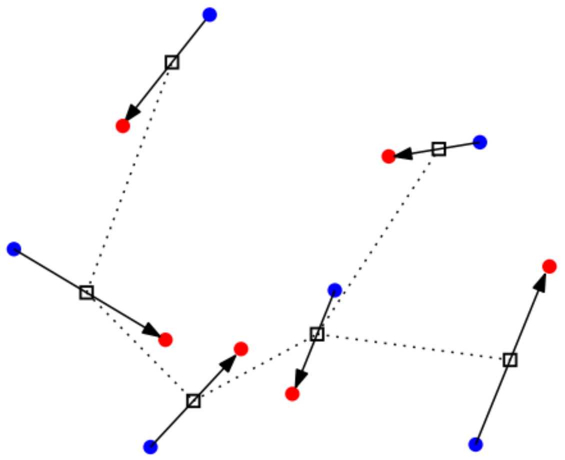
Insights into the SoA in Graph Theory & Algorithms

Vasileios Papastergios (Academic ID: 3651)

May 25, 2023



Aristotle University of Thessaloniki
School of Informatics

SCHOOL OF INFORMATICS

Course name:	Graph Theory & Algorithms
Course ID:	NIS-06-06
Semester:	6th
Instructor:	Charalampos Kouzinopoulos

Contents

List of Figures

1 Introduction

1.1 General Information

The present document serves as technical report for the bibliographic assignment in the course of *Graph Theory & Algorithms*. The author attended the course during their 6th semester of BSc. studies at the School of Informatics, AUTh. The topic of the assignment is a *literature survey* on the *State of the Art* (SoA) in the field of Graph Theory & Algorithms.

1.2 Abstract

In this literature survey, we provide insights into SoA work and results published within 2018 – 2023 related to *spanning tree problems*. At first, we investigate problems where, given a set $\mathcal{V}$ of n vertices, the goal is to find a tree that connects all n vertices in a way that the maximum weight of an edge in the graph (bottleneck) is minimized. This is called the *Minimum Bottleneck Spanning Tree* (MBST) problem. We study two interesting variations of the MBST problem; the first imposes a degree upper bound for the vertices of the constructed tree and the latter lets the vertices be linearly moving points in $\mathbb{R}^{\text{dim}}$. However, there exist problems, where minimizing the weight of the *whole tree* is required, rather than only its maximum-cost edge. This is called the *Minimum Spanning Tree* (MST) problem. Within the context of this assignment, we elaborate on MST problems where the vertices are moving points, the edges represent the distances between them under some distance function and the objective is to construct a single MST for the whole motion.

1.3 Preliminaries

It is well-known that we can use graph theory in order to represent a wide variety of real-life situations and/or problems. Computer networks, spatial data, social networks, traffic management, web pages ranking are just (very) few of these applications. In all these cases, the set $\mathcal{V}$ of vertices models real-world components and the set $\mathcal{E}$ of edges expresses the relations between them. For example, when examining a computer network, we can consider the network devices as the vertices and the connection channels between them as the edges of a graph that represents the network architecture. Taking one step further, we can assign weights on the edges based on the latency over any channel that connects two devices in the network.

Having transferred the problem into the field of Graph Theory, we can, then, apply graph algorithms on it, aiming to solve a wide range of problems. A quite often case asks for a way to keep all components connected with the minimum total cost. Back in our example, this can be translated as asking for the most suitable network architecture that, given the latency over any channel of the network, keeps all devices connected with one another with the minimum possible latency for the network as a whole, as well as the minimum number of established connections (channels). That is the Minimum Spanning Tree (MST) problem.

In other applications, we are interested in minimizing only the largest-cost edge, not necessarily the total cost of the graph. In the example of the computer network, one can ask for minimizing the maximum latency over a communication channel. Such an approach can be easily explained if we take into account that the maximum latency over a specific channel of the network, usually, acts as a bottleneck for the data flow rate, imposing an upper bound on the speed of all connections (even if they can support higher ones). As its name suggests, what we have just described is the Minimum Bottleneck Spanning Tree (MBST) problem. The latter is the first to discuss, right in the next section.

2 The Minimum Bottleneck Spanning Tree Problem

In this section we discuss about the latest publications concerning advances in the MBST problem. In the first place, we limit the problem down to two dimensions, the Euclidean plane, and we, also, impose an upper-bound for the degree of all the MBST vertices. Under these limitations, we attempt to analyze the recent improvements in the degree-bounded Euclidean MBST ratios [2]. Right after that, we remove one by one these limitations: at first we focus on computing the MBST on a set of moving points in the Euclidean plane [10], and, next we generalize the MBST computation for a set of moving points in $\mathbb{R}^{\text{dim}}$ [1].

2.1 Paper 1: "Euclidean Bottleneck Bounded-Degree Spanning Tree Ratios" [2]

In this subsection we emphasize on Euclidean Minimum BSTs, imposing bounds on the degree of their vertices. We investigate the supremum ratios between the bottleneck of the degree-bounded to the bottleneck of the degree-free BST with the same set of vertices and edges.

2.1.1 Definitions

Before stepping into the SoA results, we consider useful to define the fundamental concepts we are going to use throughout the lines that follow.

- **Euclidean MBST**: an MBST whose vertices are points in the 2D plane and its edges have weight (length) equal to the Euclidean distance between the endpoints
- **degree-K-bounded MBST** or simply **degree-K BST**: an MBST that all its vertices have degree at most K, for an integer $K \geq 2$
- **supremum ratio** β_K: the ratio of the largest edge-length of the degree-K BST to the largest edge-length of the BST for the same set $\mathcal{V}$ of vertices, for an integer $K \geq 2$

2.1.2 Paper results

In the lines that follow we present the results provided by Biniaz [2]. To start with, the paper is about the supremum ratios β_K of degree-K BSTs. The author starts by describing the context of their work, based on related previous work conducted by other researchers in this particular area. One of the most important context baselines is the fact that all finite point sets in the Euclidean plane have an MBST of degree at most 5 [9, 2]. That automatically means $\beta_K = 1$, for any integer $K \geq 5$, since for these values of K the MST is identical with the MBST for any given finite set point in the plane.

However, that is not the case for lower values of K; in particular $2 \leq K \leq 4$. For this range, the degree-K BST in not always identical with the MST and that is where the supremum ratios appear. Before the paper we investigate, the scientific community had proven the following bounds: $2 \leq \beta_2 \leq 3$, $\sqrt{2} \leq \beta_3 \leq 2$, $1.175 \leq \beta_4 \leq \sqrt{3}$. The lower bounds for all three supremum ratios can be easily proved using simple geometry and the Pythagorean Theorem, as illustrated in figure 1

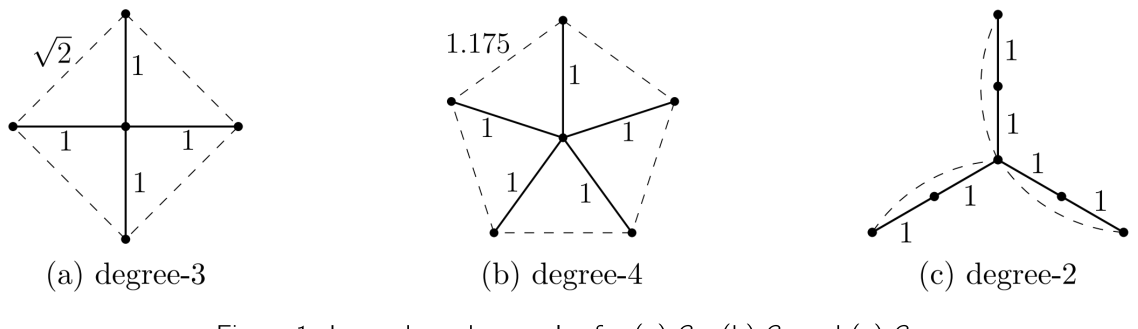


Figure 1: Lower bound examples for (a) β_3, (b) β_4 and (c) β_2

Biniaz [2] presents new, improved bounds. In order to do so, they utilize previously proven lemmas and theorems. The latter place lower and upper bounds for the angles that are shaped between the edges of a vertex on an MST with the same points. The presented algorithm is slightly altered (specialized) for different values of K in $2, 3, 4$. However, the fundamental reasoning of the algorithm remains the same. More specifically, the algorithm takes as input a degree-K MST, which is identical with the degree-K BST, as we mentioned before. Based on that tree, the algorithm tries to transform it by adding and removing edges, in order to construct a k-dergee BST. Getting a deeper insight into the implementation details, the algorithm starts from the root and recursively transforms the sub-trees of the currently examined node. During this process, the algorithm tries to bound edges from vertices that have high degree ($> K$) with simultaneous addition of edges with the minimum possible overhead. At the same time, the algorithm secures that the outcome stays connected. Figure 2 presents in a graphical way an abstract execution of the algorithm.

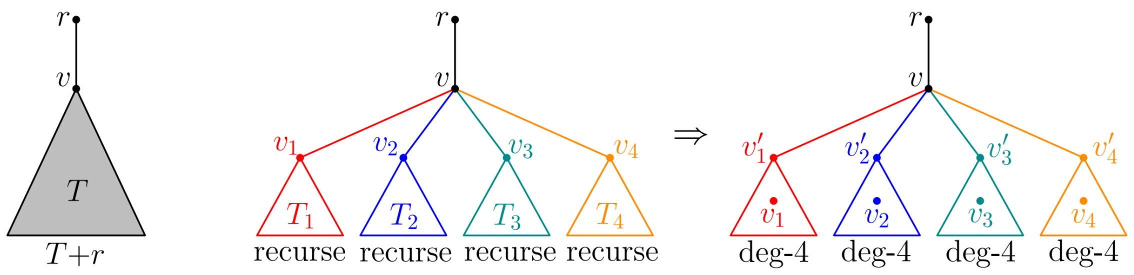


Figure 2: Abstract representation of the presented algorithm for k-degree BST construction based on an MST

In every recursion step, the algorithm differentiates its behavior, based on the number of children of the examined vertex. In all versions of the algorithm (for the different values of the degree bound K), the algorithm re-examines in every step the sub-tree that consists of the parent + child, i.e. if the examined node has 4 children, the algorithm re-examines 4 sub-trees, and all of them contain the parent node, plus the separate branch. As a result, at the end of every edge, the currently examined node may or may not has an edge to its initial children. Biniaz [2] defines as *natural* the children that remain connected to the parent after transformations, and *adopted* the children that are changed in the process (emerge as a result of the transformations). Figure 3 illustrates the different transformations (edge removals and additions) the algorithm performs, switching cases based on the number of children k (lowercase) of the currently examined node.

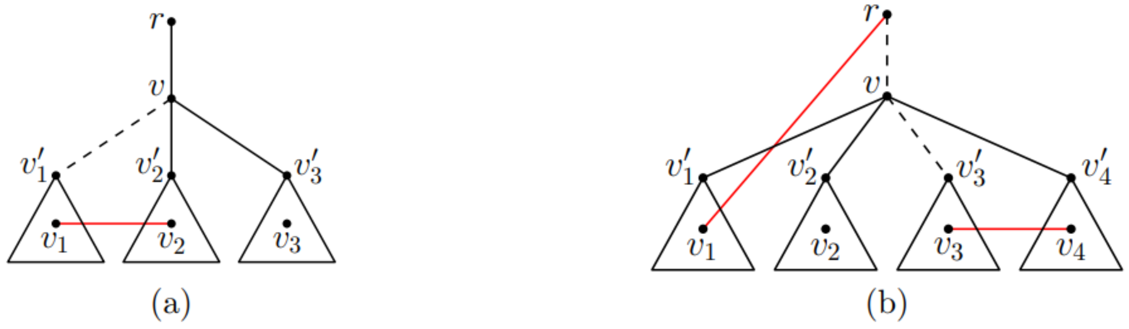
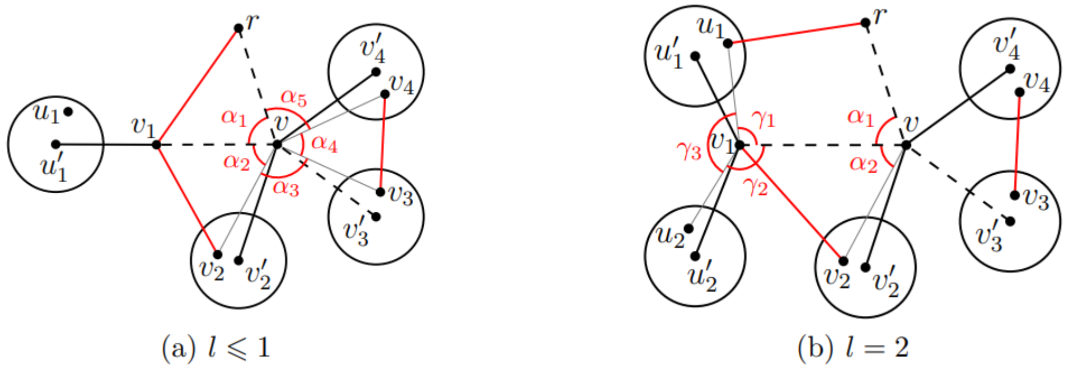


Figure 3: The transformations performed by the algorithm when (a) $k = 3$ and (b) $k = 4$

On that point, we would like to get a deeper insight into the slight, yet interesting and important, difference that Biniaz [2] suggests for the 3-degree BST construction algorithm. The difference lies on the sub-tree that is examined in each recursion step of the algorithm. Up to that point, the sub-tree consists of the parent and its child, for as many children as the examined node had. In the 3-degree BST construction version of the algorithm, however, one more level is added to the examined sub-trees, i.e. the parent, the child and the *grandchildren*. In a similar reasoning, the algorithm distinguishes different cases, depending on the number of grandchildren l the currently examined node has. Figure 4 depicts the different transformations of the 3-degree

BST construction algorithm, based on a 3-leveled sub-tree and re-examining the edges of the node, its children and its grandchildren.



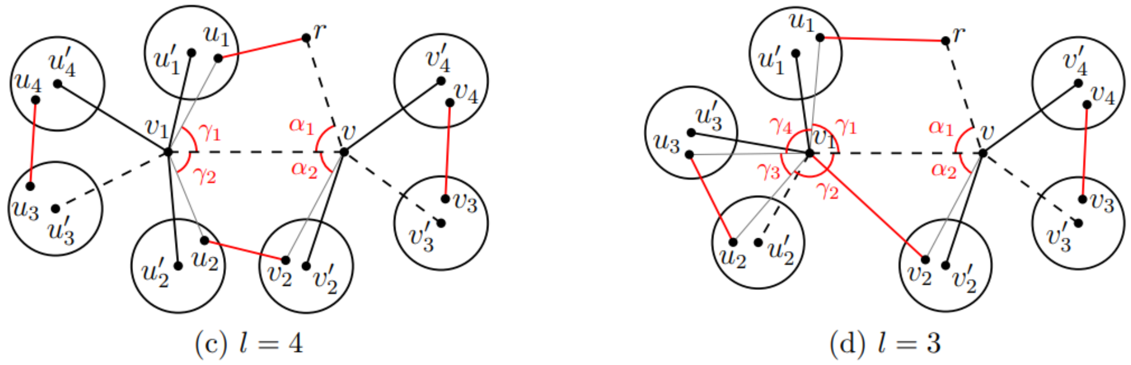


Figure 4: The transformations performed by the algorithm when $K = 3$ and the first children of the examined node has (a) $l \leq 1$, (b) $l = 2$, (c) $l = 4$ and (d) $l = 3$ children

From a theoretical scope, the algorithm recursively constructs smaller parts of the solution. More specifically, the algorithm traverses down to the leaves of the given MST and makes appropriate transformations, in order to satisfy the degree-bound. Based on these smaller solutions (bottleneck spanning sub-trees), the algorithm progressively builds on them, towards the final solution, i.e. construct the K-degree BST. Clear it is, therefore, that the algorithm belongs to the category of *dynamic programming* algorithms. Biniaz [2] utilizes this algorithm in combination with strict theorems, both existing and newly introduced by himself. The result of this process is the definition of new, improved, ratios: $\beta_3 \leq \sqrt{3}$ and $\beta_4 \leq \sqrt{2}$.

Apart from the upper bounds, the author presents an improvement on the lower bound of β_2. Up to that point, the known lower bound for the 2-degree supremum ratio was $\beta_2 \geq 2$. This bound is improved to $\beta_2 \geq \sqrt{7}$. In order to do so, they present a specific point set, whose 2-degree BST bottleneck edge is *at least* $\sqrt{7}$ times larger compared to the (no degree-bounded) BST. Such an ascertainment, immediately implies the improved lower bound. Figure 5 shows the point set of the provided counter-example.

The point set contains 19 points in the Euclidean plane. Every angle at each degree-3 vertex is 120°, and every angle at each degree-2 vertex is 180°. Biniaz [2] partitions all vertices, except for p, into three different sets, $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$, that group the vertices topologically. The figure also exhibits the MST of the given point set (in black edges), where every edge has length equal to 1.

On that point, we consider it important to clarify that finding the 2-degree BST is, actually identical with finding a minimum bottleneck *Hamiltonian path* δ that covers all 19 vertices with the minimum largest-edge length. That notice is a key point in the author's argument in proving the previously mentioned lower bound. More specifically, the author presents a proof by exhaustion (case analysis) of the incoming and outgoing edges of the vertex p.

All the possible combinations are examined, concerning the source and the destination of the edges that come in or out the vertex p. In several cases the author introduces strict theorems and lemmas proven be

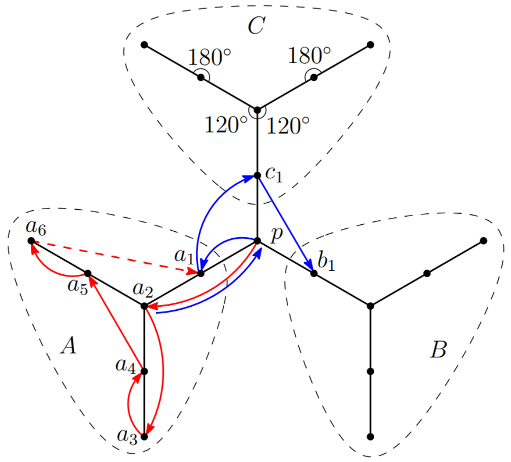


Figure 5: The counter-example point set that proves $\beta_2 \geq \sqrt{7}$

previous researchers concerning Hamiltonian paths. The latter theorems provide the mathematical background that validates the proof. In the end, all cases are reduced down to having at least one edge that is (according to the author's terminology) *long* i.e. greater than or equal to $\sqrt{7}$.

To sum up, Biniaz [2] provides important improvements about the supremum ratio β_K for different values of K. These improved bounds are both lower and upper and their mathematical foundation help the scientific community obtain more accurate approximation bounds on algorithms that utilize K-degree BSTs. Nevertheless, we should not forget that the work provided by the author is only applicable when the vertices are fixed points in the two dimensional plane, under the Euclidean distance metric. As Biniaz quotes, "*The study of worst-case ratios in higher dimensions is more vital as the maximum degree of an MST and a BST can be much larger.*". In the analysis that follow, we are going to gradually remove these limitations, starting from the *fixed-points* one, right in the next subsection.

2.2 Paper 2: "Computing the Minimum Bottleneck Moving Spanning Tree" [10]

In this subsection we remain in the Euclidean plane, but, this time, we let the vertices be *moving* points in $\mathbb{R}^2$. Our goal is to compute a single minimum BST that connects the points throughout the motion.

2.2.1 Definitions

Besides the basic concepts we introduced for the shake of the previous paper, there exist a few more that we are going to reference later on, below. The latter are task-specific for the special problem the paper focuses on. In the first place, we define a set $\mathcal{P}$ of n moving points in $\mathbb{R}^2$. The points are moving in the plane following linear trajectories with constant velocity. Without loss of generality, we consider that the time interval is $t \in [0, 1]$.

Given that point set, we can define a complete graph G_P as the graph whose vertex set is $\mathcal{P}$ and has all possible edges between them. Specifically, the complete graph G_P has an edge between all different pairs $p, q \in \mathcal{P}$, $p \neq q$, such that the edge-length is the distance between p and q, under the Euclidean metric. In full accordance with what we have already presented, based on G_P, we can compute a *Euclidean minimum spanning tree (EMST)*, whose bottleneck edge has the minimum length.

Adjusting the latter definition to the newly introduced condition of *moving* points, we can compute a single spanning tree (that does not change its connection topologically as the points move) that has the minimum bottleneck edge in total. The important note here is the fact that we opt for a *single*, *moving* EMBST during the whole time interval of the motion. Throughout that process, we consider that the *bottleneck* $b(T)$ of a spanning tree T is the maximum *instantaneous bottleneck* $b_T(t)$ of the spanning tree T at time $t \in [0, 1]$, which is the defined time interval. The *minimum* moving-BMST (MBMST) is defined as T^* and its bottleneck as $b(T^*)$.

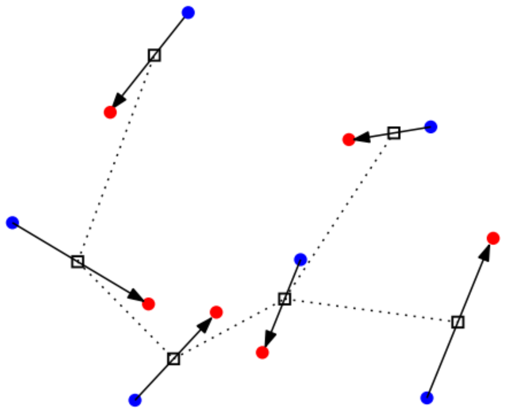


Figure 6: Each pair of red and blue points connected by a black arrow represents a moving point. Blue points denote locations at $t = 0$ and red points are locations at $t = 1$. Black boxes are locations of these moving points at certain time and the dashed segments form a spanning tree. [10]

2.2.2 Paper results

Wang and Zhao [10] present an algorithm for computing an EMBST on a set $\mathcal{P}$ of linearly moving points in the Euclidean plane. The algorithm runs in $O(n^{4/3}log^3 n)$ time, which is better than the general $\mathbb{R}^{\text{dim}}$ algorithm proposed by Akitaya et al [1], which has quadratic time complexity $O(n^2)$. In order to do so, the authors of [10] use as their base observations, theorems and results presented in [1]. We are going to analyze the general $\mathbb{R}^{\text{dim}}$ algorithm in the next section.

One of the most important observations they utilize for their improved algorithm is that $b(T^*)$ must be equal to the maximum Euclidean distance $|pq|_{max}$ for two moving points p and $q \in \mathcal{P}$, where $|pq|_{max} = max\{|p(0)q(0)|, |p(1)q(1)|\}$, i.e. $b(T^*) \in \{|pq|_{max}|p, q \in \mathcal{P}\}$ [1]. In simple words, the bottleneck $b(T^*)$ of the EMBST T^* is maximized either at time $t = 0$ or $t = 1$, given that the time interval is $t \in [0, 1]$. Or, simpler, the maximum distance between two points moving linearly in the Euclidean plane is either at the start or at the end of their motion and not in-between[1, 10].

Nevertheless, it is not correct to consider that the EMBMST $b(T^*)$ is necessarily attained at time $t = 0$ or $t = 1$. Akitaya et al [1] present a counter example (figure 7) that illustrates this exact argument. The figure depicts four points, a, b, c and d that move from time 0 to time 1, as demonstrated by arrays. In figure 7(a), the red tree $\mathcal{R}$ is their MBST at time $t = 0$ and the blue one (tree $\mathcal{B}$) is the MBST at time $t = 1$. However, neither $\mathcal{R}$ nor $\mathcal{B}$ are the MBMST for this point set, since they both have $b(\mathcal{R}) = b(\mathcal{B}) = 3$, while there is a tree $\mathcal{T}_{opt} = \{ac, cb, cd\}$ with bottleneck $b_{opt} = 2$ (figure 7(b)).

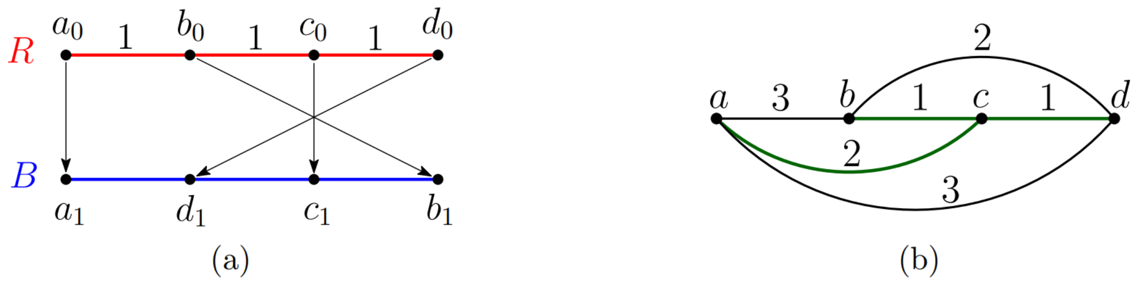


Figure 7: A counter example that EMBMST is not necessarily attained at the start or at the finish of the motion

Based on this observation, Wang & Zhao [10] propose an algorithm for computing the MBMST by dividing the problem into a *decision problem* and an *optimization problem*, which are solved separately. The combination of the two partial solutions produce the utter solution to the problem of computing the EMBMST for a set of moving points in the Euclidean plane. Below, we analyze each one of the two problems thoroughly.

The decision problem: given any $\lambda > 0$, the decision problem is to decide whether for the bottleneck $b(T^*)$ of the EMBST T^* holds $b(T^*) \leq \lambda$. Akitaya et al [10] approach this decision problem by recalling the notion of *unit-disk* graphs. The *unit-disk graph* $G_\lambda(\mathcal{Q})$ for a set $\mathcal{Q}$ of points in the plane with respect to a parameter λ is an undirected graph whose vertex set is $\mathcal{Q}$ such that an edge connects two points $p, q \in \mathcal{Q}$ if $|pq| \leq \lambda$. In other words, the *unit-disk graph* $G_\lambda(\mathcal{Q})$ can be viewed as the intersection graph of the the set of congruous disks centered at the points of Q with radius $\lambda/2$ [10]. Two vertices are connected if and only if their disks intersect, as shown in figure 8.

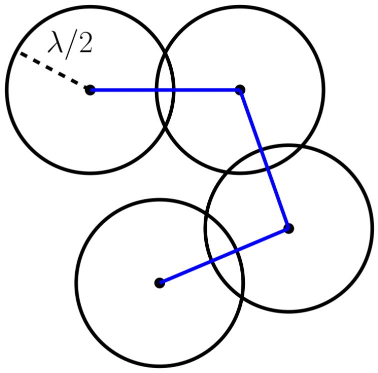


Figure 8: A unit-disk graph $G_\lambda(\mathcal{Q})$ for a set $\mathcal{Q}$ of points [10]

The authors of [10] use two unit-disk graphs in the algorithm that solves the described decision problem. The first is $G_\lambda(0)$ that corresponds to the position of the points at time $t = 0$ and $G_\lambda(1)$ that corresponds to their position at $t = 1$. Based on these two unit-disk graphs, the algorithm tries to investigate whether there is a *common spanning tree* of $G_\lambda(0)$ and $G_\lambda(1)$, i.e. if the intersection of the two graphs is a tree that connects all vertices. In order to determine whether such a property holds for the intersection graph of $G_\lambda(0)$ and $G_\lambda(1)$, the algorithm performs two *simultaneous Breadth First Search* (BFS) executions in combination with a batched range searching technique presented in [7]. That way, quadratic time complexity is avoided and an $O(n^{4/3} log^2 n)$ one is achieved, instead [10, 7].

Getting a deeper insight into the way the algorithm decides whether $b(T^*) \leq \lambda$, the authors of [10] suggest an alteration of the BFS algorithm. At each step (iteration) i the algorithm constructs a matrix and fills in the shortest paths that start from (row) and ends to (column) with path-length equal to i. The objective is, given a vertex s to start from, for each iteration $i = 0, 1, 2 \dots$, to compute the set P_i of points whose shortest paths from s have length $l = i$.

As a matter of fact, the two unit-disks are never explicitly computed and that is an important key-point for the algorithm behavior. Apart from the batched range searching technique [7], the algorithm maintains a *Unit-disk, delete only, range emptiness query* data structure, *UDRE* query in short. The latter relies on the following theorem that is introduced in [10]:

Theorem 1 *Given a value λ and a set $\mathcal{Q}$ of n points in the plane, we can build a data structure of $O(n)$ space in $O(nlogn)$ time such that the following first operation can be performed in $O(logn)$ worst case time while the second operation can be performed in $O(logn)$ amortized time.*

1. Unit-disk range emptiness (UDRE) query: *Given a point p, determine whether there exists a point $q \in \mathcal{Q}$, such that $|pq| \leq \lambda$, and if yes, return such a point q.*

2. Deletion: *delete a point from $\mathcal{Q}$.*

In an overview, the decision problem algorithm consists of a BFS execution and UDRE queries on a special data structure that supports range emptiness queries in $O(logn)$ time and point deletion in $O(logn)$, as well.

The data structure needs $O(n)$ space and $O(nlogn)$ time to be initialized. As a result, the decision problem can be solved in $O(n^{4/3}log^2n)$, based on our previous analysis. To that end, we know that we can decide, under that complexity, whether the bottleneck $b(T^*)$ of the Euclidean Minimum Bottleneck Moving Spanning tree is at most λ, for a given parameter λ.

The optimization problem: Having found a way to resolve the decision problem, utilizing this particular solution as a "black-box", in order to solve the initial EMBMST problem comes naturally as the next step of our reasoning. Up to that point, we can decide whether $b(T^*) \leq \lambda$, only when the parameter λ is "magically" provided to us. In the lines that follow, we thoroughly present how we utilize the decision problem solution and how we optimize searching the frontier for the appropriate value of λ.

The basic idea for the optimization problem algorithm is that an interval $(a_0, b_0]$, which is initialized to $(0, \infty]$ is maintained during the whole process. As a matter of fact, this interval represents the search frontier for the value of λ. Based on this search frontier-interval, the algorithm repeatedly dichotomizes the interval into two sub-intervals, by taking the median value, applying the principles of binary search. Wang & Zhao suggest as an initial value for the first iteration of the algorithm the edge with the k-th smallest length of the complete graph, where $k = 1/2 \cdot \binom{n}{2} = \frac{n(n-1)}{4}$.

In particular, the algorithm performs a binary search on the search interval, starting from the median edge-length. The selected (pivot) edge is put into the decision problem as the value of λ. The latter value of λ is used for the partition of the search interval into two sub-sets, let $\mathcal{A}$ & $\mathcal{B}$: $\mathcal{A}$ contains the edge lengths that are smaller than the selected pivot, and $\mathcal{B}$ contains the edge lengths that are equal to or greater than this. In case the answer to the decision problem is positive, i.e. $b(T^*) \leq \lambda$, then the algorithm continues the same process on the sub-set $\mathcal{A}$, else on the sub-set $\mathcal{B}$.

The algorithm keeps this binary search-like behavior until an interval $(a1, b1]$ is found such that decision problems responds positively, whilst for the subset $(a1, b1)$ the response to the decision problem is negative. It is obvious to understand that b_1 is the desired bottleneck. In addition to the bottleneck, the optimization problem computes the whole Minimum Bottleneck Moving Spanning tree with the found bottleneck, utilizing the UDRE data structure queries on the intersection graph G_λ. The final time complexity of the EMBMST computation is $O(n^{4/3}log^2n)$ for the decision problem $\times$ $O(logn)$ steps in which the decision problem algorithm is called, due to the binary search-like search technique; thus $O(n^{4/3}log^3n)$ time in total. That is the end of the algorithm analysis for computing the EMBMST of moving points in the plane, under the Euclidean distance metric.

2.3 Paper 3: "The Minimum Moving Spanning Tree Problem" [1], pp. 1 – 6

In this subsection, we remove the last limitation; the Euclidean plane. This time, we let the vertex set of the examined graphs be *moving* points in $\mathbb{R}^{\text{dim}}$, for some constant integer $dim \in \mathbb{N}^*$. The goal is, again, to compute a single Minimum Bottleneck Moving Spanning Tree (MBMST) for the whole time interval. We keep the same notation and assumptions, as we did back in the EMBMST subsection. Despite its title, the paper does relate to the Moving *Bottleneck* Spanning Tree problem, alongside with the classic MST one. In this subsection we focus on the paper results concerning the MBMST problem. The results on the Moving MST are investigated right in the next section.

2.3.1 Paper results

In this paper, Akitaya et al [1] investigate the problem of finding a Minimum Moving Bottleneck Spanning tree, for a set of linearly moving points, under any space dimension and under any *convex distance metric*. The environment they worked in is a lot more general than all the previous papers we examined in this literature survey. The authors suggest a general $O(n^2)$ algorithm for computing the MBMST, for a given set of moving points in $\mathbb{R}^{\text{dim}}$, for some constant dim.

One of the basic ideas the paper results are based on, is the notice that the distance (under any convex metric) between two linearly moving points p, q is maximized either at the start or at the end of their motion.

In order to prove such an ascertainment, the edges are represented as linear equations $p(t) = a + tu$ and $q(t) = b + tv$, standing for the position of the points p and q respectively at time $t \in [0, 1]$. However, as we analyzed back in the previous subsection, it is not correct to assert that the MBMST is attained at the start or at the end of the motion. Akitaya et al provide an illustrative counter example, where the minimum bottleneck is attained in the middle of the time interval.

For that reason, it is not enough to examine the position of the points at just $t = 0$ and $t = 1$. Akitaya et al overcome this obstacle by introducing the notion of the *upper bound graph*. The upper bound graph is the fully connected graph that has an edge between all possible combinations of points p, q, whose edge-weight is $|pq| = max\{|p(0)q(0)|, |p(1)q(1)|\}$. In simple words, the upper bound graph connects all points with each other with the maximum distance these particular points have throughout the motion.

Having defined the upper bound graph, computing the MBMST for the given set of points is quite simple. As a matter of fact, the authors of [1] show that the MBMST is identical in all cases with the simple Minimum Bottleneck Spanning Tree of the upper bound graph. As a result, computing an MBST of the upper bound graph is enough to provide a solution to the original MBMST problem. The latter computation on the upper bound graph is done with a known algorithm presented in [3]. The algorithm computes the MBST of a graph in time linear in the size, i.e. $|V|$, of the graph. Our, custom constructed, upper bound graph has all possible edges between the n points, which results in $\frac{n(n-1)}{2}$ edges and $O(n^2)$ time complexity in total.

3 The Minimum Spanning Tree Problem

In this section, we discuss about the latest work and results on the MST problem. At first, we investigate approximation algorithms for computing a single moving MST for a set of moving points in $\mathbb{R}^{\text{dim}}$ under any convex distance metric [1]. For the latter algorithms, we focus on their approximation factor, as well as their time complexity. Right after that, we remove the single-tree limitation, emphasizing on the number of different moving (parametric) MSTs that can emerge as a parameter λ varies [6]. For this family of MSTs, we get a deeper insight into a new, improved lower bound of the different combinations of such trees.

3.1 Paper 3: "The Minimum Moving Spanning Tree Problem" [1], pp. 6 – 15

In this subsection, we examine a set of linearly moving points in $\mathbb{R}^{\text{dim}}$ for the time interval $t \in [0, 1]$, where dim is a constant integer. The weight of an edge $|pq|$ in the graph between two points p, q is equal to the distance *under any convex distance metric* of the two points at a specific time frame t. Without loss of generality, the time interval is defined to be $t \in [0, 1]$. Our goal is to compute a single Minimum Moving Spanning for the whole time interval. Such an approach is highly applicable in the concept of moving networks of mobile nodes, where stability is really important in the established connections between the devices of the network.

3.1.1 Paper results

Akitaya et al [1] start their analysis on the Minimum Moving Spanning Tree (MMST) problem by proving its belonging to the NP-Hard class. In order to show such a claim, they recall the notion of the *Partition problem*. In one formulation of the Partition problem, a set of n positive integers $\alpha_0, \dots, \alpha_i$, is given and the goal is to decide whether there is a subset $S \subseteq \{0, \dots, n-1\}$ such that

$$\sum_{i \in S} \alpha_i = \frac{1}{2} \sum_{i=0}^{n-1} \alpha_i$$

The authors of [1] use as a building block the proven NP-Hard class of the Partition problem [8] and construct a reduction from it that fits the MMST problem. For that purpose, a specific instance of the MMST problem is defined, using points A_i, B_i, C_i, D_i, E_i, for $i \in \{0, \dots, n-1\}$. The edges of the constructed graph are divided into two different sets, K_0 and K_1. K_0 contains the edges $A_i b_i$ and $A_i A_{i+1}$, while K_1 contains the edges among B_i, C_i, D_i and E_i, together with K_0. Figure 9 depicts the constructed instance of the MMST problem. The dashed-line edges are the ones belonging to K_0, whilst the solid-line edges are $K_1 \backslash K_0$.

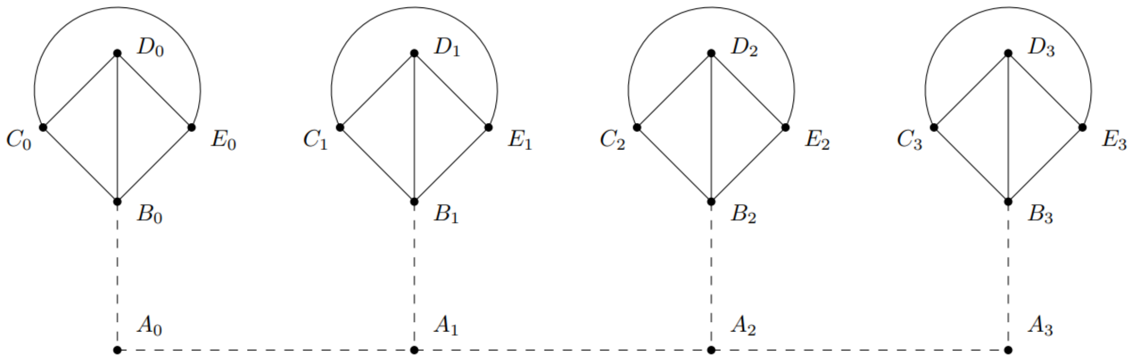


Figure 9: Reduction from the Partition problem that proves MMST computation is NP-Hard

The authors present an exhaustive case analysis proof that the constructed instance is isomorphic to the formulation of the Partition problem examined in [8]. As a result, it is proven that computing the Minimum Moving Spanning Tree of a set of moving points in $\mathbb{R}^{\text{dim}}$ under any convex distance metric is NP-Hard.

Based on that proven argument, the authors of [1] propose two algorithms for approximately solving the described problem. The first algorithm is an $O(n^2)$ 2-approximation one. The basic idea of the algorithm is quite simple and straightforward. The algorithm constructs an upper bound graph G_S on the given vertex set. The edge weight between any two points p, q is $|pq| = max\{|p(0)q(0)|, |p(1)q(1)|\}$. The MST on G_S is computed using Prim's MST algorithm with Fibonacci heaps. The total complexity of the algorithm is linear with the size of the graph; thus the time complexity of the 2-approximation algorithm is $O(n^2)$, since G_S has $\Theta(n^2)$ edges. The approximation factor 2 is proven to be tight.

Apart from this algorithm, one more is suggested, which has reduced running time of $O(nlogn)$ but achieves a $(2+\varepsilon)$-approximation factor. For the shake of analyzing this algorithm the authors of [1] recall the notion of *doubling dimension* and *convex set fatness*.

Definition: Let (V, dist) be a metric space. For any point u in V and any real number $\rho > 0$, the ball with center u and radius ρ is the set

$$\text{ball}_{\text{dist}}(u, \rho) = \{u \in V : \text{dist}(u, v) \leq \rho\}$$

Let τ be the smallest integer such that for every real number $\rho > 0$, every ball of radius ρ can be covered by at most τ balls of radius $\rho/2$. Note that all balls must be centered at points of the set V. The doubling dimension of (V, dist) is defined to be $log\tau$. It is proven in [1] that the doubling dimension of the metric space $(V, ||\cdot||_\infty$ of a set V of moving points in $\mathbb{R}^{\text{dim}}$ is *at most* dim $\cdot log5$

The algorithm works as following: for every moving point p in the given set S defines the point $P = (p(0), p(1))$ in a set $S' \in \mathbb{R}^{2\,\cdot\,\text{dim}}$. Mapping all points from S into S' lets us define the distance between two any points P, $Q \in S'$ to be

$$\text{dist}(P, Q) = max(\text{dist}_C(p(0), q(0)), \text{dist}_C(p(1), q(1)))$$

The authors prove that dist is a metric and result in showing that we can compute in $O(\log n)$ expected time a $(2+\varepsilon)$-approximation of the Minimum Moving Spanning Tree for any set S of linearly moving points in $\mathbb{R}^{\text{dim}}$, under any convex distance function in $\mathbb{R}^{\text{dim}}$.

3.2 Paper 4: "A Stronger Lower Bound on Parametric Minimum Spanning Trees" [6]

In this subsection, we no longer opt for a *single* tree that minimizes its maximum weight throughout the whole time interval of the points' motion. Instead, we focus on the different Minimum Spanning Trees that can be acquired in moving (parametric) spanning trees. In particular, we analyze a new, stronger lower bound on that number, as presented in [6].

3.2.1 Definitions

The notion of the *parametric* minimum spanning tree is rather similar with the MMST of linearly moving points under constant velocity. We, again, are given an input graph G, whose edge weights vary. However, this time, there is not a time interval, but a parameter λ. The edge labels are linear functions of this parameter λ, which is semantically close to the linear position functions of the time variable in the MMST variation of the problem.

It is easily understood that, for any value of $\lambda \in \mathbb{R}$, a spanning tree T_λ can be obtained as the minimum spanning tree of the weight functions, evaluated at that specific value of λ. A discreet sequence of trees is produced, as λ varies continuously from $-\infty$ to ∞, each of which is minimum within some range of values of λ. Before the publication of the paper, the know bounds for the number of trees in a graph with n vertices and m edges were $\Omega(m\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function [5], and is always $O(mn^{1/3})$ [4]. The paper focuses on improving the lower bound $\Omega(m\alpha(n))$ to $\Omega(mlogn)$.

3.2.2 Paper results

The authors of [6] approach the parametric minimum spanning tree problem by presenting a simple, yet important and intuitive observation. According to the latter, the number of different spanning trees for the different values of the parameter $\lambda \in \mathbb{R}$ is closely related to the number of *crossings* between these lines in the (λ, weight) plane.

Such an ascertainment can be justified by recalling that the known MST computation algorithms (Prim, Kruskal, etc.) are based on the sorted ordering of the edge weights, rather than their actual value. Combining this observation with the self-proven fact that the sorted ordering of the edges in the parametric version of the MST problem can only change if two or more lines cross, results in the argument that the MST can only change when λ passes through one of these crossing points, as its varies from $-\infty$ to ∞.

As a consequence, the number of different parametric MSTs for a given set of vertices and edge labels is *finite*. Each tree is MST for a specific range of values of λ and change at specific *breakpoints*, that are closely related to the crossings of the line functions. As a matter of fact, at each crossing point one or more edge *swaps* take place. If this edge swap results in an alteration in the computed MST, then it is considered a *breakpoint*.

The proposed algorithm by the authors of [6] is based on the notion of *2-trees*. A 2-tree can be defined as a graph that can be produced starting from the K_2 by repeatedly adding new degree-two vertices, adjacent to pairs vertices that were previously adjacent. Alternatively, two-trees are obtained by iterative replacing of the edges with triangles. During each repetitive step, we *simultaneously* replace all the edges of the initial graph G with triangles, producing the new 2-tree graph G^+.

In the next step, the newly constructed G^+ graph is the new input G, whose edges are replaced by triangles (all at the same time) in order to produce the next-iteration 2-tree. The 2-trees are symbolized by the authors with the letter T and a subscript that corresponds to the number of steps taken to reach that tree, starting from K_2. Specifically, the 2-tree T_0 is defined to be the K_2 graph, and then for every $i > 0$, T_i is defined to be the graph obtained by replacing all edges of T_{i-1} by triangles. Is is shown that a 2-tree T_i has 3^i edges and $(3^i + 3)/2$ vertices. Figure 10 illustrates the recursive construction of a family of 2-trees starting from $K_2 \equiv T_0$ (leftmost) up to T_3 (rightmost).

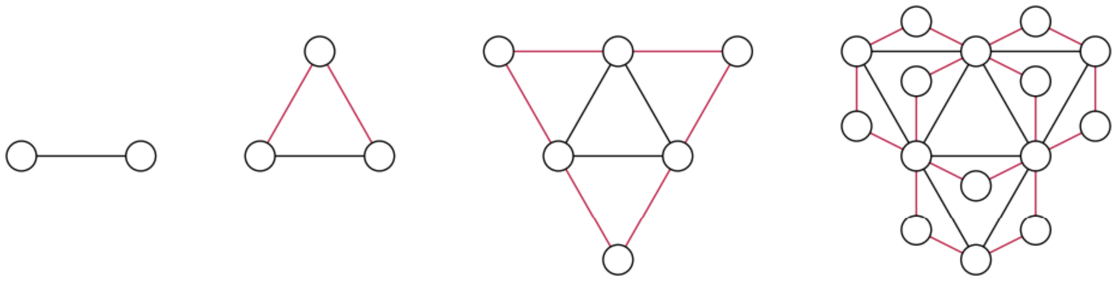


Figure 10: Recursively constructing a family of 2-trees T_i for $i = 0, 1, 2, 3$ (left to right). Two (red) edges and one vertex are added for each edge of the graph at each step, in this rightmost construction [6].

Having defined the fundamental building blocks of the algorithm, we can, now, get a deeper insight into the reason why these 2-trees are used and how they result in the new, stronger lower bound. To start with, the whole idea of constructing such two-trees meets the dynamic programming principles. The algorithm is recursive and constructs a solution based on a smaller (simpler) solution that has been previously constructed with the same process. The objective is to define the number of different parametric MSTs as well as the MSTs themselves that are produced with the given set of edge label functions of λ, as λ varies from $-\infty$ to ∞.

In math formulation, we compute the parametric MSTs on T_i for the different values of the parameter λ, by adding and removing edges of T_{i-1}. In detail, the authors of [6] thoroughly present the algorithmic steps of constructing the parametric MSTs. In specific, in case the weight functions of T_{i-1} are not computed in some previous step of the algorithm, then they are recursively computed. Next, a series of linear transformations are applied on these computed weight functions, in combination with the edge-by-triangle replacement we already analyzed.

The edges of T_i are divided into three disjoint edge sets, *blue*, *red* and *green*, based on their relation to the T_{i-1} edge set. In particular, the edges remaining from T_{i-1} into T_i are colored green, while the two newly introduced edges are arbitrarily colored as blue and red without any other requirement. The authors suggest the application of different linear transformations to each edge set (color). The utter objective of such a process is to perform a line arrangement in the (λ, weight) plane.

In simple words, the line (parameter) functions are transformed and scaled, in order to get placed in a monochromatic neighborhood. The advantage of such a result, is the fact that, when transformed in strictly defined color neighborhoods, the lines can be fully monitored concerning their crossings with specific edges, belonging to monitored edge sets. Figure 11 depicts the recursive construction and the line arrangements of the graphs T_i.

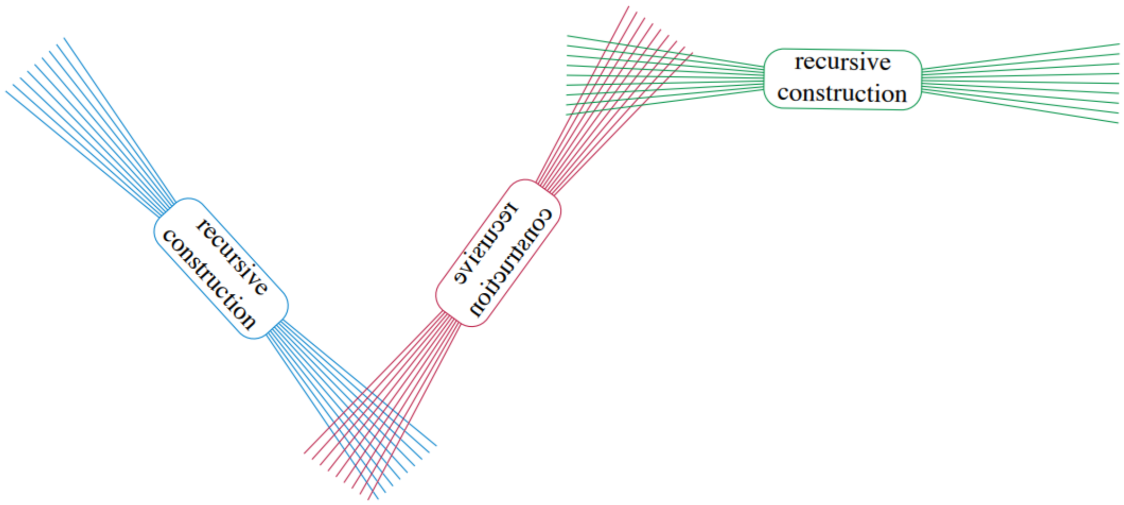


Figure 11: Line arrangements performed when computing the parametric MSTs of T_i, based on T_{i-1}. The reversed text in the central recursive construction indicates that the construction is reversed left-to-right relative to the other two copies. [6]

Based on this arrangement, the authors of [6] present and prove the argument that the number of parametric spanning trees for T_i is at least as large as

$$N(i) = \frac{i3^i}{2} + \frac{3^i + 3}{4}.$$

The proof is conducted by induction and utilizes the previously proven lemma about the number of vertices and edges that T_i has. This proof is a key-point for the whole paper, since it paves the way for claiming that the number t of different parametric MSTs for T_i is $\Omega(n \log n)$. Such a claim can be easily verified by replacing the number of vertices $n = 3^i$ to the above equation:

$$t \geq N(i) = N(\log_3 n) = \frac{n \log n}{2} + \frac{n+3}{4} \to \Omega(n \log n).$$

Applying this formula for $i = 0, 1, 2, \ldots$ gives the following numbers of different parametric spanning trees

$$1, 3, 12, 48, 183, 669, 2370, 8202, 27885, 93495 \ldots$$

Except for calculating the number of the different parametric spanning trees, the authors of [6] provide visualizations on the different spanning trees that can be computed for the different values of the parameter λ, based on the linear transformations we already presented in detail. Figure 12 illustrates the 12 different parametric spanning trees, as well as the λ intervals for which they hold. The figure also shows the crossings between the different line functions.

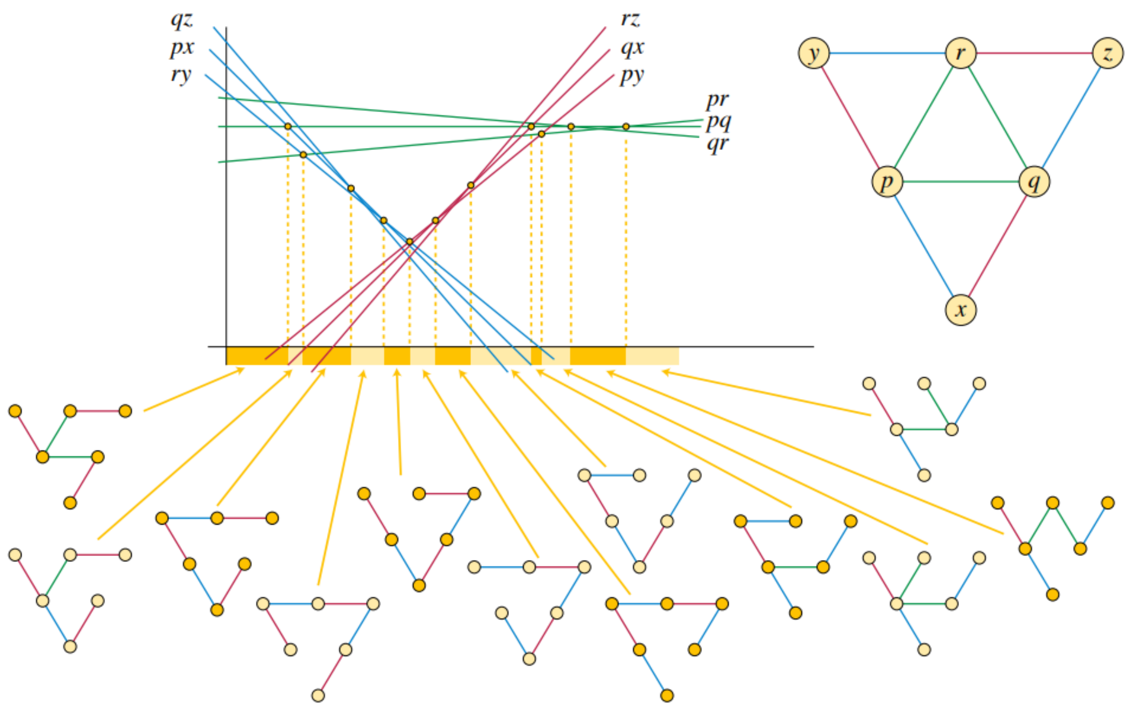


Figure 12: T_2 (upper right) as parametrically weighted, with the graphs of each weight function shown as lines in the (λ, w) plane (upper left), and the resulting sequence of 12 parametric minimum spanning trees (bottom). The marked yellow crossings of pairs of lines correspond to breakpoints in the sequence of trees. [6]

4 Summary

To that end, the current literature survey comes to its semantic completion. We started with covering the Minimum Bottleneck Spanning Tree problem and its alterations. We investigated algorithms for computing the MBST under different limitations. The Euclidean plane and the Euclidean distance metric was an important, but not the only one. We covered all the way through linearly moving points in $\mathbb{R}^{\text{dim}}$ and under any convex distance metric. The latter was the starting point for the second spanning tree problem we examined, the Minimum Spanning Tree problem. We investigated two of its variations. In the first place, we opted for computing a single, moving minimum spanning tree that spans a set of moving points in any dimensional space and connects the points without topological changes throughout the time interval of the motion. The problem is shown to be NP-hard, so we focused on two approximation algorithms, their approximation factors and their running time complexities. Secondly, we examined a similar problem, the Parametric Minimum Spanning Tree problem, from a different point of view. More specifically, that time, we studied the different minimum spanning trees that can be obtained on a graph whose edges are labeled with λ-parametered linear functions. Throughout that process we presented the current advances about a new, improved lower bound on that number.

References

[1] Hugo A. Akitaya et al. "The Minimum Moving Spanning Tree Problem". In: *Journal of Graph Algorithms and Applications* 27.1 (2021), pp. 1–18. DOI: 10.7155/jgaa.00607.

[2] Ahmad Biniaz. "Euclidean Bottleneck Bounded-Degree Spanning Tree Ratios". In: *arXiv e-prints*, arXiv:1911. 08529 (Nov. 2019), arXiv:1911. 08529. DOI: arXiv.1911.08529. arXiv: 1911.08529 [cs.CG].

[3] Paolo M. Camerini. "The Min-Max Spanning Tree Problem and Some Extensions". In: *Inf. Process. Lett.* 7 (1978), pp. 10–14.

[4] T. K. Dey. "Improved Bounds for Planar k -Sets and Related Problems". In: *Discrete & Computational Geometry* 19.3 (Mar. 1998), pp. 373–382. ISSN: 1432-0444. DOI: 10.1007/PL00009354. URL: https://doi.org/10.1007/PL00009354.

[5] D. Eppstein. "Geometric Lower Bounds for Parametric Matroid Optimization". In: *Discrete & Computational Geometry* 20.4 (Dec. 1998), pp. 463–476. ISSN: 1432-0444. DOI: 10.1007/PL00009396. URL: https://doi.org/10.1007/PL00009396.

[6] David Eppstein. "A Stronger Lower Bound on Parametric Minimum Spanning Trees". In: *Algorithms and Data Structures*. Ed. by Anna Lubiw, Mohammad Salavatipour, and Meng He. Cham: Springer International Publishing, 2021, pp. 343–356. ISBN: 978-3-030-83508-8.

[7] Matthew J. Katz and Micha Sharir. "An Expander-Based Approach to Geometric Optimization". In: *Proceedings of the Ninth Annual Symposium on Computational Geometry*. SCG '93. San Diego, California, USA: Association for Computing Machinery, 1993, pp. 198–207. ISBN: 0897915828. DOI: 10.1145/160985.161137. URL: https://doi.org/10.1145/160985.161137.

[8] Garey Michael and Johnson David. "Computers and intractability". In: 174 (1979).

[9] Clyde L. Monma and Subhash Suri. "Transitions in geometric minimum spanning trees (extended abstract)". In: *SCG '91*. 1991.

[10] Haitao Wang and Yiming Zhao. "Computing the Minimum Bottleneck Moving Spanning Tree". In: *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*. Ed. by Stefan Szeider, Robert Ganian, and Alexandra Silva. Vol. 241. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 82:1–82:15. ISBN: 978-3-95977-256-3. DOI: 10.4230/LIPIcs.MFCS.2022.82. URL: https://drops.dagstuhl.de/opus/volltexte/2022/16880.