and $d(q'_x, p_z) \leq \delta$. We replace u_x with a path from q_x to q'_x in the component u_x.
4. For each vertex $s_x \in \{u_2, u_3, \ldots, u_{m-1}\}$ from $\mathcal{S}$, we replace s_x with the corresponding point $p_x \in s_x$.

It is clear that we transformed the path $u_1 u_2 u_3 \cdots u_m$ in τ into a walk in the graph G from p_i to p_j. Hence, G is connected, which implies $\delta \geq \delta^*$.

For the other direction, assume $\delta \geq \delta^*$. Let points $p_i \in s_i$, for each $i \in [k]$, be such that any MBST on points $p_1, p_2, \ldots, p_n$ has a bottleneck edge of length δ^*. Let T' be a MBST of the complete graph $K_{\{p_1, \ldots, p_n\}}$ such that each vertex of T' has degree at most 5. Such a tree T' exists by Claim 2. Because of Lemma 3 and because $\delta \geq \delta^*$, there is no edge in T' between points from distinct components from $\mathcal{C}$. Let Γ be a graph with $k + \ell$ nodes $\mathcal{C} \cup \mathcal{S}$ and the following edges:

- $s_i s_j$, whenever $p_i p_j$ is an edge in T',
- $s_i C_j$, whenever there exists a point p in C_j such that $p_i p$ is an edge in T'.

Such a graph Γ is not necessarily a tree, however we claim that it is connected. This follows from the fact that any path in T' can be mapped into a walk in Γ by

replacing each vertex from a component C_i with the component C_i, each vertex from a line segment s_j with the line segment s_j, and then deleting the potential consecutive repetitions of vertices of Γ. Because the tree T' has maximum degree 5, it follows by definition of edges of Γ that the degree of each node $s_i \in \mathcal{S}$ of Γ has degree at most 5. Let τ be any spanning tree of Γ. It is clear that τ is a δ-realizable topology tree. □

Recall that Lemma 4 states that, in case $\delta \geq \delta^*$, it holds that $\ell \leq 4k + 1$.

Lemma 6 *If $\ell \leq 4k + 1$, then there are at most $(\mathcal{O}(k))^{5k}$ topology trees and they can be generated in $(\mathcal{O}(k))^{5k}$ time.*

Proof Using Cayley's formula for the number of spanning trees of a labeled complete graph, we get that that the number of topology trees is at most $(k + \ell)^{k+\ell-2} \leq (5k + 1)^{5k-1}$. To construct a topology tree, it is enough to determine the neighbors of each node from $\mathcal{S}$. Because each node from $\mathcal{S}$ has degree at most 5 in a topology tree, we can use brute force to generate all topology trees on $k + \ell \leq 5k + 1$ nodes in $(\mathcal{O}(k))^{5k}$ time. □

Note that, even with more careful estimates, we could not get a bound below $k^{\mathcal{O}(k)}$ because there are at least $\frac{k!}{2} = k^{\Omega(k)}$ ways to construct a path of length k out of elements of $\mathcal{S}$.

DStep 3. Given a topology tree τ, in this step we will verify whether it is δ-realizable. Therefore, because of Lemma 6, we will execute this step $k^{\mathcal{O}(k)}$ times. For the discussion, we consider a fixed topology tree τ.

We observe that if a node C from $\mathcal{C}$ is a separating vertex of τ, which is equivalent to saying that C has degree at least 2 in τ, then we can treat each part of the tree τ that is "separated" by C independently. This motivates the following definition of a *significant topology subtree* of τ.

If we remove the nodes $\mathcal{C}$ from τ, we get a forest. To each tree τ' in this forest, we add all nodes from $\mathcal{C}$ that are adjacent in τ to some vertex in τ', with the corresponding edges. The resulting tree is a *significant topology subtree* of τ. See Fig. 8. Let us state a few observations about significant topology subtrees that can be checked easily.

(a) Each significant topology subtree of τ is an induced subtree of τ.
(b) Each node from $\mathcal{C}$ that is part of a significant topology subtree τ' of τ, has degree exactly 1 in τ'. If there is another vertex of degree 1 in τ', it must have degree 1 in τ as well.
(c) Each significant topology subtree has maximum degree at most 5.
(d) The union of all significant topology subtrees of τ is the whole τ.
(e) An intersection of any two significant topology subtrees of τ is either empty or a graph with one node from $\mathcal{C}$.
(f) Each node from $\mathcal{S}$ belongs to exactly one significant topology subtree of τ.
(g) There are at most k significant topology subtrees of τ.

The definition of δ-realizability can now be naturally extended to significant topology subtrees. We say that a significant topology subtree τ' is *δ-realizable*, if there exist points $p_j \in s_j$, for each vertex $s_j \in \mathcal{S}$ from τ', such that

(a) for each edge $s_i s_j$ in τ', where $s_i, s_j \in \mathcal{S}$, it holds $d(p_i, p_j) \leq \delta$, and

(b) for each edge $C_i s_j$ in τ', where $C_i \in \mathcal{C}$ and $s_j \in \mathcal{S}$, there exists a point $p \in C_i$ such that $d(p, p_j) \leq \delta$.

Lemma 7 *The topology tree τ is δ-realizable if and only if all of its significant topology subtrees are δ-realizable.*

Proof If the topology tree τ is δ-realizable, it is clear that each of the significant topology subtrees of τ is also δ-realizable. To prove the opposite direction, assume that all of significant topology subtrees of τ are δ-realizable. For each significant topology subtree τ' of τ, we can choose points $p_j \in s_j$ in each node $s_j \in \mathcal{S}$ of τ', such that

(a) for each edge $s_i s_j$ in τ', where $s_i, s_j \in \mathcal{S}$, it holds $d(p_i, p_j) \leq \delta$, and
(b) for each edge $C_i s_j$ in τ', where $C_i \in \mathcal{C}$ and $s_j \in \mathcal{S}$, there exists a point $p \in C_i$ such that $d(p, p_j) \leq \delta$.

This way we uniquely defined points $p_j \in s_j$, for each $j \in [k]$. This is because, for each $s_j \in \mathcal{S}$, there exists exactly one significant topology subtree τ' of τ that has s_j as node. It is clear that such a choice of points $p_j \in s_j$, over all $j \in [k]$, shows that the topology tree τ is δ-realizable. □

We just showed that, to describe **DStep** 3, it is enough to describe how to verify whether a given significant topology subtree τ' of τ is δ-realizable. To describe the latter, we restrict our attention to a fixed significant topology subtree τ'. Let the set of nodes of τ' be $V' \subseteq \mathcal{S} \cup \mathcal{C}$. We denote $\mathcal{S}' = \mathcal{S} \cap V'$ and $\mathcal{C}' = \mathcal{C} \cap V'$. Therefore V' is the disjoint union of $\mathcal{S}'$ and $\mathcal{C}'$. By definition of τ', we know that $\mathcal{S}'$ is not empty. Let us choose a root $s_r \in \mathcal{S}'$ for τ'. For each segment $s_i \in \mathcal{S}'$, let $\tau'(s_i)$ be the subtree of τ' rooted at s_i. In particular $\tau'(s_r) = \tau'$.

Next, we will use dynamic programming bottom-up along τ' to compute the possible locations of points $p_i \in s_i$ on line segments $s_i \in \mathcal{S}'$ that can yield δ-realizability for the subtree of τ' rooted at s_i. More exactly, for each $s_i \in \mathcal{S}'$, we define

$$\begin{aligned} X_i = \{p_i \in s_i \mid\ & \text{we can select one point } q_\ell \in C_\ell, \text{ for each node } C_\ell \in \mathcal{C} \cap V(\tau'(s_i)), \\ & \text{and one point } p_j \in s_j, \text{ for each node } s_j \in \mathcal{S} \cap V(\tau'(s_i)) \text{ with } j \neq i, \\ & \text{such that for each edge } s_\ell s_j \text{ of } \tau'(s_i) \text{we have } d(p_\ell, p_j) \leq \delta \\ & \text{and for each edge } s_j C_\ell \text{ of } \tau'(s_i) \text{we have } d(p_j, q_\ell) \leq \delta\}. \end{aligned}$$

We begin with leaves of τ'. If we have a leaf s_i from $\mathcal{S}'$, we then have $X_i = s_i$. For internal nodes of τ', we have the following recursive property.

Lemma 8 *Let s_i be an internal node in τ' from $\mathcal{S}'$. Reindexing the nodes, if needed, let us assume that the children of s_i in τ' are $s_1, \ldots, s_t$ and $C_1, \ldots, C_u$. Then*

$$X_i = \bigcap_{\ell=1}^{t} \big(X_\ell \oplus D(0, \delta)\big) \bigcap_{\ell=1}^{u} \big(C_\ell \oplus D(0, \delta)\big) \bigcap s_i.$$

Proof Consider any point p_i in X_i. From the definition of X_i this means that we can select one point $q_\ell \in C_\ell$, for each node $C_\ell \in \mathcal{C} \cap V(\tau'(s_i))$ and one point $p_j \in s_j$ for each node $s_j \in \mathcal{S} \cap V(\tau'(s_i))$ with $j \neq i$ such that

- For each edge $s_j s_\ell$ of $\tau'(s_i)$ we have $d(p_j, p_\ell) \leq \delta$, and
- For each edge $s_j C_\ell$ of $\tau'(s_i)$ we have $d(p_j, q_\ell) \leq \delta$.

Looking at the edges connecting s_i to its children, we obtain

- for each $\ell \in [t]$ we have $d(p_i, p_\ell) \leq \delta$, and
- for each $\ell \in [u]$ we have $d(p_i, q_\ell) \leq \delta$.

Moreover, because the definition of X_i includes a condition for the whole subtree $\tau'(s_i)$ and, for each child s_ℓ of s_i, we have $\tau'(s_\ell) \subset \tau'(s_i)$, we have

- for each $\ell \in [t]$, the point p_ℓ belongs to X_ℓ.

We conclude that the point p_i belongs to $\bigcap_{j=1}^{t} \big(X_j \oplus D(0, \delta)\big)$. Because for each $\ell \in [u]$ we also have $q_\ell \in C_\ell$, we also conclude that p_i belongs to $\bigcap_{\ell=1}^{u} \big(C_\ell \oplus D(0, \delta)\big)$. This finishes the proof that X_i is included in the right hand side.

To see the other inclusion, consider one point p_i on the right hand side of the equality we want to prove. We then have:

- $p_i \in s_i$;
- for each $\ell \in [t]$, the point p_i belongs to $X_\ell \oplus D(0, \delta)$;
- for each $\ell \in [u]$, the point p_i belongs to $C_\ell \oplus D(0, \delta)$.

We can rewrite these properties as

- $p_i \in s_i$;
- for each $\ell \in [t]$, there is some point $p_\ell \in X_\ell$ such that $d(p_i, p_\ell) \leq \delta$;
- for each $\ell \in [u]$, there is some point $q_\ell \in C_\ell$ such that $d(p_i, q_\ell) \leq \delta$.

For each $\ell \in [t]$, the property that $p_\ell \in X_\ell$ implies that we can find points in all the nodes in $\tau'(s_\ell)$ satisfying the definition for X_ℓ. Since the subtrees $\tau'(s_1), \ldots, \tau'(s_t)$ are disjoint, the selection of points for those subtrees are for different nodes, and thus they do not interact. The points $p_i, p_1, \ldots, p_t, q_1, \ldots, q_u$ and the ones selected for the condition of $X_1, \ldots, X_t$ certify that $p_i \in X_i$ because each edge of $\tau'(s_i)$ appears in one of the conditions. □

All geometrical computations needed to compute X_i are described in Sect. 3. We can compute X_i with t operations $\big(X_\ell \oplus D(0, \delta)\big) \cap s_i$, described in VII), u operations $\big(C_\ell \oplus D(0, \delta)\big) \cap s_i$ described in VI), and $t + u - 1$ intersections of segmentations described in IV). For each $\ell \in [u]$, the size of the segmentation $\big(C_\ell \oplus D(0, \delta)\big) \cap s_i$ is at most $|C_\ell|$, and using induction on the structure of τ' (the base of the induction are the leaves), we can see that each point from $\mathcal{P}$ contributes at most one segment to X_i. Using that τ' has at most k leaves from $\mathcal{S}$, we see that the size of X_i is at most $|\mathcal{P}| + |\mathcal{S}| = n$. Finally, note that $t + u \leq 5$ because τ' has maximum degree at most 5. Assuming that X_ℓ is already available for each child $s_\ell \in \mathcal{S}'$ of s_i (thus for all $\ell \in [t]$), and assuming that the Voronoi diagrams on s_i for C_ℓ are available for each $\ell \in [u]$, we can compute X_i in $\mathcal{O}(n)$ time. Recall that the Voronoi diagrams inside s_i for each C_ℓ were computed in **DStep** 2, and thus are available.

The significant topology subtree τ' is δ-realizable if and only if X_r is not empty. We can compute the sets X_i for all $s_i \in \mathcal{S}'$ bottom-up. At each node of τ' from $\mathcal{S}'$ we spend $\mathcal{O}(n)$ time.

We have to repeat the test for each significant topology subtree. Recall that, by Lemma 7, a topology tree is δ-realizable if and only if all its significant topology subtrees are δ-realizable. Since each node of $\mathcal{S}$ appears exactly in one significant topology subtree, for each $s_j \in \mathcal{S}$ we compute the corresponding set X_j exactly once. It follows that we spend $\mathcal{O}(kn)$ time for a topology tree. We summarize.

Lemma 9 *Assume we have already performed* ***DStep*** 1 *and* ***DStep*** 2*. For any given topology tree τ, we can decide whether τ is δ-realizable performing $\mathcal{O}(kn)$ operations. Here, an operation may include manipulating a number that has a computation tree of depth $\mathcal{O}(k)$ whose internal nodes are additions, subtractions, multiplications, divisions or square root computations and whose leaves contain input numbers (including δ).*

Proof Correctness and the bound on the number of operations follows from the discussion. It only remains to show the property about the computation tree of numbers. Each component of a segmentation X_i that corresponds to a node v in some rooted topological subtree τ' of τ is computed with $\mathcal{O}(1)$ operations from input numbers or components of the segmentations that correspond to the children of v in τ'. Since each significant topology tree τ' has depth at most $k+1$, the claim follows. (Note that the numbers may participate in many more comparisons.) □

In **DLoop**, we try each topology tree τ and perform **DStep** 3 for τ. If for some topology tree we find that it is δ-realizable, we return *TRUE*. If the loop finishes without finding any δ-realizable topology tree, we return *FALSE*.

Because of our future use in the optimization version of the problem, we decouple the running time of **DStep** 1 and **DStep** 2.

Theorem 10 *Assume that we have an instance for* CONNECTIVITY *with k line segments and $n-k$ points (δ is not part of the input). After a preprocessing of $\mathcal{O}(k^2 n \log n)$ time, for any given δ, we can solve the decision version* DCONNECTIVITY *performing $k^{\mathcal{O}(k)} n$ operations. Here, an operation may include manipulating a number that has a computation tree of depth $\mathcal{O}(k)$ whose internal nodes are additions, subtractions, multiplications, divisions or square root computations and whose leaves contain input numbers (including δ).*

Proof We have shown that **DStep** 3 requires linear number of steps in n and that the number of repetitions of **DLoop** depends only on k. Hence, to prove the theorem, we need to reduce the $\log n$ factor from **DStep** 1 and **DStep** 2 with preprocessing.

The main part of **DStep** 1 is computing a MST on $n-k$ points, which takes $\mathcal{O}(n \log n)$ time and is independent of δ. Hence, it can be done with preprocessing. The rest of **DStep** 1 (ie. defining the clusters $\mathcal{C}$) can be implemented in time $\mathcal{O}(k)$ for any δ.

For **DStep** 2, we observe that δ can be classified into $\mathcal{O}(k)$ different intervals of values that will give the same clusters $\mathcal{C}$ and for which **DStep** 2 is the same. More precisely, let $e_1, \ldots, e_{4k+1}$ be $4k+1$ longest edges in the MST T for $\mathcal{P}$, obtained

after preprocessing for **DStep** 1 described above, sorted such that $|e_1| \geq |e_2| \geq \ldots \geq |e_{4k+1}|$. For each $i \in [4k]$ and each δ in the interval $(|e_i|, |e_{i+1}|]$ we will have the same family $\mathcal{C}$ of $i+1$ connected componenents, namely those in the graph $T - \{e_1, \ldots, e_i\}$. For each $\delta \geq |e_1|$, we have a single component in $\mathcal{C}$. For each $\delta < |e_{4k+1}|$, we know that $\delta < \delta^*$ because of Lemma 4. Therefore, we can consider the $\mathcal{O}(k)$ different connected components that appear in the graphs $T_0 = T$ and $T_i = T_{i-1} - e_i$, where $i \in [4k]$. For each such connected component C and each segment $s \in \mathcal{S}$, we compute the Voronoi diagram on s of the points of C using **DStep** 2. In total we compute $\mathcal{O}(k^2)$ Voronoi diagrams, and each of them takes $\mathcal{O}(n \log n)$ time. This can all be done with preprocessing, hence **DStep** 2 can be implemented in time $\mathcal{O}(1)$ for any δ.

The depths of the computation trees of numbers used in **DStep** 1 and **DStep** 2 are $\mathcal{O}(1)$.

Consider now that we are given a value δ after the just described preprocessing. If $\delta < |e_{4k+1}|$, we return *FALSE*. Otherwise, **DStep** 1 and **DStep** 2 now require only $\mathcal{O}(k)$ time. We perform **DLoop** iterating over all topology trees. The correctness is proven with Lemma 5 and Lemma 9.

By Lemma 6, **DStep** 3 is repeated $k^{\mathcal{O}(k)}$ times, and each such iteration performs $\mathcal{O}(kn)$ operations because of Lemma 9. In total we perform $k^{\mathcal{O}(k)}n$ operations. Numbers in each iteration of **DLoop** are computed independently of the numbers computed in another iteration, and therefore we can use the bound on the depth of computation trees of Lemma 9 for each of them. □

Corollary 11 *The decision problem* DCONNECTIVITY *for k line segments and $n-k$ points can be solved performing $k^{\mathcal{O}(k)}n \log n$ operations. Here, an operation may include a number that has a computation tree of depth $\mathcal{O}(k)$ whose internal nodes that are additions, subtractions, multiplications, divisions or square root computations and whose leaves contain input numbers (including the input value δ).*

5 Introducing *h*-square Root Functions

When using parametric search, we will need to trace the boundary of the segmentations $X_i(\delta)$ as a function of δ. In this section we introduce and discuss the properties of the functions that will appear.

For any natural[1] number h, we define h-square root functions recursively. A 0-square root function is any linear function. For $h \geq 1$, an h-square root function is any function of the form $f(x) = a_1 g(x) + a_2 + a_3\sqrt{\pm x^2 + a_4 g(x)^2 + a_5 g(x) + a_6}$, where $a_1, a_2, a_3, a_4, a_5, a_6 \in \mathbb{R}$, and $g(x)$ is a $(h-1)$-square root function. The domain of an h-square root function is all such $x \in \mathbb{R}$ for which all the square roots that appear inside them have non-negative arguments. Note that an h-square root function is also an h'-square root function for all $h' \geq h$ because we may take $a_1 = 1$ and $a_2 = a_3 = 0$.

The following lemma presents the setting where we will meet the h-square root functions. Note that this setting can occur in computations in II), that is, when computing the intersection of a circle with a line.

[1] We define that 0 is a natural number.

Lemma 12 *Let q, e and f be vectors in $\mathbb{R}^2$, $||e|| = ||f|| = 1$ and let $g(x)$ be an $(h-1)$-square root function, for some $h \in \mathbb{Z}^+$. Then any continuous function $t(x)$ that solves the equation*

$$||q + t(x)e - g(x)f||^2 = x^2$$

is an h-square root function.

Proof The equation $||q + t(x)e - g(x)f||^2 = x^2$ is equivalent to

$$t(x)^2 + \big[2(q - g(x)f) \cdot e\big]t(x) + ||q - g(x)f||^2 - x^2 = 0.$$

The discriminant of this quadratic equation in $t(x)$ is

$$\begin{aligned}\Delta(x) &= 4((q - g(x)f) \cdot e)^2 - 4(||q - g(x)f||^2 - x^2)\\ &= 4\Big((q \cdot e)^2 - 2(q \cdot e)(f \cdot e)g(x) + (e \cdot f)^2 g(x)^2\\ &\quad - ||q||^2 + 2(q \cdot f)g(x) - g(x)^2 + x^2\Big)\\ &= 4\Big(x^2 + \big[(e \cdot f)^2 - 1\big]g(x)^2 + \big[2(q \cdot e)(f \cdot e)\\ &\quad + 2(q \cdot f)\big]g(x) + \big[(q \cdot e)^2 - ||q||^2\big]\Big).\end{aligned}$$

If we denote $\tilde{\Delta}(x) = \frac{1}{4}\Delta(x)$, we have, for all x in the domain of g for which $\Delta(x) \geq 0$,

$$\begin{aligned}t_1(x) &= e \cdot (g(x)f - q) - \sqrt{\tilde{\Delta}(x)} = \big[e \cdot f\big]g(x) - \big[e \cdot q\big] - \sqrt{\tilde{\Delta}(x)},\\ t_2(x) &= e \cdot (g(x)f - q) + \sqrt{\tilde{\Delta}(x)} = \big[e \cdot f\big]g(x) - \big[e \cdot q\big] + \sqrt{\tilde{\Delta}(x)},\end{aligned}$$

which are both h-square root functions. □

The next lemma will help us solve equations with h-square root functions.

Lemma 13 *Let $f(x)$ and $g(x)$ be h-square root functions for $h \in \mathbb{N}$. Then all of the solutions of $f(x) = g(x)$ are also roots of a polynomial of degree at most 4^h in x. The coefficients of this polynomial can be computed from parameters in f and g in $2^{\mathcal{O}(h)}$ steps by using only multiplications, additions and subtractions.*

Proof Let $f_h(x) = f(x)$ be an h-square root function obtained from an $(h-1)$-square root function $f_{h-1}(x)$, which was obtained from an $(h-2)$-square root function $f_{h-2}(x)$, …, which was obtained from a 0-square root function $f_0(x)$. In a similar way we define the functions $g_h(x), g_{h-1}(x), \ldots, g_0(x)$.

We will transform the equation $f_h(x) = g_h(x)$ into the desired polynomial equation by squaring it at most $2h$ times, each time also rearranging the terms a bit and using the replacement rule $\sqrt{u}^2 = u$ multiple times. These transformations may introduce additional solutions, but we keep all the original solutions.

We show by induction on i that, for each $i = 0, \ldots, h$, there is a polynomial $P_i(X, Y_i, Z_i)$ of degree 4^i such that the solutions of $P_i(x, f_{h-i}(x), g_{h-i}(x)) = 0$ include the solutions of $f(x) = g(x)$. For the base case, $i = 0$, it is obvious that the polynomial $P_0(X, Y_i, Z_i) = Z_i - Y_i$ satisfies the condition because $P_0(x, f_h(x), g_h(x)) = 0$ is equivalent to $f_h(x) - g_h(x) = 0$.

Assume that we have the polynomial $P_i(X, Y_i, Z_i)$ for some i. We show how to compute $P_{i+1}(X, Y_{i+1}, Z_{i+1})$. The polynomial P_i can be written as

$$P_i(X, Y_i, Z_i) = \sum_{\substack{\alpha+\beta+\gamma \le 4^i \\ \alpha,\beta,\gamma \in \mathbb{N}}} c_{\alpha,\beta,\gamma}\, X^\alpha Y_i^\beta Z_i^\gamma,$$

for some coefficients $c_{\alpha,\beta,\gamma} \in \mathbb{R}$, and we have $P_i(x, f_{h-i}(x), g_{h-i}(x)) = 0$. We substitute in the latter equation $f_{h-i}(x)$ and $g_{h-i}(x)$ by their definition using $f_{h-i-1}(x)$ and $g_{h-i-1}(x)$, respectively. More precisely, and to shorten the expressions, we have for some $a_1, \ldots, a_6, b_1, \ldots, b_6 \in \mathbb{R}$,

$$\begin{aligned}
A_1(x) &= a_1 f_{h-i-1}(x) + a_2 \\
A_2(x) &= \pm x^2 + a_4 f_{h-i-1}(x)^2 + a_5 f_{h-i-1}(x) + a_6 \\
f_{h-i}(x) &= A_1(x) + a_3\sqrt{A_2(x)} \\
B_1(x) &= b_1 g_{h-i-1}(x) + b_2 \\
B_2(x) &= x^2 + b_4 g_{h-i-1}(x)^2 + b_5 g_{h-i-1}(x) + b_6 \\
g_{h-i}(x) &= B_1(x) + b_3\sqrt{B_2(x)}.
\end{aligned}$$

We thus get the equation

$$0 = \sum_{\substack{\alpha+\beta+\gamma \le 4^i \\ \alpha,\beta,\gamma \in \mathbb{N}}} c_{\alpha,\beta,\gamma}\, x^\alpha \Big(A_1(x) + a_3\sqrt{A_2(x)}\Big)^\beta \Big(B_1(x) + b_3\sqrt{B_2(x)}\Big)^\gamma.$$

We expand the terms $(A_1(x) + a_3\sqrt{A_2(x)})^\beta$ and $(B_1(x) + b_3\sqrt{B_2(x)})^\gamma$ using the binomial theorem and, in the resulting equation, we replace

- each term $(\sqrt{A_2(x)})^{\beta'}$ with even β' by $A_2(x)^{\beta'/2}$;
- each term $(\sqrt{A_2(x)})^{\beta'}$ with odd β' by $A_2(x)^{(\beta'-1)/2}\sqrt{A_2(x)}$;
- each term $(\sqrt{B_2(x)})^{\gamma'}$ with even γ' by $A_2(x)^{\gamma'/2}$;
- each term $(\sqrt{B_2(x)})^{\gamma'}$ with odd γ' by $A_2(x)^{(\gamma'-1)/2}\sqrt{B_2(x)}$.

We group the terms with a factor $\sqrt{A_2(x)}$ or $\sqrt{B_2(x)}$ remaining. For some polynomial $Q_1(X, Y, Z)$ of degree at most 4^i, polynomials $Q_2(X, Y, Z)$ and $Q_3(X, Y, Z)$ of degree at most $4^i - 1$ and polynomial $Q_4(X, Y, Z)$ of degree at most $4^i - 2$ we get an equation

$$\begin{aligned} 0 \;=\; & Q_1(x, f_{h-i-1}(x), g_{h-i-1}(x))+ \\ & Q_2(x, f_{h-i-1}(x), g_{h-i-1}(x))(\sqrt{A_2(x)})+ \\ & Q_3(x, f_{h-i-1}(x), g_{h-i-1}(x))(\sqrt{B_2(x)})+ \\ & Q_4(x, f_{h-i-1}(x), g_{h-i-1}(x))(\sqrt{A_2(x)}\sqrt{B_2(x)}). \end{aligned}$$

We pass the terms with $\sqrt{A_2(x)}$ to one side and all the other terms to the other side. We square the equation, expand each side, replace each $(\sqrt{A_2(x)})^2$ with $A_2(x)$, and replace each $(\sqrt{B_2(x)})^2$ with $B_2(x)$. We are left with an equation where some terms include the factor $\sqrt{B_2(x)}$; all the other terms are polynomial in x, $f_{h-i-1}(x)$ and $g_{h-i-1}(x)$. We collect on one side the terms with $\sqrt{B_2(x)}$, square both sides of the equation, and replace each $(\sqrt{B_2(x)})^2$ with $B_2(x)$. We are left with an equation that is polynomial in x, $f_{h-i-1}(x)$ and $g_{h-i-1}(x)$; this equation defines the polynomial $P_{i+1}(X, Y_{i+1}, Z_{i+1})$. Since we have done reorganizations and have squared both sides of the equation twice, the degree of the polynomial P_{i+1} is at most 4 times the degree of P_i. Therefore P_{i+1} has degree at most 4^{i+1}.

For $i = h$, we obtain a polynomial $P_h(X, Y_h, Z_h)$ of degree at most 4^h such that the solutions to $P_h(x, f_0(x), g_0(x)) = 0$ contains the solutions for $f(x) = g(x)$. Note that the equation $P_h(x, f_0(x), g_0(x)) = 0$ may contain some additional solutions that are added through the algebraic manipulation, possibly also solutions that are not in the domains of $f(x)$ or $g(x)$. The equation $P_h(x, f_0(x), g_0(x)) = 0$ is a polynomial of degree at most 4^h in x because $f_0(x)$ and $g_0(x)$ are linear.

For each $i \in [h]$, because the polynomial $P_i(X, Y_i, Z_i)$ has degree at most 4^i, it is defined by $2^{\mathcal{O}(i)}$ coefficients, and each of its coefficients comes from making $2^{\mathcal{O}(i)}$ operations through the computation. We conclude that all the polynomials can be computed in $2^{\mathcal{O}(h)}$ time. □

6 Parametric Version

In this section we will solve the initial optimisation problem Connectivity for uncertainty regions given as line segments. Given a set $\mathcal{S} = \{s_1, s_2, \ldots, s_k\}$ of segments and a set $\mathcal{P} = \{p_{k+1}, p_{k+2}, \ldots, p_n\}$ of points in the plane, find δ^*, which is the smallest $\delta \geq 0$, such that the decision problem DConnectivity on inputs $\mathcal{S}$, $\mathcal{P}$ and δ has the answer *TRUE*. We can shortly write $\delta^* = \text{Connectivity}(\mathcal{S}, \mathcal{P}) = \min\{\delta \mid \text{DConnectivity}(\mathcal{S}, \mathcal{P}, \delta)\}$.

We will use parametric search. The idea is to simulate the decision algorithm described in Sect. 4 for the unknown value δ^*. Through the algorithm we maintain two values $\delta_m < \delta_M$ such that the interval $(\delta_m, \delta_M]$ contains δ^* and such that, for any $\delta \in (\delta_m, \delta_M)$, the algorithm branches in the same way, that is, the combinatorial decisions of the algorithm are the same. Thus, the algorithm has the same outline as it was used for describing the algorithm for the problem DConnectivity. This means that it will be given in 4 parts: **Step** 1, **Step** 2, **Loop** and **Step** 3. These 4 parts will be analogous to the parts with the corresponding names in the algorithm for the problem DConnectivity.

Throughout our algorithm, we will constantly update δ_m and δ_M such that the value of δ_m will never decrease, the value of δ_M will never increase and δ^* will be in the interval $(\delta_m, \delta_M]$. We will mostly update δ_m and δ_M by using *parametric search among some set of values* $\Delta = \{\delta_1, \delta_2, \ldots, \delta_N\}$. This means that we will discard the values from Δ outside the interval (δ_m, δ_M) and, for the sake of simpler description, we will add the values δ_m and δ_M to Δ. Then, we will sort the values in Δ and we will do a binary search to determine two consecutive values $\delta'_1 < \delta'_2$, such that DCONNECTIVITY$(\mathcal{S}, \mathcal{P}, \delta'_1) = FALSE$ and DCONNECTIVITY$(\mathcal{S}, \mathcal{P}, \delta'_2) = TRUE$. We will update the values $\delta_M = \delta'_2$ and $\delta_m = \delta'_1$. Clearly, it will hold $\delta_m < \delta^* \leq \delta_M$ and none of the values that were initially in Δ will be in the interval (δ_m, δ_M). For this step in parametric search, we spend $\mathcal{O}(N \log N)$ time plus the time needed to solve $\mathcal{O}(\log N)$ decision problems.

We use the preprocessing of Theorem 10: after a preprocessing of $\mathcal{O}(k^2 n \log n)$ time, we can solve each decision problem performing $k^{\mathcal{O}(k)} n$ operations. The preprocessing is performed only once. Afterwards, each parametric search among a set of N values takes $\mathcal{O}(N \log N) + k^{\mathcal{O}(k)} n \log N$ steps.

Step 1. To get an upper bound on δ^*, we first choose arbitrary points $p_1 \in s_1, \ldots, p_k \in s_k$ and compute a MST on points $p_1, p_2, \ldots, p_n$. We define δ_M as the maximum length of an edge in this MST. To set a proper lower bound on δ^*, we run the decision algorithm for $\delta = 0$ and, if it returns *TRUE*, we return $\delta^* = 0$. Otherwise we define $\delta_m = 0$. We see that $\delta_m < \delta^* \leq \delta_M$.

To continue with **Step** 1, we compute a minimum spanning tree T for the $n - k$ points in $\mathcal{P}$. Let $e_1, \ldots, e_{n-k-1}$ be the edges of T sorted by length such that $|e_1| \geq |e_2| \geq \cdots \geq |e_{n-k-1}|$. We do a parametric search among the values $|e_1|, |e_2|, \ldots, |e_{\min\{n-k-1, 4k+1\}}|$ to update δ_m and δ_M.

Next, we remove all edges of T that are at least as long as δ_M. By Lemma 4, if there are any remaining edges in T, δ_m is the length of the longest of the remaining edges. Let the remaining connected components of the tree T be $\mathcal{C} = \{C_1, C_2, \ldots, C_\ell\}$. Note that we removed exactly $\ell - 1$ edges. If $\ell > 4k + 1$, we return $\delta^* = \delta_M$. This can be done because of Lemma 4. The following lemma clearly holds.

Lemma 14 *For each $\delta \in (\delta_m, \delta_M)$, the algorithm for the problem* DCONNECTIVITY *on input $(\mathcal{S}, \mathcal{P}, \delta)$ produces the same MST T and the set of components $\mathcal{C}$ in* ***DStep*** 1 *as were obtained after* ***Step*** 1.

In **Step** 1 we used the algorithm for the decision problem DCONNECTIVITY $\mathcal{O}(\log k)$ times. Hence, **Step** 1 runs in time $\mathcal{O}(n \log n) + k^{\mathcal{O}(k)} n \log k = k^{\mathcal{O}(k)} n \log n$.

Step 2. As in **DStep** 2, at the end of this step, we would like, for each component $C_i \in \mathcal{C}$ and for each line segment $s_j \in \mathcal{S}$, to have the Voronoi diagram on s_j of the points in C_i. Note that this was already computed during the preprocessing of Theorem 10, this means before **Step** 1. Hence, we do nothing on this "step".

Loop. We treat each line segment from $\mathcal{S}$ and each component from $\mathcal{C}$ as an abstract vertex and we iterate over all topology trees τ on these $k + \ell$ vertices.

Step 3. We simulate **DStep** 3 while doing parametric search. Given a topology tree τ, we iterate over all of its significant topological subtrees. We restrict our attention to

one fixed significant topological subtree τ'. Let the set of vertices of τ' be $V' \subseteq \mathcal{S} \cup \mathcal{C}$. We denote $\mathcal{S}' = \mathcal{S} \cap V'$ and $\mathcal{C}' = \mathcal{C} \cap V'$. By definition of τ', we know that $\mathcal{S}'$ is not empty. Let us choose a root $s_r \in \mathcal{S}'$ of τ'. For each node $s_i \in \mathcal{S}'$ of τ', let $\tau'(s_i)$ be the subtree of τ' rooted at s_i, and let its height $h(s_i) \in \mathbb{N}$ be the number of edges on a longest path in τ' that begins in s_i and is contained in $\tau'(s_i)$. Note that each such a longest path must end in a leaf of τ'.

As in **DStep** 3, we use dynamic programming bottom-up along τ'. For each segment node $s_i \in \mathcal{S}'$ of τ', we compute $X_i(\delta)$, as defined in **DStep** 3, but taking δ as a parameter that takes values inside the interval (δ_m, δ_M). It will be convenient to use that $X_i(\delta)$ increases with δ: whenever $\delta' < \delta$, we have $X_i(\delta') \subseteq X_i(\delta)$.

If we have a leaf s_i from $\mathcal{S}'$, then we have $X_i(\delta) = s_i$. Consider now a segment node $s_i \in \mathcal{S}'$ of τ'. As in Lemma 8, we may reindex the nodes, if needed, and assume that the children of s_i in τ' are $s_1, \ldots, s_t$ and $C_1, \ldots, C_u$. Then, by Lemma 8 we have

$$X_i(\delta) = \bigcap_{\ell=1}^{t} \big(X_\ell(\delta) \oplus D(0,\delta)\big) \bigcap_{\ell=1}^{u} \big(C_\ell \oplus D(0,\delta)\big) \bigcap s_i.$$

We will use parametric search to determine the size of the segmentation $X_i(\delta)$ and we will represent its components as at most $h(s_i)$-square root functions of δ, as defined in Sect. 5. Because we process the tree τ' bottom-up, we can assume that the sizes of segmentations $X_\ell(\delta)$, for $\ell \in [t]$, are already fixed in the interval $\delta \in (\delta_m, \delta_M)$ and that their components are at most $(h(s_i) - 1)$-square root functions of δ.

Computing an intersection of a δ-neighborhood of a segmentation with a line segment. Let us first describe how we can compute each of the t operations

$$Y_\ell(\delta) = \big(X_\ell(\delta) \oplus D(0,\delta)\big) \cap s_i.$$

We will often leave out in the notation the dependency on ℓ and i. We will closely follow the algorithm VII) from Sect. 3.

Let the line segment s_i be $s_i = (p_s, e_s, a_s, b_s)$. Let the segmentation X_ℓ be $X_\ell(\delta) = (p_X, e_X, a_1(\delta), b_1(\delta), \ldots, a_N(\delta), b_N(\delta))$, where $a_j(\delta)$ and $b_j(\delta)$, for $j \in [N]$ are $(h(s_i) - 1)$-square root functions of δ. For $j \in [N]$, let $\sigma_j(\delta)$ be the segment $\sigma_j(\delta) = (p_X, e_X, a_j(\delta), b_j(\delta))$, let $\gamma_j(\delta)$ be the boundary of $\sigma_j(\delta) \oplus D(0,\delta)$, and let $\eta_j(\delta)$ be the intersection of s_i with $\sigma_j(\delta) \oplus D(0,\delta)$. Recall Fig. 6.

We first narrow the interval defined by $\delta_m < \delta_M$ in such a way that, for each single $j \in [N]$, the intersection $\eta_j(\delta)$ is empty for all δ in the interval (δ_m, δ_M) or nonempty for all δ in the interval (δ_m, δ_M). For each $j \in [N]$, we compute the value of δ_j such that $\eta_j(\delta_j)$ is non-empty but $\eta_j(\delta)$ is empty for all $\delta < \delta_j$. Because $X_\ell(\delta)$ increases with δ, there is at most one single δ_j that may satisfy this condition. It may be that δ_j does not exist because $\eta_j(\delta)$ is always non-empty; in this case we set $\delta_j = \delta_m$. Each such value δ_j is a solution to some equation involving the segment s_i and a circle of radius δ centered at $a_j(\delta)$ or $b_j(\delta)$, or lines parallel to e_X at distance δ from σ_j. Because of Lemmas 12 and 13, the value δ_j is a root of a polynomial in δ of degree at most $4^{h(s_i)} \leq 4^k$. We then do a parametric search among the values $\{\delta_1, \ldots, \delta_N\}$ to

update δ_m and δ_M. We can then assume that, for each $j \in [N]$, the segment $\eta_j(\delta)$ is empty for all δ with $\delta_m < \delta < \delta_M$ or non-empty for all δ with $\delta_m < \delta < \delta_M$.

For each $j \in [N]$, we have the (possibly empty) segment $\eta_j(\delta) = (p_s, e_s, a'_j(\delta), b'_j(\delta))$. All these segments have the same reference point p_s and vector e_s. The functions $a'_j(\delta)$ and $b'_j(\delta)$ are at most $h(s_i)$-square root functions because of Lemma 12. To merge the non-empty segments that are overlapping, we have to sort the values $a'_j(\delta), b'_j(\delta), a'_{j+1}(\delta), b'_{j+1}(\delta)$, for each single $j \in [N-1]$. For this, we perform a step of parametric search among the solutions of the equations $a'_j(\delta) = a'_{j+1}(\delta)$, $a'_j(\delta) = b'_{j+1}(\delta)$, $b'_j(\delta) = a'_{j+1}(\delta)$ and $b'_j(\delta) = b'_{j+1}(\delta)$, for all $j \in [N-1]$. Because of Lemma 13, these solutions are roots of polynomials of degree at most 4^k.

To summarize, spending $k^{\mathcal{O}(k)} N \log N$ time to manipulate segments, polynomials of degree at most 4^k and their roots, and performing $O(\log N)$ calls to the decision problem, we have an interval (δ_m, δ_M) where $\big(X_\ell(\delta) \oplus D(0,\delta)\big) \cap s_i$ is described by the same combinatorial structure. In particular, it is described by a segmentation

$$Y_\ell(\delta) \;=\; (p_s, e_s, \tilde{a}_1(\delta), \tilde{b}_1(\delta), \ldots, \tilde{a}_{N'}(\delta), \tilde{b}_{N'}(\delta)) \quad \text{for all } \delta \in (\delta_m, \delta_M).$$

Note that N' depends on δ and ℓ, but it is constant for all $\delta \in (\delta_m, \delta_M)$.

We perform this procedure for each $\ell \in [t]$, where $t \le 5$. If, for some $\delta \in (\delta_m, \delta_M)$ and hence for all $\delta \in (\delta_m, \delta_M)$, we get that $\big(X_\ell(\delta) \oplus D(0,\delta)\big) \cap s_i$ is empty, then we know that the topology tree τ under consideration is not δ-realizable for any $\delta < \delta_M$ and therefore we can move on to the next topology tree τ inside **Loop**.

Computing an intersection of a δ-neighborhood of a set of points with a line segment. Next, we describe each of the u operations $Z_\ell(\delta) = \big(C_\ell \oplus D(0,\delta)\big) \cap s_i$. We will often leave out in the notation the dependency on ℓ and i. Let $(a_1, J_1), \ldots, (a_N, J_N)$ be a Voronoi diagram on s_i for points from C_ℓ. Then

$$Z_\ell(\delta) = \bigcup_{j=1}^{N} \big(D(a_j, \delta) \cap J_j\big).$$

This implies that when δ goes from δ_m to δ_M, the set $Z_\ell(\delta)$ goes through $\mathcal{O}(N)$ combinatorial changes. This is because for each $j \in [N]$, $D(a_j, \delta) \cap J_j$ goes through at most 3 combinatorial changes: when $D(a_j, \delta) \cap J_j \neq \emptyset$ for the first time, when one endpoint of J_j is included in $D(a_j, \delta)$ and when both endpoints of J_j are included in $D(a_j, \delta)$. See Fig. 9.

We compute in time $\mathcal{O}(N)$ all $\mathcal{O}(N)$ values of δ for which these combinatorial changes occur and we do a parametric search among them to update δ_m and δ_M. After that, for any $\delta \in (\delta_m, \delta_M)$, the segmentation $Z_\ell(\delta)$ has fixed size and its components are at most 1-square root functions of δ.

We perform this procedure for each $\ell \in [u]$. Using that $|C_\ell| \le n$ for all $\ell \in [u]$ and that $u \le 5$, we spend a total of $\mathcal{O}(n \log n)$ time plus $\mathcal{O}(\log n)$ calls to the decision problem. If, for some $\delta \in (\delta_m, \delta_M)$ and hence for all $\delta \in (\delta_m, \delta_M)$, we get that for some $\ell \in [u]$ the segmentation $Z_\ell(\delta)$ is empty, then we know that the topology tree τ

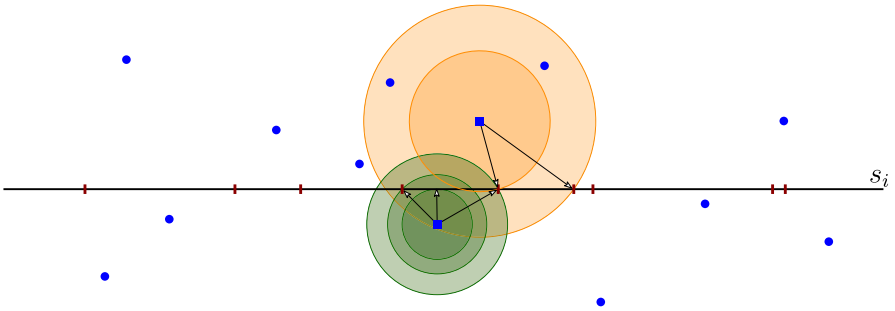

Fig. 9 Example showing for two different points the moments when $D(a_j, \delta) \cap J_j$ combinatorially change. In one case (orange) there are two changes, in the other case (green) there are three changes. The value of δ corresponds to the radius of the disk. The Voronoi diagram is from Fig. 4

under consideration is not δ-realizable for any $\delta < \delta_M$ and therefore we can move on to the next topology tree τ inside **Loop**.

Computing intersections of segmentations. Let us describe how we can compute the intersections of segmentations needed to finish the computation of $X_i(\delta)$. As in the statement of Lemma 8, let t and u denote the number of children of each type for s_i in τ'. At this point we have:

- two values $\delta_m < \delta_M$;
- segmentations $Y_\ell(\delta) = \big(X_\ell(\delta) \oplus D(0, \delta)\big) \cap s_i$, for all $\ell \in [t]$; and
- segmentations $Z_\ell(\delta) = \big(C_\ell \oplus D(0, \delta)\big) \cap s_i$, for all $\ell \in [u]$;

such that for all $\delta \in (\delta_m, \delta_M)$ each segmentation has immutable size (number of segments) and each value describing any part of any segmentation is an $h(s_i)$-square root function.

We have to compute the intersection of these $u + t$ segmentations on s_i. Recall that $u + t$ is bounded by 5 because it is the degree of s_i in τ'. We do this by pairs, which means that we have to compute $t + u - 1 \leq 4$ intersections of pairs of segmentations. We describe how to perform the merge of two segmentations.

Consider two of the segmentations that may appear through the process:

$$X(\delta) = (p_s, e_s, a_1(\delta), b_1(\delta), \ldots, a_N(\delta), b_N(\delta)) \quad \text{of size } N$$
$$X'(\delta) = (p_s, e_s, a'_1(\delta), b'_1(\delta), \ldots, a'_{N'}(\delta), b'_{N'}(\delta)) \quad \text{of size } N'$$

We want to compute $X(\delta) \cap X'(\delta)$. For this, it suffices to sort the values

$$a_1(\delta) \leq b_1(\delta) < a_2(\delta) \leq b_2(\delta) < \ldots < a_N(\delta) \leq b_N(\delta), \quad \text{and}$$
$$a'_1(\delta) \leq b'_1(\delta) < a'_2(\delta) \leq b'_2(\delta) < \ldots < a'_{N'}(\delta) \leq b'_{N'}(\delta)$$

for any $\delta \in (\delta_m, \delta_M)$. After sorting the endpoints, we can easily compute the intersection $X(\delta) \cap X'(\delta)$ in $O(N + N')$ time. Since we are merging two lists that are sorted, we can use Cole's technique [10] for parametric search on networks applied to the bitonic sorting network [15, Section 4.4]. This gives a running time of $\mathcal{O}((N + N') \log(N + N'))$ plus $O(\log(N + N'))$ calls to the decision problem.

Using that $N + N' = \mathcal{O}(n)$, we get a running time of $\mathcal{O}(n \log n) + k^{\mathcal{O}(k)} n \log n$ for the intersection of two segmentations.

Since Cole's technique is complex, we provide an alternative, simpler way of achieving the same time bound to compute the intersection of two segmentations and that uses properties of our setting. The key insight is that the segmentations we are considering do not decrease with δ in the following sense: if $0 \leq \delta_1 < \delta_2$, then $X(\delta_1) \subseteq X(\delta_2)$ and $X'(\delta_1) \subseteq X'(\delta_2)$. This follows from the definition of $X_i(\delta)$.

This implies that the functions $a_j(\delta)$, for $j \in [N]$, and the functions $a'_j(\delta)$, for $j \in [N']$, are (not necessarily strictly) decreasing on the interval (δ_m, δ_M), while the functions $b_j(\delta)$, for $j \in [N]$, and the functions $b_j(\delta)$, for $j \in [N']$ are (not necessarily strictly) increasing on the interval (δ_m, δ_M). Moreover, all these functions are $h(s_i)$-square root functions, defined (at least) on the interval (δ_m, δ_M). By continuity, they are also defined on the interval $(\delta_m, \delta_M]$.

Lemma 15 *There are at most $4^k\, \mathcal{O}(N + N')$ values of δ in the interval (δ_m, δ_M) where the boundary of some segment in $X(\delta)$ may intersect with a boundary of some segment in $X'(\delta)$. We are only considering such pairs of boundaries, where not both boundaries are constant on the interval (δ_m, δ_M). These values can be computed in $2^{\mathcal{O}(k)}(N + N') + \mathcal{O}((N + N') \log(N + N'))$ time.*

Proof Let $\sigma_1(\delta), \ldots, \sigma_N(\delta)$ be the segments in $X(\delta)$; let $\sigma'_1(\delta), \ldots, \sigma'_{N'}(\delta)$ be the segments in $X'(\delta)$. If for some δ the boundary of some segments $\sigma_i(\delta)$ and $\sigma'_j(\delta)$ intersect, then, because the segments are monotonely increasing, $\sigma_i(\delta_M)$ and $\sigma'_j(\delta_M)$ intersect. Here we are only inserting $\delta = \delta_M$ into the boundaries of segments σ_i and σ'_j and we are not considering a possible combinatorial change of $X(\delta)$ or $X'(\delta)$ at $\delta = \delta_M$. This is because we are only interested in limits when $\delta \in (\delta_m, \delta_M)$ approaches δ_M. Because $\sigma_1(\delta_M), \ldots, \sigma_N(\delta_M)$ are pairwise interior disjoint, and $\sigma'_1(\delta_M), \ldots, \sigma'_{N'}(\delta_M)$ are pairwise interior disjoint, there may be at most $\mathcal{O}(N + N')$ pairs of indices

$$\Pi = \{(i, j) \in [N] \times [N'] \mid \sigma_i(\delta_M) \text{ and } \sigma'_j(\delta_M) \text{ intersect}\}.$$

Therefore, it suffices to compute those pairs Π and, for each $(i, j) \in \Pi$ consider the 4 equations $c_i(\delta) = c'_j(\delta)$ with $c_i \in \{a_i, b_i\}$ and $c'_j \in \{a'_i, b'_i\}$. The solutions to those equations are roots of a polynomial of degree at most $4^{h(s_i)}$ because of Lemma 13.

The computation of Π takes $\mathcal{O}((N + N') \log(N + N'))$ time, and then we have to compute the roots of the resulting $\mathcal{O}(N + N')$ polynomials of degree $4^{h(s_i)}$. □

Using the lemma, we compute in $2^{\mathcal{O}(k)}(N + N') + \mathcal{O}((N + N') \log(N + N')) = 2^{\mathcal{O}(k)} n \log n$ time the $4^k\, \mathcal{O}(N + N') = 2^{\mathcal{O}(k)} n$ values of δ where the boundaries of the segments may intersect and do a parametric search among them to update δ_m and δ_M. After that, for any $\delta \in (\delta_m, \delta_M)$, the endpoints of the segments in $X(\delta)$ and $X'(\delta)$ are sorted in the same way, and we can easily compute $X(\delta) \cap X'(\delta)$. Note that the endpoints of the resulting segmentation $X(\delta) \cap X'(\delta)$ keep being described by $h(s_i)$-square root functions because for each endpoint there was an endpoint in $X(\delta)$ or $X'(\delta)$.

We repeat $t+u-1 \leq 4$ times the computation of intersection of two segmentations on s_i, until we obtain

$$X_i(\delta) = \bigcap_{\ell=1}^{t} Y_\ell(\delta) \bigcap_{\ell=1}^{u} Z_\ell(\delta) = \bigcap_{\ell=1}^{t} \big(X_\ell(\delta) \oplus D(0,\delta)\big) \bigcap_{\ell=1}^{u} \big(C_\ell \oplus D(0,\delta)\big) \bigcap s_i.$$

Altogether, we used $k^{\mathcal{O}(k)} n \log n$ steps.

Summary of Step 3. We perform the computation of $X_i(\delta)$ bottom-up along the significant topology tree τ'. For each node $s_i \in \mathcal{S}'$ of τ' we spend $k^{\mathcal{O}(k)} n \log n$ time. At end of processing the significant topology tree τ', we have computed in $k^{\mathcal{O}(k)} n \log n$ time values $\delta_m < \delta_M$ such that the set $X_r(\delta)$ is either empty, for all $\delta_m < \delta < \delta_M$, or non-empty, for all $\delta_m < \delta < \delta_M$. This is because there are no combinatorial changes for $\delta \in (\delta_m, \delta_M)$. After we compute the set $X_r(\delta)$ for each significant topology subtree of τ, we know that at least one of these sets is empty. If all of the sets X_r were non-empty, then δ^* should be at most δ_m by continuity, which cannot be the case.

We repeat **Step** 3 for each topology tree τ. After **Loop** finishes, we return $\delta^* = \delta_M$. Since there are $k^{\mathcal{O}(k)}$ different topology trees to consider, and for each topology tree we spend $k^{\mathcal{O}(k)} n \log n$ time, the algorithm takes $k^{\mathcal{O}(k)} k^{\mathcal{O}(k)} n \log n = k^{\mathcal{O}(k)} n \log n$ time in total.

Theorem 16 *The optimization problem* CONNECTIVITY *for k line segments and $n-k$ points can be solved performing $k^{\mathcal{O}(k)} n \log n$ operations. Here, an operation may include a number that has a computation tree of depth $\mathcal{O}(k)$ whose internal nodes are additions, subtractions, multiplications, divisions or square root computations and whose leaves contain input numbers and a root of a polynomial of degree at most 4^k with coefficients that are obtained from the input numbers using $2^{\mathcal{O}(k)}$ multiplications, additions and subtractions.*

Proof The correctness of the algorithm was argued as the algorithm was described. It remains to discuss the depth of computation tree of the numbers being computed through the algorithm. The depths of the computation tree of numbers used in the preprocessing, **Step** 1 and **Step** 2 are $\mathcal{O}(1)$. Numbers in each iteration of **Loop** are computed independently of the numbers computed in another iteration. The depth of computation trees of numbers used in **Step** 3 is $\mathcal{O}(k)$, but in the calls to the decision problem we are using a root of a polynomial of degree 4^k. Therefore, we are using Theorem 10 with an input number that is a root of a polynomial of degree 4^k that is computed by using Lemma 13. The result follows. □

Without diving into the time needed for the algebraic operations performed by the algorithm and trying to optimize them, we obtain the following.

Corollary 17 *The optimization problem* CONNECTIVITY *for k line segments and $n-k$ points can be solved in $f(k)n \log n$ time for some computable function $f(\cdot)$.*

7 Conclusions

We have shown that the CONNECTIVITY problem for k segments and $n-k$ points in the plane can be solved in $f(k)n\log n$ time, for some computable function $f(\cdot)$. The precise function f depends on the time to manipulate algebraic numbers. The decision problem is simpler, while the algorithm for the optimization problem uses parametric search.

The algorithms can be extended to $\mathbb{R}^d$, for any fixed dimension d, when the uncertain regions are segments. The main observations are the following:

- A MST for a set of points in $\mathbb{R}^d$ has maximum degree $c_d = 2^{\mathcal{O}(d)}$. Indeed, Claim 2 shows that any two edges incident to a point need to have angle at least $\pi/3$. This implies that the maximum degree of the MST is bounded by the kissing number in dimension d, which is known to be $c_d = 2^{\mathcal{O}(d)}$ using a simple volume argument.
- A MST for a set $\mathcal{P}$ of n points in $\mathbb{R}^d$ can be computed in $\mathcal{O}(dn^2)$ time by constructing the complete graph $K_{\mathcal{P}}$ explicitly and using a generic algorithm for MST in dense graphs. The term $\mathcal{O}(d)$ is added because it is needed to compute each distance. More efficient algorithms with a time complexity of $\mathcal{O}(n^{2-\frac{2}{\lceil d/2\rceil+1}+\varepsilon})$, for any $\varepsilon > 0$, are known [1] for any fixed dimension d. (The constant hidden in the O-notation depends on ε.)
- In Lemma 4, we have to consider the components obtained by removing up to kc_d edges of the MST of $\mathcal{P}$.
- The rest of the discussion follows as written. When constructing the Voronoi diagram restricted to a segment s_i and all the other geometric constructions, the dimension of the ambient space does not matter. In fact, when considering a segment s_i, we could just replace the input points by points that are coplanar with the segment and have the same distances to the line supporting the segment.

All together, when d is constant, we get an algorithm for k uncertain line segments and $n-k$ points in $\mathbb{R}^d$ that uses $\mathcal{O}(n^{2-\frac{2}{\lceil d/2\rceil+1}+\varepsilon})+f(k)n\log n$ time, for some computable function $f(\cdot)$. Interestingly, when k is constant, the bottleneck of the computation in our algorithm is obtaining the MST; after that step, we need $O(f(k)n\log n)$ time. When d is unbounded, we get an exponential dependency on d because the number of components in the MST that have to be considered is $O(kc_d)$.

Acknowledgements We are grateful to the reviewers for their feedback and suggestions to improve the paper. Funded in part by the Slovenian Research and Innovation Agency (P1-0297, J1-1693, J1-2452, N1-0218, N1-0285). Funded in part by the European Union (ERC, KARST, project number 101071836). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Author Contributions Equal contribution by the authors.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Agarwal, P.K., Edelsbrunner, H., Schwarzkopf, O.: Euclidean minimum spanning trees and bichromatic closest pairs. Discret. Comput. Geom. **6**, 407–422 (1991). https://doi.org/10.1007/BF02574698
2. Bae, S.W., Choi, S., Lee, C., Tanigawa, S.: Exact algorithms for the bottleneck Steiner tree problem. Algorithmica **61**(4), 924–948 (2011). https://doi.org/10.1007/s00453-011-9553-y
3. Bae, S.W., Lee, C., Choi, S.: On exact solutions to the Euclidean bottleneck Steiner tree problem. Inf. Process. Lett. **110**(16), 672–678 (2010). https://doi.org/10.1016/j.ipl.2010.05.014
4. Bandyapadhyay, S., Lochet, W., Lokshtanov, D., Saurabh, S., Xue, J.: Euclidean bottleneck Steiner tree is fixed-parameter tractable, 2023. To appear in SODA (2024). Preprint available at https://arxiv.org/abs/2312.01589
5. Basu, S., Pollack, R., Roy, M.-F.: *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer, Berlin, 2nd edition, (2006)
6. Bose, P., D'Angelo, A., Durocher, S.: On the restricted k-steiner tree problem. J. Comb. Optim. **44**(4), 2893–2918 (2022). https://doi.org/10.1007/s10878-021-00808-z
7. Brazil, M., Ras, C.J., Swanepoel, K.J., Thomas, D.A.: Generalised k-Steiner tree problems in normed planes. Algorithmica **71**(1), 66–86 (2015). https://doi.org/10.1007/s00453-013-9780-5
8. Burnikel, C., Funke, S., Mehlhorn, K., Schirra, S., Schmitt, S.: A separation bound for real algebraic expressions. Algorithmica **55**(1), 14–28 (2009). https://doi.org/10.1007/s00453-007-9132-4
9. Chambers, E.W., Erickson, A., Fekete, S.P., Lenchner, J., Sember, J., Venkatesh, S., Stege, U., Stolpner, S., Weibel, C., Whitesides, S.: Connectivity graphs of uncertainty regions. Algorithmica **78**(3), 990–1019 (2017). https://doi.org/10.1007/s00453-016-0191-2
10. Cole, R.: Slowing down sorting networks to obtain faster sorting algorithms. J. ACM **34**(1), 200–208 (1987). https://doi.org/10.1145/7531.7537
11. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, (2008). https://doi.org/10.1007/978-3-540-77974-2
12. Eppstein, D.: Quasiconvex programming. In Goodman, J.E. Pach J. and Welzl E., editors, *Combinatorial and Computational Geometry*, volume 52 of *MSRI Publications*. Cambridge Univ. Press, 2005. Preprint at http://arxiv.org/abs/cs/0412046
13. Ganley, J.L., Salowe, J.S.: Optimal and approximate bottleneck Steiner trees. Oper. Res. Lett. **19**(5), 217–224 (1996). https://doi.org/10.1016/S0167-6377(96)00028-4
14. Georgakopoulos, G., Papadimitriou, C.H.: The 1-Steiner tree problem. J. Algorithms **8**(1), 122–130 (1987). https://doi.org/10.1016/0196-6774(87)90032-0
15. Joseph, F.: JáJá. Addison-Wesley, An Introduction to Parallel Algorithms (1992)
16. Kettner, L., Mehlhorn, K., Pion, S., Schirra, S., Yap, C.-K.: Classroom examples of robustness problems in geometric computations. Comput. Geom. **40**(1), 61–78 (2008). https://doi.org/10.1016/j.comgeo.2007.06.003
17. Li, C., Pion, S., Yap, C.-K.: Recent progress in exact geometric computation. J. Log. Algebraic Methods Program. **64**(1), 85–111 (2005). https://doi.org/10.1016/j.jlap.2004.07.006
18. Löffler, M., van Kreveld, M.: Largest and smallest convex hulls for imprecise points. Algorithmica **56**(2), 235–269 (2010). https://doi.org/10.1007/s00453-008-9174-2
19. Löffler, M., van Kreveld, M.: Largest bounding box, smallest diameter, and related problems on imprecise points. Comput. Geom. **43**(4), 419–433 (2010). https://doi.org/10.1016/j.comgeo.2009.03.007
20. Megiddo, N.: Combinatorial optimization with rational objective functions. Math. Oper. Res. **4**(4), 414–424 (1979). https://doi.org/10.1287/moor.4.4.414

21. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. J. ACM **30**(4), 852–865 (1983). https://doi.org/10.1145/2157.322410
22. Mehlhorn, K., Schirra, S.: Exact computation with leda_real—theory and geometric applications. In Alefeld, G., Rohn, J., Rump, S. and Yamamoto T., editors, *Symbolic Algebraic Methods and Verification Methods*, pages 163–172. Springer, (2001)
23. Monma, C., Suri, S.: Transitions in geometric minimum spanning trees. Discret. Comput. Geom. **8**(3), 265–293 (1992)
24. Sarrafzadeh, M., Wong, C.K.: Bottleneck Steiner trees in the plane. IEEE Trans. Comput. **41**(3), 370–374 (1992). https://doi.org/10.1109/12.127452
25. Shamos, M.I., Hoey, D.: Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (SFCS 1975)*, pages 151–162, (1975). URL: https://doi.org/10.1109/SFCS.1975.8
26. Sharma, V., Yap, C.K.: Robust geometric computation. In Goodman, J.E. O'Rourke, J. and Tóth, C.D. editors, *Handbook of Discrete and Computational Geometry*, pages 1189–1223. Chapman and Hall/CRC, 3rd edition, (2017). URL: https://doi.org/10.1201/9781315119601
27. van Kreveld, M., Löffler, M.: Approximating largest convex hulls for imprecise points. J. Discrete Algorithms **6**(4), 583–594 (2008). https://doi.org/10.1016/j.jda.2008.04.002
28. Wang, L., Ding-Zhu, D.: Approximations for a bottleneck steiner tree problem. Algorithmica **32**(4), 554–561 (2002). https://doi.org/10.1007/s00453-001-0089-4
29. Wang, L., Li, Z.: An approximation algorithm for a bottleneck k-steiner tree problem in the euclidean plane. Inf. Process. Lett. **81**(3), 151–156 (2002). https://doi.org/10.1016/S0020-0190(01)00209-5
30. Yap, C.-K.: *Fundamental problems of algorithmic algebra*. Oxford University Press, (1999)