Streaming Facility Location in High Dimension via Geometric Hashing

(Full version available at arXiv:2204.02095)

Artur Czumaj[*]
University of Warwick
A.Czumaj@warwick.ac.uk

Shaofeng H.-C. Jiang[†]
Peking University
shaofeng.jiang@pku.edu.cn

Robert Krauthgamer[‡]
Weizmann Institute of Science
robert.krauthgamer@weizmann.ac.il

Pavel Veselý[§]
Charles University
vesely@iuuk.mff.cuni.cz

Mingwei Yang
Peking University
yangmingwei@pku.edu.cn

***Abstract*—In *Euclidean Uniform Facility Location*, the input is a set of clients in $\mathbb{R}^d$ and the goal is to place facilities to serve them, so as to minimize the total cost of opening facilities plus connecting the clients. We study the classical setting of dynamic geometric streams, where the clients are presented as a sequence of insertions and deletions of points in the grid $\{1,\dots,\Delta\}^d$, and we focus on the *high-dimensional regime*, where the algorithm's space complexity must be polynomial (and certainly not exponential) in $d\cdot\log\Delta$.**

We present a new algorithmic framework, based on importance sampling from the stream, for $O(1)$-approximation of the optimal cost using only $\operatorname{poly}(d\cdot\log\Delta)$ space. This framework is easy to implement in two passes, one for sampling points and the other for estimating their contribution. Over random-order streams, we can extend this to a one-pass algorithm by using the two halves of the stream separately. Our main result, for arbitrary-order streams, computes $O(d^{1.5})$-approximation in one pass by using the new framework but combining the two passes differently. This improves upon previous algorithms that either need space exponential in d or only guarantee $O(d\cdot\log^2\Delta)$-approximation, and therefore our algorithms for high-dimensional streams are the first to avoid the $O(\log\Delta)$-factor in approximation that is inherent to the widely-used quadtree decomposition. Our improvement is achieved by employing a geometric hashing scheme that maps points in $\mathbb{R}^d$ into buckets of bounded diameter, with the key property that every point set of small-enough diameter is hashed into at most $\operatorname{poly}(d)$ distinct buckets.

Finally, we complement our results with a proof that every streaming 1.085-approximation algorithm requires space exponential in $\operatorname{poly}(d\cdot\log\Delta)$, even for insertion-only streams.

***Index Terms*—sublinear algorithms, facility location, streaming algorithms, hash functions, high dimension**

Shaofeng H.-C. Jiang is partially supported by a national key R&D program of China No. 2021YFA1000900.

[*] 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.

[†] Research partially supported by a startup fund from Peking University, and the Advanced Institute of Information Technology, Peking University.

[‡] Work partially supported by ONR Award N00014-18-1-2364, by the Israel Science Foundation grant #1086/18, by a Weizmann-UK Making Connections Grant, by a Minerva Foundation grant, and the Weizmann Data Science Research Center.

[§] Partially supported by GA ČR project 22-22997S and by Center for Foundations of Modern Computer Science (Charles University project UNCE/SCI/004).

I. Introduction

Facility Location is a classical problem in combinatorial optimization and operations research, and models a scenario where one wishes to find a placement of facilities that optimizes the total service cost for a given set of customers. The cost has two different parts: opening a facility at a location incurs a so-called *opening cost*, and serving each customer incurs a so-called *connection cost*. The goal is to minimize the sum of both costs. Typical examples of applications include placement of servers in a network and location planning.

Euclidean Uniform Facility Location: In the *Uniform Facility Location (UFL)* problem, all possible facilities have the same opening cost $\mathfrak{f} > 0$. This is essentially a clustering problem; it is similar to k-median, except that the number of clusters is not prescribed in advance, but rather optimized by adding to the objective a regularization term (proportional to the number of clusters).

We consider the Euclidean version of UFL, where the data points and facilities all lie in $\mathbb{R}^d$. Formally, given as input a dataset $P \subseteq \mathbb{R}^d$ and $\mathfrak{f} > 0$, the goal is to *open* a set $F \subseteq \mathbb{R}^d$ of *facilities*, so as to minimize the total *connection cost* (the cost of connecting each data point to its nearest facility according to the Euclidean distance) plus the total *opening cost* (opening each facility costs $\mathfrak{f}$). That is, the goal is to find $F \subseteq \mathbb{R}^d$ that minimizes

$$\operatorname{cost}(P,F) := \sum_{p\in P} \operatorname{dist}(p,F) + \mathfrak{f}\cdot|F| \ ,$$

where we denote $\operatorname{dist}(p,F) := \min_{q\in F}\operatorname{dist}(p,q)$ and $\operatorname{dist}(p,q) := \|p-q\|_2$ is the Euclidean distance. This problem has been studied extensively for decades in many algorithmic settings, including offline, online, dynamic, sublinear-time, streaming, and so forth; see Section I-C for an overview and references.

Streaming Setting: Euclidean UFL has also played a special role in the study of algorithms for dynamic geometric streams. Indeed, the seminal paper by Indyk [1], which introduced this model, considered UFL as one of its benchmark problems, together with the minimum spanning tree, minimum-weight (bichromatic) matching, and k-median problems.

In the setting of *dynamic geometric streams*, the input P is presented as a stream of insertions and deletions of points from $[\Delta]^d := \{1, 2, \ldots, \Delta\}^d$. The algorithms should be *space-efficient*, ideally using space that is polylogarithmic in Δ^d, which is a natural benchmark because it is polynomial in the representation of a single point using $O(d \cdot \log \Delta)$ bits. Due to this constraint, we cannot store the actual solution which could take space $\Omega(\Delta^d)$, hence the algorithms only approximate the optimal *cost* OPT, which in case of UFL is $\mathrm{OPT} := \min_{F \subseteq \mathbb{R}^d} \mathrm{cost}(P, F)$. We say that a randomized algorithm achieves *α-approximation*, for $\alpha \geq 1$, if it outputs an estimate E that with probability at least $2/3$ satisfies $\mathrm{OPT} \leq E \leq \alpha \cdot \mathrm{OPT}$. (In many cases, this success probability can be amplified using standard methods.)

In this dynamic geometric streaming model, algorithms generally exhibit a dichotomy, as their space complexity is either polynomial or exponential in the dimension d. Obviously, algorithms whose space complexity is exponential in d are applicable only in a low-dimensional setting, say $d = O(1)$ or $d = O(\log \log \Delta)$. This class contains many algorithms that achieve an $O(1)$-approximation, or even $(1+\varepsilon)$-approximation for arbitrary fixed $\varepsilon > 0$, from facility location [2], [3] to minimum spanning tree [4], and several other basic geometric problems [5]–[7]. The list would extend even further if we included algorithms for the insertion-only model.

One would clearly prefer algorithms whose space complexity is polynomial in d. Such algorithms are known for facility location and several other problems, but unfortunately their approximation ratio is usually high, for example $O(d \cdot \log \Delta)$ or even higher; see, e.g., [1], [8], [9]. A major open problem in the area is to break this barrier and significantly improve the approximation factor, say to $O(1)$ if not $1 + \varepsilon$, to match the ratios obtainable in low dimension, while using space polynomial in $d \cdot \log \Delta$. This question was partially resolved for k-median [10] and k-means [11], by designing coresets, to achieve $(1 + \varepsilon)$-approximation using space $k \cdot \mathrm{poly}(d \cdot \log \Delta)$ for a fixed $\varepsilon > 0$; however, these bounds are low-space only for small values of k.

As for UFL, the state-of-the-art streaming algorithms achieve only: $O(d \cdot \log^2 \Delta)$-approximation using space $\mathrm{poly}(d \cdot \log \Delta)$ [1]; $O(1)$-approximation for constant d [2] (which seems to generalize to $2^{O(d)}$-approximation using $2^{O(d)}$-space); and $(1 + \varepsilon)$-approximation for $d = 2$ [3] (even if this approach could be extended to general d, its space bound seems to be $(\log \Delta)^{\Omega(d)}$). Since one can expect many applications of UFL (or clustering in general) to require high dimension d, the above bounds would not be satisfactory for either having large approximation ratio or using prohibitively large space.

The main barrier for closing this gap is the lack of suitable techniques for high-dimensional spaces. In particular, the quadtree subdivision is a natural geometric decomposition technique used in almost all previous geometric streaming algorithms, allowing for a lot of strong results in a low dimension [3]–[7]. However, a quadtree in a high dimension, as used by Indyk [1] and more recent follow-up papers [8], [9], gives a tree embedding, which is very useful algorithmically, but necessarily distorts distances by an $\Omega(d \cdot \log \Delta)$-factor. It seems that new techniques must be developed in order to get a better approximation ratio in a high dimension, but surprisingly, very little, if at all, is known beyond the quadtree techniques. This technical barrier is not specific to UFL, but rather applies to many geometric problems.

A. Results

We break this barrier of existing techniques, and devise a geometric *importance-sampling* approach that is based on a *geometric hashing scheme*. Using this technique, we obtain improved approximation for UFL in high-dimensional streams, i.e., using $\mathrm{poly}(d \cdot \log \Delta)$ space. We present our new streaming bounds for UFL below, and defer discussion of our technical contributions to Section I-B. Throughout, we assume that all data points in P are distinct, and that the streaming algorithm knows in advance the parameters Δ and $\mathfrak{f}$. These are relatively standard assumptions in the model of dynamic geometric streams, and we omit them from the statements below.[1]

We begin by presenting a *two-pass* streaming algorithm that computes an $O(1)$-approximation to the cost of UFL. The underlying idea is to apply in the first pass the abovementioned importance-sampling method to select a sample of points from P, and then in the second pass to compute the contribution of each sampled point (for details see Section I-B). This result demonstrates the significant advantage of our technique over the quadtree/tree-embedding based methods, which do not yield an $O(1)$-approximation, even in two passes. This limitation of previous techniques can be seen also in recent work [9] that achieves $O(\log n)$-approximation for bichromatic matching (earth mover distance) in two passes, where $n = |P|$ is the number of input points (their approach extends to one pass, albeit with an additive error).

Theorem I.1 (***Two-Pass Algorithm***). *There is a two-pass randomized algorithm that computes an $O(1)$-approximation to Uniform Facility Location of an input $P \subseteq [\Delta]^d$ presented as a dynamic geometric stream, using $\mathrm{poly}(d \cdot \log \Delta)$ bits of space.*

We can extend the approach from Theorem I.1 to *random-order insertion-only streams* (where the stream is a uniformly random permutation of P and we assume $|P|$ is given), by using the first half of the stream to simulate the first pass of the algorithm from Theorem I.1, and then using the second half to approximate the estimator used in the second pass of the algorithm. There is clearly correlation between the two halves of the stream, and neither half represents the full input faithfully, but we can circumvent these obstacles and obtain a one-pass $O(1)$-approximation algorithm, as stated below. We

[1]It suffices to know an $O(1)$-approximation to $\log \Delta$ and to $\mathfrak{f}$ for all our results. The assumption about distinct points was made also in [1], and can sometimes be removed easily, e.g. in insertion-only streams using a random perturbation.

are not aware of previous geometric streaming algorithms in high dimension, particularly those based on quadtrees, that obtain $O(1)$-approximation in random-order streams (apart from those for k-median [10] and k-means [11], though these bounds are low-space only for small k).

Theorem I.2 (***Random-Order Algorithm***)**.** *There is a one-pass randomized algorithm that computes an $O(1)$-approximation to Uniform Facility Location of an input $P \subseteq [\Delta]^d$ presented as a random-order insertion-only stream, using $\mathrm{poly}(d \cdot \log \Delta)$ bits of space.*

Finally, to deal with arbitrary-order dynamic streams in one pass, we have to significantly extend our methods. Specifically, our previous scheme of first obtaining samples and then computing their estimations cannot work, and we need our importance sampling to provide additional structure. This more involved step crucially relies on a powerful property in our geometric hashing scheme, namely, that every point set of small-enough diameter is hashed to only $\mathrm{poly}(d)$ distinct buckets. However, this also introduces the $O(d^{1.5})$-factor in the approximation ratio, which corresponds to a key performance parameter of the hashing.

Theorem I.3 (***One-Pass Algorithm***)**.** *There is a one-pass randomized algorithm that computes an $O(d^{1.5})$-approximation to Uniform Facility Location of an input $P \subseteq [\Delta]^d$ presented as a dynamic geometric stream, using $\mathrm{poly}(d \cdot \log \Delta)$ bits of space.*

While the approximation ratio of $O(d^{1.5})$ may look prohibitively large, it is useful to note that many geometric problems, including facility location, are reducible to the case $d = O(\log n)$, where $n = |P|$, by using standard techniques relying on the Johnson–Lindenstrauss lemma. Furthermore, the previously best streaming algorithm for UFL with $\mathrm{poly}(d \cdot \log \Delta)$ space, by Indyk [1], has an approximation factor of $O(d \cdot \log^2 \Delta)$ and thus, for $d = \Theta(\log n)$ and $\Delta = \mathrm{poly}(n)$, we improve the approximation factor from $O(\log^3 n)$ to $O(\log^{1.5} n)$.

Other streaming algorithms for UFL in the literature focus solely on the case of constant d, or even just $d = 2$, and for general d seem to use space exponential in d. In particular, Lammersen and Sohler [2] gave an $\exp(O(d))$-approximation using $\exp(\Omega(d)) \cdot \mathrm{polylog}\,\Delta$ space (which thus has worse approximation ratio than our algorithm for $d = \omega(1)$), and Czumaj et al. [3] designed a $(1+\varepsilon)$-approximation for UFL in the Euclidean plane, which for a fixed $\varepsilon > 0$ seems to require space $(\log \Delta)^{\Omega(d)}$ in order to be extended to dimension d, meaning that the space would be superlogarithmic for $d = \omega(1)$.

The results above are complemented by the following lower bound, which follows by a reduction from the one-way communication complexity of the Boolean Hidden Matching problem.

Theorem I.4 (***Streaming Lower Bound***)**.** *For any dimension d, any one-pass randomized algorithm that for a given insertion-only stream of points from $[\Delta]^d$ for $\Delta = 2^{O(d)}$ approximates Uniform Facility Location within ratio better than 1.085, requires $2^{\mathrm{poly}(d \cdot \log \Delta)}$ space.*

The lower bound in particular implies the impossibility of designing a streaming $(1+\varepsilon)$-approximation for any $\varepsilon > 0$ with $f(\varepsilon) \cdot \mathrm{poly}(d \cdot \log \Delta)$ space, even for insertion-only streams. In terms of $n = |P|$, the space lower bound is $\Omega(\sqrt{n})$ in the setting with $d = \Theta(\log n)$ and $\Delta = \mathrm{poly}(n)$.

B. Technical Overview

For simplicity, in this section we consider the insertion-only setting, and assume that the instance is scaled so that the opening cost is $\mathfrak{f} = 1$. Then the input $P \subseteq \mathbb{R}^d$ of size $n = |P|$ is not restricted to a discrete grid. Our overall strategy is to design a streaming implementation of an estimator known to achieve an $O(1)$-approximation. This estimator was proposed in [12] in the context of sublinear-time algorithms (based on an offline $O(1)$-approximation algorithm in [13]). The idea is to associate to every data point $p \in P$ a value $r_p \in [1/n, 1]$ (formally defined in Definition II.1), that satisfies two key properties (Fact II.2): First, the sum of all r_p's gives an $O(1)$-approximation to UFL, i.e., $\sum_{p \in P} r_p = \Theta(\mathrm{OPT})$. Second, r_p is roughly the inverse of the number of points inside the ball $B(p, r_p)$ centered at p with radius r_p, i.e., $|P \cap B(p, r_p)| = \Theta(1/r_p)$. It follows that r_p can be estimated (see Fact II.3) by just counting the number of points in the balls $B(p, 2^{-j})$ for $j = 1, \ldots, \log_2 n$, which is easy if p is known at the beginning of the stream. However, if p is given as a *query* at the end of the stream, then any finite approximation requires $\Omega(n)$ space, by a reduction from the communication complexity of indexing.

Naïve Approach: Uniform Sampling: Consider initially making two passes over the stream, the first one samples a few points, and the second pass estimates the r_p value for each sampled point p. Since all $r_p \in (0, 1]$, one immediate idea is to perform *uniform sampling*, and argue using Chernoff bounds that the resulting scaled estimate is likely to be $\Theta(\sum_{p \in P} r_p) = \Theta(\mathrm{OPT})$. However, to obtain decent concentration, one needs the expectation $\sum_{p \in P} r_p$ to be large enough, which need not hold. Indeed, consider Example I.5 below, where uniform sampling needs to draw $\Omega(\sqrt{n})$ samples to have a decent chance to see even one point p with $r_p = 1$, which is necessary for obtaining a nontrivial approximation.

Example I.5. Let $P = P_1 \cup P_2$, where P_1 consists of $\sqrt{n}$ points, whose pairwise distances are at least 1, and P_2 consists of $n - |P_1| = \Theta(n)$ points whose pairwise distances are (approximately) $1/n$; these two sets are at distance at least 1 from each other. (A realization of this point set is possible in dimension $\Theta(\log n)$.) One can easily verify from the definition that all $x \in P_1$ have $r_x = 1$, and all $x \in P_2$ have $r_x = \Theta(1/n)$, thus $\mathrm{OPT} = \Theta(\sum_{p \in P} r_p) = \Theta(\sqrt{n})$.

To bypass the limitation of uniform sampling, we can employ *importance sampling*: sample one point $p^* \in P$, such that each $p \in P$ is picked with probability roughly proportional to

r_p, and construct an unbiased estimator $\widehat{Z} = 1/\Pr[p^*] \cdot r_{p^*}$. By a standard analysis, such an estimator has low variance, and thus averaging a few independent samples yields an accurate estimate. To get some intuition, in Example I.5, when sampling proportionally to r_p, the total sampling probability of the points p with $r_p = 1$ is far larger than that of the remaining points.

However, this importance sampling idea is difficult to implement in streaming. At first glance, it is a chicken-and-egg problem: importance sampling aims to estimate $\sum_p r_p$, but it requires knowing the r_p values. The crux is that a coarse estimation for r_p suffices for importance sampling, but as noted above, computing r_p for point queries with any finite ratio requires $\Omega(n)$ space. Moreover, even if the r_p values of the sampled points could be estimated at the end of the stream, how would a streaming algorithm draw samples proportionally to these estimates?

New Idea: Geometric Importance Sampling: Instead of trying to estimate the value of r_p, we implement importance sampling indirectly (in Theorem III.1), using a geometric hashing scheme $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ that "isolates" points with large r_p. As usual in hashing, the codomain of φ is somewhat arbitrary (e.g., in applications it could be $[\Delta]^d$), and a *bucket* refers to a preimage $\varphi^{-1}(z)$, i.e., the set of points mapped to the same image z. Ideally, we would like the aforementioned points (with large r_p) to each get its own bucket, and the others (small r_p) to collide, say, into one bucket per cluster, and thus take up only a few buckets. Given such a hashing scheme, we apply it to all the points in P and pick a non-empty bucket at random. This is equivalent to sampling uniformly from the hash values (of all points in P), and is easily implemented in streaming using a well-known tool called the ℓ_0-sampler; see e.g. [14] (this tool produces a uniform sample from the *distinct elements* of a stream, and applying it here will sample uniformly from the distinct hash values).

The geometric hashing scheme, when combined with subsampling, guarantees that with high probability,

a) the number of non-empty buckets is bounded by $\operatorname{poly}(d \cdot \log \Delta) \cdot \mathrm{OPT}$, and
b) every bucket with at least one point of large r_p contains at most $\operatorname{poly}(d \cdot \log \Delta)$ points.

We may assume that points of large r_p constitute a significant fraction of $\sum_p r_p = \Theta(\mathrm{OPT})$ (because we can anyway neglect a subset of points whose contribution is low), and employ the following *two-level uniform sampling*: First sample uniformly a non-empty bucket of the hashing scheme, and then sample uniformly a point from that bucket. Now, the probability of sampling a point p of large r_p is at least $1/\operatorname{poly}(d \cdot \log \Delta)$ (this holds whenever $r_p > 1/\operatorname{poly}(d \cdot \log \Delta)$). Thus, by taking $\operatorname{poly}(d \cdot \log \Delta)$ samples we are likely to hit at least one point of large r_p, which in fact leads to a robust estimator. The aforementioned two-level uniform sampling is implemented by extending a standard construction of the ℓ_0-sampler (in Lemma III.3), inspired by a different extension in [4].

This implementation of importance sampling bypasses the straightforward approach of first estimating the desired values (in our case r_p) and then sampling accordingly, as done previously in some fast algorithms, e.g., for counting [15] and for geometric problems [16], and in "data-compression" algorithms, e.g., constructing graph sparsifiers [17], [18] and geometric coresets [19], [20]. Previously, such a non-straightforward implementation of importance sampling was employed in streaming algorithms for matrices [21], [22], for the same reason that the needed values are hard to compute in a streaming fashion.

Consistent Geometric Hashing with Bounded Gap: We now elaborate on the geometric hashing, which plays a central role in our importance-sampling algorithm and is formally defined as follows. Throughout, $\operatorname{diam}(S)$ denotes the diameter of $S \subseteq \mathbb{R}^d$.

Definition I.6 (Consistent Hashing)**.** A mapping $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ is called a Γ-*gap* Λ-*consistent hash* with diameter bound $\ell > 0$, or simply (Γ, Λ)-*hash*,[2] if it satisfies:

1) Diameter: for every image $z \in \varphi(\mathbb{R}^d)$, we have $\operatorname{diam}(\varphi^{-1}(z)) \leq \ell$; and
2) Consistency: for every $S \subseteq \mathbb{R}^d$ with $\operatorname{diam}(S) \leq \ell/\Gamma$, we have $|\varphi(S)| \leq \Lambda$.

Intuitively, the first condition (diameter) requires that further apart points are never hashed (mapped) to the same bucket, and the second one (consistency) requires that highly-clustered points, even if their number is very large, are hashed to only a few different buckets.

We present this definition with a general parameter Λ, but in our application we always require $\Lambda := \operatorname{poly}(d)$, which is sufficient as our algorithms have their approximation ratios independent of Λ and space only polynomially depending on Λ. However, the "gap" parameter Γ, equal to the ratio between the diameter bound ℓ and the consistency diameter ℓ/Γ, goes into the approximation factor of our one-pass streaming algorithm (Theorem I.3), besides affecting the space complexity polynomially.

Comparison to Related Geometric Decompositions: Our definition of consistent hashing is essentially equivalent to the notion of sparse partitions that was introduced by Jia, Lin, Noubir, Rajaraman, and Sundaram [23]. Their definition concerns a *partition* of $\mathbb{R}^d$, if we view each part in that partition as a bucket in a hashing, then their definition is the same as Definition I.6. However, in our setting it is more natural to think of a hash function, because the part/bucket that contains each input point $x \in \mathbb{R}^d$ must be computed *using a small amount of memory*, and a mere partition of $\mathbb{R}^d$ may not suffice. The construction in [23] has consistency $\Lambda = 2^d$ (it is a straightforward partition into cubes), and is thus not useful in our context. Notably, Filtser [24] designed a sparse partition with consistency $\Lambda = \operatorname{poly}(d)$ and gap $\Gamma = O(d/\log d)$, however, in this partition the description of a part takes $\Omega(2^d)$

[2]While the parameter ℓ is important when applying the hashing, when constructing the hashing one can assume by scaling that $\ell = 1$.

bits, and thus does not directly imply a hash function that can be evaluated on a point in small space.

We construct a consistent hashing scheme with gap $\Gamma = O(d^{1.5})$ (Lemma III.6) that can be evaluated at a point $x \in \mathbb{R}^d$ in space and time $\mathrm{poly}(d)$. We stress that the hash function is *data-oblivious*, because Definition I.6 requires the function to be defined on the entire $\mathbb{R}^d$. This gap parameter Γ can potentially be improved to $O(d/\log d)$, to match the non-streaming construction in [24].

In fact, the same gap bound of $\Gamma = O(d^{1.5})$ was recently obtained by Dunkelman et al. [25]. It is not stated there explicitly (but can be verified by inspecting their analysis) because their construction is designed for a different notion, called *consistent rounding*, which is incomparable to our Definition I.6 primarily because they require each bucket to have a bounded volume (instead of diameter), and their guarantee on the number of intersections is also slightly different. Their construction is somewhat similar to ours except that it works top-down, whereas ours works bottom-up.

We note that the second condition in Definition I.6 does not easily follow from many known methods in the literature. Indeed, if we partition the space $\mathbb{R}^d$ using standard methods, like hypercube subdivisions as in a quadtree (see, e.g., [26]), it is hard to avoid clusters from intersecting $2^{\Omega(d)}$ buckets, instead of only $\mathrm{poly}(d)$. Other geometric decompositions, such as padded decomposition [27]–[29], and Locality-Sensitive Hashing [30], aim for different guarantees that are not directly comparable to those of Definition I.6; see Section I-C for a broader comparison.

Remark I.7. By a private communication with Arnold Filtser, we were informed of the existence of hash functions with $\Gamma = O(d/\log d)$ that uses space $\mathrm{poly}(d)$, and this gap bound is known to be tight [24]. Plugging in this new hash function, the approximation ratio of Theorem I.3 may be improved to $O(d/\log d)$. However, it takes time $\exp(d)$ to evaluate the hash value for a single point which is significantly worse than the $\mathrm{poly}(d)$ time as in our construction, and thus results in a worse running time in Theorem I.3.[3] It is therefore open to design a $\mathrm{poly}(d)$-consistent geometric hash function with $\Gamma = O(d/\log d)$ that can be evaluated in $\mathrm{poly}(d)$ space *and* time.

Construction of the Hashing: Our $O(d^{1.5})$-gap construction first partitions the entire $\mathbb{R}^d$ into unit hypercubes. Then we apply an identical decomposition/partition on every unit hypercube, and the partition of the entire space is the collection of all the parts (across all unit hypercubes). The partition into hypercubes immediately implies a diameter bound. Consider now the partition of an (arbitrary) unit hypercube H; intuitively, we partition H into $d+1$ *groups* of regions, such that regions in the same group are at distance $\geq \varepsilon := \ell/\Gamma$ of each other (where $\ell = O(\sqrt{d})$ when we start with unit hypercubes). This way, a subset of diameter $< \varepsilon$ can only intersect one region from each of the $d+1$ groups, which ensures consistency. Roughly, each of the $d+1$ groups, say the i-th group, corresponds to all the i-dimensional faces of the hypercube. However, simply taking these i-dimensional faces cannot work, since they intersect and the minimum distance is 0. To make them separated, we employ a sequential process, where iteration $i = 0, 1, 2, \ldots$ takes as our next group the i-dimensional faces and their close ℓ_∞ neighborhoods, excluding (i.e., removing) points that are sufficiently close (in the ℓ_∞ distance) to $(i-1)$-dimensional faces. In particular, we crucially use the following geometric fact.

[3]Even though not explicitly stated, the running time per update and query in Theorem I.3 is $\mathrm{poly}(d \cdot \log \Delta)$.

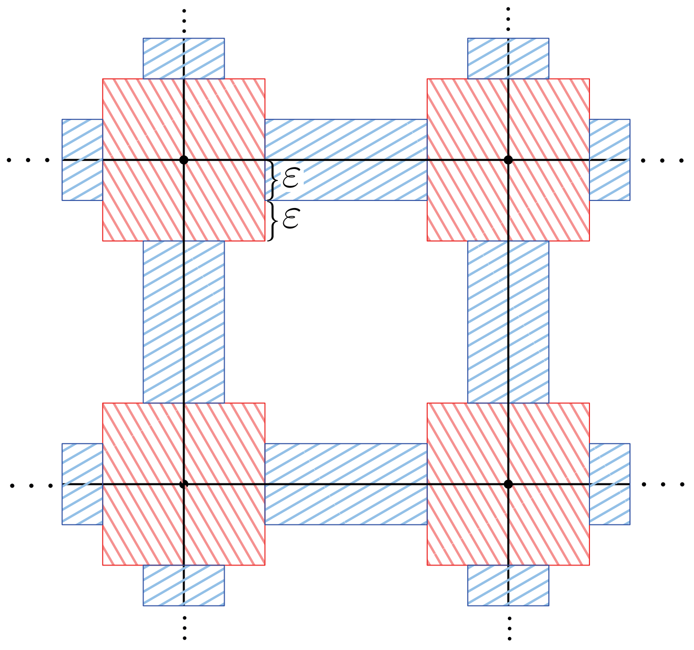

Fig. 1. An illustration of the construction of a $(\Gamma, d+1)$-hash in 2D, showing only a vicinity of a single square $[0,1]^2$ of the initial partition (in black). The 2ε-neighborhoods of vertices, which form $\mathcal{S}_0^{+2\varepsilon}$, are depicted in red (north-west shaded areas), and the ε-neighborhoods of edges, that is, set $\mathcal{S}_1^{+\varepsilon} \setminus \mathcal{S}_0^{+2\varepsilon}$, are depicted in blue (north-east shaded areas).

Fact I.8. *Consider two orthogonal i-dimensional subspaces S_1, S_2 and denote their intersection by I. Denoting by A^{+t} the ℓ_∞ neighborhood of A of radius t, define $S_i' := S_i^{+\varepsilon} \setminus I^{+2\varepsilon}$ for $i = 1, 2$. Then $\mathrm{dist}(S_1', S_2') > \sqrt{2}\varepsilon$.*

Our construction iterates over $i = 0, \ldots, d-1$ and repeatedly uses the above fact. Denoting by $\mathcal{S}_i$ the union of the i-dimensional faces, the i-th group consists of the connected regions of $\mathcal{S}_i^{+(d-i)\varepsilon} \setminus \mathcal{S}_{i-1}^{+(d-i+1)\varepsilon}$. Finally, the d-th group consists of a single region corresponding to the hypercube interior after removing the previous groups. See Figure 1 for an illustration.

To intuitively see why this geometric hashing helps, consider the instance in Example I.5, and let us focus on estimating the number of points with $r_p = 1$ (for which uniform sampling does not work). In Example I.5, points are grouped into natural clusters: the whole P_2 forms a cluster that consists of $O(n)$ points each with $r_p = O(1/n)$, and each point in P_1 forms a singleton cluster whose $r_p = 1$. We construct a $(\Gamma, \mathrm{poly}(d))$-hash with diameter bound $\ell = 1/2$ (as guaranteed in Lemma III.6), and let $\varepsilon := \ell/\Gamma$. By the second guarantee, every small ball of radius $O(1/n) \ll \varepsilon$ is mapped to $\mathrm{poly}(d)$ points/buckets, even if the ball originally could have $\Omega(n)$ points. Hence, after the hashing, the entire cluster P_2 with $O(n)$ points gets mapped to $\mathrm{poly}(d)$ points, while points in P_1 are preserved because of the diameter bound of the buckets. Hence, applying the two-level uniform sampling, we hit one point in P_1 with at least $\mathrm{poly}(d)$ samples on average.

Finally, we note that the actual implementation of this whole idea is more involved and requires additional steps. For instance, the nice cluster structure in Example I.5 might not be present in a general input, and our analysis needs to explicitly define a clustering where a cluster containing p has diameter roughly r_p. Another issue is that our overview focused on $r_p = 1$. For general r_p, we use a subsampling at rate 2^{-i} to "reduce" the case $r_p = 2^{-i}$ to the case $r_p = 1$, which is conceptually similar to the subsampling used in the construction of ℓ_0-samplers. At the end, this algorithm implements our desired importance sampling task, namely, it uses space $\mathrm{poly}(d \cdot \log \Delta)$ and produces a sample $p^* \in P$ such that every $p \in P$ is picked with probability proportional to at least $r_p/\mathrm{poly}(d \cdot \log \Delta)$.

Streaming Implementations: The above-mentioned geometric importance sampling (Theorem III.1) can be implemented in one pass using small space. However, it only returns a set of *samples* $S \subseteq P$, and to actually estimate $\sum_{p \in P} r_p$ one still needs to estimate the value r_p for each $p \in S$. This limitation is a consequence of our importance-sampling approach, which bypasses estimating the r_p values on purpose.

Two Passes and Random-Order Streams: Our two-pass streaming algorithm is quite simple: the first pass computes a sample $S \subseteq P$ using importance sampling (Theorem III.1), and the second pass estimates r_p for each $p \in S$ using Fact II.3 and straightforward counting. Its space complexity is $\mathrm{poly}(d \cdot \log \Delta)$ per point $p \in S$, and we need $|S| = \mathrm{poly}(d \cdot \log \Delta)$.

A similar approach can be applied also in the random-order model (i.e., the stream is a uniformly random permutation of P): the first half of the stream is used to generate a sample S, and the second half is used to estimate the r_p value of the points $p \in S$ sampled in the first half. However, more technical steps are needed in the analysis, due to the correlation between the two halves of the stream, and the fact that a random half does not represent the full stream accurately.

One-pass Implementation: The one-pass setting is significantly more difficult. We estimate $\sum_{p \in P} r_p$, by partitioning P into levels $i = 1, \ldots, d \cdot \log_2 \Delta$, namely, we let $P_i := \{p \in P : r_p \in (2^{-i}, 2^{-i+1}]\}$ and $W_i := \sum_{p \in P_i} r_p$. We build an estimator for W_i separately for each i (recall that in this section we assume $\mathfrak{f} = 1$).

For simplicity, we focus here on $i = 0$, so we now only care about points $p \in P_0$, meaning that $r_p = \Theta(1)$. Then $W_0 = \sum_{p \in P_0} r_p = \Theta(|P_0|)$, and it suffices to estimate $|P_0|$. To this end, it is natural to use the estimator $1/\Pr[x] \cdot I(x \in P_0)$, where x is a sample generated by our importance sampling. However, the indicator $I(x \in P_0)$ turns out to be very sensitive, and it is difficult to estimate it in one pass even within a constant factor (e.g., distinguish between $r_x \geq 1/2$ and $r_x \leq 1/10$). Hence, we have to design a more relaxed tester and analyze how this affects the approximation ratio.

To implement this tester, we construct an $(\Gamma, \mathrm{poly}(d))$-hash with diameter bound $\ell = 1/10$. Recalling that the r_p values satisfy that $|P \cap B(p, r_p)| = \Theta(1/r_p)$, we observe that if a hashing bucket contains a point with $r_p \geq 1/2$, then the number of points in this bucket is bounded by $O(1)$. This means that a bucket either consists only of points with large r_p values, or no such point at all. Using this observation, we maintain a counter for the number of points mapped to every bucket, and when a point is sampled, we retrieve also the counter for its bucket, and if the counter is small, we use it as a proxy for the event that the bucket consists only of points with large r_p values, including in particular the sampled point.

However, a subtle technical issue is that some point y with $r_y \ll 1$ can possibly lie on the "boundary" of the bucket, and then the number of points in that bucket is small, while $P \cap B(y, r_y)$ contains many nearby points that lie in other buckets. Hence, we need to count the number of points in a slightly enlarged region, i.e., for a bucket $Q \subseteq \mathbb{R}^d$ we need to count the points in P that fall inside the r_y-neighborhood of Q, denoted here as $B(Q, r_y)$ and defined as the set of points in $\mathbb{R}^d$ at distance at most r_y from Q. Hence, if we maintain the counter for $P \cap B(Q, \beta)$ for some $\beta > 0$ then the information of the counter suffices for rejecting y's whose $r_y \leq O(\beta)$.

Now, suppose we are to maintain the counter for some β. To implement this, whenever we see a data point x arrives, we should increase the counter for all buckets Q such that $x \in B(Q, \beta)$ (a similar trick appears e.g. in [4]). However, this becomes challenging in the streaming setting, primarily due to the fact that the number of buckets Q such that $x \in B(Q, \beta)$ can be huge (e.g., 2^d), and more importantly, most of them may be "fake", in that they do not contain any data point. Consequently, these fake nonempty buckets can enlarge the support of the ℓ_0-sampler significantly, which makes it difficult to obtain a uniform sample.

To tackle this challenge, we make use of the guarantee from the geometric hashing, that any subset of $\mathbb{R}^d$ with diameter less than $\varepsilon = O(1/\Gamma)$ is mapped to $\mathrm{poly}(d)$ buckets (see Definition I.6). Hence, if we choose $\beta = \varepsilon/2$, the effect of enlarging the buckets by an additive $\beta = \varepsilon/2$ is essentially making every data point x a ball $B(x, \varepsilon/2)$, and add the image of this ball to the buckets/counters. The $\mathrm{poly}(d)$ intersection bound ensures that the number of fake buckets is still well bounded.

Finally, the approximate tester is off by a factor of Γ, which is the gap of the geometric hashing, and we show that this factor goes into the approximation ratio, which will thus be $O(\Gamma)$.

C. Related Work

Facility Location Problem: The facility location problem is one of the fundamental problems in operations research and combinatorial optimization, and has received extensive studies in the past. In the offline setting with uniform opening costs, the facility location problem has been proved to be NP-hard [31], and is hard to approximate within factor 1.463 [32] (unless $\mathbf{NP} \subseteq \mathbf{DTIME}\left[n^{O(\log\log n)}\right]$). For the upper bounds, the state of the art for general metrics is a 1.488-approximation by Li [33], and PTAS's are known for special metric spaces, specifically, doubling metrics [34], minor-free graphs [35], and near-linear time PTAS's for (bounded-dimensional) $\mathbb{R}^d$ [36] and planar graphs [37]. For the online setting, Meyerson [38] gave an $O(\log n)$-competitive algorithm, and Fotakis [39] proved that it has ratio $\Theta(\log n/\log\log n)$ and that this ratio is asymptotically optimal.

Geometric Decomposition: Geometric decomposition is a topic that was studied extensively, with many different definitions, even beyond $\mathbb{R}^d$, that are motivated by numerous applications. For brevity, we only mention a few that are closer to our work. One basic genre, often called *space partitioning*, refers to a partition of $\mathbb{R}^d$, perhaps using a variant of the standard grid (quadtree) partition, e.g. [40]. Sometimes it is convenient to use multiple space partitions, or a probability distribution over space partitions (e.g., a few shifts or a random shift of the grid partition [26], [41], [42]). Another standard requirement is that every part in the partition has a bounded diameter (e.g., padded and separating decomposition [27]–[29]), or alternatively that every part has a bounded volume (e.g., [25], [43]). The above examples ask that nearby points lie in the same part, but another type of decomposition, called Locality-Sensitive Hashing (LSH) [30], only asks that close-by points fall in the same part with noticeably higher probability than far-away points.

Two-pass and Random-order Streaming Algorithms: Besides the most studied streaming model of one-pass algorithms over an arbitrary (non-random) order streams, algorithms requiring a few passes or assuming that the stream order is random have received significant attention as well. For graph streams, two-pass streaming algorithm have been designed, for example, for graph spanners [44], [45], maximum matching [46], and triangle counting [47]. Apart from graph streams, other examples of two-pass algorithms include those for matrix norm estimation [48], set cover [49], and geometric earth mover distance [9]. The one-pass random-order (insertion-only) setting was studied for problems including matching [50], [51], quantile estimation [52], graph connected components and minimum spanning tree [53], and frequency moment estimation [54].

D. Paper Organization

Due to the space limit, we only present the technical details for geometric importance sampling (Section III). The details for the streaming implementations and the lower bounds are omitted, and they can be found in the full version [55].

II. Preliminaries

Notation: We use the usual notation $[n] := \{1,\dots,n\}$, and for a function $\varphi : X \to Y$ and $y \in Y$, we denote $\varphi^{-1}(y) := \{x \in X : \varphi(x) = y\}$. The *d-dimensional ball* centered at $x \in \mathbb{R}^d$ with radius $r \geq 0$ is defined as $B(x,r) := \{y \in \mathbb{R}^d : \operatorname{dist}(x,y) \leq r\}$.

Definitions and Facts from Mettu-Plaxton (MP) Algorithm: We will need some machinery from the MP algorithm [12], [13]. We first introduce the definition of r_p, and then recall useful facts, particularly that it suffices to approximate $\sum_{p\in P} r_p$, because it $O(1)$-approximates OPT.

Definition II.1 ([13])**.** For every $p \in P$, let r_p be the number such that

$$\sum_{x\in P\cap B(p,r_p)} \big(r_p - \operatorname{dist}(p,x)\big) = \mathfrak{f}. \tag{1}$$

It is easy to see that r_p is well-defined and $\frac{\mathfrak{f}}{|P|} \leq r_p \leq \mathfrak{f}$. Indeed, using the notation $z^+ = \max(z,0)$, we can write the left-hand side of (1) as $\sum_{x\in P}(r_p - \operatorname{dist}(p,x))^+$, which is easily seen to be non-decreasing with r_p. For illustration, suppose p is one of k points whose pairwise distances are all equal to $a \in (0,\mathfrak{f})$, and all other points are at distance at least $\mathfrak{f}$ from p; then $r_p = \Theta(a + \mathfrak{f}/k)$.

Fact II.2 (Lemmas 1 and 2 in [12])**.** *The following holds.*

- *For every $p \in P$, it holds that $|P \cap B(p,r_p)| \geq \mathfrak{f}/r_p$ and $|P \cap B(p,r_p/2)| \leq 2\mathfrak{f}/r_p$.*
- *$\sum_{p\in P} r_p = \Theta(\mathrm{OPT})$.*

We assume without loss of generality (w.l.o.g.) that Δ is a power of two. Let $L := d \cdot \log_2 \Delta \geq \log_2 |P|$ as $|P| \leq \Delta^d$ by the assumption that points are distinct. The first point of Fact II.2 implies that the r_p value can be approximated within a constant factor by counting the number of points in balls of geometrically increasing radii.

Fact II.3. *Let j_0 be the maximum $j \in \{0,\dots,L\}$ such that $|P \cap B(p, 2^{-j}\mathfrak{f})| \geq 2^j$. Then*

$$r_p \in (2^{-j_0-1}\mathfrak{f}, 2^{-j_0+1}\mathfrak{f}]. \tag{2}$$

Moreover, there is a one-pass deterministic streaming algorithm that given the opening cost $\mathfrak{f} > 0$ and a point p in advance of the data set P presented as a dynamic stream, returns an estimate $\hat{r}_p$ such that $r_p \leq \hat{r}_p \leq O(r_p)$ using space of $O(L^2)$ bits.

Proof: By Fact II.2, if $r_p \leq 2^{-j_0-1}\mathfrak{f}$ then we arrive at the contradiction $|P \cap B(p, 2^{-j_0-1}\mathfrak{f})| \geq |P \cap B(p,r_p)| \geq \mathfrak{f}/r_p \geq 2^{j_0+1}$. And if $r_p > 2^{-j_0+1}\mathfrak{f}$ then we arrive at the contradiction $|P \cap B(p, 2^{-j_0}\mathfrak{f})| \leq |P \cap B(p, r_p/2)| \leq 2\mathfrak{f}/r_p < 2^{j_0}$. To implement the estimation in dynamic streams, we count the number of points in each of the balls $B(p, 2^{-j}\mathfrak{f})$ for $j = 0,\dots,L$ using an L-bit counter. ■

III. Importance Sampling via Geometric Hashing

In this section, we develop a streaming algorithm for importance sampling on P, where the probability to report each point $x \in P$ is (at least) proportional to its contribution to $\sum_{x\in P} r_x$. Similarly to other streaming algorithms for sampling (e.g., ℓ_p-samplers), our algorithm might fail with a tiny but non-zero probability, in our case $q_{\text{fail}} = 1/\operatorname{poly}(\Delta^d)$, and the analysis can effectively ignore these events by a union bound.[4] While the algorithm's goal is to sample from P, we also allow it to return $\perp$, which is *not* considered a failure, as long as it returns points from P with sufficiently large probability.[5] The output $\perp$ is useful in the algorithm's design, as it can replace the use of a fixed point from P, and also handle properly the corner case $P = \emptyset$.

Theorem III.1. *There is a one-pass randomized algorithm that, given $P \subseteq [\Delta]^d$ presented as a dynamic geometric stream, samples a random point $p^* \in P \cup \{\perp\}$ such that*

$$\forall x \in P, \qquad \Pr[p^* = x] \geq \Omega\left(\frac{1}{\operatorname{poly}(d \cdot \log \Delta)}\right) \cdot \frac{r_x}{\sum_{y\in P} r_y},$$

and also reports a 2-approximation $\widehat{\Pr}[p^*]$ for the probability of sampling this point, i.e., $\Pr[p^* = x] \leq \widehat{\Pr}[p^*] \leq 2\Pr[p^* = x]$. This algorithm uses $\operatorname{poly}(d \cdot \log \Delta)$ bits of space, and fails with probability at most $1/\operatorname{poly}(\Delta^d)$.

Proof of Theorem III.1: We first provide an algorithm that samples points $x \in P$ at a given level $i \in \{1, \ldots, L\}$, which refers to points with r_x value roughly $2^{-i}\mathfrak{f}$. We present it as an offline algorithm in Algorithm 1, and discuss below how to implement it as a streaming algorithm. The main guarantee about its output is given in Lemma III.2 below, whose proof appears in Section III-B. We remark that the algorithm returns $\perp$ in case $\operatorname{sub}_i(P)$ is empty, i.e., no point survives the subsampling. (We will see later that Theorem III.1 follows by simply executing a streaming implementation of this algorithm with a random level i.)

Lemma III.2. *Algorithm 1 returns a random point $p^* \in P \cup \{\perp\}$ such that $\forall x \in P_i$,*

$$\Pr[p^* = x] \geq \Omega\left(\frac{1}{\operatorname{poly}(d \cdot \log \Delta)}\right) \cdot 2^{-i} \cdot \frac{\mathfrak{f}}{\mathrm{OPT}},$$

where $P_i := \{x \in P : 2^{-i}\mathfrak{f} < r_x \leq 2^{-i+1}\mathfrak{f}\}$.

[4]Failure in Theorem III.1 or Lemma III.3 means that the algorithm may behave arbitrarily, e.g., not return anything or even return a point outside P, and it is not easy to verify if the point is in P. Formally, having failure probability q_{fail} means that the total variation distance between the algorithm's output distribution and desired distribution (e.g., uniform over a certain set in the case of ℓ_0-sampler) is at most q_{fail}.

[5]For example, an acceptable output distribution may be $\perp$ with probability $\frac{1}{2}$, and every $x \in P$ with probability $\frac{1}{2} r_x / \sum_{y\in P} r_y$.

Algorithm 1 Importance sampling for single level i

```
1: let $\ell \leftarrow 0.1 \cdot 2^{-i}\mathfrak{f}$, let $\varphi_i$ be a $(\Gamma, \operatorname{poly}(d))$-hash of $\mathbb{R}^d$ with diameter bound $\ell$   $\triangleright$ use Lemma III.6
2: subsample $P$ with rate $2^{-i}$   $\triangleright$ denote the subsampled subset of $P$ by $\operatorname{sub}_i(P)$
3: sample uniformly $a \in \varphi_i(\operatorname{sub}_i(P))$, then uniformly $p^* \in \varphi_i^{-1}(a) \cap \operatorname{sub}_i(P)$
4: return $p^*$   $\triangleright$ if such $p^*$ does not exist (e.g., if $\operatorname{sub}_i(P) = \emptyset$), return $\perp$.
```

A. Streaming Implementation of Algorithm 1.

Line 1 uses Lemma III.6 to get a data-oblivious function φ_i, hence this step can be executed before the algorithm starts to process the stream.

Subsampling in Dynamic Streams: Line 2 performs subsampling with rate 2^{-i}, that is, each point in P is independently sampled with probability 2^{-i}. If the stream is insertion-only we just sample each newly added point independently with probability 2^{-i}. However, in dynamic streams we need consistency between insertions and deletions of a point, and we thus apply a random hash function $h : [\Delta]^d \to \{0, 1\}$ such that for every point p we have $\Pr[h(p) = 1] = 2^{-i}$. We draw this hash function at the beginning of the stream, and then for each insertion/deletion of a point p, we evaluate $h(p)$ to determine whether p is subsampled.

In the analysis, we assume that these subsampling events are independent for all points, i.e., that h is fully random. To deal with the fact that storing such a hash function takes Δ^d bits, we use Nisan's pseudorandom generator (PRG) [56] that has the following guarantee: For any parameters R and S, given a seed of $\Omega(S \cdot \log R)$ truly random bits, the PRG generates R bits that cannot be distinguished from truly random bits by any algorithm running in space S. Naturally, we use this PRG with S being the space cost of our algorithm and $R = i \cdot \Delta^d$ (which is the number of independent fair coin flips needed to generate h).

Two-level Uniform Distinct Sampling: To implement the final sampling step of Algorithm 1 (in line 3), we present in Lemma III.3 a two-level ℓ_0-sampler, which is more convenient to describe as sampling from a frequency matrix. is an extension of a standard ℓ_0-sampler (from a frequency vector); see e.g. [14]. We are not aware of such a sampler in the literature, although similar extensions were devised before, e.g. in [4].[6]

Lemma III.3 (Two-Level ℓ_0-Sampler)**.** *There is a randomized algorithm, that given as input a matrix $M \in \mathbb{R}^{m\times n}$, with $m \leq n$ and integer entries bounded by $\operatorname{poly}(n)$, that is presented as a stream of additive entry-wise updates, returns an index-pair (i, j) of M, where i is chosen uniformly at random (u.a.r.)*

[6]The notion of *ℓ_p-sampling with meta-data*, which was recently introduced in [9], sounds related but is quite different, as each index i arrives with an associated value λ_i; in fact, their approach builds on Precision Sampling [57] and is applicable only for $p > 0$. The use of a two-level structure and its representation as a matrix were introduced in [58], [59] as cascaded aggregates/norms, however their algorithms estimate these norms, not sampling an index by the norm. A sampler for cascaded $\ell_{p,2}$-norm was designed in [22], building on properties of the Gaussian distribution and ℓ_p-samplers.

from the non-zero rows, and then j is chosen u.a.r. from the non-zero columns in that row i. The algorithm uses space $\mathrm{poly}(\log n)$*, fails with probability at most* $1/\mathrm{poly}(n)$*, and can further report the corresponding row-sum* $\sum_{j'} M_{i,j'}$.

It is straightforward to implement line 3 of Algorithm 1 using this sampler. Simply convert the updates to P, on the fly, into updates to a frequency matrix M, whose rows correspond to all hash buckets (images of φ_i) and columns correspond to all grid points ($[\Delta]^d$). This is clearly a huge matrix, but it is not maintained explicitly. The reported row-sum $\sum_{j'} M_{i,j'}$ corresponds to the number of points in P that are hashed (mapped) to the bucket returned by the sampler. Hence, we can implement Algorithm 1 in one pass over a dynamic geometric stream. The success probability depends on the two-level ℓ_0-sampler, and is thus $1-1/\mathrm{poly}(n) \geq 1-1/\mathrm{poly}(\Delta^d)$, and assuming success, the output distribution is as described in Lemma III.2.

We can now complete the proof of Theorem III.1. The algorithm draws uniformly at random a level $i^* \in \{1,\dots,L\}$ and executes Algorithm 1 for this level i^*. Now consider a point $x \in P$, and let j be the level for which $x \in P_j$, i.e., $2^{-j}\mathfrak{f} < r_x \leq 2^{-j+1}\mathfrak{f}$. Then by Lemma III.2, the probability to sample this point x is

$$\begin{aligned}\Pr[p^*=x] &\geq \Pr[i^*=j]\cdot\frac{1}{\mathrm{poly}(d\cdot\log\Delta)}\cdot 2^{-j}\cdot\frac{\mathfrak{f}}{\mathrm{OPT}}\\ &\geq \frac{1}{L}\cdot\frac{1}{\mathrm{poly}(d\cdot\log\Delta)}\cdot\Omega\left(\frac{r_x}{\sum_{y\in P} r_y}\right). \qquad (3)\end{aligned}$$

Recall that the algorithm needs to report also an estimate $\widehat{\Pr}[p^*]$ for the probability of sampling the specific point p^* that is reported. Given the randomly chosen level i^* (which might differ from the level j of x), for the algorithm to pick x, it must first subsample x, which happens with probability 2^{-i^*}, then pick the bucket of x under φ_{i^*}, while there are $|\varphi_{i^*}(\mathrm{sub}_{i^*}(P))|$ non-empty buckets, and finally, it has to pick this point x from its bucket, which contains $|\varphi_{i^*}^{-1}(\varphi_{i^*}(x))\cap\mathrm{sub}_{i^*}(P)|$ subsampled points. Thus,

$$\Pr\left[p^*=x \mid i^*\right] = \frac{2^{-i^*}}{|\varphi_{i^*}(\mathrm{sub}_{i^*}(P))|\cdot|\varphi_{i^*}^{-1}(\varphi_{i^*}(x))\cap\mathrm{sub}_{i^*}(P)|}.$$

Furthermore, the algorithm can accurately estimate all these quantities; indeed, the bucket size is known from the two-level ℓ_0-sampler (recall that Lemma III.3 reports also the corresponding row-sum), and to estimate the number of non-empty buckets the algorithm can run in parallel a standard streaming algorithm for counting distinct elements (see e.g. [60]). ■

B. Proof of Lemma III.2

Subsampling: The first step is to subsample every point in P independently with probability $1/2^i$. For every subset $S \subseteq P$, let $\mathrm{sub}_i(S) \subseteq S$ be the random subset after the subsampling. The following describes several standard facts about the subsampling.

Fact III.4. *$\forall S \subseteq \mathbb{R}^d$ and $t \geq 2$, the following holds.*

- *If* $|S| \geq 2^i$*, then* $\Pr[|\mathrm{sub}_i(S)| \geq t\cdot|S|\cdot 2^{-i}] \leq \exp(-\Theta(t))$.
- *If* $|S| \leq 2^i$*, then* $\Pr[|\mathrm{sub}_i(S)| \geq t] \leq \exp(-\Theta(t))$.

Proof: For every $u \in S$, let $X_u \in \{0,1\}$ be the indicator random variable such that $X_u = 1$ if and only if $u \in \mathrm{sub}_i(S)$, so $\Pr[X_u = 1] = 2^{-i}$ for every $u \in S$. Then $|\mathrm{sub}_i(S)| = \sum_{u\in S} X_u$, and $\mathbb{E}[|\mathrm{sub}_i(S)|] = |S|\cdot 2^{-i}$. Let $\mu := \mathbb{E}[\mathrm{sub}_i(S)]$.

- If $|S| \geq 2^i$, then $\mu \geq 1$. By Chernoff bound,

$$\begin{aligned}&\Pr[|\mathrm{sub}_i(S)| \geq t\cdot|S|\cdot 2^{-i}]\\ &= \Pr[|\mathrm{sub}_i(S)| - \mu \geq (t-1)\cdot\mu]\\ &\leq \exp(-\Theta(t)\cdot\mu)\\ &\leq \exp(-\Theta(t)).\end{aligned}$$

- If $|S| \leq 2^i$, then $\mu \leq 1$. By Chernoff bound,

$$\begin{aligned}\Pr[|\mathrm{sub}_i(S)| \geq t] &= \Pr[|\mathrm{sub}_i(S)| - \mu \geq (t/\mu - 1)\cdot\mu]\\ &\leq \exp(-\Theta(t)).\end{aligned}$$

■

Geometric Hashing: We consider bounded consistent hashing schemes that do not map small "clusters" of points into too many buckets. We restate the definition below, and we prove the existence of such hashing schemes, with certain parameters, in Lemma III.6. As mentioned in Section I, Jia et al. [23] introduced an essentially equivalent notion called "sparse partitions", although we further require that evaluating the hash function at a point is efficient.

Definition III.5 (Restatement of Definition I.6)**.** A mapping $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ is called a Γ*-gap* Λ*-consistent hash* with diameter bound $\ell > 0$, or simply (Γ,Λ)*-hash*, if it satisfies:

1) Diameter: for every image $z \in \varphi(\mathbb{R}^d)$, we have $\mathrm{diam}(\varphi^{-1}(z)) \leq \ell$; and
2) Consistency: for every $S \subseteq \mathbb{R}^d$ with $\mathrm{diam}(S) \leq \ell/\Gamma$, we have $|\varphi(S)| \leq \Lambda$.

Lemma III.6. *There exists a $(\Gamma, d+1)$-hash $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ for $\Gamma = \Theta(d\sqrt{d})$. Furthermore, one can evaluate $\varphi(x)$ for input $x \in \mathbb{R}^d$ in time and space* $\mathrm{poly}(d)$.

Proof: The proof can be found in the full version. ■

Sampling on $\varphi_i(\mathrm{sub}_i(P))$: Suppose we apply Lemma III.6 to find a $(\Gamma, \mathrm{poly}(d))$-hash φ_i with diameter bound $\ell = 2^{-i}\mathfrak{f}/10$, and define $\varepsilon := \ell/\Gamma$ which is the magnitude of the consistency guarantee. Then $\varphi_i(\mathrm{sub}_i(P))$ essentially maps points in $\mathrm{sub}_i(P)$ into buckets, and our plan is to sample from these buckets. Next, we wish to upper bound $|\varphi_i(\mathrm{sub}_i(P))|$, which is the support of sampling, in terms of OPT. Since the guarantee on φ_i in Lemma III.6 is about clusters/subsets, we need to first define a clustering of the point set (Lemma III.7) such that the number of points in each cluster C is upper bounded by $O(\mathfrak{f}/\mathrm{diam}(C))$. Then, in Lemma III.8, we use the guarantee of the geometric hashing on the clusters resulting from Lemma III.7 to bound $|\varphi_i(\mathrm{sub}_i(P))|$. In general, there are two types of clusters according to the diameter: i) "small" with diameter at most

$\varepsilon = \ell/\Gamma$, for which we use the consistency guarantee of our geometric hashing, i.e., the second point of Definition I.6 (note that for small clusters, $\mathfrak{f}/\operatorname{diam}(C)$ is not a useful bound on $|C|$), and ii) "large", for which $O(1/\operatorname{diam}(C))$ is not too large and the subsampling leaves only $\operatorname{poly}(d \cdot \log \Delta)$ points for each "large" cluster with high probability. (In the lemma below, if $\operatorname{diam}(C) = 0$, the bound $O(\mathfrak{f}/\operatorname{diam}(C))$ is defined to be Δ^d.)

Lemma III.7 (Extended MP-clustering)**.** *There exists a partition $\mathcal{C}$ of P such that $\mathfrak{f} \cdot |\mathcal{C}| \le O(d \cdot \log \Delta) \cdot \mathrm{OPT}$ and for every $C \in \mathcal{C}$, $|C| \le O(\mathfrak{f}/\operatorname{diam}(C))$.*

Proof: The following algorithm from [13], called MP algorithm, finds a 3-approximation for UFL.

1) List P in non-decreasing order of r_p.
2) Examine $p \in P$ in order, and if there is no open facility in $B(p, 2r_p)$, then open the facility at p.

Denote the set of the facilities opened by MP algorithm as F^{MP}. We use the following steps to construct a partition of P.

1) For every $p \in P$, assign it to the nearest point in F^{MP}.
2) For every $p \in F^{\mathrm{MP}}$, let $C(p) \subseteq P$ be the set of points that are assigned to p.
3) For every $p \in F^{\mathrm{MP}}$ and every $j = 1, \dots, L$, let $C(p)_j := C(P) \cap P_j$.
4) For every $p \in F^{\mathrm{MP}}$ and every j, arbitrarily divide $C(p)_j$ into subsets of size 2^j, possibly with a unique subset that has size $< 2^j$, and include these subsets into $\mathcal{C}$.

Now we show that $\mathcal{C}$ is the collection satisfying Lemma III.7. Clearly, $\mathcal{C}$ covers P, since every point in p is assigned to some point in F^{MP}.

To upper bound the number of points in each cluster, suppose $C \in \mathcal{C}$ is created by dividing $C(p)_j$ for some p and j. Then $\forall q \in C$, $p \in B(q, 2r_q)$. To see this, suppose for the contrary that $p \notin B(q, 2r_q)$. By the construction of $C(p)$, $p \in F^{\mathrm{MP}}$ is the closest to q, hence $p \notin B(q, 2r_q)$ implies that no point in F^{MP} belongs to $B(q, 2r_q)$. However, by the MP algorithm, this means q should have been added to F^{MP}, which is a contradiction. Hence for every $q_1, q_2 \in C \subseteq P_j$,

$$\begin{aligned}\operatorname{dist}(q_1, q_2) &\le \operatorname{dist}(q_1, p) + \operatorname{dist}(p, q_2) \\ &\le O(2^{-j}\mathfrak{f}) + O(2^{-j}\mathfrak{f}) = O(2^{-j}\mathfrak{f}),\end{aligned}$$

which implies that $\operatorname{diam}(C) = O(2^{-j}\mathfrak{f})$. By Step 4 of the construction, we know that $|C| \le 2^j$. Therefore, when $\operatorname{diam}(C) > 0$, $|C| \le O(\mathfrak{f}/\operatorname{diam}(C))$ holds.

Finally, we bound the number of clusters. For every $p \in F^{\mathrm{MP}}$ and every j, the number of subsets that we obtain from the division is at most $\lceil |C(p)_j|/2^j \rceil \le |C(p)_j|/2^j + 1$. Summing over p and j,

$$\begin{aligned}\mathfrak{f} \cdot |\mathcal{C}| &\le \mathfrak{f} \cdot \sum_{p \in F^{\mathrm{MP}}} \sum_{j=1}^{L} 1 + |C(p)_j)|/2^j \\ &\le O(d \cdot \log \Delta) \cdot |F^{\mathrm{MP}}| \cdot \mathfrak{f} + O(1) \cdot \sum_{p \in F^{\mathrm{MP}}} \sum_j \sum_{q \in C(p)_j} r_q \\ &\le O(d \cdot \log \Delta) \cdot |F^{\mathrm{MP}}| \cdot \mathfrak{f} + O(\mathrm{OPT}) \\ &\le O(d \cdot \log \Delta) \cdot \mathrm{OPT}),\end{aligned}$$

where the second inequality uses that $C(p)_j \subseteq P_j$ and $\sum_{p \in P} r_p = \Theta(1) \cdot \mathrm{OPT}$. This completes the proof of Lemma III.7. ∎

Lemma III.8. *With probability at least $1 - 1/\operatorname{poly}(\Delta^d)$, $\mathfrak{f} \cdot |\varphi_i(\mathrm{sub}_i(P))| \le \operatorname{poly}(d \cdot \log \Delta) \cdot \Gamma \cdot \mathrm{OPT} \le \operatorname{poly}(d \cdot \log \Delta) \cdot \mathrm{OPT}$.*

Proof: Let $\mathcal{C}$ be the collection of subsets guaranteed by Lemma III.7. Then

$$|\varphi_i(\mathrm{sub}_i(P))| \le \sum_{C \in \mathcal{C}} |\varphi_i(\mathrm{sub}_i(C)| \le |\mathcal{C}| \cdot \max_{C \in \mathcal{C}} |\varphi_i(\mathrm{sub}_i(C))|.$$

Since $\mathfrak{f} \cdot |\mathcal{C}| \le O(d \cdot \log \Delta) \cdot \mathrm{OPT}$, it suffices to prove that with probability at least $1 - 1/\operatorname{poly}(\Delta^d)$, $|\varphi_i(\mathrm{sub}_i(C))| \le \operatorname{poly}(d \cdot \log \Delta)$ for every $C \in \mathcal{C}$.

We wish to bound $|\varphi_i(\mathrm{sub}_i(C))|$ for C with "small" diameter and "large" diameter separately. For C with $\operatorname{diam}(C) \le \varepsilon$, by Definition I.6 and Lemma III.6, we have $|\varphi_i(C)| \le \operatorname{poly}(d)$, which implies that

$$|\varphi_i(\mathrm{sub}_i(C))| \le |\varphi_i(C)| \le \operatorname{poly}(d)\,.$$

For C with $\operatorname{diam}(C) > \varepsilon = \Theta(\mathfrak{f}/(2^i\Gamma))$, by Lemma III.7, we have $|C| \le O(\mathfrak{f}/\operatorname{diam}(C)) \le O(2^i \cdot \Gamma)$. By Fact III.4, with probability at least $1 - 1/\operatorname{poly}(\Delta^d)$, $|\mathrm{sub}_i(C)| \le O(d \cdot \log \Delta) \cdot \Gamma$. Finally, applying the union bound to all subsets $C \in \mathcal{C}$ with $\operatorname{diam}(C) > \varepsilon$ concludes the proof of Lemma III.8. ∎

The next lemma states that if the close neighborhood (at distance r_p) of any point p in P_i does not contain too many points after subsampling P, then for any subsampled point $p \in P_i$ there are not too many subsampled points mapped by φ_i into the same bucket as p.

Lemma III.9. *With probability at least $1 - 1/\operatorname{poly}(\Delta^d)$, for every point p such that $r_p \ge 2^{-i}\mathfrak{f}$, $|\varphi_i^{-1}(\varphi_i(p)) \cap \mathrm{sub}_i(P)| \le O(d \cdot \log \Delta)$.*

Proof: Observe that if $r_p \ge 2^{-i}\mathfrak{f}$, then $|\varphi_i^{-1}(\varphi_i(p)) \cap P| \le O(2^i)$. This is a consequence of having $\ell = 2^{-i}\mathfrak{f}/10$, and the fact that $|B(p, r_p/2) \cap P| \le O(2^i)$ (by Fact II.2). Hence, Lemma III.9 follows by applying Fact III.4 on the set $\varphi_i^{-1}(\varphi_i(p)) \cap P$. ∎

Proof of Lemma III.2: If $P_i = \emptyset$, then Algorithm 1 always returns an arbitrary point of P or $\perp$, and the guarantee of the lemma trivially holds. Now suppose $P_i \ne \emptyset$ and fix $x \in P_i$. Let $P' = P \backslash \{x\}$. Let $\mathcal{E}$ be the event that the following happens (over the randomness of sub_i).

1) $\mathfrak{f} \cdot |\varphi_i(\mathrm{sub}_i(P'))| \le \operatorname{poly}(d \cdot \log \Delta) \cdot \mathrm{OPT}$.

2) for every point p with $r_p \geq 2^{-i}\mathfrak{f}$, $|\varphi_i^{-1}(\varphi_i(p)) \cap \mathrm{sub}_i(P')| \leq O(d \cdot \log \Delta)$.

By Lemma III.8 and Lemma III.9, $\Pr[\mathcal{E}] \geq 1 - 1/\mathrm{poly}(\Delta^d)$. Moreover, $\mathcal{E}$ implies that

$$\mathfrak{f}\cdot|\varphi_i(\mathrm{sub}_i(P))| \leq \mathfrak{f}\cdot|\varphi_i(\mathrm{sub}_i(P'))|+\mathfrak{f} \leq \mathrm{poly}(d\cdot\log\Delta)\cdot\mathrm{OPT},$$

and for every point p with $r_p \geq 2^{-i}\mathfrak{f}$,

$$\begin{aligned}|\varphi_i^{-1}(\varphi_i(p)) \cap \mathrm{sub}_i(P)| &\leq |\varphi_i^{-1}(\varphi_i(p)) \cap \mathrm{sub}_i(P')| + 1 \\ &\leq O(d \cdot \log \Delta).\end{aligned}$$

Note that $\mathcal{E}$ is independent to whether or not x survives the subsampling. Let $A_x := \varphi_i^{-1}(\varphi_i(x)) \cap \mathrm{sub}_i(P)$ be the set of points that lie in the same bucket of φ_i as x. Suppose Algorithm 1 returns a random point p^*. Then a point $x \in P_i$ is sampled with probability

$$\begin{aligned}\Pr[p^* = x] &\geq \Pr[p^* = x \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] \\ &\geq \Omega(1) \cdot 2^{-i} \cdot \frac{1}{|\varphi_i(\mathrm{sub}_i(P))| \cdot |A_x|} \\ &\geq \Omega(1) \cdot \frac{1}{\mathrm{poly}(d \cdot \log \Delta)} \cdot 2^{-i} \cdot \frac{\mathfrak{f}}{\mathrm{OPT}}.\end{aligned}$$

This completes the proof of Lemma III.2. ■

IV. Future Directions

Improved Hash Functions: As mentioned in Remark I.7, it is an interesting open question to design an $O(d/\log d)$-gap $\mathrm{poly}(d)$-consistent hashing that is computable in $\mathrm{poly}(d)$ space and time. Note that this is the optimal gap bound as shown by Filtser [24].

New One-Pass Approach: Since the gap bound determines the approximation ratio of the one-pass algorithm, our framework cannot be directly used for a better than $O(d/\log d)$-approximation in one pass using $\mathrm{poly}(d \cdot \log \Delta)$ space. The main open problem is thus to design a one-pass streaming $O(1)$-approximation algorithm for Uniform Facility Location in high-dimensional Euclidean spaces.

Multiple Passes: It is also interesting to explore the power of multiple passes. In particular, is it possible to achieve $(1+\varepsilon)$-approximation using, e.g., $\mathrm{poly}(d\cdot\log\Delta)$ passes? Our lower bound works only for one pass, and it would be interesting to strengthen it to $O(1)$-passes using $\mathrm{poly}(d \cdot \log \Delta)$ space.

Acknowledgment

We are indebted to Arnold Filtser for telling us, very shortly after we announced our results (on the arXiv), about the definition of sparse partitions and the prior work in [23], [24].

References

[1] P. Indyk, "Algorithms for dynamic geometric problems over data streams," in *36th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2004, pp. 373–380.

[2] C. Lammersen and C. Sohler, "Facility location in dynamic geometric data streams," in *16th Annual European Symposium on Algorithms (ESA)*, 2008, pp. 660–671.

[3] A. Czumaj, C. Lammersen, M. Monemizadeh, and C. Sohler, "$(1+\varepsilon)$-approximation for facility location in data streams," in *24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2013, pp. 1710–1728.

[4] G. Frahling, P. Indyk, and C. Sohler, "Sampling in dynamic data streams and applications," *International Journal of Computational Geometry and Applications*, vol. 18, no. 1/2, pp. 3–28, 2008.

[5] G. Frahling and C. Sohler, "Coresets in dynamic geometric data streams," in *37th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2005, pp. 209–217.

[6] A. Andoni, K. D. Ba, P. Indyk, and D. P. Woodruff, "Efficient sketches for earth-mover distance, with applications," in *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2009, pp. 324–330.

[7] A. Czumaj, S. H. Jiang, R. Krauthgamer, and P. Veselý, "Streaming algorithms for geometric Steiner forest," in *49th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2022, pp. 47:1–47:20. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICALP.2022.47

[8] A. Andoni, P. Indyk, and R. Krauthgamer, "Earth mover distance over high-dimensional spaces," in *19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2008, pp. 343–352.

[9] X. Chen, R. Jayaram, A. Levi, and E. Waingarten, "New streaming algorithms for high dimensional EMD and MST," in *54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2022, pp. 222–233. [Online]. Available: https://doi.org/10.1145/3519935.3519979

[10] V. Braverman, G. Frahling, H. Lang, C. Sohler, and L. F. Yang, "Clustering high dimensional dynamic data streams," in *34th International Conference on Machine Learning (ICML)*, 2017, pp. 576–585.

[11] Z. Song, L. F. Yang, and P. Zhong, "Nearly optimal dynamic k-means clustering for high-dimensional data," *CoRR*, vol. abs/1802.00459, 2018.

[12] M. Bădoiu, A. Czumaj, P. Indyk, and C. Sohler, "Facility location in sublinear time," in *32nd International Colloquium on Automata, Languages, and Programming (ICALP)*, 2005, pp. 866–877.

[13] R. R. Mettu and C. G. Plaxton, "The online median problem," *SIAM Journal on Computing*, vol. 32, no. 3, pp. 816–832, 2003.

[14] G. Cormode and D. Firmani, "A unifying framework for ℓ_0-sampling algorithms," *Distributed Parallel Databases*, vol. 32, no. 3, pp. 315–335, 2014.

[15] R. M. Karp and M. Luby, "Monte-Carlo algorithms for enumeration and reliability problems," in *24th Annual Symposium on Foundations of Computer Science (FOCS)*, 1983, pp. 56–64.

[16] P. Indyk, "A near linear time constant factor approximation for Euclidean bichromatic matching (cost)," in *8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007, pp. 39–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=1283383.1283388

[17] A. A. Benczúr and D. R. Karger, "Randomized approximation schemes for cuts and flows in capacitated graphs," *SIAM Journal on Computing*, vol. 44, no. 2, pp. 290–319, 2015.

[18] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1913–1926, Dec. 2011.

[19] D. Feldman and M. Langberg, "A unified framework for approximating and clustering data," in *43rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2011, pp. 569–578.

[20] D. Feldman, M. Schmidt, and C. Sohler, "Turning big data into tiny data: Constant-size coresets for k-means, PCA, and projective clustering," *SIAM Journal on Computing*, vol. 49, no. 3, pp. 601–657, 2020.

[21] Y. Li and D. P. Woodruff, "On approximating functions of the singular values in a stream," in *48th Annual ACM symposium on Theory of Computing (STOC)*, 2016, pp. 726–739.

[22] V. Braverman, R. Krauthgamer, A. Krishnan, and R. Sinoff, "Schatten norms in matrix streams: Hello sparsity, goodbye dimension," in *37th International Conference on Machine Learning (ICML)*, 2020, pp. 1100–1110.

[23] L. Jia, G. Lin, G. Noubir, R. Rajaraman, and R. Sundaram, "Universal approximations for TSP, Steiner tree, and set cover," in *37th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2005, pp. 386–395.

[24] A. Filtser, "Scattering and sparse partitions, and their applications," in *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2020, pp. 47:1–47:20.

[25] O. Dunkelman, Z. Geyzel, C. Keller, N. Keller, E. Ronen, A. Shamir, and R. J. Tessler, "Error resilient space partitioning (invited talk)," in *48th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2021, pp. 4:1–4:22.

[26] S. Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," *Journal of the ACM*, vol. 45, no. 5, pp. 753–782, 1998.

[27] N. Linial and M. E. Saks, "Low diameter graph decompositions," *Combinatorica*, vol. 13, no. 4, pp. 441–454, 1993.

[28] Y. Bartal, "Probabilistic approximations of metric spaces and its algorithmic applications," in *37th Annual Symposium on Foundations of Computer Science (FOCS)*, 1996, pp. 184–193.

[29] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. A. Plotkin, "Approximating a finite metric by a small number of tree metrics," in *39th Annual Symposium on Foundations of Computer Science (FOCS)*, 1998, pp. 379–388.

[30] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *30th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 1998, pp. 604–613.

[31] N. Megiddo and K. J. Supowit, "On the complexity of some common geometric location problems," *SIAM Journal on Computing*, vol. 13, no. 1, pp. 182–196, 1984.

[32] S. Guha and S. Khuller, "Greedy strikes back: Improved facility location algorithms," *Journal of Algorithms*, vol. 31, no. 1, pp. 228–248, 1999.

[33] S. Li, "A 1.488 approximation algorithm for the uncapacitated facility location problem," *Information and Computation*, vol. 222, pp. 45–58, 2013.

[34] V. Cohen-Addad, A. E. Feldmann, and D. Saulpic, "Near-linear time approximation schemes for clustering in doubling metrics," *Journal of the ACM*, vol. 68, no. 6, pp. 44:1–44:34, 2021.

[35] V. Cohen-Addad, P. N. Klein, and C. Mathieu, "Local search yields approximation schemes for k-means and k-median in Euclidean and minor-free metrics," *SIAM Journal on Computing*, vol. 48, no. 2, pp. 644–667, 2019.

[36] S. G. Kolliopoulos and S. Rao, "A nearly linear-time approximation scheme for the Euclidean k-median problem," *SIAM Journal on Computing*, vol. 37, no. 3, pp. 757–782, 2007.

[37] V. Cohen-Addad, M. Pilipczuk, and M. Pilipczuk, "A polynomial-time approximation scheme for facility location on planar graphs," in *60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2019, pp. 560–581.

[38] A. Meyerson, "Online facility location," in *42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001, pp. 426–431.

[39] D. Fotakis, "On the competitive ratio for online facility location," *Algorithmica*, vol. 50, no. 1, pp. 1–57, 2008.

[40] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[41] U. Feige and R. Krauthgamer, "Stereoscopic families of permutations, and their applications," in *5th Israel Symposium on the Theory of Computing and Systems (ISTCS)*, 1997, pp. 85–95.

[42] T. M. Chan, S. Har-Peled, and M. Jones, "On locality-sensitive orderings and their applications," *SIAM Journal on Computing*, vol. 49, no. 3, pp. 583–600, 2020.

[43] G. Kindler, R. O'Donnell, A. Rao, and A. Wigderson, "Spherical cubes and rounding in high dimensions," in *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2008, pp. 189–198.

[44] M. Kapralov and D. P. Woodruff, "Spanners and sparsifiers in dynamic streams," in *33rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2014, pp. 272–281.

[45] A. Filtser, M. Kapralov, and N. Nouri, "Graph spanners by sketching in dynamic streams and the simultaneous communication model," in *32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2021, pp. 1894–1913. [Online]. Available: https://doi.org/10.1137/1.9781611976465.113

[46] C. Konrad and K. K. Naidu, "On two-pass streaming algorithms for maximum bipartite matching," in *APPROX-RANDOM*, 2021, pp. 19:1–19:18.

[47] G. Cormode and H. Jowhari, "A second look at counting triangles in graph streams (corrected)," *Theoretical Computer Science*, vol. 683, pp. 22–30, 2017.

[48] V. Braverman, S. R. Chestnut, R. Krauthgamer, Y. Li, D. P. Woodruff, and L. F. Yang, "Matrix norms in data streams: Faster, multi-pass and row-order," in *35th International Conference on Machine Learning (ICML)*, 2018, pp. 648–657.

[49] S. Assadi, "Tight space-approximation tradeoff for the multi-pass streaming set cover problem," in *36th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2017, pp. 321–335.

[50] M. Kapralov, S. Khanna, and M. Sudan, "Approximating matching size from random streams," in *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014, pp. 734–751.

[51] S. Assadi and S. Behnezhad, "Beating two-thirds for random-order streaming matching," in *48th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2021, pp. 19:1–19:13.

[52] S. Guha and A. McGregor, "Stream order and order statistics: Quantile estimation in random-order streams," *SIAM Journal on Computing*, vol. 38, no. 5, pp. 2044–2059, 2009.

[53] P. Peng and C. Sohler, "Estimating graph parameters from random order streams," in *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2018, pp. 2449–2466.

[54] D. P. Woodruff and S. Zhou, "Separations for estimating large frequency moments on data streams," in *48th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2021, pp. 112:1–112:21.

[55] A. Czumaj, S. H. Jiang, R. Krauthgamer, P. Veselý, and M. Yang, "Streaming facility location in high dimension via geometric hashing," *arXiv*, vol. abs/2204.02095, 2022.

[56] N. Nisan, "Pseudorandom generators for space-bounded computation," *Combinatorica*, vol. 12, no. 4, pp. 449–461, 1992. [Online]. Available: https://doi.org/10.1007/BF01305237

[57] A. Andoni, R. Krauthgamer, and K. Onak, "Streaming algorithms via precision sampling," in *52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2011, pp. 363–372.

[58] G. Cormode and S. Muthukrishnan, "Space efficient mining of multigraph streams," in *24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2005, p. 271–282.

[59] T. S. Jayram and D. P. Woodruff, "The data stream space complexity of cascaded norms," in *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2009, pp. 765–774.

[60] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2010, pp. 41–52.