Streaming Euclidean Max-Cut: Dimension vs Data Reduction*

Xiaoyu Chen
yuchen21@stu.pku.edu.cn
Peking University
China

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

Robert Krauthgamer‡
rober.krauthgamer@weizmann.ac.il
Weizmann Institute of Science
Israel

ABSTRACT

Max-Cut is a fundamental problem that has been studied extensively in various settings. We design an algorithm for Euclidean Max-Cut, where the input is a set of points in $\mathbb{R}^d$, in the model of dynamic geometric streams, where the input $X \subseteq [\Delta]^d$ is presented as a sequence of point insertions and deletions. Previously, Frahling and Sohler [STOC 2005] designed a $(1+\varepsilon)$-approximation algorithm for the low-dimensional regime, i.e., it uses space $\exp(d)$.

To tackle this problem in the high-dimensional regime, which is of growing interest, one must improve the dependence on the dimension d, ideally to space complexity $\mathrm{poly}(\varepsilon^{-1} d \log \Delta)$. Lammersen, Sidiropoulos, and Sohler [WADS 2009] proved that Euclidean Max-Cut admits dimension reduction with target dimension $d' = \mathrm{poly}(\varepsilon^{-1})$. Combining this with the aforementioned algorithm that uses space $\exp(d')$, they obtain an algorithm whose overall space complexity is indeed polynomial in d, but unfortunately exponential in ε^{-1}.

We devise an alternative approach of *data reduction*, based on importance sampling, and achieve space bound $\mathrm{poly}(\varepsilon^{-1} d \log \Delta)$, which is exponentially better (in ε) than the dimension-reduction approach. To implement this scheme in the streaming model, we employ a randomly-shifted quadtree to construct a tree embedding. While this is a well-known method, a key feature of our algorithm is that the embedding's distortion $O(d \log \Delta)$ affects only the space complexity, and the approximation ratio remains $1 + \varepsilon$.

CCS CONCEPTS

• **Theory of computation → Streaming, sublinear and near linear time algorithms**.

KEYWORDS

max cut, streaming, data reduction, dimension reduction

*Full version of the paper is available at arXiv:2211.05293.

†Research partially supported by a national key R&D program of China No. 2021YFA1000900, and a startup fund from Peking University.

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

STOC '23, June 20–23, 2023, Orlando, FL, USA

ACM ISBN 978-1-4503-9913-5/23/06…$15.00
https://doi.org/10.1145/3564246.3585170

ACM Reference Format:
Xiaoyu Chen, Shaofeng H.-C. Jiang, and Robert Krauthgamer. 2023. Streaming Euclidean Max-Cut: Dimension vs Data Reduction. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3564246.3585170

1 INTRODUCTION

Max-Cut is a fundamental problem in multiple domains, from constraint satisfaction (CSP) and linear equations to clustering. It was studied extensively in many computational models and for types of inputs, and many (nearly) tight bounds were obtained, oftentimes leading the way to even more general problems. For instance, in the offline setting, Max-Cut admits a polynomial-time 0.878-approximation for general graphs [20], and this approximation factor is tight under the Unique Games Conjecture [31]. In contrast, if the input is a dense unweighted graph, or a metric space (viewed as a weighted graph), then a PTAS exists [14, 15]. In the graph-streaming setting, $(1+\varepsilon)$-approximation can be obtained using $\tilde{O}(n)$ space [3], and this space bound is tight [30].

However, the streaming complexity of Max-Cut is only partially resolved in the geometric setting, i.e., for Euclidean points. A known algorithm, due to Frahling and Sohler [19], achieves $(1+\varepsilon)$-approximation but uses space $\exp(d)$, which is prohibitive when the dimension is high. Combining this algorithm with a dimension reduction result, based on the Johnson-Lindenstrauss Lemma but specialized to Max-Cut and has target dimension $\mathrm{poly}(\varepsilon^{-1})$ [34, 35], one can achieve polynomial dependence on d, but at the expense of introducing to the space complexity an undesirable $\exp(\mathrm{poly}(\varepsilon^{-1}))$-factor. It was left open to obtain in the high-dimension regime space complexity that is truly efficient, i.e., $\mathrm{poly}(\varepsilon^{-1} d)$.

We answer this question by providing the first streaming algorithms that achieve $(1+\varepsilon)$-approximation for Max-Cut using space $\mathrm{poly}(d\varepsilon^{-1})$. We consider the setting of *dynamic geometric streams*, introduced by Indyk [24], where the input is a dataset $X \subseteq [\Delta]^d$ that is presented as a stream of point insertions and deletions. The goal of the algorithm is to approximate (multiplicatively) the so-called Max-Cut value, defined as

$$\textsc{Max-Cut}(X) := \max_{S \subseteq X} \sum_{x \in S, y \in X \setminus S} \|x - y\|_2$$

(see Section 2 for general metric spaces). We say that $\textsc{Max-Cut}(X)$ is *α-approximated*, for $\alpha \geq 1$, by a value $\eta \geq 0$ if $\textsc{Max-Cut}(X)/\alpha \leq \eta \leq \textsc{Max-Cut}(X)$.[1] We assume throughout that X contains distinct points (and is not a multiset), hence $n := |X| \leq |\Delta|^d$. In the *high-dimension regime*, algorithms can use at most $\mathrm{poly}(d \log \Delta)$ bits of space, which is polynomial in the number of bits required to

[1] We will actually aim at $\eta \in (1 \pm \varepsilon) \cdot \textsc{Max-Cut}(X)$, for $0 < \varepsilon < 1/2$, which can be scaled to achieve $(1 + O(\varepsilon))$-approximation.

represent a point in $[\Delta]^d$, and also allows counting to $n \le |\Delta|^d$. In the *low-dimension regime*, algorithms may have bigger space complexity, e.g., exponential in d.

A central challenge in the area of geometric streaming algorithms is to achieve good accuracy (approximation) in the high-dimension regime. Indeed, algorithms for many basic problems (like diameter, minimum spanning tree, facility location, and Max-Cut), achieve good approximation, say for concreteness $O(1)$ or even $1+\varepsilon$, using space that is exponential in d [1, 11, 13, 18, 19, 36, 43]. In contrast, algorithms that use space polynomial in d are fewer and they typically achieve far worse approximation ratio [10, 12, 24, 42], and obtaining $O(1)$-approximation remains open. In particular, Indyk [24] tackled the high-dimension regime using a technique of randomized tree embedding, which is rather general and economical in space, but unfortunately distorts distances by a factor of $O(d \log \Delta)$ that goes directly into the approximation ratio. Attempts to improve the approximation ratio had only limited success so far; for example, the algorithm of [2] (for diameter) works only in insertion-only streams, and the algorithms of [10, 12] (for MST and for facility location) fall short of the desired $O(1)$-approximation in one pass.

1.1 Our Results

We bypass the limitation of dimension reduction via a data reduction approach, and design a streaming algorithm that $(1+\varepsilon)$-approximates Max-Cut using $\mathrm{poly}(\varepsilon^{-1} d \log \Delta)$ space, thereby closing the gap of high dimension (for Max-Cut). Our approach works not only under Euclidean norm, but also when distances are calculated using ℓ_p norm, for $p \ge 1$.

Data Reduction via Importance Sampling. We present an algorithm that is based on the data-reduction approach, namely, it uses the dataset X to construct a small instance X' that has a similar Max-Cut value, then solve Max-Cut on it optimally and report this value Max-Cut(X'). Following a common paradigm, X' is actually a re-weighted subset of X, that is picked by non-uniform sampling from X, known as importance sampling.

Theorem 1.1 (Streaming Max-Cut in ℓ_p Norm)**.** *There is a randomized streaming algorithm that, given $0 < \varepsilon < 1/2$, $p \ge 1$, integers $\Delta, d \ge 1$, and an input dataset $X \subseteq [\Delta]^d$ presented as a dynamic stream, uses space $\mathrm{poly}(\varepsilon^{-1} d \log \Delta)$ and reports an estimate $\eta > 0$ that with probability at least $2/3$ is a $(1+\varepsilon)$-approximation to Max-Cut(X) in ℓ_p.*

This data-reduction approach was previously used for several clustering problems. For k-median and related problems, such an instance X' is often called a coreset, and there are many constructions, see e.g. [8, 16, 17, 19, 22, 23]. Earlier work [41] has designed importance-sampling based algorithms for a problem closely related to Max-Cut, but did not provide a guarantee that X and X' have a similar Max-Cut value (see Section 1.3 for a more detailed discussion). Recently, importance sampling was used to design streaming algorithms for facility location in high dimension [12], although their sample X' is not an instance of facility location. Overall, this prior work is not useful for us, and we have to devise our own sampling distribution, prove that it preserves the Max-Cut value, and design a streaming algorithm that samples from this distribution.

The approximation ratio $1+\varepsilon$ in Theorem 1.1 is essentially the best one can hope for using small space, because finding the Max-Cut value exactly, even in one dimension, requires $\Omega(\Delta)$ space, as shown in Claim A.3. Compared with the dimension-reduction approach (based on [19]), our Theorem 1.1 has the advantage that it works for all ℓ_p norms ($p \ge 1$) and not only ℓ_2. The result of [19] is stronger in another aspect, of providing a "cut oracle", i.e., an implicit representation of the approximately optimal cut that can answer the side of the cut that each data point $x \in X$ (given as a query) belongs to. This feature extends also to high dimension, as it is easy to combine with the dimension reduction. For completeness, we give a (somewhat simplified) proof of the dimension reduction in Section B, followed by a formal statement of this "cut oracle" in Corollary B.3. It remains open to design a streaming algorithm that computes such an implicit representation using space $\mathrm{poly}(\varepsilon^{-1} d \log \Delta)$.

1.2 Technical Overview

In order to estimate Max-Cut using importance sampling, we must first identify a sampling distribution for which Max-Cut(X') indeed approximates Max-Cut(X), and then we have to design a streaming algorithm that samples from this distribution.

Sampling Probability. One indication that geometric Max-Cut admits data reduction by sampling comes from the setting of dense unweighted graphs, where it is known that Max-Cut can be $(1+\varepsilon)$-approximated using a *uniform sample* of $O(\varepsilon^{-4})$ vertices, namely, by taking the Max-Cut value in the induced subgraph and scaling appropriately [4, 40] (improving over [21]). However, sampling points uniformly clearly cannot work in the metric case – if a point set X has many co-located points and a few distant points that are also far from each other, then uniform sampling from X is unlikely to pick the distant points, which have a crucial contribution to the Max-Cut value. It is therefore natural to employ importance sampling, i.e., sample each point with probability proportional to its contribution. The contribution of a point x to the Max-Cut value is difficult to gauge, but we can use instead a simple proxy – its total distance to all other points $q(x) := \sum_{y \in X} \mathrm{dist}(x,y)$, which is just its contribution to twice the total edge weight $\sum_{x,y \in X} \mathrm{dist}(x,y)$. For any fixed cut in X, this sampling works well and the estimate will (likely) have additive error $\varepsilon \sum_{x,y \in X} \mathrm{dist}(x,y) = \Theta(\varepsilon) \cdot$ Max-Cut(X). While the analysis is straightforward for a fixed cut, say a maximum one, proving that the sampling preserves the Max-Cut value is much more challenging, as one has to argue about all possible cuts.

We show in Theorem 3.1 that $O(\varepsilon^{-4})$ independent samples generated with probability proportional to $q(x)$ preserve the Max-Cut value. This holds even if the sampling probabilities are dampened by a factor $\lambda \ge 1$, at the cost of increasing the number of samples by a $\mathrm{poly}(\lambda)$ factor. To be more precise, it suffices to sample from any probability distribution $\{p_x : x \in X\}$, where $p(x) \ge \frac{1}{\lambda} \frac{q(x)}{\sum_{y \in X} q(y)}$ for all $x \in X$. A small technicality is that we require the sampling procedure to report a random sample x^* together with its corresponding $p(x^*)$, in order to re-weight the sample x^* by factor $1/p(x^*)$, but this extra information can often be obtained using the same methods.

This sampling distribution, i.e., probabilities proportional to $\{q(x) : x \in X\}$, can be traced to two prior works. Schulman [41] used essentially the same probabilities, but his analysis works only for one fixed cut, and extends to a multiple cuts by a union bound.[2] The exact same probabilities were also used by [15] as weights to convert a metric instance to a dense unweighted graph, and thereby obtain a PTAS (without sampling or any data reduction).

In fact, our proof combines several observations from [15] about a uniform sample of $O(\varepsilon^{-4})$ vertices in unweighted graphs [4, 40]. In a nutshell, we relate sampling from X proportionally to $\{q(x) : x \in X\}$ with sampling uniformly from a certain dense unweighted graph, whose cut values corresponds to those in X, and we thereby derive a bound on the Max-Cut value of the sample X'.

Streaming Implementation. Designing a procedure to sample proportionally to $\{q(x) : x \in X\}$, when X is presented as a stream, is much more challenging and is our main technical contribution (Lemma 4.2). The main difficulty is that standard tools for sampling from a stream, such as ℓ_p-samplers [25, 28, 39], are based on the frequency vector and oblivious to the geometric structure of X. Indeed, the literature lacks geometric samplers, which can be very useful when designing geometric streaming algorithms. Our goal of sampling proportionally to $q(x)$, which is the total distance to all other points in X, seems like a fundamental geometric primitive, and therefore our sampler (Lemma 4.2) is a significant addition to the geometric-streaming toolbox, and may be of independent interest.

High-Level Picture. At a high level, our sampling procedure is based on a randomly-shifted quadtree T that is defined on the entire input domain $[\Delta]^d$ and captures its geometry. This randomly-shifted quadtree T is data oblivious, and thus can be picked (constructed implicitly) even before the stream starts (as an initialization step), using space $O(\text{poly}(d \log \Delta))$. This technique was introduced by Indyk [24], who noted that the quadtree essentially defines a tree embedding with expected distortion of distances $O(d \log \Delta)$. Unfortunately, the distortion is fundamental to this approach, and directly affects the accuracy (namely, goes into the approximation ratio) of streaming algorithms that use this technique [10, 24].[3] While our approach still suffers this distortion, we can avoid its effect on the accuracy; instead, it affects the importance-sampling distribution, namely, the dampening factor λ grows by factor $O(d \log \Delta)$, which can be compensated by drawing more samples. This increases the space complexity moderately, which we can afford, and overall leads to $(1+\varepsilon)$-approximation using space $\text{poly}(\varepsilon^{-1} d \log \Delta)$.

Comparison to Other Sampling Approaches. A different importance sampling method was recently designed in [12] for facility location in high dimension. Their sampling procedure relies on a geometric hashing (space partitioning), and completely avoids a quadtree.[4] Our sampling technique may be more easily applicable to other problems, as it uses a more standard and general tool of a randomly-shifted quadtree. Another importance-sampling method was recently designed in [37] in the context of matrix streams. Geometrically, their importance-sampling can be viewed as picking a point (row in the matrix) proportionally to its length, but after the points are projected, at query time, onto a subspace. This sampling procedure is very effective for linear-algebraic problems, like column-subset selection and subspace approximation, which can be viewed also as geometric problems.

Previously, quadtrees were employed in geometric sampling results, but mostly a fixed quadtree and not a randomly-shifted one, and to perform *uniform sampling* over its non-empty squares at each level, as opposed to our importance sampling [11, 18, 19]. Furthermore, in [18], uniform sampling was augmented to report also all the points nearby the sampled points, and this was used to estimate the number of connected components in a metric threshold graph.

Sampling on Quadtree: Bypassing the Distortion. Finally, we discuss how to perform the importance sampling, based on a randomly-shifted quadtree T, but without the $O(d \log \Delta)$ distortion going into the approximation ratio. As explained above, our importance sampling (Theorem 3.1) requires that every point $x \in X$ is sampled with some probability $p(x) \geq \frac{1}{\lambda} \frac{q(x)}{Q}$ for $Q := \sum_{y \in X} q(y)$, and the dampening factor λ can be up to $O(\text{poly}(d \log \Delta))$. We will exploit the fact that there is no upper bound on the probability $p(x)$. As usual, a randomly-shifted quadtree T is obtained by recursively partitioning of the grid $[2\Delta]^d$, each time into 2^d equal-size grids, which we call squares, and building a tree whose nodes are all these squares, and every square is connected to its parent by an edge whose weight is equal to that square's Euclidean diameter. Furthermore, this entire partitioning is shifted by a uniformly random $v_{\text{shift}} \in [\Delta]^d$. (See Section 4.1 for details.) The key is that every grid point $x \in [\Delta]^d$ is represented by a tree leaf, and thus T defines distances on $[\Delta]^d$, also called a tree embedding. Define q_T and Q_T analogously to q and Q, but using the *tree distance*, that is, $q_T(x)$ is the total tree distance from x to all other points in X, and $Q_T := \sum_{y \in X} q_T(y)$.

The tree-embedding guarantees, for all $x \in [\Delta]^d$, that $q_T(x)$ overestimates $q(x)$ (with probability 1), and that $\mathbb{E}_T[q_T(x)] \leq O(d \log \Delta) \cdot q(x)$. It thus suffices to have one event, $Q_T \leq O(d \log \Delta) \cdot Q$, which happens with constant probability by Markov's inequality, and then we would have $\frac{q_T(x)}{Q_T} \geq \frac{1}{O(d \log \Delta)} \cdot \frac{q(x)}{Q}$ simultaneously for all $x \in X$. As mentioned earlier, the algorithm picks (constructs implicitly) T even before the stream starts, and the analysis assumes the event mentioned above happens. In fact, the rest of the sampling procedure works correctly for arbitrary T, i.e., regardless of that event. We stress, however, that our final algorithm actually needs to generate multiple samples that are picked independently from the same distribution, and this is easily achieved by parallel executions that share the same random tree T but use independent coins in all later steps. We thus view this T as a preprocessing step that is run only once, particularly when we refer to samples as independent.

A Two-level Sampling by Using Tree Structure. The remaining challenge is to sample (in a streaming fashion) with probability proportional to q_T for a fixed quadtree T. To this end, we make heavy use of the structure of the quadtree. In particular, the quadtree

[2] We gloss over slight technical differences, e.g., he deals with squared Euclidean distances, and his sampling and re-weighting processes are slightly different.

[3] A randomly-shifted quadtree was used also in Arora's approximation algorithm for TSP [6], but as the basis for dynamic programming rather than as a tree embedding, and similarly in streaming applications of this technique [11].

[4] A key parameter in that hashing, called consistency, turns out to be $\text{poly}(d)$, and unfortunately affects also the final approximation ratio for facility location, and not only the importance-sampling parameter λ (and thus the space).

T has $O(d \log \Delta)$ levels, and every data point $x \in X$ forms a distinct leaf in T. Clearly, each internal node in T represents a subset of X (all its descendants in T, that is, the points of X inside its square). At each level i (the root node has level 1), we identify a level-i *heavy node* h_i that contains the maximum number of points from X (breaking ties arbitrarily). We further identify a *critical level* k, such that $h_1, \ldots, h_k$ (viewed as subsets of X), all contain more than $0.5|X|$ points, but h_{k+1} does not. This clearly ensures that $h_1, \ldots, h_k$ forms a path. Let $X(h_i) \subseteq X$ denote the subset of X represented by node h_i. The tree structure and the path property of $(h_1, \ldots, h_k)$ guarantee that, for each $i < k$, all points $x \in X(h_i) \setminus X(h_{i+1})$ have roughly the same $q_T(x)$ values. For the boundary case $i = k$, a similar claim holds for $x \in X(h_k)$. Hence, a natural algorithm is to employ two-level sampling: first draw a level $i^* \le k$, then draw a uniform sample from $X(h_{i^*}) \setminus X(h_{i^*+1})$. The distribution of sampling i^* also requires a careful design, but we omit this from the current overview, and focus instead on how to draw a uniform sample from $X(h_i) \setminus X(h_{i+1})$.

Unfortunately, sampling from $X(h_i) \setminus X(h_{i+1})$ still requires $\Omega(\Delta)$ space in the streaming model, even for $d = 1$. (We prove this reduction from INDEX in Claim A.2.) In fact, it is even hard to determine whether $X(h_i) \setminus X(h_{i+1}) = \emptyset$; the difficulty is that h_i is not known in advance, otherwise it would be easy. We therefore relax the two-level sampling, and replace sampling from $X(h_i) \setminus X(h_{i+1})$, with its superset $X \setminus X(h_{i+1})$. This certainly biases the sampling probability, in fact some probabilities might change by an unbounded factor (Remark 4.12), nevertheless we prove that the increase in the dampening parameter λ is bounded by an $O(\log \Delta)$ factor (Lemma 4.13), which we can still afford. This part crucially uses the tree and the path property of $(h_1, \ldots, h_k)$.

Sampling from Light Parts. The final remaining step is to sample from $X \setminus X(h_i)$ for a given $i \le k$ (Lemma 4.14). We can assume here $X(h_i)$ contains more than $0.5|X|$ points, and thus $X \setminus X(h_i)$ is indeed the "light" part, containing few (and possibly no) points. To let the light points "stand out" in a sampling, we hash the *tree nodes* (instead of the points) randomly into two buckets. Since $|X(h(i))| > 0.5|X|$, the heavy node h_i always lies in the bucket that contains more points, and we therefore sample only from the light bucket. (To implement this in streaming, we actually generate two samples, one from each bucket, and in parallel estimate the two buckets' sizes to know which of the two samples to use.) Typically, the light bucket contains at least half of the points from $X \setminus X(h_i)$, which is enough. Overall, this yields a sampling procedure that uses space $\mathrm{poly}(\varepsilon^{-1} d \log \Delta)$.

1.3 Related Work

Geometric streaming in the low-dimension regime was studied much more extensively than in the high-dimensional case that we investigate here. In [19], apart from the $O(\varepsilon^{-d} \operatorname{poly} \log \Delta)$-space $(1+\varepsilon)$-approximation for Max-Cut that we already mentioned, similar results were obtained also for k-median, maximum spanning tree, maximum matching and similar maximization problems. For minimum spanning tree, a $(1+\varepsilon)$-approximation $O((\varepsilon^{-1} \log \Delta)^d)$-space algorithm was devised in [18], alongside several useful techniques for geometric sampling in low dimension. In [5], an $O(\varepsilon^{-1})$-approximation $\tilde{O}(\Delta^{\varepsilon})$-space streaming algorithm was obtained for computing earth-mover distance in dimension $d = 2$. For facility location in dimension $d = 2$, a $(1+\varepsilon)$-approximation $\mathrm{poly}(\varepsilon^{-1} \log \Delta)$-space algorithm was designed in [13]. Recently, Steiner forest (a generalization of Steiner tree, which asks to find a minimum-weight graph that connects k groups of points), was studied in [12] for dimension $d = 2$, and they obtain $O(1)$-approximation using space $(\mathrm{poly}(k \log \Delta))$.

Our sampling distribution may seem reminiscent of an importance sampling procedure devised by Schulman [41], however his results and techniques are not useful in our context. First, the problem formulation differs, as it asks to approximate the complement objective of sum of distances inside each cluster (which is only stronger than our MAX-CUT objective), but the objective function sums squared Euclidean (rather than Euclidean) distances. Second, his algorithm is for the offline setting and is not directly applicable in streaming. Third, his analysis does not provide a guarantee on Max-Cut(X'), but rather only on certain cuts of X' (not all of them), and his approximation guarantee includes a non-standard twist.

2 PRELIMINARIES

Consider a metric space (V, dist). The Euclidean case is $V = \mathbb{R}^d$ and $\mathrm{dist} = \ell_2$, and the ℓ_p case is $V = \mathbb{R}^d$ and $\mathrm{dist} = \ell_p$. For $X \subseteq V$, the cut function $\mathrm{cut}_X : 2^X \to \mathbb{R}$ is defined as $\mathrm{cut}_X(S) := \sum_{x \in S, y \in X \setminus S} \mathrm{dist}(x, y)$. The Max-Cut value of a dataset $X \subseteq V$ is defined as

$$\textsc{Max-Cut}(X) := \max_{S \subseteq X} \mathrm{cut}_X(S).$$

We shall use the following standard tools as building blocks in our algorithms. Recall that a turnstile (or dynamic) stream can contain insertions and deletions of items from some domain $[N]$, and it naturally defines a frequency vector $x \in \mathbb{R}^N$, where every possible item has a coordinate that counts its net frequency, i.e., insertions minus deletions. In general, this model allows frequencies to be negative. However, in our setting where $X \subset [\Delta]^d$ is presented as a dynamic geometric stream, then frequency vector has Δ^d coordinates all in the range $\{0, 1\}$, and is just the incidence vector of X.

Lemma 2.1 (ℓ_0-Norm Estimator [29])**.** *There exists a streaming algorithm that, given $0 < \varepsilon, \delta < 1$, integers $N, M \ge 1$, and a frequency vector $x \in [-M, M]^N$ presented as a turnstile stream, where we denote its support by $X := \{i \in [N] : x_i \neq 0\}$, uses space $\mathrm{poly}(\varepsilon^{-1} \log(\delta^{-1} MN))$ to return $r^* \ge 0$, such that $\Pr[r^* \in (1 \pm \varepsilon)|X|] \ge 1 - \delta$.*

Lemma 2.2 (ℓ_0-Sampler [28])**.** *There exists a streaming algorithm that, given $0 < \delta < 1$, integers $N, M \ge 1$, and a frequency vector $x \in [-M, M]^N$ presented as a turnstile stream, where we assume that its support $X := \{i \in [N] : x_i \neq 0\}$ is non-empty, uses space $\operatorname{poly} \log(\delta^{-1} MN)$, to return a sample $i^* \in X \cup \{\perp\}$, such that with probability at least $1 - \delta$,*

$$\forall i \in X, \qquad \Pr[i^* = i] = \frac{1}{|X|}.$$

3 APPROXIMATING MAX-CUT BY IMPORTANCE SAMPLING

In this section, we consider a general metric space (V, dist) (which includes Euclidean spaces by setting $V = \mathbb{R}^d$ and $\mathrm{dist} = \ell_2$), and show that a small importance-sample S on a dataset $X \subseteq V$ may be used to estimate Max-Cut(X) by simply computing Max-Cut(S).

Point-weighted Set [15]. Since we apply importance sampling (as opposed to using a uniform probability for every point), the sampled points need to be re-weighted so that the estimation is unbiased. Hence, we consider the notion of *point-weighted* sets. This notion was first considered in [15] to reduce the metric Max-Cut to Max-Cut in (dense) weighted graphs. Specifically, a point-weighted set $S \subseteq V$ is a subset of V that is associated with a point-weight function $w_S : S \to \mathbb{R}_+$. For a point-weighted set S, the distance $\mathrm{dist}_S(x, y)$ between $x, y \in S$ is also re-weighted such that $\mathrm{dist}_S(x, y) := \frac{\mathrm{dist}(x,y)}{w_S(x) w_S(y)}$. Under this weighting, when an edge $\{x, y\}$ appears in a cut, its contribution is still accounted as $w_S(x) \cdot w_S(y) \cdot \mathrm{dist}_S(x, y) = \mathrm{dist}(x, y)$.

We prove (in Theorem 3.1) that for every dataset $X \subseteq V$, if one construct a point-weighted subset $S \subseteq X$ by drawing i.i.d. samples from a distribution on X, where each $x \in X$ is sampled proportional to $q(x) = \sum_{y \in X} \mathrm{dist}(x, y)$ which is the sum of distances to other points in X, up to an error factor of λ, then Max-Cut(S) is $(1+\varepsilon)$-approximation to Max-Cut(X) with high probability.

Theorem 3.1. *Given $\varepsilon, \delta > 0, \lambda \geq 1$, metric space (V, dist) and dataset $X \subseteq V$, let $\mathcal{D}$ be a distribution $(p_x : x \in X)$ on X such that $\forall x \in X, p_x \geq \frac{1}{\lambda} \cdot \frac{q(x)}{Q}$, where $q(x) = \sum_{y \in X} \mathrm{dist}(x, y)$ and $Q = \sum_{x \in X} q(x)$. Let S be a point-weighted set that is obtained by an i.i.d. sample of $m \geq 2$ points from $\mathcal{D}$, weighted by $w_S(x) := \hat{p}_x$ such that $p_x \leq \hat{p}_x \leq (1+\varepsilon) \cdot p_x$. If $m \geq O(\varepsilon^{-4} \lambda^{-8})$, then with probability at least 0.9, the value $\frac{\text{Max-Cut}(S)}{m^2}$ is a $(1+\varepsilon)$-approximation to Max-Cut(X).*

The $O(\varepsilon^{-4})$ dependence on ε of Theorem 3.1 matches a similar $O(\varepsilon^{-4})$ sampling complexity bound for unweighted graphs in [4, 40][5]. To the best of our knowledge, this $O(\varepsilon^{-4})$ is the state-of-the-art even for the case of unweighted graphs. Although our proof of Theorem 3.1 is obtained mostly by using the bound in [4, 40] as a black box, the generalization to metric spaces, as well as the allowance of λ which is the error in sampling probability, is new.

3.1 Proof of Theorem 3.1

As mentioned, the plan is to apply the sampling bound proved in [4, 40], which we restate in Lemma 3.2. In fact, the original statement in [4, 40] was only made for unweighted graphs, i.e., edge weights in $\{0, 1\}$. However, we observe that only the fact that the edges weights are between $[0, 1]$ was used in their proof. Hence, in our statement of Lemma 3.2 we make this stronger claim of $[0, 1]$ edge weights.

Here, for a graph $G(V, E)$ with weight function $\mathrm{len}_G : E \to \mathbb{R}$ ($\mathrm{len}_G(\cdot) = 1$ for unweighted graphs), we define the cut function for $S \subseteq X \subseteq V$ (as well as Max-Cut) similarly, as $\mathrm{cut}_X(S) = \sum_{x \in S, y \in X \setminus S : \{x,y\} \in E} \mathrm{len}_G(x, y)$.

Lemma 3.2 ([4, 40]). *Consider a weighted graph $G(V, E)$ with weights in $[0, 1]$. Let $D \subseteq V$ be a uniformly independent sample from V (possibly with repetitions) of $O(\varepsilon^{-4})$ points. Then with probability at least 0.9,*

$$\left| \frac{1}{|D|^2} \textit{Max-Cut}(D) - \frac{1}{|V|^2} \textit{Max-Cut}(V) \right| \leq \varepsilon.$$

Hence, our plan is to define an auxiliary graph G' whose edge weights are in $[0, 1]$, such that our importance sampling may be interpreted as a uniform sampling from vertices in G'. Eventually, our sampling bound would follow from Lemma 3.2.

Defining Auxiliary Graph. Since we focus on approximate solutions, we can assume that p_x's ($x \in X$) are of finite precision. Then, let N be a sufficiently large number such that for all $x \in X$, Np_x is an integer. We define an auxiliary graph $G'(X', E' := X' \times X')$, such that X' is formed by copying each point $x \in X$ for Np_x times, and edge weight $\mathrm{len}_{G'}(x, y) := \frac{1}{4\lambda^2 Q} \cdot \frac{\mathrm{dist}(x,y)}{\hat{p}_x \hat{p}_y}$. Clearly, if we let x^* be a uniform sample from X', then for every $x \in X$, $\Pr[x^* = x] = p_x$. Hence, this uniform sample x^* is identically distributed as an importance-sample from distribution $\mathcal{D}$ on X. Furthermore, for $x, y \in S$, it holds that

$$\mathrm{dist}_S(x, y) = \frac{\mathrm{dist}(x, y)}{\hat{p}_x \hat{p}_y} = 4\lambda^2 Q \cdot \mathrm{len}_{G'}(x, y).$$

Hence, we conclude the following fact.

Fact 3.3. *Let $S' \subseteq X'$ be m uniform samples from X'. Then, the value $4\lambda^2 Q \cdot$ Max-Cut(S') and Max-Cut(S) are identically distributed.*

Therefore, it suffices to show the S' from Fact 3.3 satisfies that $4\lambda^2 Q \cdot \frac{\text{Max-Cut}(S')}{|S'|^2}$ is a $(1+\varepsilon)$-approximation to Max-Cut(X), with constant probability. Our plan is to apply Lemma 3.2, but we first need to show, in Lemma 3.4, that the edge weights of G' are in $[0, 1]$, and in Lemma 3.6 that Max-Cut(X') is a $(1+\varepsilon)$-approximation to Max-Cut(X) up to a scaling.

Lemma 3.4. *For all $x, y \in X'$, $\mathrm{len}_{G'}(x, y) \leq 1$.*

Proof. We need the following fact from [15] (which was proved in [15, Lemma 7]).

Lemma 3.5 ([15]). *For all $x \in X$, it holds that*

$$\frac{\mathrm{dist}(x, y)}{q(x) q(x)} \leq \frac{4}{Q}.$$

Applying Lemma 3.5,

$$\mathrm{len}_{G'}(x, y) = \frac{1}{4\lambda^2 Q} \cdot \frac{\mathrm{dist}(x, y)}{\hat{p}_x \hat{p}_y} \leq \frac{1}{4\lambda^2 Q} \cdot \frac{\mathrm{dist}(x, y)}{p_x p_y} \leq 1.$$

□

Lemma 3.6. $\frac{4\lambda^2 Q}{N^2} \textit{Max-Cut}(X') \in (1 \pm \varepsilon) \cdot \textit{Max-Cut}(X)$.

Proof. Let $\widetilde{X}$ be a point-weighted set formed by re-weight points in X with $w_{\widetilde{X}}(x) = Np_x$. The following lemma from [15] shows that Max-Cut(X) = Max-Cut$(\widetilde{X})$.

[5] A slightly weaker bound of $O(\varepsilon^{-4} \mathrm{poly}\log(\varepsilon^{-1}))$ was obtained in [4], and [40] gave an improved technical lemma which can be directly plugged into [4] to obtain the $O(\varepsilon^{-4})$ bound.

Lemma 3.7 ([15, Lemma 5 and Lemma 6]). *Let (U, dist_U) be a metric space, and $W \subseteq U$ be a dataset. Suppose for every $x \in W$, $\mu_x > 0$ is an integer weight. Then the point-weighted set W' obtained from re-weighting each point $x \in W$ by μ_x, satisfies that*

$$\textsc{Max-Cut}(W') = \textsc{Max-Cut}(W).$$

Now, we observe that $\widetilde{X}$ can be naturally interpreted a weighted complete graph $\widetilde{G}$, where we copy $x \in X$ for $w_{\widetilde{X}}$ times to form the vertex set, and the edge length is defined as $\mathrm{dist}_{\widetilde{X}}(x, y)$. Notice that the vertex set of $\widetilde{G}$ is exactly X', and that the edge length

$$\mathrm{dist}_{\widetilde{X}}(x, y) = \frac{\mathrm{dist}(x, y)}{N^2 p_x p_y} \in (1 \pm \varepsilon) \cdot \frac{4\lambda^2 Q}{N^2} \cdot \mathrm{len}_{G'}(x, y).$$

Therefore, we conclude that $\textsc{Max-Cut}(X) = \textsc{Max-Cut}(\widetilde{X}) \in (1 \pm \varepsilon) \cdot \frac{4\lambda^2 Q}{N^2} \cdot \textsc{Max-Cut}(X')$. This finishes the proof. □

Now, we are ready to apply Lemma 3.2. Let $S' \subseteq X'$ such that $|S'| = O(\varepsilon^{-4})$ be the resultant set by applying Lemma 3.2 with $G = G'$ (recalling that the promise of $[0, 1]$ edge weights is proved in Lemma 3.4). Then

$$\frac{1}{|S'|^2}\textsc{Max-Cut}(S') \in \frac{\textsc{Max-Cut}(X')}{|X'|^2} \pm \varepsilon.$$

Applying Lemma 3.6, and observe that $|X'| = N$, the above equivalents to

$$\begin{aligned}\frac{4\lambda^2 Q}{|S'|^2}\textsc{Max-Cut}(S') &\in (1 \pm \varepsilon) \cdot \textsc{Max-Cut}(X) \pm \varepsilon \cdot 4\lambda^2 Q \\ &\in (1 \pm O(\lambda^2 \varepsilon)) \cdot \textsc{Max-Cut}(X)\end{aligned}$$

where the last inequality follows from $\textsc{Max-Cut}(X) \geq \Omega(Q)$. We finish the proof by rescaling ε.

4 STREAMING IMPLEMENTATIONS

Theorem 4.1 (Streaming Euclidean Max-Cut). *There is a randomized streaming algorithm that, given $0 < \varepsilon < 1/2$, $p \geq 1$, integers $\Delta, d \geq 1$, and an input dataset $X \subseteq [\Delta]^d$ presented as a dynamic stream, uses space $\mathrm{poly}(\varepsilon^{-1} d \log \Delta)$ and reports an estimate $\eta > 0$ that with high probability (at least $2/3$) is a $(1+\varepsilon)$-approximation to $\textsc{Max-Cut}(X)$ in ℓ_p.*

Our algorithm employs importance sampling as formulated in Theorem 3.1, and thus needs (enough) samples from a distribution $\mathcal{D}$ that corresponds to the input $X \subseteq [\Delta]^d$ with small parameter $\lambda > 0$. Given these samples, the algorithm can estimate $\textsc{Max-Cut}(X)$ by a brute-force search on the (point-weighted) samples, which can be done using small space. Note that Theorem 3.1 works for a general metric space, hence it also applies to the ℓ_p case as we require. We thus focus henceforth on performing importance sampling from a dataset X that is presented as a dynamic stream, as formalized next in Lemma 4.2.

Lemma 4.2 (Importance-Sampling Algorithm). *There is a randomized streaming algorithm $\mathcal{A}$ that, given $0 < \varepsilon < 1/2$, $p \geq 1$, integers $\Delta, d \geq 1$, and an input dataset $X \subseteq [\Delta]^d$ presented as a dynamic stream, it uses space $\mathrm{poly}(\varepsilon^{-1} d \log \Delta)$ and reports $z^* \in X \cup \{\perp\}$ together with $p^* \in [0, 1]$. The algorithm has a random initialization with success probability at least 0.99,[6] and conditioned on a successful initialization, its random output satisfies: (1) with probability at least $1 - 1/\mathrm{poly}(\Delta^d)$,*

$$\forall x \in X, \qquad \Pr[z^* = x] \geq \frac{1}{\lambda}\frac{q(x)}{Q},$$

for $q(x) := \sum_{y \in X} \mathrm{dist}(x, y)$, $Q := \sum_{x \in X} q(x)$, $\mathrm{dist} = \ell_p$, and $\lambda := \mathrm{poly}(d \log \Delta)$; and (2) whenever $z^* \neq \perp$,

$$z^* = x \in X \quad \Longrightarrow \quad p^* \in (1 \pm \varepsilon) \cdot \Pr[z^* = x].$$

The somewhat intricate statement of Lemma 4.2 is very useful to generate many samples with a large success probability. The obvious approach to generate t samples is to run t executions of this algorithm (all in parallel on the same stream) using independent coins, but then the success probability is only 0.99^t. Consider instead running t parallel executions, using the *same initialization coins* but otherwise independent coins, which requires total space $t \cdot \mathrm{poly}(\varepsilon^{-1} d \log \Delta)$. Then with probability at least 0.99 the initialization succeeds, in which case the t executions produce t independent samples, each of the form (z^*, p^*) and satisfies the two guarantees in the lemma.

4.1 The Importance-Sampling Algorithm (Proof of Lemma 4.2)

Our plan is to implement the importance sampling on a tree metric generated by a randomized embedding of the input dataset. The notion of randomized tree embedding was first proposed in [7] for arbitrary metric spaces, and the specific embedding that we employ was given by [24] for ℓ_p metrics presented as a stream of points. We describe this tree embedding below. We stress that our algorithm can be easily implemented in low space because it does not need to compute the entire embedding explicitly; for instance, the algorithm's initialization picks random coins, which determine the embedding but do not require any further computation.

Initialization Step: Randomized Tree Embedding [7, 9, 24]. Assume without loss of generality that $\Delta \geq 1$ is an integral power of 2, and let $L := 1 + d \log \Delta$. Let $\{\mathcal{G}_i\}_{i=0}^{L}$ be a recursive partitioning of the grid $[2\Delta]^d$ into squares,[7] as follows. Start with $\mathcal{G}_0$ being a trivial partitioning that has one part corresponding to the entire grid $[2\Delta]^d$, and for each $i \geq 0$, subdivide every square in $\mathcal{G}_i$ into 2^d squares of half the side-length, to obtain a partition $\mathcal{G}_{i+1}$ of the entire grid $[2\Delta]^d$. Thus, every $\mathcal{G}_i$ is a partition into squares of side-length 2^i. The recursive partitioning $\{\mathcal{G}_i\}_i$ naturally defines a rooted tree T, whose nodes are the squares inside all the $\mathcal{G}_i$'s, that if often called a *quadtree decomposition* (even though every tree node has 2^d children rather than 4). Finally, make the quadtree T random by shifting the entire recursive partitioning by a vector $-v_{\mathrm{shift}}$, where v_{shift} is chosen uniformly at random from $[\Delta]^d$. (This is equivalent to shifting the dataset $[\Delta]^d$ by v_{shift}, which explains why we defined the recursive partitioning over an extended grid

[6] It is convenient to separate the random coins of the algorithm into two groups, even though they can all be tossed before the stream starts. We refer to the coin tosses of the first group as an initialization step, and condition on their "success" when analyzing the second group of coins. The algorithm cannot tell whether its initialization was successful, and thus this event appears only in the analysis (in Lemma 4.4).

[7] Strictly speaking, these squares are actually hypercubes (sometimes called cells or grids), but we call them squares for intuition.

$[2\Delta]^d$.) Every node in T has a *level* (or equivalently, depth), where the root is at level 1, and the level of every other node is one bigger than that of its parent node. The *scale* of a tree node is the side-length of the corresponding square. Observe that leaves of T have scale 2^0 and thus correspond to squares that contain a single grid point; moreover, points $x \in [\Delta]^d$ correspond to distinct leaves in T. Define the weight of an edge in T between a node u at scale 2^i and its parent as $d^{\frac{1}{p}} \cdot 2^i$ (i.e., the diameter of u's square). Define a *tree embedding* of $[\Delta]^d$ by mapping every point $x \in [\Delta]^d$ to its corresponding leaf in T, and let the *tree distance* between two points $x, y \in [\Delta]^d$, denoted $\mathrm{dist}_T(x, y)$, be the distance in T between their corresponding leaves. The following lemma bounds the distortion of this randomized tree embedding. We remark that a better distortion of $O(d^{\max\{\frac{1}{p}, 1-\frac{1}{p}\}} \log \Delta)$ may be obtained via a different technique that is less suitable for streaming [9].

Lemma 4.3 ([24, Fact 1])**.** *Let T be a randomized tree as above. Then for all $x, y \in [\Delta]^d$,*

$$\mathrm{dist}_T(x, y) \geq \mathrm{dist}(x, y);$$
$$\mathbb{E}[\,\mathrm{dist}_T(x, y)] \leq O(d \log \Delta)\, \mathrm{dist}(x, y).$$

Streaming Implementation of Randomized Tree Embedding. We emphasize that in our definition of the quadtree T is non-standard as it contains the entire grid $[\Delta]^d$ as leaves (the standard approach is to recursively partition only squares that contain at least one point from the dataset X). The advantage of our approach is that the tree is defined obliviously of the dataset X (e.g., of updates to X). In particular, the leaf-to-root path from a point $x \in [\Delta]^d$ is well-defined regardless of X and can be computed on-the-fly (without constructing the entire tree T) using time and space $\mathrm{poly}(d \log \Delta)$, providing sufficient information for evaluating the tree distance.

Our streaming algorithm samples such a tree T as an initialization step, i.e., before the stream starts, which requires small space because it can be done implicitly by picking $\mathrm{poly}(d \log \Delta)$ random bits that describe the random shift vector v. Next, we show in Lemma 4.4 that this initialization step succeeds with 0.99 probability, and on success, every distance $\mathrm{dist}(x, y)$ for $x, y \in X$ is well-approximated by its corresponding $\mathrm{dist}_T(x, y)$. In this case, the sampling of points x with probability proportional to $q(x)$ can be replaced by sampling with probabilities that are derived from the tree metric. More specifically, the probability of sampling each $x \in X$ deviates from the desired probability $\frac{q(x)}{Q}$ by at most a factor of $\mathrm{poly}(d \log \Delta)$. We remark that the event of success does depend on the input X, but the algorithm does not need to know whether the initialization succeeded.

Lemma 4.4. *For $x \in X$, let $q_T(x) := \sum_{y \in X} \mathrm{dist}_T(x, y)$ and let $Q_T := \sum_{x \in X} q_T(x)$. Then*

$$\Pr_T\left[\forall x \in X,\ \frac{q_T(x)}{Q_T} \geq \frac{1}{O(d \log \Delta)} \frac{q(x)}{Q}\right] \geq 0.99.$$

Proof. Fix some $x \in X$. By Lemma 4.3,

$$q_T(x) = \sum_{y \in X} \mathrm{dist}_T(x, y) \geq \sum_{y \in X} \mathrm{dist}(x, y) = q(x) \tag{1}$$

and

$$\mathbb{E}\left[\sum_{y \in X} q_T(y)\right] = \mathbb{E}\left[\sum_{y, y' \in X} \mathrm{dist}_T(y, y')\right] \leq O(d \log \Delta) \sum_{y, y' \in X} \mathbb{E}[\mathrm{dist}(y, y')].$$

By Markov's inequality, with high constant probability,

$$\sum_{y \in X} q_T(y) \leq O(d \log \Delta) \sum_{y, y' \in X} \mathbb{E}[\mathrm{dist}(y, y')]. \tag{2}$$

We finish the proof by combining (2) and (1). □

Sampling w.r.t. Tree Distance. In the remainder of the proof, we assume that the random tree T was already picked and condition on its success as formulated in Lemma 4.4. This lemma shows that it actually suffices to sample each x with probability proportional to $q_T(x)$. Next, we provide in Fact 4.5 a different formula for $q_T(x)$ that is based on x's ancestors in the tree T, namely, on counting how many data points (i.e., from X) are contained in the squares that correspond to these ancestors. To this end, we need to set up some basic notation regarding X and T.

The Input X in the Tree T. Let $n := |X|$ be the number of input points at the end of the stream. For a tree node $v \in T$, let $X(v) \subseteq X$ be the set of points from X that are contained in the square corresponding to v. For $x \in X$ and $i \geq 1$, let $\mathrm{anc}_i(x)$ be the level-i ancestor of x in T (recalling that x corresponds to a leaf). By definition, $\mathrm{anc}_{L+1}(x) := x$. For $0 \leq i \leq L$, let $\beta_i := d^{\frac{1}{p}} \cdot 2^{L+1-i}$, which is the edge-length between a level-i node u and its parent (since the scale of a level-i node is 2^{L+1-i}). Due to the tree structure, we have the following representation of $q_T(x)$.

Fact 4.5. *For every $x \in X$, we have $q_T(x) = 2\sum_{i=0}^{L} \beta_i \cdot (n - |X(\mathrm{anc}_i(x))|)$.*

For each level i, let h_i be a level-i node whose corresponding square contains the most points from X, breaking ties arbitrarily. Next, we wish to identify a *critical* level k; ideally, this is the last level going down from the root, i.e., largest i, such that $|X(h_i)| \geq 0.6n$ (the constant 0.6 is somewhat arbitrary). However, it is difficult to find this k exactly in a streaming algorithm, and thus we use instead a level $\tilde{k}$ that satisfies a relaxed guarantee that only requires estimates on different $|X(h_i)|$, as follows. Let us fix henceforth two constants $0.5 < \sigma^- \leq \sigma^+ \leq 1$.

Definition 4.6 (Critical Level)**.** Level $1 \leq \tilde{k} < L + 1$ is called (σ^-, σ^+)-*critical*, if $|X(h_{\tilde{k}})| \geq \sigma^- n$ and $|X(h_{\tilde{k}+1})| \leq \sigma^+ n$.

Suppose henceforth that $\tilde{k}$ is a (σ^-, σ^+)-critical level. (Such a critical level clearly exists, although its value need not be unique.) Since $|X(h_i)| \geq |X(h_{i+1})|$ for every $i < \tilde{k}$ (because h_i contains the most points from X at level i), we know that $|X(h_i)| \geq \sigma^- n$ for every $i \leq \tilde{k}$ (not only for $i = \tilde{k}$), and $|X(h_i)| \leq \sigma^+ n$ for every $i > \tilde{k}$.

Fact 4.7. *Each h_i is the parent of h_{i+1} for $1 \leq i \leq \tilde{k} - 1$, hence $(h_1, \ldots, h_{\tilde{k}})$ forms a path from the root of T.*

Next, we further "simplify" the representation of $q_T(x)$, by introducing an approximate version of it that requires even less information about x. Specifically, we introduce in Definition 4.8 a

sequence of $O(L)$ values that are independent of x, namely, one value $\tilde{q}_i$ for each level $i \le \tilde{k}$, and then we show in Lemma 4.9 that for every $x \in X$, we can approximate $q_T(x)$ by one of these $O(L)$ values, namely, by $\tilde{q}_i$ for a suitable level $i = \ell(x)$.

Definition 4.8 (Estimator for q_T)**.** For $1 \le i \le \tilde{k}$, define

$$\tilde{q}_i := n\beta_i + \sum_{j \le i} \beta_j \cdot (n - |X(h_j)|).$$

Relating q_T and $\tilde{q}$. For $x \in X$, let $\ell(x)$ be the maximum level $1 \le j \le \tilde{k}$ such that $\mathrm{anc}_j(x) = h_j$. This is well-defined, because $j = 1$ always satisfies that $\mathrm{anc}_j(x) = h_j$. The next lemma shows that $q_T(x)$ can be approximated by $\tilde{q}_i$ for $i = \ell(x)$.

Lemma 4.9. *Let $\tilde{k}$ be a (σ^-, σ^+)-critical level. Then*

$$\forall x \in X, \qquad \tilde{q}_{\ell(x)} = \Theta(1) \cdot q_T(x).$$

Proof.

$$\begin{aligned}
\frac{1}{2} q_T(x) &= \sum_{i=0}^{L} \beta_i \cdot (n - |X(\mathrm{anc}_i(x))|) \\
&= \sum_{i \le \ell(x)} \beta_i \cdot (n - |X(h_i)|) + \sum_{i > \ell(x)} \beta_i \cdot (n - |X(\mathrm{anc}_i(x))|) \qquad (3) \\
&\in \sum_{i \le \ell(x)} \beta_i \cdot (n - |X(h_i)|) \\
&\qquad + [\min\{\sigma^-, 1 - \sigma^+\}, 1] \cdot n \sum_{i > \ell(x)} \beta_i \qquad (4) \\
&\in \sum_{i \le \ell(x)} \beta_i \cdot (n - |X(h_i)|) + [\min\{\sigma^-, 1 - \sigma^+\}, 1] \cdot n\beta_{\ell(x)} \\
&\in [\min\{\sigma^-, 1 - \sigma^+\}, 1] \cdot \tilde{q}_{\ell(x)}.
\end{aligned}$$

In the above, (3) follows from the fact that $\mathrm{anc}_i(x) = h_i$ for $i \le \ell(x)$ (by the definition of $\ell(x)$ and the property that $(h_1, \ldots, h_{\tilde{k}})$ forms a path from Fact 4.7). (4) follows from the definition of (σ, μ)-critical and the definition of ℓ. □

The next lemma shows that the sequence $\tilde{q}_1, \ldots, \tilde{q}_{\tilde{k}}$ is non-increasing.

Fact 4.10. *$\tilde{q}_1 = \beta_1 n$, and for every $2 \le i \le \tilde{k}$, we have $\tilde{q}_i \le \tilde{q}_{i-1}$.*

Proof. The fact for $i = 1$ is immediate. Now consider $i \ge 2$. We have

$$\tilde{q}_{i-1} - \tilde{q}_i = n(\beta_{i-1} - \beta_i) - \beta_i \cdot (n - |X(h_i)|) = \beta_i \cdot |X(h_i)| \ge 0,$$

which verifies the lemma. □

Alternative Sampling Procedure. Recall that level $\tilde{k}$ is assumed to be (σ^-, σ^+)-critical for fixed constants $0.5 < \sigma^- \le \sigma^+ \le 1$. We plan to sample $x \in X$ with probability proportional to $\tilde{q}_{\ell(x)}$, and by Lemma 4.9 this only loses an $O(1)$ factor in the bound λ needed for importance sampling (as in Lemma 4.2). For $1 \le i \le \tilde{k}$, define $X_i := \{x \in X \mid \ell(x) = i\}$. Notice that $\{X_i\}_{i=1}^{\tilde{k}}$ forms a partition of X, and

$$X_i = \begin{cases} X(h_i) \setminus X(h_{i+1}) & \text{if } 1 \le i \le \tilde{k} - 1; \\ X(h_{\tilde{k}}) & \text{if } i = \tilde{k}. \end{cases} \qquad (5)$$

By definition, points in the same X_i have the same $\tilde{q}_{\ell(x)}$, and thus also the same sampling probability. A natural approach to sampling a point from X with the desired probabilities is to first pick a random $i \in [\tilde{k}]$ (non-uniformly) and then sample uniformly a point from that X_i. But unfortunately, it is impossible to sample uniformly from X_i in streaming (this is justified in Claim A.2), and thus we shall sample instead from an "extended" set $X_i^{\mathrm{ext}} \supseteq X_i$, defined as follows.

$$X_i^{\mathrm{ext}} := \begin{cases} X \setminus X(h_{i+1}) & \text{if } 1 \le i \le \tilde{k} - 1; \\ X(h_{\tilde{k}}) & \text{if } i = \tilde{k}. \end{cases} \qquad (6)$$

The path structure of $\{h_i\}_i$ (Fact 4.7) implies the following.

Fact 4.11. *For every $1 \le i < \tilde{k}$, we have $X_i^{\mathrm{ext}} = X_1 \cup \ldots \cup X_i$.*

We describe in Algorithm 1 a procedure for sampling $x \in X$ with probability proportional to $\tilde{q}_{\ell(x)}$, based on the above approach of picking a random $i \in [\tilde{k}]$ (from a suitable distribution) and then sampling uniformly a point from that X_i^{ext}. We then prove in Lemma 4.13 that this procedure samples from X with probabilities proportional to $\tilde{q}_{\ell(x)}$, up to an $O(L)$ factor.

Remark 4.12. Sampling from the extended sets (X_i^{ext} instead of X_i) can significantly bias the sampling probabilities, because the "contribution" of a point $x \in X$ can increase by an unbounded factor. On the one hand, this can increase the sampling probability of that x, which is not a problem at all. On the other hand, it might increase the total contribution of all points (and thus decrease some individual sampling probabilities), but our analysis shows that this effect is bounded by an $O(L)$ factor. The intuition here is that $q(x)$ represents the sum of distances from x to all other points $y \in X$, and we can rearrange their total $\sum_x q(x)$ by the "other" point $y \in X$, and the crux now is that the contribution of each $y \in X$ increases by at most $O(L)$ factor.

Algorithm 1 Alternative sampling procedure (offline)

1: draw a random i^* where each $1 \le i \le \tilde{k}$ is picked with probability $r_i := \frac{|X_i^{\mathrm{ext}}| \tilde{q}_i}{\sum_{j=1}^{\tilde{k}} |X_j^{\mathrm{ext}}| \tilde{q}_j}$
2: draw $x \in X_i^{\mathrm{ext}}$ uniformly at random
3: return $z^* = x$ as the sample, together with $p^* = \sum_{i=\ell(x)}^{\tilde{k}} \frac{r_i}{|X_i^{\mathrm{ext}}|}$ as its sampling probability

Lemma 4.13. *Algorithm 1 samples every $x \in X$ with probability $\Pr[z^* = x] = \sum_{i=\ell(x)}^{\tilde{k}} \frac{r_i}{|X_i^{\mathrm{ext}}|}$, exactly as line 3 reports in p^*, and furthermore this is bounded by $\Pr[z^* = x] \ge \Omega\left(\frac{1}{L}\right) \frac{\tilde{q}_{\ell(x)}}{\sum_{x \in X} \tilde{q}_{\ell(x)}}$.*

Proof. Observe that $x \in X_{\ell(x)}$, and by Fact 4.11, this point x can only be sampled for $i \ge \ell(x)$. Therefore, $\Pr[z^* = x] = \sum_{i=\ell(x)}^{\tilde{k}} \frac{r_i}{|X_i^{\mathrm{ext}}|} = p^*$. We bound this probability by

$$\begin{aligned}
\Pr[z^* = x] &= \sum_{i \ge \ell(x)} \frac{r_i}{|X_i^{\mathrm{ext}}|} = \sum_{i \ge \ell(x)} \frac{\tilde{q}_i}{\sum_{j=1}^{\tilde{k}} |X_j^{\mathrm{ext}}| \tilde{q}_j} \\
&= \frac{\sum_{i \ge \ell(x)} \tilde{q}_i}{\sum_{j=1}^{\tilde{k}} |X_j^{\mathrm{ext}}| \tilde{q}_j} \ge \frac{\tilde{q}_{\ell(x)}}{\sum_{j=1}^{\tilde{k}} |X_j^{\mathrm{ext}}| \tilde{q}_j}. \qquad (7)
\end{aligned}$$

Next, to bound the denominator $\sum_{j=1}^{\tilde{k}} |X_j^{\text{ext}}|\tilde{q}_j$, observe that $|X_j^{\text{ext}}| = \sum_{i=1}^{j} |X_i|$ for all $j < \tilde{k}$ (by Fact 4.11), and therefore

$$\begin{aligned}\sum_{j=1}^{\tilde{k}} |X_j^{\text{ext}}|\tilde{q}_j &= |X_{\tilde{k}}|\tilde{q}_{\tilde{k}} + \sum_{j=1}^{\tilde{k}-1}\sum_{i=1}^{j} |X_i|\tilde{q}_j \\ &= |X_{\tilde{k}}|\tilde{q}_{\tilde{k}} + \sum_{i=1}^{\tilde{k}-1}\sum_{j=i}^{\tilde{k}-1} |X_i|\tilde{q}_j \\ &\le |X_{\tilde{k}}|\tilde{q}_{\tilde{k}} + \tilde{k}\cdot\sum_{i=1}^{\tilde{k}-1} |X_i|\tilde{q}_i \\ &\le (L+1)\sum_{i=1}^{\tilde{k}} |X_i|\tilde{q}_i = (L+1)\sum_{x\in X}\tilde{q}_{\ell(x)},\end{aligned}$$

where the first inequality is by the monotonicity of $\tilde{q}_i$'s (Fact 4.10). Combining this with (7), the lemma follows. □

Implementing Algorithm 1 in Streaming. To implement Algorithm 1 in streaming, we first need a streaming algorithm that finds a critical level $\tilde{k}$ using space $O(\text{poly}(d \log \Delta))$. We discuss this next.

Finding $\tilde{k}$. For each level i, we draw $\text{poly}(d \log \Delta)$ samples $S_i \subseteq X$ uniformly at random from X. We then count the number of samples that lie in each tree node (square) at level i, and let m_i be the maximum count. We let $\tilde{k}$ be the largest level i such that $\frac{m_i}{|S_i|} \ge 0.6$. By a standard application of Chernoff bound, with probability at least $1 - \text{poly}(\Delta^d)$, this level $\tilde{k}$ is $(0.55, 0.65)$-critical. Moreover, this process can be implemented in streaming using space $\text{poly}(d \log \Delta)$, by maintaining, for each level i, only $|S_i| = \text{poly}(d \log \Delta)$ independent ℓ_0-samplers (Lemma 2.2) on the domain $[\Delta]^d$. A similar approach can be used to $(1+\varepsilon)$-approximate the size of $X(h_i)$ for every $i \le \tilde{k}$, and also sample uniformly from these sets, using space $O(\text{poly}(\varepsilon^{-1} d \log \Delta))$ and with failure probability $1 - 1/\text{poly}(\Delta^d)$ (by using Lemmas 2.1 and 2.2).

Estimating and Sampling from $X \setminus X(h_i)$. We also need to estimate $\tilde{q}_i$ and $|X_i^{\text{ext}}|$, and to sample uniformly at random from X_i^{ext}, for every $i \le \tilde{k}$. The case $i = \tilde{k}$ was already discussed, because $X_{\tilde{k}}^{\text{ext}} = X(h_{\tilde{k}})$. It remains to consider $i < \tilde{k}$, in which case we need to $(1\pm\varepsilon)$-approximate the size of $X \setminus X(h_i)$, and also to sample uniformly at random from that set, and we can assume that $|X(h_i)| > 0.5n$. We provide such a streaming algorithm in Lemma 4.14 below, which we prove in Section 4.2. This lemma is stated in a more general form that may be of independent interest, where the input is a frequency vector $x \in \mathbb{R}^N$ (i.e., a stream of insertions and deletions of items from domain $[N]$) and access to a function $\mathcal{P} : [N] \to [N']$, for $N' \le N$, that can be viewed as a partition of the domain into N' parts. In our intended application, the domain $[N]$ will be the grid $[\Delta]^d$, and the partition $\mathcal{P}$ will be its partition into squares of a given level i; observe that it is easy to implement $\mathcal{P}$ as a function that maps each grid point to its level-i square. Roughly speaking, the streaming algorithm in Lemma 4.14 samples uniformly from the support set $\text{supp}(x) = \{i \in [N] : x_i \ne 0\}$, but excluding indices that lie in the part of $\mathcal{P}$ that is heaviest, i.e., has most nonzero indices, assuming it is sufficiently heavy. In our intended application, this method samples uniformly from the input $X \subset [\Delta]^d$ but excluding points that lie in the heaviest square, i.e., uniformly from $X \setminus X(h_i)$. (square with the largest number of input points).

Lemma 4.14 (Sampling from Light Parts)**.** *There exists a streaming algorithm, that given $0 < \varepsilon, \delta, \sigma < 0.5$, integers $N, N', M \ge 1$, a mapping $\mathcal{P} : [N] \to [N']$, and a frequency vector $x \in [-M, M]^N$ that is presented as a stream of additive updates, uses space*

$$O(\text{poly}(\varepsilon^{-1}\sigma^{-1}\log(\delta^{-1}MN))),$$

and reports a sample $i^* \in [N] \cup \{\perp\}$ and a value $r^* \ge 0$. Let $X := \{i \in [N] \mid x_i \ne 0\}$ be the support of x, and let $j_{\max} := \arg\max_{j\in[N']} |\mathcal{P}^{-1}(j) \cap X|$ be the heaviest $\mathcal{P}$ with respect to X. If $X_{\text{heavy}} := \mathcal{P}^{-1}(j_{\max}) \cap X$ satisfies $|X_{\text{heavy}}| \ge (0.5+\sigma)|X|$, then with probability at least $1-\delta$,

- *$r^* \in (1 \pm \varepsilon) \cdot |X_{\text{light}}|$ where $X_{\text{light}} := X \setminus X_{\text{heavy}}$, and*
- *unless X_{light} is empty, $i^* \in X_{\text{light}}$ and moreover for all $i \in X_{\text{light}}$, it holds that $\Pr[i^* = i] = \frac{1}{|X_{\text{light}}|}$ (provided that $|X_{\text{light}}| \ne 0$).*

In our application, we will apply Lemma 4.14 in parallel for every level i, with $N = \Delta^d$, i.e., the items being inserted and deleted are points in $[\Delta]^d$, and a mapping $\mathcal{P}$ defined by the level-i squares (tree nodes), i.e., for $x \in [\Delta]^d$ we define $\mathcal{P}(x)$ as the level-i node that contains x. We will set the failure probability to be $\delta = 1/\text{poly}(\Delta^d)$ and a fixed $\sigma = 0.05$. This way, conditioning on the success of Lemma 4.14, we can compute $\tilde{q}_i$, $|X_i^{\text{ext}}|$ with error $(1 \pm \varepsilon)$, and sampled from X_i^{ext} uniformly.

Concluding Lemma 4.2. In conclusion, our streaming algorithm initializes with sampling a randomly-shifted quadtree T which defines a tree embedding, all in an implicit way. Then, assume T is obtained and condition on the success of it, specifically Lemma 4.4 (with probability 0.99), we use the streaming implementation of Algorithm 1, as outlined above. The resultant z^* and p^* are the return value. The error bound on z^* and p^* and the bound of $\lambda = O(\text{poly}(d \log \Delta))$ follow by Lemma 4.4 and Lemma 4.13, plus an additional error and failure probability introduced by streaming, which is bounded in the previous paragraphs. This finishes the proof.

4.2 Sampling from The Light Parts (Proof of Lemma 4.14)

An Offline Algorithm. Notice that $\{\mathcal{P}^{-1}(y)\}_{y\in[N']}$ defines a partition of $[N]$. In our proof, we interpret $\mathcal{P} = \{P_i\}_i$ as such a partition. Let $P_{\max} := \mathcal{P}^{-1}(y_{\max})$ be the part of $\mathcal{P}$ that contains the most from X, so $X_{\text{heavy}} = P_{\max} \cap X$. We start with an offline algorithm, summarized in Algorithm 2. In the algorithm, we consider a set of $s = \Theta(\log(N\delta^{-1}))$ random hash functions $h_1, \ldots, h_s$ that randomly map each part in $\mathcal{P}$ to one of $u = 2$ buckets (as in line 2).

Then, consider some h_t for $t \in [s]$. Let B_j ($j \in [u]$) be the elements from all parts that are mapped by h_t to the bucket j (in line 4). We find j^* as the bucket that contains the most elements from X (in line 5). Since we assume $|X_{\text{heavy}}| \ge (0.5+\sigma)|X| > 0.5|X|$, we know the bucket $h_t(P_{\max})$ contains *more* than $0.5|X|$ elements form X (recalling that $P_{\max} = \mathcal{P}^{-1}(y_{\max})$ is the part that contains the most from X), and this implies $h_t(P_{\max})$ must be the bucket

Algorithm 2 Sampling and estimating from the light part (offline)

1: let $u \leftarrow 2$, $s \leftarrow \Theta(\log(N\delta^{-1}))$
2: let $\mathcal{H} \leftarrow \{h_1, \ldots, h_s\}$ be a collection of independent random hash functions, where each $h \in \mathcal{H}$ ($h : \mathcal{P} \to [u]$) satisfies $\forall P \neq P'$, $\Pr[h(P) = h(P')] \leq 1/u$
3: **for** $t \in [s]$ **do**
4: for $j \in [u]$, let $B_j \leftarrow \left(\bigcup_{P \in \mathcal{P}: h_t(P) = j} P\right) \cap X$
5: let $j^* \leftarrow \arg\max_j |B_j|$
6: let $D_t \leftarrow X \setminus \bigcup_{P \in \mathcal{P}: h_t(P) = j^*} P$
7: **end for**
8: compute $D_{\text{all}} \leftarrow \bigcup_{t \in [s]} D_t$
9: return a uniform sample $i^* \in D_{\text{all}}$, and report $r^* := |D_{\text{all}}|$ as the estimate for $|X_{\text{light}}|$

that contains the most elements from X. Hence,

$$j^* = h_t(P_{\max}). \tag{8}$$

Next, we drop the elements that lie in the bucket j^*, and take the remaining elements, as D_t (in line 6). While D_t certainly does not contain any element from X_{heavy} (by (8) and the definition of D_t's), D_t is only a subset of X_{light}. Hence, we take the union of all D_t's (over $t \in [s]$), denoted as D_{all} (in line 8), which equals X_{light} with high probability.

Analysis of D_{all}. For every $i \in X_{\text{light}}$, every $t \in [s]$,

$$\Pr[i \notin D_t] = \Pr[h_t(P_i) = h_t(P_{\max})] \leq \frac{1}{u} = \frac{1}{2},$$

where $P_i \in \mathcal{P}$ is the part that i belongs to. Therefore, by the independence of h_t's, we know for every $i \in X_{\text{light}}$,

$$\Pr[i \notin D_{\text{all}}] = \Pr[\forall t \in [s], i \notin D_t] \leq \frac{1}{2^s} = \frac{\delta}{\text{poly}(N)}.$$

Taking a union bound over $i \in X_{\text{light}}$, we have

$$\Pr[\exists i \in X_{\text{light}}, i \notin D_{\text{all}}] \leq \frac{\delta}{\text{poly}(N)} |X_{\text{light}}| \leq \delta.$$

Hence, we conclude that

$$\Pr[D_{\text{all}} = X_{\text{light}}] \geq 1 - \delta.$$

Conditioning on the event that $D_{\text{all}} = X_{\text{light}}$, we conclude that $r^* = |D_{\text{all}}| = |X_{\text{light}}|$, and that $\forall i \in X_{\text{light}}$, $\Pr[i^* = i] = \frac{1}{|D_{\text{all}}|} = \frac{1}{|X_{\text{light}}|}$.

Streaming Algorithm. It remains to give a streaming implementation for Algorithm 2. Before the stream starts, we initialize several streaming data structures. We start with building the hash functions $\mathcal{H}$, and this can be implemented using space $\text{poly}(\log N)$, by using hash families of limited independence. Next, we maintain for every $t \in [s]$, for every bucket $j \in [u]$, an ℓ_0-sampler $\mathcal{L}_j^{(t)}$ (Lemma 2.2) with failure probability $O(\frac{\delta}{us})$, as well as an ℓ_0-norm estimator $\mathcal{K}_j^{(t)}$ (Lemma 2.1) with failure probability $O(\frac{\delta}{us})$ and error guarantee $\varepsilon\sigma \leq \min\{\varepsilon, \sigma\}$, both on domain $[n]$. The setup of the failure probabilities immediately implies that with probability $1 - \delta$, all data structures succeed simultaneously, and we condition on their success in the following argument. Since we need to combine the linear sketches $\mathcal{L}_j^{(t)}$'s in later steps, for every $t \in [s]$ and $j \in [u]$, we use the same random seeds among all ℓ_0-samplers $\{\mathcal{L}_j^{(t)}\}$'s, so that they can be "combined" by simply adding up. Also do the same for $\mathcal{K}_j^{(t)}$'s. Another detail is that, strictly speaking, we need $O(1)$ independent "copies" of every $\mathcal{K}$ and $\mathcal{L}$, since we need to query each of them $O(1)$ times. As this only enlarges the space by an $O(1)$ factor, we omit this detail for the sake of presentation.

Whenever an update for element $i \in [n]$ is received, we update $\mathcal{L}_{j_i}^{(t)}$ and $\mathcal{K}_{j_i}^{(t)}$ for every $t \in [s]$, where $j_i := h_t(P_i)$, and $P_i \in \mathcal{P}$ is the unique part that contains i.

When the stream terminates, we proceed to generate the sample $i^* \in X_{\text{light}}$ and the estimate r^* for X_{light}. For $t \in [s]$, $j \in [u]$, query $\mathcal{K}_j^{(t)}$ to obtain an estimator for $|B_j|$ (line 4) within $(1 \pm \varepsilon\sigma)$ factor. Use these estimations to find j^* (line 5). Note that this j^* is the same as computing using exact $|B_j|$ values. To see this, the key observation is that, $|X_{\text{heavy}}| \geq (0.5 + \sigma)|X|$, while for every $P \in \mathcal{P} \setminus P_{\max}$ we have $|P| \leq (0.5 - \sigma)|X|$. Hence, to precisely find j^*, it suffices to distinguish between subsets P, P' such that $|P| \geq (0.5+\sigma)|X|$ and $|P'| \leq (0.5-\sigma)|X|$. Even with a $(1\pm\varepsilon\sigma)$ error (which is the error of our $\mathcal{K}$'s), this gap is still $\frac{0.5+\sigma}{0.5-\sigma} \cdot \frac{1+\varepsilon\sigma}{1-\varepsilon\sigma} > 1$ which is large enough.

Next, compute $\mathcal{L}^{(t)} := \sum_{j \in [u] \setminus \{j^*\}} \mathcal{L}_j^{(t)}$ as the ℓ_0-sampler that corresponds to D_t (line 6), and obtain $\mathcal{K}^{(t)} := \sum_{j \in [u] \setminus \{j^*\}} \mathcal{K}_j^{(t)}$ similarly. We can do this since we use the same random seeds among $\mathcal{L}_j^{(t)}$'s (and the same has been done to $\mathcal{K}$). We further compute $\mathcal{L} := \sum_{t \in [s]} \mathcal{L}^{(t)}$ whose support corresponds to D_{all}. Define $\mathcal{K} := \sum_{t \in [s]} \mathcal{K}^{(t)}$ similarly. The final return values i^* and r^* are given by querying $\mathcal{L}$ and $\mathcal{K}$. Note that on the success of the ℓ_0-sampler $\mathcal{L}$, the probability for $i^* = i$ for each $i \in W$ is exactly $\frac{1}{|W|}$ (Lemma 2.2). However, the r^* deviates from $|W|$ by a multiplicative $(1 \pm \varepsilon)$ factor.

In conclusion, the analysis of Algorithm 2 still goes through by using the estimated values as in the above procedure, except that one needs to rescale ε and δ by a constant factor, to compensate the error and failure probability introduced by the streaming data structures. This finishes the proof of Lemma 4.14.

Acknowledgments

We thank Christian Sohler for pointing us to the dimension reduction result in [34, 35]. We also thank an anonymous reviewer for pointing out how to simplify our proof of this result (Theorem B.1).

A LOWER BOUNDS BASED ON INDEX

Definition A.1 (INDEX Problem)**.** Alice is given a message $x \in \{0,1\}^n$, and Bob is given an index $i \in [n]$. Alice can send Bob exactly one message M, and Bob needs to use his input i and this message M, to compute $x_i \in \{0,1\}$.

It is well known that the INDEX problem requires $\Omega(n)$ combination to succeed with constant probability, i.e., $M = \Omega(n)$ (see e.g., [26, 32, 33]).

Claim A.2. *For every integer $\Delta \geq 1$, given access to a quadtree T on $[\Delta]$, any streaming algorithm that tests with constant success probability whether $X(h_{i^*}) \setminus X(h_{i^*+1}) = \emptyset$ for every $X \subseteq [\Delta]$ and i^* presented as an insertion-only point stream must use space $\Omega(\sqrt{\Delta})$,*

where h_i is defined as in Section 4 which is the level-i node of T that contains the most from X, and it is promised that both $|X(h_{i^*})|$ and $|X(h_{i^*+1})|$ is larger than $0.5|X|$.

Proof. Pick the level i^* in T such that the squares (i.e., intervals) are of side-length $m := 10\sqrt{\Delta}$. We reduce to INDEX with $n := \Delta/m = \frac{1}{10}\sqrt{\Delta}$, corresponding to the level-i^* nodes. (Note that i^* is public knowledge between Alice and Bob). Suppose Alice receives $x \in \{0,1\}^n$. For $j \in [n]$, if $x_j = 1$, insert a point at coordinate $(j-1)\cdot m + 1$, which is the first coordinate in the j-th interval at level i^*, and do nothing if $x_j = 0$. Feed this input to the algorithm, and send the internal state of the algorithm to Bob.

When Bob receives $i \in [n]$, Bob inserts one point to each coordinate in the right/larger sub-interval of the i-th level-i^* interval, namely, $((i-1)\cdot m + \frac{m}{2}, i\cdot m - 1]$. Clearly, h_{i^*} is this i-th interval at level i^*, h_{i^*+1} is the right sub-interval of h_{i^*}, and both $|X(h_{i^*})|$ and $|X(h_{i^*+1})|$ contain more than $0.5|X|$ points. Observe that $X(h_{i^*}) \setminus X(h_{i^*+1}) = \emptyset$ if and only if $x_i = 0$. This finishes the proof. □

Claim A.3. *For every integer $\Delta \geq 1$, any algorithm that with constant success probability computes Max-Cut(X) exactly for every $X \subseteq [\Delta]$ presented as an insertion-only point stream must use space $\Omega(\text{poly}(\Delta))$.*

Proof. In our proof, we assume an algorithm $\mathcal{A}$ returns $\eta \geq 0$, such that for every $X \subseteq [\Delta]$,

$$\Pr[\eta = \text{Max-Cut}(X)] \geq 1 - 1/\Delta^c,$$

for some sufficiently large $c \geq 1$. We show such $\mathcal{A}$ must use space $\Omega(\text{poly}(\Delta))$. Note that the assumption about the $1/\Delta^c$ probability is without loss of generality.

We reduce to INDEX with $n := \Delta^{0.1}$. Let $m := \frac{\Delta}{n} = \Delta^{0.9} = n^9$. Suppose Alice receives $x \in \{0,1\}^n$. For $j \in [n]$, if $x_j = 1$, insert a point at coordinate $(j-1)m$. [8] If $x_j = 0$ do nothing. Alice feeds this input to $\mathcal{A}$, and sends the internal state of $\mathcal{A}$ to Bob.

Now, suppose Bob receives $i \in [n]$. Bob resumes algorithm $\mathcal{A}$, and use $\mathcal{A}$ to do the following: for every $j \in [n]$, insert to the stream $P_j := \{(j-1)m + \frac{m}{2} + t \mid 1 \leq t \leq n^3\}$ and query the Max-Cut value. Restore to the initial received states of $\mathcal{A}$ after every iteration of j. This may be done, by saving the received state, insert the points P_j for one j, query, and fall back to the saved state. Eventually, since $\mathcal{A}$ succeeds with probability $1 - 1/\Delta^c$, by a union bound, with constant probability, all the queries are answered correctly simultaneously. We condition on this constant-probability event.

Let $X := \{i \in [n] \mid x_i \neq 0\}$ be the support of Alice's input x, and let $X' := \{(i-1)m \mid i \in X\}$ be the set of points inserted by Alice. Let $\widehat{X}_j := X' \cup P_j$. Notice that now we have the exact value of Max-Cut$(\widehat{X}_j)$ for every $j \in [n]$. It remains to show this information suffices to deduce the x_i value, which the INDEX problem asks for. To this end, we need to show the following lemma, that gives the (rough) value of Max-Cut$(\widehat{X}_j)$, which is basically $\sum_{i\in X} |i - j - 0.5|$ (up to scaling and a neglectable additive error).

[8] Here we may insert a point at 0 which is not in $[\Delta]$. This may be resolved by e.g., enlarging Δ by 1 and shifting the coordinates, but we omit this detail for the sake of presentation.

Lemma A.4. *For every $j \in [n]$, Max-Cut$(\widehat{X}_j) \in n^{12}\sum_{i\in X} |i - j - 0.5| \pm O(n^4)$.*

Proof. Fix some j. Note that $P_j \cap X' = \emptyset$. Observe that the cut value of the cut (X', P_j) is

$$\begin{aligned}
\sum_{x\in X', y\in P_j} \text{dist}(x,y) &= \sum_{i\in X}\sum_{t\in[n^3]} \left|(i-1)m - (j-1)m - \frac{m}{2} - t\right| \\
&= \sum_{i\in X}\sum_{t\in[n^3]} |(i-j-0.5)n^9 - t| \\
&\in \sum_{i\in X}\sum_{t\in[n^3]} |i-j-0.5|\cdot n^9 \pm t \\
&\in \sum_{i\in X} |i-j-0.5|\cdot n^{12} \pm O(n^4).
\end{aligned}$$

It suffices to show the cut (P_j, X') achieves Max-Cut$(\widehat{X}_j)$. Define the notation $\text{cut}(W,Z) := \sum_{x\in W, y\in Z} \text{dist}(x,y)$. Consider a cut (S,T) of $\widehat{X}_j$ such that $S \neq P_j$ and $T \neq X'$, and we show $\text{cut}(S,T) < \text{cut}(X', P_j)$. A useful fact which follows from the definition is that $\text{dist}(X', P_j) \geq \frac{m}{2}$.

Easy Case. First, we consider the (easy) case such that $S \subseteq P_j$ or $T \subseteq P_j$. In this case, there is some $\emptyset \neq W \subset P_j$ such that $\text{cut}(S,T) = \text{cut}(P_j \setminus W, X' \cup W)$. Then,

$$\begin{aligned}
\text{cut}(S,T) = \text{cut}(P_j \setminus W, X' \cup W) &= \text{cut}(P_j \setminus W, X') + \text{cut}(P_j, W) \\
&< \text{cut}(P_j \setminus W, X') + \frac{|P_j|^2}{4}\cdot n^3 \\
&= \text{cut}(P_j, X') - \text{cut}(W, X) + \frac{n^9}{4} \\
&\leq \text{cut}(P_j, X') - \frac{m}{2} + \frac{n^9}{4} \\
&< \text{cut}(P_j, X').
\end{aligned}$$

General Case. Now, we handle the remaining case where neither S nor T is a subset of P_j. Then $|S \cap X'| > 0$ and $|T \cap X'| > 0$. Combining these facts and assumptions, we have

$$\begin{aligned}
&\text{cut}(P_j, X') - \text{cut}(S,T) \\
&= \text{cut}(P_j, X') - \text{cut}(S\cap P_j, T\cap X') - \text{cut}(S\cap X', T\cap P_j) \\
&\qquad - \text{cut}(S\cap X', T\cap X') - \text{cut}(S\cap P_j, T\cap P_j) \\
&> \text{cut}(S\cap P_j, S\cap X') + \text{cut}(T\cap P_j, T\cap X') - \frac{|X'|^2}{4}\cdot\Delta - |P_j|^2\cdot n^3 \\
&\geq \frac{m}{2}\cdot(|S\cap X'|\cdot|S\cap P_j| + |T\cap X'|\cdot|T\cap P_j|) - \frac{n^{12}}{4} - n^9 \\
&\geq \frac{m}{2}\cdot(|S\cap P_j| + |T\cap P_j|) - \frac{n^{12}}{4} - n^9 \\
&= \frac{n^{12}}{4} - n^9 > 0.
\end{aligned}$$

This finishes the proof of Lemma A.4. □

Observe that, the function $f : [n] \to \mathbb{R}_+$ such that $f(j) := 2\sum_{i\in X} |i - j - 0.5|$ is integer-valued, so $2n^{12}\sum_{i\in X} |i-j-0.5|$ must be a multiple of n^{12}. Therefore, by Lemma A.4, we round the value of $2\cdot$ Max-Cut$(\widehat{X}_j)$ to the nearest multiple of n^{12} (so that the additive error $O(n^4)$ in Lemma A.4 is ignored), and we obtain the exact value of $2n^{12}\sum_{i\in X} |i-j-0.5|$. In other words, we know

the value of $f(j)$ for all $j \in [n]$. Finally, because of the piece-wise linear structure of f, one can make use of all the $f(j)$ values to recover all of X. This finishes the proof of Claim A.3. □

B DIMENSION REDUCTION FOR MAX-CUT

For completeness, we prove below a dimension reduction for Max-Cut, similarly to the one given in [35], with proof details appearing in [34].

Theorem B.1. *Let $X \subset \mathbb{R}^d$ be a finite set, $0 < \varepsilon, \delta < 1$ and $\pi : \mathbb{R}^d \to \mathbb{R}^{d'}$ be a JL Transform as in Definition B.2 with target dimension $d' = O\left(\varepsilon^{-2}\log(\varepsilon^{-1}\delta^{-1})\right)$. Then with probability $1-\delta$,*

$$\textsc{Max-Cut}(\pi(X)) \in (1 \pm \varepsilon) \cdot \textsc{Max-Cut}(X). \tag{9}$$

Our bound is not directly comparable with that in [34, 35], since we improve the dependence on δ from $O(\delta^{-2})$ to $O(\log(\delta^{-1}))$, but we also introduced an additional $\log(\varepsilon^{-1})$ factor. Our argument is simpler, achieved by making use of a certain formulation of the Johnson-Lindenstrauss (JL) transform [27], that differs from the one used in [34, 35].

The JL transform formulation that we use (see Definition B.2) is similar to the one previously used in [38], to obtain dimension reduction results for clustering problems. Its second property (the expectation bound), may not hold for all constructions of the JL transform (and was not used in [34, 35]), but it can be realized by a random matrix with independent sub-Gaussian entries.

Definition B.2 (JL Transform [27, 38])**.** For every integer $d, d' \geq 1$, there exists a randomized mapping $\pi : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that for all $x \neq y \in \mathbb{R}^d$,

$$\Pr_{\pi}\Big[\operatorname{dist}(\pi(x),\pi(y)) \notin (1\pm\varepsilon)\cdot\operatorname{dist}(x,y)\Big] \leq e^{-Cd'\varepsilon^2} \tag{10}$$

$$\mathop{\mathbb{E}}_{\pi}\left[\max\left\{\left|\frac{\operatorname{dist}(\pi(x),\pi(y))}{\operatorname{dist}(x,y)} - 1\right| - \varepsilon, 0\right\}\right] \leq e^{-Cd'\varepsilon^2}, \tag{11}$$

for some universal constant $C > 0$.

Our version is similar but not identical to that in [38], because we introduce an absolute value (to bound the error from both sides), but the proof is similar.

Proof of Theorem B.1. Observe that it suffices to show with probability $1-\delta$,

$$\sum_{x,y\in X} |\operatorname{dist}(\pi(x),\pi(y)) - \operatorname{dist}(x,y)| \leq O(\varepsilon) \sum_{x,y\in X} \operatorname{dist}(x,y). \tag{12}$$

Let $P := \{(x,y) \in X : |\operatorname{dist}(\pi(x),\pi(y)) - \operatorname{dist}(x,y)| > \varepsilon \operatorname{dist}(x,y)\}$ be the set of "bad" point pairs whose distances are distorted by more than ε error. Since by definition we have (with probability 1)

$$\sum_{(x,y)\in X\times X\setminus P} |\operatorname{dist}(\pi(x),\pi(y)) - \operatorname{dist}(x,y)| \leq \varepsilon \cdot \sum_{x,y\in X} \operatorname{dist}(x,y),$$

hence, it remains to show the following holds with probability $1-\delta$

$$\sum_{(x,y)\in P} |\operatorname{dist}(\pi(x),\pi(y)) - \operatorname{dist}(x,y)| \leq \varepsilon \sum_{x,y\in X} \operatorname{dist}(x,y). \tag{13}$$

By the guarantee of Definition B.2, we have

$$\begin{aligned}
&\mathbb{E}\Big[\sum_{(x,y)\in P} |\operatorname{dist}(\pi(x),\pi(y)) - \operatorname{dist}(x,y)| - \varepsilon \operatorname{dist}(x,y)\Big] \\
=&\,\mathbb{E}\Big[\sum_{x,y\in P} \max\{|\operatorname{dist}(\pi(x),\pi(y)) - \operatorname{dist}(x,y)| - \varepsilon \operatorname{dist}(x,y), 0\}\Big] \\
\leq&\,\mathbb{E}\Big[\sum_{x,y\in X} \max\{|\operatorname{dist}(\pi(x),\pi(y)) - \operatorname{dist}(x,y)| - \varepsilon \operatorname{dist}(x,y), 0\}\Big] \\
\leq&\, e^{-Cd'\varepsilon^2} \sum_{x,y\in X} \operatorname{dist}(x,y) \leq \varepsilon\delta \sum_{x,y\in X} \operatorname{dist}(x,y).
\end{aligned}$$

Therefore, by Markov's inequality, we conclude that (13) holds with probability $1-\delta$. This finishes the proof. □

Theorem B.1 implies the corollary below about a streaming algorithm that reports an encoding of a near-optimal cut (and not just its value). The most natural way to report a cut of X is to somehow represent of a 2-partition of X, but this is not possible because that contains X itself, which requires $\Omega(n)$ bits to store. Instead, we let the algorithm report a function $f : \mathbb{R}^d \to \{0,1\}$ (using some encoding), and then f implicitly defines the cut $(X \cap f^{-1}(0), X \cap f^{-1}(1))$. In other words, the algorithm essentially reports an "oracle" that does not know X, but can determine, for each input point $x \in X$, its side in the cut. This formulation was suggested by [19], and in fact we rely on their solution and combine it with our dimension reduction.

Corollary B.3 (Cut Oracle)**.** *There is a randomized streaming algorithm that, given $0 < \varepsilon < 1/2$, integers $\Delta, d \geq 1$, and an input dataset $X \subseteq [\Delta]^d$ presented as a dynamic stream, the algorithm uses space $\exp(\operatorname{poly}(\varepsilon^{-1}))\operatorname{poly}(d\log\Delta)$, and reports (an encoding of) a mapping $f : \mathbb{R}^d \to \{0,1\}$, such that with constant probability (at least 2/3), $\operatorname{cut}_X(X \cap f^{-1}(0)) \geq (1-\varepsilon)\cdot\textsc{Max-Cut}(X)$.*

Proof. As noted in [19], there exists an algorithm $\mathcal{A}$ that finds an f with the same guarantee and failure probability, except that the space usage is $\varepsilon^{-O(d)} \cdot \operatorname{poly}(\log\Delta)$. Hence, we can use this $\mathcal{A}$ as a black with Theorem B.1 to conclude the theorem.

Specifically, let $\pi : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that $d' = O(\varepsilon^{-2}\log(\varepsilon^{-1}))$ be a mapping that satisfies Theorem B.1. Then, for every update of point $x \in [\Delta]^d$ in the stream, we map it to $\pi(x)$ and feed it to $\mathcal{A}$. When the stream terminates, we use $\mathcal{A}$ to compute an $f' : \mathbb{R}^{d'} \to \{0,1\}$. Then, to define the final $f : \mathbb{R}^d \to \{0,1\}$ is defined as $f(x) := f'(\pi(x))$. This finishes the proof. □

REFERENCES

[1] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. 2004. Approximating Extent Measures of Points. *J. ACM* 51, 4 (2004), 606–635. https://doi.org/10.1145/1008731.1008736

[2] Pankaj K. Agarwal and R. Sharathkumar. 2015. Streaming Algorithms for Extent Problems in High Dimensions. *Algorithmica* 72, 1 (2015), 83–98. https://doi.org/10.1007/s00453-013-9846-4

[3] Kook Jin Ahn and Sudipto Guha. 2009. Graph Sparsification in the Semi-streaming Model. In *ICALP (2) (Lecture Notes in Computer Science, Vol. 5556)*. Springer, 328–338. https://doi.org/10.1007/978-3-642-02930-1_27

[4] Noga Alon, Wenceslas Fernandez de la Vega, Ravi Kannan, and Marek Karpinski. 2003. Random sampling and approximation of MAX-CSPs. *J. Comput. Syst. Sci.* 67, 2 (2003), 212–243. https://doi.org/10.1016/S0022-0000(03)00008-4

[5] Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David P. Woodruff. 2009. Efficient Sketches for Earth-Mover Distance, with Applications. In *FOCS*. IEEE Computer Society, 324–330. https://doi.org/10.1109/FOCS.2009.25

[6] Sanjeev Arora. 1998. Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems. *J. ACM* 45, 5 (1998), 753–782. https://doi.org/10.1145/290179.290180

[7] Yair Bartal. 1996. Probabilistic Approximations of Metric Spaces and Its Algorithmic Applications. In *FOCS*. IEEE Computer Society, 184–193. https://doi.org/10.1109/SFCS.1996.548477

[8] Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. 2017. Clustering High Dimensional Dynamic Data Streams. In *ICML (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, 576–585. http://proceedings.mlr.press/v70/braverman17a.html

[9] Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge A. Plotkin. 1998. Approximating a Finite Metric by a Small Number of Tree Metrics. In *FOCS*. IEEE Computer Society, 379–388. https://doi.org/10.1109/SFCS.1998.743488

[10] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. 2022. New streaming algorithms for high dimensional EMD and MST. In *STOC*. ACM, 222–233. https://doi.org/10.1145/3519935.3519979

[11] Artur Czumaj, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Pavel Veselý. 2022. Streaming Algorithms for Geometric Steiner Forest. In *ICALP (LIPIcs, Vol. 229)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 47:1–47:20. https://doi.org/10.4230/LIPIcs.ICALP.2022.47

[12] Artur Czumaj, Shaofeng H.-C. Jiang, Robert Krauthgamer, Pavel Veselý, and Mingwei Yang. 2022. Streaming Facility Location in High Dimension via Geometric Hashing. In *FOCS*. IEEE, 450–461. https://doi.org/10.1109/FOCS54457.2022.00050

[13] Artur Czumaj, Christiane Lammersen, Morteza Monemizadeh, and Christian Sohler. 2013. $(1+\varepsilon)$-approximation for facility location in data streams. In *SODA*. SIAM, 1710–1728. https://doi.org/10.1137/1.9781611973105.123

[14] Wenceslas Fernandez de la Vega and Marek Karpinski. 2000. Polynomial time approximation of dense weighted instances of MAX-CUT. *Random Struct. Algorithms* 16, 4 (2000), 314–332.

[15] Wenceslas Fernandez de la Vega and Claire Kenyon. 2001. A Randomized Approximation Scheme for Metric MAX-CUT. *J. Comput. Syst. Sci.* 63, 4 (dec 2001), 531–541. https://doi.org/10.1006/jcss.2001.1772

[16] Dan Feldman and Michael Langberg. 2011. A unified framework for approximating and clustering data. In *STOC*. ACM, 569–578. https://doi.org/10.1145/1993636.1993712

[17] Dan Feldman, Melanie Schmidt, and Christian Sohler. 2020. Turning Big Data Into Tiny Data: Constant-Size Coresets for k-Means, PCA, and Projective Clustering. *SIAM J. Comput.* 49, 3 (2020), 601–657. https://doi.org/10.1137/18M1209854

[18] Gereon Frahling, Piotr Indyk, and Christian Sohler. 2008. Sampling in Dynamic Data Streams and Applications. *Int. J. Comput. Geom. Appl.* 18, 1/2 (2008), 3–28. https://doi.org/10.1142/S0218195908002520

[19] Gereon Frahling and Christian Sohler. 2005. Coresets in dynamic geometric data streams. In *STOC*. ACM, 209–217. https://doi.org/10.1145/1060590.1060622

[20] Michel X. Goemans and David P. Williamson. 1995. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM* 42, 6 (1995), 1115–1145. https://doi.org/10.1145/227683.227684

[21] Oded Goldreich, Shafi Goldwasser, and Dana Ron. 1998. Property Testing and its Connection to Learning and Approximation. *J. ACM* 45, 4 (1998), 653–750. https://doi.org/10.1145/285055.285060

[22] Sariel Har-Peled and Soham Mazumdar. 2004. On coresets for k-means and k-median clustering. In *STOC*. ACM, 291–300. https://doi.org/10.1145/1007352.1007400

[23] Wei Hu, Zhao Song, Lin F. Yang, and Peilin Zhong. 2018. Nearly Optimal Dynamic k-Means Clustering for High-Dimensional Data. *CoRR* abs/1802.00459 (2018).

[24] Piotr Indyk. 2004. Algorithms for dynamic geometric problems over data streams. In *STOC*. ACM, 373–380. https://doi.org/10.1145/1007352.1007413

[25] Rajesh Jayaram and David Woodruff. 2021. Perfect L_p Sampling in a Data Stream. *SIAM J. Comput.* 50, 2 (2021), 382–439. https://doi.org/10.1137/18M1229912

[26] T. S. Jayram, Ravi Kumar, and D. Sivakumar. 2008. The One-Way Communication Complexity of Hamming Distance. *Theory Comput.* 4, 1 (2008), 129–135. https://doi.org/10.4086/toc.2008.v004a006

[27] W. B. Johnson and J. Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*. Amer. Math. Soc., 189–206.

[28] Hossein Jowhari, Mert Saglam, and Gábor Tardos. 2011. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In *PODS*. ACM, 49–58. https://doi.org/10.1145/1989284.1989289

[29] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. 2010. An optimal algorithm for the distinct elements problem. In *PODS*. ACM, 41–52. https://doi.org/10.1145/1807085.1807094

[30] Michael Kapralov and Dmitry Krachun. 2019. An optimal space lower bound for approximating MAX-CUT. In *STOC*. ACM, 277–288. https://doi.org/10.1145/3313276.3316364

[31] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. 2007. Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs? *SIAM J. Comput.* 37, 1 (2007), 319–357. https://doi.org/10.1137/S0097539705447372

[32] Ilan Kremer, Noam Nisan, and Dana Ron. 1999. On Randomized One-Round Communication Complexity. *Comput. Complex.* 8, 1 (1999), 21–49. https://doi.org/10.1007/s000370050018

[33] Eyal Kushilevitz and Noam Nisan. 1997. *Communication Complexity*. Cambridge University Press.

[34] Christiane Lammersen. 2011. *Approximation Techniques for Facility Location and Their Applications in Metric Embeddings*. Ph. D. Dissertation. TU Dortmund. https://doi.org/10.17877/DE290R-8506

[35] Christiane Lammersen, Anastasios Sidiropoulos, and Christian Sohler. 2009. Streaming Embeddings with Slack. In *WADS (Lecture Notes in Computer Science, Vol. 5664)*. Springer, 483–494. https://doi.org/10.1007/978-3-642-03367-4_42

[36] Christiane Lammersen and Christian Sohler. 2008. Facility Location in Dynamic Geometric Data Streams. In *ESA (Lecture Notes in Computer Science, Vol. 5193)*. Springer, 660–671. https://doi.org/10.1007/978-3-540-87744-8_55

[37] Sepideh Mahabadi, Ilya P. Razenshteyn, David P. Woodruff, and Samson Zhou. 2020. Non-adaptive adaptive sampling on turnstile streams. In *Proccedings of the 52nd Annual ACM Symposium on Theory of Computing, STOC 2020*. ACM, 1251–1264. https://doi.org/10.1145/3357713.3384331

[38] Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. 2019. Performance of Johnson-Lindenstrauss transform for k-means and k-medians clustering. In *STOC*. ACM, 1027–1038. https://doi.org/10.1145/3313276.3316350

[39] Morteza Monemizadeh and David P. Woodruff. 2010. 1-Pass Relative-Error L_p-Sampling with Applications. In *Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '10)*. SIAM, 1143–1160. https://doi.org/10.1137/1.9781611973075.92

[40] Mark Rudelson and Roman Vershynin. 2007. Sampling from large matrices: An approach through geometric functional analysis. *J. ACM* 54, 4 (2007), 21. https://doi.org/10.1145/1255443.1255449

[41] L. J. Schulman. 2000. Clustering for edge-cost minimization. In *32nd Annual ACM Symposium on Theory of Computing* (Portland, Oregon, United States). ACM, 547–555. https://doi.org/10.1145/335305.335373

[42] David P. Woodruff and Taisuke Yasuda. 2022. High-Dimensional Geometric Streaming in Polynomial Space. In *FOCS*. IEEE, 732–743. https://doi.org/10.1109/FOCS54457.2022.00075

[43] Hamid Zarrabi-Zadeh. 2011. An Almost Space-Optimal Streaming Algorithm for Coresets in Fixed Dimensions. *Algorithmica* 60, 1 (2011), 46–59. https://doi.org/10.1007/s00453-010-9392-2

Received 2022-11-07; accepted 2023-02-06