Streaming Algorithms for Geometric Steiner Forest

Artur Czumaj
University of Warwick, Coventry, UK

Shaofeng H.-C. Jiang
Peking University, Beijing, China

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel

Pavel Veselý
Charles University, Prague, Czech Republic

Abstract

We consider an important generalization of the Steiner tree problem, the *Steiner forest problem*, in the Euclidean plane: the input is a multiset $X \subseteq \mathbb{R}^2$, partitioned into k color classes $C_1, C_2, \ldots, C_k \subseteq X$. The goal is to find a minimum-cost Euclidean graph G such that every color class C_i is connected in G. We study this Steiner forest problem in the streaming setting, where the stream consists of insertions and deletions of points to X. Each input point $x \in X$ arrives with its color $\mathsf{color}(x) \in [k]$, and as usual for dynamic geometric streams, the input is restricted to the discrete grid $\{0, \ldots, \Delta\}^2$.

We design a single-pass streaming algorithm that uses $\mathrm{poly}(k \cdot \log \Delta)$ space and time, and estimates the cost of an optimal Steiner forest solution within ratio arbitrarily close to the famous Euclidean Steiner ratio α_2 (currently $1.1547 \le \alpha_2 \le 1.214$). This approximation guarantee matches the state of the art bound for streaming Steiner tree, i.e., when $k = 1$. Our approach relies on a novel combination of streaming techniques, like sampling and linear sketching, with the classical Arora-style dynamic-programming framework for geometric optimization problems, which usually requires large memory and has so far not been applied in the streaming setting.

We complement our streaming algorithm for the Steiner forest problem with simple arguments showing that any finite approximation requires $\Omega(k)$ bits of space.

2012 ACM Subject Classification Theory of computation → Sketching and sampling; Theory of computation → Streaming models; Theory of computation → Computational geometry

Keywords and phrases Steiner forest, streaming, sublinear algorithms, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.47

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2011.04324`

Funding Work partially supported by the National key R & D program of China No. 2021YFA1000900.
Artur Czumaj: Research partially supported by the Centre for Discrete Mathematics and its Applications (DIMAP), by a Weizmann-UK Making Connections Grant, by an IBM Faculty Award, and by EPSRC award EP/V01305X/1.
Shaofeng H.-C. Jiang: Part of this work was done when the author was at the Weizmann Institute of Science and Aalto University. Partially suppported by a startup fund from Peking University, and the Advanced Institute of Information Technology, Peking University.
Robert Krauthgamer: Work partially supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, a Weizmann-UK Making Connections Grant, the Weizmann Data Science Research Center, and a Minerva Foundation grant.
Pavel Veselý: Part of this work was done when the author was at the University of Warwick. Partially supported by European Research Council grant ERC-2014-CoG 647557, by a Weizmann-UK Making Connections Grant, by GA ČR project 19-27871X, and by Charles University project UNCE/SCI/004.

1 Introduction

We study combinatorial optimization problems in dynamic geometric streams, in the classical framework introduced by Indyk [36]. In this setting, focusing on low dimension $d = 2$, the input point set is presented as a stream of insertions and deletions of points restricted to the discrete grid $[\Delta]^2 := \{0, \dots, \Delta\}^2$. Geometric data is very common in applications and has been a central object of algorithmic study, from different computational paradigms (like data streams, property testing and distributed/parallel computing) to different application domains (like sensor networks and scientific computing). Research on geometric streaming algorithms has been very fruitful, and in particular, streaming algorithms achieving $(1 + \varepsilon)$-factor estimation (i.e., approximation of the optimal value) have been obtained for fundamental geometric problems, such as k-clustering [15, 28, 35], facility location [24, 43], and minimum spanning tree (MST) [27].

Despite this significant progress, some similarly looking problems are still largely open. Specifically, for the TSP and Steiner tree problems, which are the cornerstone of combinatorial optimization, it is a major outstanding question (see, e.g., [50]) whether a streaming algorithm can match the $(1+\varepsilon)$-approximation known for the offline setting [7, 46]. In fact, the currently best streaming algorithms known for TSP and Steiner tree only achieve $O(1)$-approximation, and follow by a trivial application of the MST streaming algorithm.

While MST is closely related to TSP and Steiner tree – their optimal values are within a constant factor of each other – it seems unlikely that techniques built around MST could achieve $(1 + \varepsilon)$-approximation for either problem. Indeed, even in the offline setting, the only approach known to achieve $(1 + \varepsilon)$-approximation for TSP and/or Steiner tree relies on a framework devised independently by Arora [7] and Mitchell [46], that combines geometric decomposition (e.g., a randomly shifted quad-tree) and dynamic programming. These two techniques have been used *separately* in the streaming setting in the past: quad-tree decomposition in [3, 4, 20, 24, 27, 28, 37, 43] and dynamic programming, mainly for string processing problems, in [13, 16, 22, 25, 31, 48, 51]. However, we are not aware of any successful application of the Arora/Mitchell framework, which combines these two approaches, for any geometric optimization problem whatsoever.

We make an important step towards better understanding of these challenges by developing new techniques that *successfully adapt the Arora/Mitchell framework to streaming.* To this end, we consider a generalization of Steiner tree, the classical **Steiner Forest Problem (SFP)**. In this problem (also called *Generalized Steiner tree*, see, e.g., [7]), the input is a multiset of n *terminal* points $X \subseteq [\Delta]^2$, partitioned into k *color classes* $X = C_1 \sqcup \cdots \sqcup C_k$, presented as a dynamic stream. In addition, apart from the coordinates of the point $x \in X$, its color $\mathsf{color}(x) \in [k]$ is also revealed upon its arrival in the stream[1]. The goal is to find a minimum-cost Euclidean graph G such that every color class C_i is connected in G. Observe that the Steiner tree problem is a special case of SFP in which all terminal points should be connected (i.e., $k = 1$). Similar to the Steiner tree problem, a solution to SFP may use points other than X; those points are called *Steiner points.*

▶ Remark. In the literature, the term SFP sometimes refers to the *special case* where *each color class contains only a pair of points*, i.e., each $C_i = \{s_i, t_i\}$ [11, 12, 14, 17, 33]. It is not difficult to see (see [49]) that one can reduce one problem into another in the standard setting of *offline algorithms.* The special case of pairs is often simpler to present and does

[1] The points are arriving and leaving in an arbitrary order; there is no requirement that each color arrives in a batch, i.e., that its points are inserted/deleted consecutively in the stream.

not restrict algorithmic generality for offline algorithms, even though the setting considered here is more natural for applications (see [45]). Nevertheless, the definition used here allows for better parameterization over the number of colors k.

Background. While one might hope for a streaming algorithm for SFP with $o(k)$ space, we observe that *this task is impossible*, even in the one-dimensional case. In Theorem 4.1, we present a reduction that creates instances of SFP in $\mathbb{R}$ such that every streaming algorithm achieving any finite approximation ratio for SFP must use $\Omega(k)$ bits of space. This holds even for insertion-only algorithms, and even if all color classes are of size at most 2.

Even for $k = 1$, which is the famous Steiner tree problem, the only known streaming algorithm is to estimate the cost of a minimum spanning tree (MST) and report it as an estimate for SFP. It is useful to recall here the *Steiner ratio* α_d, defined as the supremum over all point sets $X \subseteq \mathbb{R}^d$, of the ratio between the cost of an MST and that of an optimal Steiner tree. The famous Steiner ratio Gilbert-Pollak Conjecture [29] speculates that $\alpha_2 = \frac{2}{\sqrt{3}} \approx 1.1547$, but the best upper bound to date is only that $\alpha_2 \le 1.214$ [23]. It follows that employing the streaming algorithm of Frahling, Indyk, and Sohler [27], which $(1+\varepsilon)$-approximates the MST cost using space $\mathrm{poly}(\varepsilon^{-1}\log\Delta)$, immediately yields a streaming algorithm that $(\alpha_2+\varepsilon)$-approximates the Steiner tree cost, with the same space bound.

1.1 Our Contribution

Our main result is a *space and time* efficient, *single-pass* streaming algorithm that estimates the optimal *cost* OPT for SFP within $(\alpha_2+\varepsilon)$ factor. Our space bound is nearly optimal in terms of the dependence in k, since any finite approximation for SFP requires space $\Omega(k)$ (Theorem 4.1), and our ratio matches the state of the art for Steiner tree (i.e., $k = 1$).

▶ **Theorem 1.1** (Informal Version of Theorem 3.1)**.** *For any integers $k, \Delta \ge 1$ and any fixed $\varepsilon > 0$, one can with high probability $(\alpha_2+\varepsilon)$-approximate the SFP cost of an input $X \subseteq [\Delta]^2$ presented as a dynamic stream, using space and query/update times bounded by* $\mathrm{poly}(k \cdot \log\Delta)$.

We notice that while the algorithm in Theorem 1.1 returns only an approximate cost of the optimal solution and it cannot return the entire approximate solution (since the output is of size $\Omega(n)$), an additional desirable feature of our algorithm in Theorem 1.1 is that it can return information about the colors in the trees in an approximate solution. That is, our algorithm can maintain a partition of the colors used in X into $I_1, \dots, I_r \subseteq \{1, \dots, k\}$, so that the sum of the costs of the minimum-cost Steiner trees for sets $\bigcup_{i\in I_j} C_i$ is an $(\alpha_2+\varepsilon)$-approximation of SFP. It is worth noting that in estimating the optimal cost, our algorithm does use Steiner points. This means that the MST costs for sets $\bigcup_{i\in I_j} C_i$ of the aforementioned partition may be by an $O(1)$ factor larger than the estimate of the algorithm.

Comparison to a Simple Exponential-time Approach. As we shall discuss in Section 1.2, a simple brute force enumeration combined with linear sketching techniques yields a streaming algorithm also with near-optimal space, but significantly worse running time that is *exponential* in k. Technically, while this approach demonstrates the amazing power of linear sketching, its core is exhaustive search rather than an algorithmic insight, and thus it is quite limited, offering no path for improvements or extensions. Furthermore, the $\mathrm{poly}(k)$ running time in Theorem 1.1 is exponentially better than the exhaustive search, which seems to be a limit of what linear sketching could possibly achieve. Therefore even though the primary focus of streaming algorithms is on their space complexity, the improvement of the running time is critical in terms of pursuing efficient algorithms and making our techniques broadly applicable. Indeed, similar exponential improvements of running time have been of key importance in the advances of various other fundamental streaming problems, for instance, for moment estimation the query time was improved from $\mathrm{poly}(\varepsilon^{-1})$ to $\mathrm{poly}\log(\varepsilon^{-1})$ [40], and for heavy hitters from $\mathrm{poly}(n)$ to $\mathrm{poly}\log(n)$ [44].

1.1.1 Technical Contribution: Adapting Arora's Framework to Streaming

We introduce a method for efficient streaming implementation of an offline Arora-style [7] dynamic-programming framework based on the quad-tree decomposition. This method, which is probably the first of its kind for geometric streams, is our main technical contribution.

In the offline setting, Borradaile, Klein, and Mathieu [14] and then Bateni and Hajiaghayi [11] extended the Arora's approach to obtain a polynomial-time approximation scheme (PTAS) for SFP. The key insight of these works is that one can tweak the optimal solution so that its cost remains nearly optimal, but it satisfies certain structural properties that allow for designing a suitable dynamic program. In Section 2, we review the structural theorem and the dynamic-programming approach for SFP from [11, 14] in more detail.

The main difficulty of using the Arora-style approach in low-space streaming is that in general, such approach requires access to all input points, that is, $\Omega(n)$ space to store $\Omega(n)$ leaves at the bottom of the quad-tree input decomposition that have to be considered as basic subproblems. In order to ensure a low-space implementation of the Arora-style framework in the streaming setting, we will use only $O(k \log \Delta)$ non-uniform leaf nodes of the quad-tree, each corresponding to a square. The definition of these leaf nodes is one of the novel ideas needed to make the dynamic-programming approach work in the streaming setting. Moreover, since each internal node in the quad-tree has degree 4, the total number of quad-tree squares to consider is thus $O(k \cdot \log \Delta)$.

The next challenge is that for the dynamic program to run, we need to find an $(\alpha_2 + \varepsilon)$-approximate estimation for each new leaf and each dynamic-programming subproblem associated with it. The definition of leaf squares will enable us to reduce it to estimating the MST cost for a certain subset of points inside the square. It would then be natural to just employ the MST sketch designed in [27] to estimate the MST cost, in a black box manner. However, the leaf squares are not known in advance as we can only find them after processing the stream and thus, it is impossible to build the MST sketch for each leaf square and each subproblem associated with it. To overcome this, we observe that in essence, the MST sketch consists of uniformly sampled points (with suitably rounded coordinates). We thus obtain the MST sketch for each color separately and only use the sampled points that are relevant for the subproblem to estimate the MST cost for the subproblem, in a way similar to [27].

However, due to restricting the attention to a single subproblem, the original analysis of the MST sketch in [27] has to be modified to deal with additional technical challenges. For instance, we may not sample any point relevant to a leaf square in case there are relatively few points in it. We need to account for the error arising from this case in a global way, by observing that then the MST cost inside the leaf square is a small fraction of the overall cost.

Further, to be able to accurately enumerate the subproblems for a leaf square, we need to know the set of color classes that intersect every leaf square, but unfortunately doing so exactly is impossible in the streaming setting. To this end, we employ a δ-net for a small-enough δ, so that the intersection test can be approximately done by only looking at the nearby net points. We show that this only introduces a small error for SFP, and that this δ-net can be constructed in a dynamic stream, using space by only a factor of $\mathrm{poly}\log(\Delta)$ larger than the net. Finally, we apply the dynamic program using our leaf nodes as basic subproblems to obtain the estimation.

1.2 Could Other Approaches Work?

A Simple Exponential-time Streaming Algorithm Based on Linear Sketching. An obvious challenge in solving SFP is to determine the connected components of an optimal (or approximate) solution. Each color class must be connected, hence the crucial information is which *colors* are connected together (even though they do not have to be). Suppose momentarily that the algorithm receives an advice with this information, which can be represented as a partition of the color set $[k] = P_1 \sqcup \cdots \sqcup P_l$. Then a straightforward approach for SFP is to solve the Steiner tree problem separately on each part P_j (i.e., the union of some color classes), and report their total cost. In our streaming model, we could apply the aforementioned MST-based algorithm [27], using space $\mathrm{poly}(\varepsilon^{-1} \log \Delta)$, to achieve $(\alpha_2 + \varepsilon)$-approximation, and we would need $l \leq k$ parallel executions of it (one for each P_j). An algorithm can bypass having such an advice by enumeration, i.e., by trying in parallel all the k^k partitions of $[k]$ and reporting the minimum of all their outcomes. This would still achieve $(\alpha_2 + \varepsilon)$-approximation, because each possible partition gives rise to a feasible SFP solution (in fact, this algorithm optimizes the sum-of-MST objective). However, this naive enumeration increases the space and time complexities by a factor of $O(k^k)$. We can drastically improve the space complexity by the powerful fact that the MST algorithm of [27] is based on a *linear sketch*, i.e., its memory contents is obtained by applying a (randomized) linear mapping to the input X. The huge advantage is that linear sketches of several point sets are mergeable. In our context, one can compute a linear sketch for each color class C_i, and then obtain a sketch for the union of some color classes, say some P_j, by simply adding up their linear sketches. These sketches are randomized, and hence, one has to make sure they use the same random coins (same linear mapping), and also to amplify the success probability of the sketches so as to withstand a union bound over all 2^k subsets $P_j \subseteq [k]$. This technique improves the space complexity and update time to be basically $\mathrm{poly}(k\varepsilon^{-1} \log \Delta)$, however the query time is still *exponential* in k. We state this result as follows, and its formal proof can be found in the full version.

▶ **Theorem 1.2.** *For any integers $k, \Delta \geq 1$ and any $0 < \varepsilon < 1/2$, one can with high probability $(\alpha_2 + \varepsilon)$-approximate SFP cost of an input $X \subseteq [\Delta]^2$ presented as a dynamic geometric stream, using space and update time of $O(k^2 \cdot \mathrm{poly}(\varepsilon^{-1} \cdot \log \Delta))$ and with query time $O(k^k) \cdot \mathrm{poly}(\varepsilon^{-1} \cdot \log \Delta)$.*

Tree Embedding. Indyk [36] incorporated the low-distortion tree embedding approach of Bartal [9] to obtain dynamic streaming algorithms with $O(\log \Delta)$ ratio for several geometric problems. This technique can be easily applied to SFP as well, but the approximation ratio is $O(\log \Delta)$ which is far from what we are aiming at.

Other $O(1)$-Approximate Offline Approaches. In the regime of $O(1)$-approximation, SFP has been extensively studied using various other techniques, not only dynamic programming. For example, in the offline setting there are several 2-approximation algorithms for SFP using the primal-dual approach and linear programming relaxations [1, 30, 38], and there is also a combinatorial (greedy-type) constant-factor algorithm called *gluttonous* [33]. Both of these approaches work in the general metric setting. While there are no known methods to turn the LP approach into low-space streaming algorithms, the gluttonous algorithm of [33] might seem amenable to streaming. Indeed, it works similarly to Kruskal's MST algorithm as it also builds components by considering edges in the sorted order by length, and the MST cost estimation in [27] is similar in flavor to Kruskal's algorithm. However, a crucial difference is that the gluttonous algorithm stops growing a component once all terminals inside the component are satisfied, i.e., for each color i, the component either contains all points of C_i, or no point from C_i. This creates a difficulty that the algorithm must know for each component whether or not it is "active" (i.e., not satisfied), and there are up to n components, requiring overall $\Omega(n)$ bits of space. This information is crucial because "inactive" components do not have to be connected to anything else, but they may help to connect two still "active" components in a much cheaper way than by connecting them directly. Furthermore, we have a simple one-dimensional example showing that the approximation ratio of the gluttonous algorithm cannot be better than 2 (moreover, its approximation guarantee in [33] is significantly larger than 2). In comparison, our dynamic-programming approach gives a substantially better ratio of $\alpha_2 + \varepsilon$. Nevertheless, an interesting open question is whether the gluttonous algorithm admits a low-space streaming implementation.

1.3 Related Work

SFP has been extensively studied in operations research and algorithmic communities for several decades. This problem has been also frequently considered as a part of a more general network design problem (see, e.g., [1, 30, 38, 45]), where one could require for some subsets of vertices to maintain some higher inter-connectivity.

In the classical, offline setting, it is known that the Steiner tree problem is APX-hard in general graphs and in high-dimensional Euclidean spaces, and the same thus holds for SFP as it is more general. In general graphs, a 2-approximation algorithm is known due to Agrawal et al. [1] (see also [30, 38]). These 2-approximation algorithms rely on linear-programming relaxations, and the only two combinatorial constant-factor approximations for SFP were recently devised by Gupta and Kumar [33] and by Groß et al. [32]. For low-dimensional Euclidean space, which is the main focus of our paper, Borradaile et al. [14] and then Bateni and Hajiaghayi [11] obtained a $(1+\varepsilon)$-approximation by applying dynamic programming and geometric space decomposition, significantly extending the approach of Arora [7]. Further extensions of the dynamic-programming approach have led to a PTAS for metrics of bounded doubling dimension [17], planar graphs, and graphs of bounded treewidth [12].

There has been also extensive work for geometric optimization problems in the dynamic (turnstile) streaming setting, with low space. Indyk [36] designed $O(\log \Delta)$-approximate algorithms for several basic problems, like MST and matching. Follow-up papers presented a number of streaming algorithms achieving approximation ratio of $1+\varepsilon$ or $O(1)$ to the cost of Euclidean MST [27], various clustering problems [28, 34], geometric facility location [24, 43], earth-mover distance [3, 36], and various geometric primitives (see, e.g., [5, 18, 19, 26]). Some papers have studied geometric problems with superlogarithmic but still sublinear space and in the multipass setting (see, e.g., [6]). We are not aware of prior results for the (Euclidean) Steiner tree problem nor SFP in the streaming context, although $(1+\varepsilon)$-approximation of the MST cost [27] immediately gives a $(\alpha_2 + \varepsilon)$-approximation of the Euclidean Steiner tree.

2 Preliminaries

For $x, y \in \mathbb{R}^2$, let $\operatorname{dist}(x,y) := \|x-y\|_2$. For $S, T \subseteq \mathbb{R}^2$, let $\operatorname{dist}(S,T) := \min_{x\in S, y\in T} \operatorname{dist}(x,y)$. For $S \subseteq \mathbb{R}^2$, let $\operatorname{diam}(S) := \max_{x,y\in S} \operatorname{dist}(x,y)$. A *$\rho$-packing* $S \subseteq \mathbb{R}^2$ is a point set such that $\forall x, y \in S$, $\operatorname{dist}(x,y) \geq \rho$. A *$\rho$-covering* of X is a subset $S \subseteq \mathbb{R}^2$ such that $\forall x \in X$, $\exists y \in S$, $\operatorname{dist}(x,y) \leq \rho$. We call $S \subseteq \mathbb{R}^2$ a *ρ-net* for X if it is both a ρ-packing and a ρ-covering for X.

▶ **Fact 2.1** (Packing Property, cf. [47]). *A ρ-packing $S \subseteq \mathbb{R}^d$ has size $|S| \leq \left(\frac{3\,\mathrm{diam}(S)}{\rho}\right)^d$.*

Metric Graphs. We call a weighted undirected graph $G = (X, E, w)$ a *metric graph* if for every edge $\{u, v\} \in E$, $w(u, v) = \mathrm{dist}(u, v)$, and we let $w(G)$ to be the sum of the weights of edges in G. A solution F of SFP may be interpreted as a metric graph. For a set of points S (e.g., S can be a square), let $F|_S$ be the subgraph of F formed by edges whose both endpoints belong to S. Note that we think of F as a *continuous* graph in which every point of an edge is itself a vertex, so $F|_S$ may be interpreted as a geometric intersection of F and S.

Randomly-Shifted Quad-trees [7]. Without loss of generality, let Δ be a power of 2, and let $L := 2\Delta$. A quad-tree sub-division is constructed on $[L]^2$. In the quad-tree, each node u corresponds to a square R_u and if it's not a leaf, it has four children, whose squares partition R_u. The squares in the quad-tree are of side-lengths that are powers of 2, and we say a square R is of level i if its side-length is 2^i (this is also the level of its corresponding node in the quad-tree, where leaves have level 0 and the root is at level $\log_2 L$). The whole quad-tree is shifted by a random vector in $[-\Delta, 0]^2$. Throughout, we assume a randomly-shifted quad-tree has been sampled from the very beginning. When we talk about a quad-tree square R, we interpret it as the point set that consists of both the boundary and the internal points. For $i = 0, \ldots \log_2 L$, let 2^i-*grid* $\mathcal{G}_i \subset \mathbb{R}^2$ be the set of centers of all level-i squares in the quad-tree.

2.1 Review of Dynamic Programming (DP) [11, 14]

The PTAS for geometric SFP in the offline setting [11, 14] is based on the quad-tree sub-division framework of Arora [7], with modifications tailored to SFP. For each square R in the (randomly-shifted) quad-tree,

- $O(\varepsilon^{-1} \log L)$ equally-spaced points on the boundary edges are designated as *portals*; and
- the $\gamma \times \gamma$ sub-squares of R are designated as *cells* of R, denoted $\mathsf{cell}(R)$, where $\gamma = \Theta(\varepsilon^{-1})$ is a power of 2.

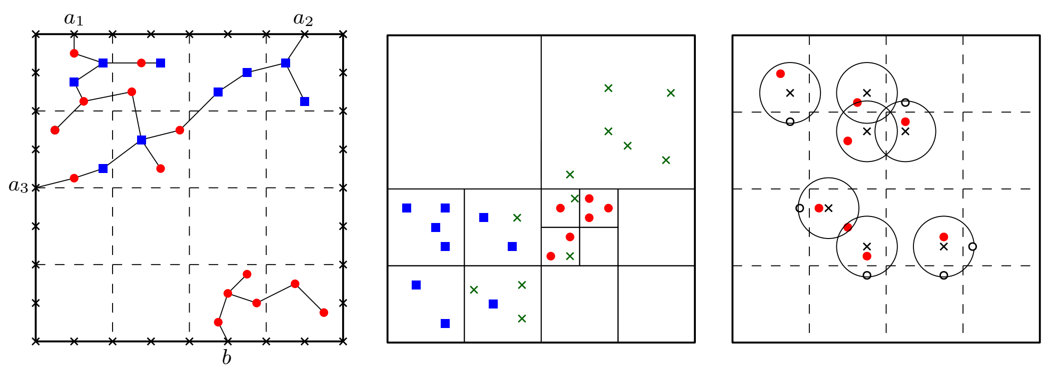
For each square R in the quad-tree, let ∂R be the boundary of R (which consists of four segments). The following is the main structural theorem from [11], and an illustration of it can be found in Figure 1a.

▶ **Theorem 2.2** ([11]). *For an optimal solution F of SFP, there is a solution F' (defined with respect to the randomly-shifted quad-tree), such that*

1. *$w(F') \leq (1+O(\varepsilon)) \cdot w(F)$ with constant probability (over the randomness of the quad-tree);*
2. *For each quad-tree square R, $F'|_{\partial R}$ has at most $O(\varepsilon^{-1})$ components, and each component of $F'|_{\partial R}$ contains a* portal *of R;*
3. *For each quad-tree square R and each cell P of R, if two points $x_1, x_2 \in X \cap P$ are connected to ∂R via F', then they are connected in $F'|_R$; this is called the* cell property.

It suffices to find the optimal solution that satisfies the structure defined in Theorem 2.2. This is implemented using dynamic programming (DP), where a subproblem of the DP is identified as a tuple (R, A, f, Π), specified as follows:

- R is a quad-tree square;
- A is a set of $O(\varepsilon^{-1})$ active portals through which the local solution enters/exits R;
- $f : \mathsf{cell}(R) \to 2^A$ s.t. for $S \in \mathsf{cell}(R)$, $f(S)$ represents the subset of A that S connects to;
- Π is a partition of A, where active portals in each part of Π have to be connected outside of R (in a larger subproblem).



(a) structural property. **(b)** simple squares. **(c)** compatibility checking.

Figure 1 Illustrations of the structural properties of Theorem 2.2 (Figure 1a), construction of simple squares by Algorithm 1 (Figure 1b) and the approximate compatibility checking idea in Section 3.2.2 (Figure 1c). Figure 1a shows a square R with portals (crosses) on ∂R, the 4×4 cells of R, and the part of solution $F'|_R$, such that $F'|_R$ passes ∂R through four portals a_1, a_2, a_3, b on the sides, and in each cell, points that are connected by F' to ∂R in the cell are connected in R. Figure 1b demonstrates the 13 simple squares constructed by Algorithm 1 for the three colors (noting that the 5 empty squares are also included as simple squares). In Figure 1c, red points are data points, cross points are the net points constructed from the data, and the black hollowed points are the added points for cells that are close-enough to a net point (for simplicity, not shown for cells containing a data point).

The use of R and A is immediate, and f is used to capture the connectivity between cells and portals (this suffices because we have the "cell property" in Theorem 2.2). Finally, Π is used to ensure feasibility, since a global connected component may be broken into several components in square R, and it is crucial to record whether or not these components still need to be connected from outside of R. An optimal solution for subproblem (R, A, f, Π) is defined as a minimum weight metric graph in R that satisfies the constraints A, f, Π. Standard combinatorial bounds show that the number of subproblems associated with each square is bounded by $(\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})}$ (see [11]).

▶ Remark 2.3. Strictly speaking, we use a simplified definition of DP subproblems, compared to [11]. Namely, one can additionally require that for any two cells $S, S' \in \mathsf{cell}(R)$, either $f(S) = f(S')$ or $f(S) \cap f(S') = \emptyset$ and that any active portal in A appears in $f(S)$ for some cell S. Then, f defines a partition of $\mathsf{cell}(R)$ and of A into local components inside R (taking into account only components connected to ∂R), and Π should encode which local components need be connected from the outside of R, implying that Π should be a partition of local components (instead of A). Thus, Π can also be thought of as a partition of the partition of A induced by f. We chose to give a more relaxed definition of DP subproblems as it is sufficient for describing how to implement the DP approach in the streaming setting.

3 Streaming Dynamic Programming: k^3-time-and-space Algorithm

In this section, we prove our main result, Theorem 1.1, restated with more precise bounds. Formally, we call the time for processing inserting/deleting one point as *update time*, and for reporting the estimate of OPT the *query time*.

▶ **Theorem 3.1.** *For any integers $k, \Delta \geq 1$ and any $0 < \varepsilon < 1/2$, one can with high probability $(\alpha_2 + \varepsilon)$-approximate the SFP cost of an input $X \subseteq [\Delta]^2$ presented as a dynamic geometric stream, using space and update time of $k^3 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \cdot \log \Delta)$ and with query time bounded by $k^3 \cdot \mathrm{poly}(\log k) \cdot (\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})}$.*

Overview. Our approach for the streaming algorithm relies on a novel modification of the known PTAS for SFP in the offline setting [11, 14], which is based on dynamic programming (DP). One important reason why the DP requires $\Omega(n)$ space is that $\Omega(n)$ leaves in the quad-tree have to be considered as basic subproblems which correspond to singletons. To make the DP use only $\widetilde{O}(\text{poly}(k))$ space, we will use only $\widetilde{O}(\text{poly}(k))$ leaf nodes. Then, since each internal node in the quad-tree has degree 4, the total number of squares to consider is $\widetilde{O}(\text{poly}(k))$. Furthermore, we design an algorithm that runs in time and space $\widetilde{O}(\text{poly}(k))$ and finds an $(\alpha_2 + \varepsilon)$-approximate estimation for each new leaf and each DP subproblem associated with it. Finally, we apply the DP using such leaves as basic subproblems to obtain the estimation. We start in Section 3.1 with a description of this approach in the offline setting, and we make it streaming in Section 3.2. The proof of Theorem 3.1 is in Section 3.3.

3.1 Offline Algorithm

New Definition of Basic Subproblems. Each of our new leaves in the DP will be a *simple square* defined below. The idea behind the definition is also simple: If no color class is contained in R, then all points inside R must be connected to ∂R, so we can make better use of the cell property in Theorem 2.2.

▶ **Definition 3.2** (Simple Squares). *We call a square R* simple *if for every $1 \le i \le k$, $C_i \cap R \neq C_i$. In other words, there is no color class totally contained in R.*

We note that the number of all possible simple squares can still be large (in particular, any empty square is simple as well as any square containing a single point of color C_i with $|C_i| \ge 2$), and we use Lemma 3.3 below to show the existence of a small subset of simple squares that covers the whole instance and can be found efficiently. Our new leaves are naturally defined using such subset of squares.

▶ **Lemma 3.3.** *There is a subset $\mathcal{R}$ of disjoint simple squares, such that the union of the squares in $\mathcal{R}$ covers X, and $|\mathcal{R}| = O(k \cdot \log \Delta)$.*

Proof. Consider the recursive procedure specified in Algorithm 1 that takes as input a square R and returns a set of disjoint simple squares $\mathcal{R}$ that covers R; see Figure 1b for an illustration of the outcome of the procedure. For our proof, we apply the procedure with R being the root square covering the whole instance. Suppose the procedure returns $\mathcal{R}$.

Algorithm 1 Algorithm for finding simple squares.

```
1: procedure SIMP-SQUARE($R$)
2:     if $R$ is simple then return $\{R\}$
3:     else
4:         let $\{R_i\}_i$ be the child squares of $R$ in the quad-tree
5:         return $\bigcup_i$ SIMP-SQUARE($R_i$)
```

We call a square R intermediate square if it is a square visited in the execution of the algorithm and it is not simple (i.e., R contains a color class). We observe that $|\mathcal{R}|$ is $O(1)$ times the number of intermediate squares. On the other hand, each color C_i can be totally contained in at most $O(\log \Delta)$ intermediate squares. Therefore, $|\mathcal{R}| = O(k \cdot \log \Delta)$. ◀

Approximation Algorithm for Subproblems on Simple Squares. Fix some simple square R. We now describe how each DP subproblem (R, A, f, Π) associated with R can be solved directly using an α_2-approximate algorithm that is amenable to the streaming setting.

Since R is a simple square, every point in R has to be connected to the outside of R, as otherwise the color connectivity constraint is violated. Hence, by the cell property of Theorem 2.2, for every cell $R' \in \mathsf{cell}(R)$, all points in R' are connected in R. Therefore, we enumerate all possible partitions of $\mathsf{cell}(R)$ that is consistent with the f constraint. For each partition, we further check whether it satisfies the constraint defined by Π. To do so, for each cell $R' \in \mathsf{cell}(R)$, we scan through all colors, and record the set of colors $\mathcal{C}_{R'} \subseteq \mathcal{C}$ that intersects R'. The $\mathcal{C}_{R'}$'s combined with the f constraint as well as the enumerated connectivity between cells suffice for checking the Π constraint.

Observe that every feasible solution of the subproblem corresponds to the above-mentioned partition of cells. Therefore, to evaluate the cost of the subproblem, we evaluate the sum of the MST costs of the parts in each partition and return the minimum one. The time complexity for evaluating each subproblem is bounded since $|A| = O(\varepsilon^{-1})$ and $|\mathsf{cell}(R)| = O(\varepsilon^{-2})$. The approximation ratio is α_2 because we use MST instead of Steiner tree for evaluating the cost. Using MST will enable us to implement this algorithm in the streaming setting.

3.2 Building Blocks for Streaming Algorithm

3.2.1 Constructing Simple Squares in the Streaming Setting

The first step is to construct a set of simple squares, as in Lemma 3.3, and an offline construction is outlined in Algorithm 1. For the streaming construction of simple squares, we observe that the key component of Algorithm 1 is a subroutine that tests whether a given square is simple or not. To implement the subroutine, we use a streaming algorithm to compute the bounding square for each color, and we test whether a given square contains any bounding square as a sub-square.

▶ **Lemma 3.4.** *Algorithm 1 can be implemented in the streaming setting, using space $O(k \operatorname{poly}\log\Delta)$ and in time $O(k \operatorname{poly}\log\Delta)$ per stream update, with success probability at least $1 - \operatorname{poly}(\Delta^{-1})$.*

Proof. The proof is ommited due to the space limit and can be found in the full version. ◀

3.2.2 Approximate Compatibility Checking

Suppose we applied Lemma 3.4 to obtain a set of simple squares $\mathcal{R}$. We proceed to evaluate the cost of subproblems associated with each simple square. Fix a simple square $R \in \mathcal{R}$. We next describe how to evaluate the cost for every subproblem associated with R, in streaming.

Suppose we are to evaluate the cost of a subproblem (R, A, f, Π). Since R is known, we have access to $\mathsf{cell}(R)$, and hence, we can enumerate the connectivity between the cells, which is a partition of $\mathsf{cell}(R)$, on-the-fly without maintaining other information about the input. Similarly, we can check the compatibility of the partition of cells with the f constraint, since the constraint only concerns the information about A and the partition. Then, when we check the compatibility of the partition of cells with Π, in the offline setting we need to compute the set of colors $\mathcal{C}_{R'} \subseteq \mathcal{C}$ that a cell R' intersects.

However, computing this set $\mathcal{C}_{R'}$ is difficult in the streaming setting, even if there is only one color C. Indeed, testing whether color C has an intersection with cell R' can be immediately reduced to the INDEX problem (see e.g. [42] for the definition), which implies an $\Omega(n)$ space bound, where n is the number of points of color C. Therefore, we need to modify the offline algorithm, and only test the intersection approximately.

To implement the approximate testing, for every color $C \in \mathcal{C}$, we impose a $\delta \cdot \operatorname{diam}(C)$-net N_C for C (see Section 2), where $\delta := O\left(\varepsilon^3 (k \log\Delta)^{-1}\right)$. A streaming algorithm in Lemma 3.5 is presented to compute this net. To be exact, the streaming algorithm in Lemma 3.5 returns a set of points N_C such that for any point $x \in N_C$ at least one point in C is within distance $\delta \cdot \operatorname{diam}(C)$ from x (so N_C does not contain net points that are far away from C). Hence, take $D_C := \operatorname{diam}(N_C)$, and we have $D_C \in (1 \pm \delta) \cdot \operatorname{diam}(C)$. Then, for each cell $R' \in \mathsf{cell}(R)$ of each simple square R, we examine each point in N_C, and if $\operatorname{dist}(R', N_C) \leq \delta \cdot D_C$, we add a new point $x \in R'$ such that $\operatorname{dist}(x, N_C) \leq \delta \cdot D_C$ to the stream, and assign it color C. Furthermore, we declare C intersects R'. This idea is visually demonstrated in Figure 1c.

▶ **Lemma 3.5.** *There is an algorithm that for every $0 < \rho \leq 1$ and every point set $S \subset \mathbb{R}^2$ provided as a dynamic geometric stream, computes a subset $N_S \subset \mathbb{R}^2$ that is a $\rho \cdot \operatorname{diam}(S)$-net for S such that for every $x \in N_S$ there exists $y \in S$ with $\operatorname{dist}(x, y) \leq \rho \cdot \operatorname{diam}(S)$, with probability at least $1 - \operatorname{poly}(\Delta^{-1})$, using space $O(\rho)^{-2} \cdot \operatorname{poly}\log\Delta$, and running in time $O(\rho)^{-2} \cdot \operatorname{poly}\log\Delta$ per stream update.*

Proof. The proof is ommited due to the space limit and can be found in the full version. ◀

In fact, such procedure of adding points is oblivious to the subproblem, and should be done only once as a *pre-processing step* before evaluating any subproblems. Therefore, the subproblems are actually evaluated on a new instance $(X', \mathcal{C}')$ after the pre-processing. Since we apply Lemma 3.5 for every color i, and by the choice of δ, the space complexity for the pre-processing step is $O\left(k^3 \cdot \operatorname{poly}(\varepsilon^{-1}\log\Delta)\right)$, and the time complexity per update is bounded by this quantity. Next, we upper bound the error introduced by the new instance.

▶ **Lemma 3.6.** *Let* OPT *be the optimal SFP solution for the original instance $(X, \mathcal{C})$, and let* OPT' *be that for $(X', \mathcal{C}')$. Then $w(\mathrm{OPT}) \leq w(\mathrm{OPT}') \leq (1 + \varepsilon) \cdot w(\mathrm{OPT})$.*

Proof. Since OPT' is a feasible solution for $(X, \mathcal{C})$, we obtain $w(\mathrm{OPT}) \leq w(\mathrm{OPT}')$ by the optimality of OPT. It remains to prove the other side of the inequality.

Recall that for every color C, we use Lemma 3.5 to obtain a $\delta \cdot \operatorname{diam}(C)$-net N_C and estimate $\operatorname{diam}(C)$ using $D_C := \operatorname{diam}(N_C)$. Now, for every cell R' of every simple square, if $\operatorname{dist}(R', N_C) \leq \delta \cdot D_C$ for some color C we add a point x to C satisfying $d(x, N_C) \leq \delta \cdot D_C$, and for any other color $C' \neq C$ with $\operatorname{dist}(R', N_{C'}) \leq \delta \cdot D_{C'}$, we add the same point x to C'. Note that we add at most one distinct point for each cell. Let $z \in N_C$ satisfy $d(x, z) \leq \delta \cdot D_C$, then adding x increases OPT by at most $2\delta \cdot D_C \leq 3\delta \cdot \operatorname{diam}(C)$, since one can connect x to $y \in C$ such that $\operatorname{dist}(y, z) \leq \delta \cdot \operatorname{diam}(C)$ (such y must exist due to Lemma 3.5).

Since there are in total at most $O(k \log\Delta \cdot \varepsilon^{-2})$ cells in all simple squares by Lemma 3.3, the total increase of the cost is at most $O(\delta \cdot k \log\Delta \cdot \varepsilon^{-2}) \cdot \max_{C \in \mathcal{C}} \operatorname{diam}(C) \leq \varepsilon \max_{C \in \mathcal{C}} \operatorname{diam}(C) \leq \varepsilon w(\mathrm{OPT})$, using the definition of δ and $w(\mathrm{OPT}) \geq \max_{C \in \mathcal{C}}(\operatorname{diam}(C))$. We conclude that $w(\mathrm{OPT}') \leq (1 + \varepsilon) \cdot w(\mathrm{OPT})$. ◀

3.2.3 Evaluating Basic Subproblems in the Streaming Setting

After we obtain the new instance $(X', \mathcal{C}')$, we evaluate the cost for every subproblem (R, A, f, Π). Because of the modification of the instance, we know for sure the subset of colors $\mathcal{C}_{R'}$ for each cell R'. To evaluate the subproblem, recall that we start with enumerating a partition of $\mathsf{cell}(R)$ that is compatible with the subproblem, which can be tested efficiently using $\mathcal{C}_{R'}$'s. Suppose now $\{P_i := R_i \cup A_i\}_{i=1}^{t}$ is a partition of $\mathsf{cell}(R) \cup A$ that we enumerated (recalling that A is the set of active portals, which needs to be connected to cells in a way that is compatible to the constraint f). Then, as in the offline algorithm, we evaluate $\mathrm{MST}(P_i)$ of each part P_i, and compute the sum of them, i.e. $\sum_{i=1}^{t} \mathrm{MST}(P_i)$, however, we need to show how to do this in the streaming setting.

Frahling et al. [27] designed an algorithm that reports a $(1+\varepsilon)$-approximation for the value of the MST of a point set presented in a dynamic stream, using space $O(\varepsilon^{-1}\log\Delta)^{O(1)}$. Furthermore, as noted in Section 1, their algorithm maintains a linear sketch. Now, a natural idea is to apply this MST sketch, that is, create an MST sketch for each color, which only takes $k \cdot O(\varepsilon^{-1}\log\Delta)^{O(1)}$ space. Then, for each $P_i = R_i \cup A_i$, we compute the set of intersecting colors, and we create a new MST sketch $\mathcal{K}$ by first adding up the MST sketches of these colors (recalling that they are linear sketches), and then adding the active portals connected to P_i to the sketch. We wish to query the sketch $\mathcal{K}$ for the cost of $\mathrm{MST}(P_i)$.

However, this idea cannot directly work, since the algorithm by [27] only gives the MST value for *all* points represented by $\mathcal{K}$, instead of the MST value for a subset P_i. Therefore, we modify the MST sketch to answer the MST cost of a subset of points of interest.

Brief Review of the MST Sketch. We give a brief overview of the algorithm of [27] before we explain how we modify it. The first observation (already from [21]) is that the MST cost can be written as a weighted sum of the number of connected components in metric threshold graphs, which are obtained from the complete metric graph of the point set by removing edges of length larger than a threshold τ. Essentially, the idea is to count the number of MST edges of length larger than τ.

To estimate the number of components in a threshold graph, we round the points to a suitable grid and sample a small number of rounded points uniformly, using ℓ_0-samplers. An ℓ_0-sampler is a data structure that processes a dynamic stream (possibly containing duplicate items), succeeds with high probability, and conditioned on it succeeding, it returns a random item from the stream such that any item in the stream is chosen with the same probability $1/n$, where n is the ℓ_0 norm of the resulting frequency vector, i.e., the number of distinct items in the stream (see Lemma 3.7 for a more precise statement). For each sampled (rounded) point y, the algorithm in [27] runs a stochastic-stopping BFS from y and in particular, it checks if it explores the whole component of y within a random number of steps. We note that this requires an extended ℓ_0-sampler that also returns the neighboring points for each sampled point, as presented in [27] and stated in Lemma 3.7. The MST cost is estimated by a weighted sum of the number of completed BFS's, summed over all levels.

▶ **Lemma 3.7** (ℓ_0-Sampler with Neighborhood Information [27, Corollary 3])**.** *There is an algorithm that for $\delta > 0$, integer $\rho, \Delta \geq 1$, every set of points $S \subseteq [\Delta]^2$ presented as a dynamic geometric stream, succeeds with probability at least $1-\delta$ and, conditioned on it succeeding, returns a point $p \in S$ such that for every $s \in S$ it holds that $\Pr[p = s] = 1/|S|$. Moreover, if the algorithm succeeds, it also returns all points from $s \in S$ such that $\mathrm{dist}(p, s) \leq \rho$. The algorithm has space and both update and query times bounded by $\mathrm{poly}(\rho \cdot \varepsilon^{-1} \cdot \log\Delta \cdot \log\delta^{-1})$, and its memory contents is a linear sketch of S.*

Generalizing the MST Algorithm to Handle Subset Queries. Fix some part P_i. Recall that the P_i's always consist of at most $O(\varepsilon^{-2})$ cells (which are quad-tree squares), plus $O(\varepsilon^{-2})$ active portal points. Hence, a natural first attempt is to make the ℓ_0-samplers to sample only on these clipping squares defined by P_i. Unfortunately, this approach would not work, since the squares are not known in advance and may be very small (i.e., degenerate to a point), so sampling a point from them essentially solves the INDEX problem.

Therefore, to estimate $\mathrm{MST}(P_i)$, we still use the original ℓ_0-samplers, and we employ a careful sampling and estimation step. We sample from the whole point set maintained by the sketch $\mathcal{K}$ but we only keep the sampled points contained in P_i. We execute the stochastic BFS from these points that are kept, restricting the BFS to the points contained in P_i.

One outstanding problem of this sampling method is that if the number of points in P_i, or to be exact, the number of non-zero entries of level-i ℓ_0-samplers, is only a tiny portion of that of the full sketch, then with high probability, we do not sample any point from P_i at all. Hence, in this case, no stochastic BFS can be performed, and we inevitably answer 0 for the number of successful BFS's. This eventually leads to an additive error. We summarize the additive error and the whole idea of the above discussions in Lemma 3.8.

▶ **Lemma 3.8.** *There is an algorithm that for every $0 < \varepsilon < 1$, integer $k, \Delta \geq 1$, and every set of points $S \subseteq [\Delta]^2$ presented as a dynamic geometric stream, maintains a linear sketch of size $k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta)$. For every query $(R, \{R_j\}_{j=1}^t, A)$ (provided after the stream ends) satisfying*

1. *R is a simple square, A is a subset of portals of R, and*
2. *$\{R_j\}_{j=1}^t \subseteq \mathsf{cell}(R)$,*

the algorithm computes from the linear sketch a real number E such that with probability at least $1 - \exp(-\log k \cdot \mathrm{poly}(\varepsilon^{-1} \log \Delta))$,

$$\mathrm{MST}(P) \leq E \leq (1+\varepsilon) \cdot \mathrm{MST}(P) + O\left(\frac{\mathrm{poly}(\varepsilon)}{k \log \Delta}\right) \cdot \mathrm{MST}(S)\,,$$

where $P = \left(\bigcup_{j=1}^t R_j\right) \cup A$. The algorithm runs in time $k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta)$ per update and the query time is also $k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta)$.

This lemma constitutes the main algorithm for the evaluation of the subproblem. Note that we only need to prove it for one point set S, since the sketch is linear. Indeed, when applying Lemma 3.8, we obtain the sketch for each color separately from the stream, and for every query, we first merge the sketches of colors relevant to the query and add query portals to the resulting sketch. By linearity, this is the same as if we obtain the sketch for all these colors and portals at once. Due to the space limit, the proof of Lemma 3.8 can be found in the full version.

3.3 Proof of Theorem 3.1

▶ **Theorem 3.1.** *For any integers $k, \Delta \geq 1$ and any $0 < \varepsilon < 1/2$, one can with high probability $(\alpha_2 + \varepsilon)$-approximate the SFP cost of an input $X \subseteq [\Delta]^2$ presented as a dynamic geometric stream, using space and update time of $k^3 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \cdot \log \Delta)$ and with query time bounded by $k^3 \cdot \mathrm{poly}(\log k) \cdot (\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})}$.*

We combine the above building blocks to prove Theorem 3.1. See Algorithm 2 for a description of the complete algorithm. The space and update time follow immediately from Algorithm 2, Theorem 2.2 and Lemmas 3.4, 3.5, and 3.8.

The query time is bounded by $O(k \cdot \log \Delta) \cdot (\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})} \cdot \varepsilon^{-O(\varepsilon^{-1})} \cdot k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta) \leq k^3 \cdot \mathrm{poly}(\log k) \cdot (\varepsilon^{-1} \log \Delta)^{O(\varepsilon^{-2})}$, where $O(k \cdot \log \Delta)$ is the number of simple squares (and thus, up to an $O(1)$ factor, the number of quad-tree squares for which we evaluate DP subproblems), $(\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})}$ is the number of subproblems associated with each square (see Section 2.1), $\varepsilon^{-O(\varepsilon^{-1})}$ is the number of MST queries evaluated for each subproblem, and each MST query takes $k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta)$ time by Lemma 3.8.

[2] We need to use the same randomness for sketches $\{\mathcal{K}_C^{(3)}\}$ among all colors C so that they can be combined later.

Algorithm 2 Main streaming algorithm.

```
1: procedure SFPINITIALIZATION($\mathcal{C}$)                         ▷ $\mathcal{C}$ is the set of colors
2:     initialize a sketch $\mathcal{K}^{(1)}$ of Lemma 3.4, a set of sketches of Lemma 3.5 $\{\mathcal{K}_C^{(2)}\}_{C\in\mathcal{C}}$ for every color $C \in \mathcal{C}$ with parameter $\delta := \mathrm{poly}(\varepsilon)(k\log\Delta)^{-1}$, and a set of (linear) sketches[2] of Lemma 3.8 $\{\mathcal{K}_C^{(3)}\}_{C\in\mathcal{C}}$ for every color $C \in \mathcal{C}$

3: procedure SFPUPDATE($x, C$, insert/delete)          ▷ insert/delete point $x$ of color $C$
4:     insert/delete point $x$ in sketches $\mathcal{K}^{(1)}, \mathcal{K}_C^{(2)}, \mathcal{K}_C^{(3)}$

5: procedure SFPQUERY                                           ▷ the stream terminates
6:     use sketch $\mathcal{K}^{(1)}$ to compute a set of simple squares $\mathcal{R}$         ▷ see Section 3.2.1
7:     for each color $C \in \mathcal{C}$, use sketch $\mathcal{K}_C^{(2)}$ to compute a set of net points $N_C$, and let $D_C := \mathrm{diam}(N_C)$          ▷ $D_C$ is a $(1 \pm \varepsilon)$-approximation for $\mathrm{diam}(C)$
8:     initialize a Boolean list $\mathcal{I}$ records whether a cell of a simple square and a color intersects          ▷ This uses space at most $O(k \cdot \log\Delta \cdot \mathrm{poly}(\varepsilon^{-1}))$
9:     for every $R \in \mathcal{R}$, $R' \in \mathsf{cell}(R)$ do
10:        if $\mathrm{dist}(N_C, R') \le \rho \cdot D_C$ for some color $C$ then
11:            let $x \in R'$ be a point such that $\mathrm{dist}(x, N_C) \le \rho \cdot D_C$
12:            for every color $C'$ with $\mathrm{dist}(N_{C'}, R') \le \rho \cdot D_{C'}$ do
13:                add $x$ to $\mathcal{K}_{C'}^{(3)}$ and record in $\mathcal{I}$ that $R'$ intersects color $C'$ ▷ see Section 3.2.2
14:    for each simple square $R$ and an associated subproblem $(R, A, f, \Pi)$ do
15:        for each partition of $\mathsf{cell}(R)$ do
16:            if the partition is compatible with the subproblem then   ▷ see Section 3.2.2
17:                for each part $R_j$ in the partition do
18:                    let $A_j \subseteq A$ be the set of active portals that $R_j$ connects to
19:                    create linear sketch $\mathcal{K}'$, by adding up $\mathcal{K}_C^{(3)}$ for every $C$ intersecting a cell in $R_j$          ▷ the intersection information is recorded in $\mathcal{I}$
20:                    add points in $A_j$ to sketch $\mathcal{K}'$
21:                    query sketch $\mathcal{K}'$ for the value of the MST of the part $R_j$ and portals $A_j$ (as in Lemma 3.8)                     ▷ see Section 3.2.3
22:                store the sum of the queried values of $\mathrm{MST}(R_j, A_j)$ as the estimated cost for the subproblem
23:    invoke the DP (as in [11]) using the values of basic subproblem estimated as above
24:    return the DP value (for the root square with no active portals)
```

To bound the failure probability, we use a union bound over the failure probabilities of all applications and queries of the streaming algorithms as well as the error bound in Theorem 2.2. We observe that Theorem 2.2 incurs an $O(1)$ failure probability, and every other steps, except for the use of Lemma 3.8, have a failure probability of $\mathrm{poly}(\Delta^{-1})$. Since we have $k \cdot (\varepsilon^{-1} \cdot \log\Delta)^{O(\varepsilon^{-2})}$ basic subproblems (see Section 2), and for each basic subproblem we need to evaluate at most $\varepsilon^{-O(\varepsilon^{-1})}$ MST queries, the total failure probability of evaluating the subproblems is at most

$$k \cdot (\varepsilon^{-1} \cdot \log\Delta)^{O(\varepsilon^{-2})} \cdot \varepsilon^{-O(\varepsilon^{-1})} \cdot \exp(-\log k \cdot \mathrm{poly}(\varepsilon^{-1}\log\Delta)) \le \mathrm{poly}(\Delta^{-1}),$$

by the guarantee of Lemma 3.8. Therefore, we conclude that the failure probability is then at most $\frac{2}{3}$. It remains to analyze the error.

Error Analysis. For the remaining part of the analysis, we condition on no failure of the sketches used in Algorithm 2 and on that the error bound in Theorem 2.2 holds. By Lemma 3.6, for the part of evaluating the basic subproblems (lines 14-22 of Algorithm 2), the actual instance that the linear sketches work on is $(1+O(\varepsilon))$-approximate. Hence, it suffices to show the DP value is accurate to that instance.

Our estimation is never an underestimate, by Lemma 3.8 and since all partitions that we enumerated are compatible with the subproblems; see Section 3.2.2. Hence, it remains to upper bound the estimation. Consider an optimal DP solution F, which we interpret as a metric graph (see Section 2). Then we create a new solution F' from F by modifying F using the following procedure. For each simple square R, we consider $F|_R$ which is the portion of F that is totally inside of R (see Section 2). For each component $S \subset R$ in $F|_R$, let S' be the point set formed by removing all Steiner points from S, except for portals of R (note that we remove portals of subsquares of R if they appear in S). Then, for each component S, we replace the subtree in F that spans S with the MST on S'. It is immediate that after the replacement, the new solution has the same connectivity of portals and terminal points as before. We define F' as the solution after doing this replacement for all simple squares.

F' is still a feasible solution. Furthermore, for every simple square R, if F is compatible with a subproblem (R, A, f, Π), then so does F'. By the construction of F', the definition of Steiner ratio α_2, and Theorem 2.2, we know that

$$w(F') \le \alpha_2 \cdot w(F) \le (1+O(\varepsilon)) \cdot \alpha_2 \cdot \mathrm{OPT}, \tag{1}$$

where the last inequality holds as we condition on that the error bound in Theorem 2.2 holds.

Now we relate the algorithm's cost to $w(F')$. Fix a simple square R, and suppose (R, A, f, Π) is the subproblem that is compatible with $F'|_R$. Then, the components in $F'|_R$ can be described by a partition of the cells plus their connectivity to active portals. Such a subproblem, together with the partition, must be examined by Algorithm 2 (in lines 14-22), and the MST value for each part is estimated in Algorithm 2. Since the algorithm runs a DP using the estimated values, the final DP value is no worse than the DP value that is only evaluated from the subproblems that are compatible to F'. Recall that our estimation for each subproblem not only has a multiplicative error of $(1+\varepsilon)$ but also an additive error by Lemma 3.8. Therefore, by the fact that F' always uses MST to connect points in components of basic subproblems, it suffices to bound the *total* additive error for the estimation of the MST cost of the components of F'.

Fix a connected (global) component Q of F', and let $\mathcal{C}_Q \subseteq \mathcal{C}$ be the subset of colors that belongs to Q. By Lemma 3.8, for every basic subproblem (R, f, A, Π) that is compatible with F', and every component P of $Q|_R$, the additive error is at most $O\left(\frac{\mathrm{poly}(\varepsilon)}{k\log\Delta}\right) \cdot \mathrm{MST}(S)$, where S is the union of color classes that intersect P plus the active portals A. Observe that $\mathcal{C}_S \subseteq \mathcal{C}_Q$ (where $\mathcal{C}_S$ is the set of colors used in S), so S is a subset of the point set of Q (note that Q contains all portals in A as F' is a portal-respecting solution and the subproblem is compatible with F') and thus $\mathrm{MST}(S) \le \mathrm{MST}(Q)$, which implies

$$O\left(\frac{\mathrm{poly}(\varepsilon)}{k\log\Delta}\right) \cdot \mathrm{MST}(S) \le O\left(\frac{\mathrm{poly}(\varepsilon)}{k\log\Delta}\right) \cdot \mathrm{MST}(Q) \le O\left(\frac{\mathrm{poly}(\varepsilon)}{k\log\Delta}\right) \cdot w(Q).$$

Observe that for each simple square R, $Q|_R$ has at most $O(\varepsilon^{-2})$ local components, hence, summing over all local components of $Q|_R$ and all simple squares R, the total additive error is bounded by $\mathrm{poly}(\varepsilon^{-1}) \cdot O(k\log\Delta) \cdot O\left(\mathrm{poly}(\varepsilon)/(k\log\Delta)\right) \cdot w(Q) \le \varepsilon \cdot w(Q)$, where use that there are at most $O(k\log\Delta)$ simple squares by Lemma 3.3. Finally, summing over all components Q of F', we conclude that the total additive error is $\varepsilon \cdot w(F')$. Combining with Equation (1), we conclude the error guarantee. This finishes the proof of Theorem 3.1.

4 Lower Bound: $\Omega(k)$ Bits are Necessary

In this section we demonstrate that any streaming algorithm for SFP achieving any finite approximation ratio for SFP requires $\Omega(k)$ bits of space.

▶ **Theorem 4.1.** *For every $k > 0$, every randomized streaming algorithm achieving a finite approximation ratio for SFP with k color classes of size at most 2 must require $\Omega(k)$ bits of space. This holds even for insertion-only algorithms and even when points are from the one-dimensional line $\mathbb{R}$.*

Proof. The proof is a reduction from the INDEX problem on k bits, where Alice holds a binary string $x \in \{0,1\}^k$, and Bob has an index $i \in [k]$. The goal of Bob is to compute the bit x_i in the one-way communication model, where only Alice can send a message to Bob and not vice versa. It is well-known that Alice needs to send $\Omega(k)$ bits for Bob to succeed with constant probability [41] (see also [42, 39]). Our reduction is from INDEX to SFP on the (discretized) one-dimensional line $[2k]$. Consider a randomized streaming algorithm ALG for SFP that approximates the optimal cost and in particular can distinguish whether the optimal cost is 0 or 1 with constant probability. We show that it can be used to solve the INDEX problem, implying that ALG needs to use $\Omega(k)$ bits of space.

Indeed, Alice applies ALG on the following stream: For each bit x_j, she adds to the stream a point of color j at location $2j + x_j$. So far $\text{OPT} = 0$. She now sends the internal state of ALG to Bob. Then, Bob continues the execution of ALG (using the same random coins) by adding one more point to the stream: Given his index $i \in [k]$, he adds a point of color i at location $2i$. After that, $\text{OPT} = 0 + x_i$, which is either 0 or 1. It follows that if ALG achieves a finite approximation with constant probability, then Bob can discover x_i and solve INDEX. ◀

5 Conclusions and Future Directions

Our paper makes a progress in the understanding of geometric streaming algorithms and of applicability of Arora's framework for low-space streaming algorithms for geometric optimization problems. Still, our work leaves a number of very interesting open problems.

Our approximation ratio $\alpha_2 + \varepsilon$ matches the current approximation ratio for the Steiner tree problem in geometric streams. Hence, any improvement to our approximation ratio would require to first improve the approximation for Steiner tree, even in insertion-only streams. This naturally leads to the main open problem of obtaining a $(1+\varepsilon)$-approximation for Steiner tree in geometric streams using only $\text{poly}(\varepsilon^{-1} \log \Delta)$ space.

Our naïve algorithm for the Steiner forest problem given in Theorem 1.2 is also an $(\alpha_2 + \varepsilon)$-approximation with $\text{poly}(k\varepsilon^{-1} \log \Delta)$ space, but its running time is exponential in k because it queries an (approximate) MST-value oracle on all possible subsets of color classes to find the minimum. We do not know if a smaller number of queries suffices here, but it is known that in a similar setup for coverage problems any oracle-based $O(1)$-approximation requires exponentially many queries to an approximate oracle [10]. Thus, it would not be surprising if a similar lower bound holds for our problem.

Our Theorem 4.1 shows that for SFP with color classes of size at most 2 one cannot achieve any bounded approximation ratio using space that is sublinear in $n \le 2k$. This strongly suggests that SFP with pairs of terminals (i.e., $C_i = \{s_i, t_i\}$) does not admit a constant-factor approximation in the streaming setting, although our lower bound construction does not extend to this case (as it requires having some size-1 color classes). We leave it as an open problem whether a constant-factor approximation in sublinear (in $n = 2k$) space is possible for this version. We notice however that for the case where both points of each terminal pair are inserted/deleted together, it is possible to get an $O(\log n)$-approximation using the metric embedding technique of Indyk [36].

The main focus of this paper is on the study of SFP for the Euclidean plane, but in principle, our entire analysis can be extended to the Euclidean space $\mathbb{R}^d$, for any fixed $d \geq 2$. However, this would require extending the arguments of [11, 14], namely, the structural result restated in Theorem 2.2, and these details were not written explicitly in these papers.

The techniques developed in this paper seem to be general enough to be applicable to other problems/objectives with *connectivity constraints*, where the connectivity is specified by the colors and a solution is feasible if the points of the same color are connected. One such closely related problem is the sum-of-MST objective, i.e., the problem of minimizing the sum of the costs of trees such that points of the same color are in the same tree (see also [2, 53] for related problems). We hope that the approach developed in our paper can lead to a $(1+\varepsilon)$-approximation of the geometric version of this problem, using $\mathrm{poly}(k\varepsilon^{-1}\log\Delta)$ space and time (for space only, one can use similar techniques as in Theorem 1.2). Moreover, it may be possible to apply our approach to solve the connectivity-constrained variants of other classical problems, especially those where dynamic programming has been employed successfully, like r-MST and TSP [7]. For example, the TSP variant could be to find a collection of cycles of minimum total length with points of the same color in the same cycle.

At a higher level, the connectivity constraints may be more generally interpreted as grouping constraints. For instance, in the context of clustering, our color constraints may be viewed as *must-link* constraints, where points of the same color have to be placed in the same cluster. Such constrained clustering framework is of significant interest in data analysis (see, e.g., [52]). Our framework, combined with coreset techniques [28] and Arora's quad-tree methods (see [8]), may be used to design streaming algorithms for such clustering problems.

Finally, we believe that the framework of optimization problems with connectivity and grouping constraints is interesting on its own, going beyond the streaming setup. Such problems may be studied also in the setting of standard (offline) algorithms, as well as of online algorithms, approximation algorithms, fixed-parameter tractability, and heuristics.

References

1 Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

2 Mattias Andersson, Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Balanced partition of minimum spanning trees. *International Journal of Computational Geometry and Applications*, 13(4):303–316, 2003.

3 Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David P. Woodruff. Efficient sketches for earth-mover distance, with applications. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 324–330, 2009.

4 Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 343–352, 2008.

5 Alexandr Andoni and Huy L. Nguyen. Width of points in the streaming model. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 447–452, 2012.

6 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 574–583, 2014.

7 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

8 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 106–113, 1998.

9 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

10 MohammadHossein Bateni, Hossein Esfandiari, and Vahab S. Mirrokni. Almost optimal streaming algorithms for coverage problems. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 13–23, 2017.

11 MohammadHossein Bateni and MohammadTaghi Hajiaghayi. Euclidean prize-collecting Steiner forest. *Algorithmica*, 62(3-4):906–929, 2012.

12 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM*, 58(5):21:1–21:37, 2011.

13 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 51–60, 2016.

14 Glencora Borradaile, Philip N. Klein, and Claire Mathieu. A polynomial-time approximation scheme for Euclidean Steiner forest. *ACM Transactions of Algorithms*, 11(3):19:1–19:20, 2015.

15 Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 576–585, 2017.

16 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 712–725, 2016.

17 T.-H. Hubert Chan, Shuguang Hu, and Shaofeng H.-C. Jiang. A PTAS for the Steiner forest problem in doubling metrics. *SIAM Journal on Computing*, 47(4):1705–1734, 2018.

18 Timothy M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computation Geometry*, 35(1-2):20–35, 2006.

19 Timothy M. Chan. Dynamic streaming algorithms for ε-kernels. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG)*, pages 27:1–27:11, 2016.

20 Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 380–388, 2002.

21 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005.

22 Kuan Cheng, Alireza Farhadi, MohammadTaghi Hajiaghayi, Zhengzhong Jin, Xin Li, Aviad Rubinstein, Saeed Seddighin, and Yu Zheng. Streaming and small space approximation algorithms for edit distance and longest common subsequence. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 54:1–54:20, 2021.

23 F. R. K. Chung and R. L. Graham. A new bound for Euclidean Steiner minimal trees. *Annals of the New York Academy of Sciences*, 440(1):328–346, 1985.

24 Artur Czumaj, Christiane Lammersen, Morteza Monemizadeh, and Christian Sohler. $(1+\varepsilon)$-approximation for facility location in data streams. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1710–1728, 2013.

25 Funda Ergün and Hossein Jowhari. On the monotonicity of a data stream. *Combinatorica*, 35(6):641–653, 2015.

26 Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2005.

27 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry and Applications*, 18(1/2):3–28, 2008.

28 Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–217, 2005.

29 Edgar N. Gilbert and Henry O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

30 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

31 Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 318–327, 2007.

32 Martin Groß, Anupam Gupta, Amit Kumar, Jannik Matuschke, Daniel R. Schmidt, Melanie Schmidt, and José Verschae. A local-search algorithm for Steiner forest. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, pages 31:1–31:17, 2018.

33 Anupam Gupta and Amit Kumar. Greedy algorithms for Steiner forest. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 871–878, 2015.

34 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–300, 2004.

35 Wei Hu, Zhao Song, Lin F. Yang, and Peilin Zhong. Nearly optimal dynamic k-means clustering for high-dimensional data, 2019. `arXiv:1802.00459`.

36 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 373–380, 2004.

37 Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *Proceedings of the 3rd International Workshop on Statistical and Computational Theories of Vision (SCTV)*, 2003. URL: `https://people.csail.mit.edu/indyk/emd.pdf`.

38 Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

39 T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of Hamming distance. *Theory of Computing*, 4(6):129–135, 2008.

40 Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 745–754, 2011.

41 Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

42 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

43 Christiane Lammersen and Christian Sohler. Facility location in dynamic geometric data streams. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA)*, pages 660–671, 2008.

44 Kasper Green Larsen, Jelani Nelson, Huy L. Nguyen, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. *Communications of the ACM*, 62(8):95–100, 2019.

45 Thomas L. Magnanti and Laurence A. Wolsey. Chapter 9: Optimal trees. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503–615. Elsevier, 1995.

46 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

47 David Pollard. *Empirical Processes: Theory and Applications*, chapter 4: Packing and Covering in Euclidean Spaces, pages 14–20. IMS, 1990.

48 Michael E. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1698–1709, 2013.

49 Guido Schäfer. Steiner Forest. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 2099–2102. Springer, New York, NY, 2016.

50 Christian Sohler. Problem 52: TSP in the streaming model. `https://sublinear.info/52`, 2012.

51 Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 336–345, 2007.

52 Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 577–584, 2001.

53 Liang Zhao, Hiroshi Nagamochi, and Toshihide Ibaraki. Greedy splitting algorithms for approximating multiway partition problems. *Mathematical Programming, Series A*, 102(1):167–183, 2005.