Multipass Linear Sketches for Geometric LP-Type Problems

N. Efe Çekirge[1]
Department of Computer Science, Dartmouth College, Hanover, NH, USA

William Gay[1]
Grainger College of Engineering, University of Illinois, Urbana-Champaign, IL, USA

David P. Woodruff
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

LP-type problems such as the Minimum Enclosing Ball (MEB), Linear Support Vector Machine (SVM), Linear Programming (LP), and Semidefinite Programming (SDP) are fundamental combinatorial optimization problems, with many important applications in machine learning applications such as classification, bioinformatics, and noisy learning. We study LP-type problems in several streaming and distributed big data models, giving ε-approximation linear sketching algorithms with a focus on the high accuracy regime with low dimensionality d, that is, when $d < (1/\varepsilon)^{0.999}$. Our main result is an $O(ds)$ pass algorithm with $O(s(\sqrt{d}/\varepsilon)^{3d/s}) \cdot \text{poly}(d, \log(1/\varepsilon))$ space complexity in words, for any parameter $s \in [1, d\log(1/\varepsilon)]$, to solve ε-approximate LP-type problems of $O(d)$ combinatorial and VC dimension. Notably, by taking $s = d\log(1/\varepsilon)$, we achieve space complexity polynomial in d and polylogarithmic in $1/\varepsilon$, presenting exponential improvements in $1/\varepsilon$ over current algorithms. We complement our results by showing lower bounds of $(1/\varepsilon)^{\Omega(d)}$ for any 1-pass algorithm solving the $(1+\varepsilon)$-approximation MEB and linear SVM problems, further motivating our multi-pass approach.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Streaming, sketching, LP-type problems

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2025.8

Category APPROX

Related Version *Full Version*: https://arxiv.org/abs/2507.11484

Funding *David P. Woodruff*: Supported by a Simons Investigator Award and Office of Naval Research award number N000142112647.

1 Introduction

LP-type problems are a class of problems fundamental to the field of combinatorial optimization. Introduced originally by Shahir and Welzl, they are defined by a finite set of elements S and a solution function f which maps subsets of S to a totally ordered domain and also satisfies the properties of locality and monotonicity [42]. Examples of LP-type problems include the Minimum Enclosing Ball (MEB) problem, the Linear Support Vector Machine(SVM) problem, and linear programming (LP) problems for which the class is named. These problems have many applications in different fields such as machine learning. For example, the MEB problem has applications in classifiers [12], and support vector machines [13, 45]. The linear SVM problem is a fundamental problem in machine learning, used in many classification problems such as text classification, character recognition, and bioinformatics [17]. Linear programs have many uses in data science problems, with [35]

[1] This work was done while at Carnegie Mellon University.

showing that most common machine learning algorithms can be expressed as LPs. One such example is the linear classification problem, which has applications in many noisy learning and optimization problems [14, 11, 25]. A related class of problems are semidefinite programming (SDP) problems, which are useful for many machine learning tasks such as distance metric learning [50] and matrix completion [16, 41, 15].

For these applications, the size of the input problem can be very large, and often too expensive to store in local memory. Further, the data set can be distributed between multiple machines which require a joint computation. In these applications, the large size of data generally means that finding exact solutions to problems is prohibitively expensive in space or communication complexity, and such many algorithms instead keep and perform computations on a small representation of the data which allows them to give an answer within a small approximation factor of the original solution.

One such way to represent data is with linear sketches. For a large dataset P of n elements from a universe of size U, which can be thought of as a vector $\boldsymbol{U} \in \{0,1\}^U$ where $\boldsymbol{U}_{\boldsymbol{p}}$ is 1 for each element $\boldsymbol{p} \in P$ and 0 otherwise, a linear sketch generally is in the form of an $N \times U$ matrix A, for $N \ll n$. Thus, one only has to store $A \cdot \boldsymbol{U}$, a vector of size N, which is much cheaper than storing the full n element data set. Linear sketches are very useful in many big data applications, such as data streams where data points are read one by one, or parallel computation where the data set is split among different machines. This is because updating the data vector $\boldsymbol{U} \leftarrow \boldsymbol{U} + \boldsymbol{e}_{\boldsymbol{p}}$ for a newly read data point $\boldsymbol{p}$, where $\boldsymbol{e}_{\boldsymbol{p}}$ is the $\boldsymbol{p}^{\text{th}}$ standard unit vector, simply requires updating the sketch as $A \cdot \boldsymbol{U} + A_{\cdot \boldsymbol{p}}$ for the $\boldsymbol{p}^{\text{th}}$ column of A, $A_{\cdot \boldsymbol{p}}$. Similarly, for data distributed among k machines as $P = \sum_{i \in [k]} P_k$, the sketch can be maintained separately by each machine, since $A \cdot P = \sum_{i \in [k]} A \cdot P_i$. As linear sketches consist of matrix-vector products, they are useful in GPU-based applications, which are particularly suited for these operations. Linear sketching algorithms have extensive study in big data applications, see [49, 33, 5] for some examples and surveys. An extension which we build our algorithm in is the multiple linear sketch model, which corresponds to making adaptive matrix-vector products in each of a small number of rounds [44].

1.1 Our Contributions

In this paper, we explore using linear sketches to solve approximate LP-type problems in high accuracy regimes, in which $d < (1/\varepsilon)^{0.999}$, where d is the dimensionality of the problem and ε is the approximation accuracy. This regime is applicable to many real world big data applications where $1/\varepsilon$ might be several orders of magnitude larger than d, such as machine learning tasks of regression or classification. Take for example [38], where $d = 2$, $1/\varepsilon \in \left[10^2, 10^5\right]$, and $n = 10^5$, or [45] where for some data sets $1/\varepsilon$ is taken to be much larger than d yet small compared to n.

We present an algorithm in the multiple linear sketch model for solving certain classes of LP-type problems, which can be utilized in several prominent big data models. The multiple linear sketch model, which we define further in Section 1.2, can be thought of as a multipass extension of the linear sketch model, where one is allowed to choose a fresh linear sketch in each pass of a multipass streaming algorithm in order to have a better representation of the data. There are two significant benefits of our sketching approach. Firstly, our algorithm works in the presence of input point deletions, such as in the turnstile model. Secondly, it allows our algorithm to work in the presence of duplicated points in the input, which generally can prove problematic for sampling based algorithms.

We are given a universe $S = \{-\Delta, -\Delta + 1, \ldots, \Delta\}^d$ of d-dimensional points, where each point can be represented by a word of size $d \log(\Delta n)$. This defines our cost model, and we give our space complexity in the number of points/words. We use a linear sketch to reduce an

input set $P \subseteq S$ of n input points into a point set $Q = \left\{(1+\varepsilon)^l \cdot \boldsymbol{e} \mid \boldsymbol{e} \in E_{\boldsymbol{p}_c}, l \in \left[\log_{1+\varepsilon} \delta\right]\right\}$, where $E_{\boldsymbol{p}_c}$ is a metric ε-net centered at a point $\boldsymbol{p}_c \in P$ and δ is an $O(1)$ approximation for the distance of the furthest point in P to $\boldsymbol{p}_c$. Our algorithm is also applicable when $S = \left\{-1, -1 + \frac{1}{\Delta}, \ldots, 1\right\}^d$. Now, we reduce P directly into the point set $Q = E$ where E is the metric ε-net centered at the origin. Intuitively, this has the effect of snapping each input point to the closest of a small number of net points, thereby reducing our input space. This linear sketch can be though of as an $N \times |S|$ matrix A for $N = l\,|E| \ll n$ where each row corresponds to one possible point $(1+\varepsilon)^l \cdot \boldsymbol{e}$, containing 1's for all the points $\boldsymbol{p} \in S$ with direction closest to $\boldsymbol{e}$ and norm closest to $(1+\varepsilon)^l$. The major benefit of using this linear sketch is that we don't actually store our metric ε-net. We can calculate where each point is snapped to quickly on the fly, so we do not suffer the exponential in d space complexity of ε-kernel based formulations.

This linear sketch of the data can now be used to run our algorithm on, treating it as a smaller virtual input compared to the original input. Our general algorithm for solving LP-type problems follows the sampling idea presented by [8], which in turn is based on Clarkson's algorithm for linear programs in low dimensions [21], where input points are assigned weights, and a weighted sampling procedure is used in order to solve the LP-type problem on smaller samples from the input. By modifying the weights throughout multiple iterations, the algorithm quickly converges on samples that contain a basis for the LP-type problem, for which the solution is the solution of the entire input set.

As our algorithm is based on linear sketches, we can utilize it in many big data models. We can solve problems on input streams that we can make multiple linear scans over in the *Multipass Streaming* model. Additionally, our algorithm works for streams with deletions, such as in the *Multipass Strict Turnstile* model. Finally, we can also formulate our algorithm for distributed inputs, minimizing communication load in the *Coordinator* and *Parallel Computation* models. A usage of these models is seeing the symmetric difference of data over time across multiple servers. Over two servers with large datasets, communication can be thought of as passes over a stream with deletions. Further, we can find the symmetric difference of server datasets in the Parallel Computation model as well. This is helpful in capturing change in data to predict current trends, see e.g. [24].

We give the following result for LP-type problems that are bounded in combinatorial and VC-dimension. More detailed bounds and time bounds are given in Sections A.1, A.2, A.3, and A.4.

▶ **Theorem 1.** *For any $s \in [1, \log N]$, there exists a randomized algorithm to compute a $(1+O(\varepsilon))$-approximation solution to an LP-type problem with VC and combinatorial dimensions in $O(d)$ with high probability, using a linear sketch of dimensionality N where $\log N \in \operatorname{poly}(d, \log(1/\varepsilon))$, where $d \log(n\Delta)$ is the bit complexity to store one word/point $\boldsymbol{p} \in P$, and where a solution $f(\cdot)$ can be stored in $O(1)$ words. This algorithm can be implemented in various models as such.*

- ***Multipass Streaming**: An $O(ds)$ pass algorithm with $O\left(sN^{3/s}\right) \cdot \operatorname{poly}(d, \log N)$ space complexity in words.*
- ***Strict Turnstile**: An $O(ds + \log\log(\Delta d))$ pass algorithm with $O\left(sN^{3/s}\right) \cdot \operatorname{poly}(d, \log N)$ space complexity in words.*
- ***Coordinator**: An $O(ds)$ round algorithm with $O\left(\operatorname{poly}\left(d, N^{1/s}\right) + k\right)$ load in words.*
- ***Parallel Computation**: An $O(ds)$ round algorithm with $O\left(\operatorname{poly}\left(d, N^{1/s}\right) + k\right)$ load in words.*

For most problems, we have that $N \in (1/\varepsilon)^{O(d)}$. With this, we get the following corollary.

▶ **Corollary 2.** *For any $s \in [1, d\log(1/\varepsilon)]$, there exists a randomized algorithm to compute a $(1+O(\varepsilon))$-approximation solution to an LP-type problem with VC and combinatorial dimensions in $O(d)$ with high probability, taking $O(ds)$ passes and $O\big(s\big(\sqrt{d}/\varepsilon\big)^{3d/s}\big) \cdot \mathrm{poly}(d, \log(1/\varepsilon))$ space complexity in words.*

Our algorithm provides powerful results when we set the parameter s equal to $\log N$. As we have that $\log N \in \mathrm{poly}(d, \log(1/\varepsilon))$, this allows us to achieve algorithms with pass and space complexities polynomial in d and polylogarithmic in $1/\varepsilon$.

We can use this general algorithm to find approximate solutions for many LP-type problems which satisfy some elementary properties, as described in Section 3. We emphasize that most LP-type problems fulfill these properties. We present applications of our general algorithm for the following problems. Note that the Linear SVM, Bounded LP, and Bounded SDP problems do not have the $\log\log(\Delta d)$ pass increase in the strict turnstile model.

- **MEB**: Given n points $P \subseteq \{-\Delta, -\Delta+1, \ldots, \Delta\}^d$, our objective is to find a minimal ball enclosing the input points. We present an algorithm to output a ball that encloses P with radius at most $(1+O(\varepsilon))$ times optimal. For the MEB problem, we have $N \in (1/\varepsilon)^{O(d)}$.
- **Linear SVM**: Given n labeled points $P \subseteq \big\{-1., -1+\frac{1}{\Delta}, \ldots, 1\big\}^d \times \{\pm 1\}$ and some $\gamma > 0$ for which the points are either linearly inseparable or γ-separable, our objective is to find a separating hyperplane of P with the largest margin. We present an algorithm to output either that the points are linearly inseparable, or a separating hyperplane $(\boldsymbol{u}, b)$ where $\|\boldsymbol{u}\|^2$ is at most $(1+O(\varepsilon))$ times optimal. For the linear SVM problem, we have $N \in (1/\varepsilon\gamma)^{O(d)}$.
- **Bounded LP**: Given an objective vector $\boldsymbol{c} \in \big\{-1, -1+\frac{1}{\Delta}, \ldots, 1\big\}^d$ such that $\|\boldsymbol{c}\| \in O(1)$ and n constraints $P \subseteq \big\{-1., -1+\frac{1}{\Delta}, \ldots, 1\big\}^{d+1}$, where for each constraint $(\boldsymbol{a}_i, b_i) \in P$, $\|\boldsymbol{a}_i\|, |b_i| \in O(1)$, our objective is to find a solution $\boldsymbol{x}$ with $\|\boldsymbol{x}\| \in O(1)$ that maximizes $\boldsymbol{c}^T\boldsymbol{x}$ while satisfying the constraints. We present an algorithm to output a solution $O(\varepsilon)$ within each constraint, and for which the objective is at most $O(\varepsilon)$ smaller than optimal. For bounded LP problems, we have $N \in (1/\varepsilon)^{O(d)}$.
- **Bounded SDP**: Given an objective matrix $C \in \big\{-1., -1+\frac{1}{\Delta}, \ldots, 1\big\}^{d\times d}$ such that $\|C\|_F \le 1$ and n constraints $P \subseteq \big\{-1., -1+\frac{1}{\Delta}, \ldots, 1\big\}^{d\times d} \times \big\{-1, -1+\frac{1}{\Delta}, \ldots, 1\big\}$, where for each constraint $(A^{(i)}, b^{(i)}) \in P$, $\big\|A^{(i)}\big\|_2, \big|b^{(i)}\big| \le 1$, $\big\|A^{(i)}\big\|_F \le F$, and the number of nonzero entries of $A^{(i)}$ is bounded by S, our objective is to find a positive semidefinite solution X of unit trace that maximizes the objective $\langle C, X\rangle_F$ while satisfying the constraints. We present an algorithm to output a solution that is $O(\varepsilon)$ within each constraint, and for which the objective is at most $O(\varepsilon)$ smaller than optimal. As we are working with $d \times d$ matrices, the combinatorial and VC dimensions are $O(d^2)$, and we have $N \in \binom{d^2}{S} \cdot (1/\varepsilon)^{O(S)} + (1/\varepsilon)^{O(d)}$. $\|\cdot\|_2$, $\|\cdot\|_F$, and $\langle\cdot,\cdot\rangle_F$ refer to the spectral and Frobenius norms and Frobenius inner product respectively, explained further in Appendix A.8.

Table 1 compares our results to those of prior work, including the *exact* LP-type problem algorithm of [8], and various ε-approximation LP-type problems.

For the approximate MEB problem, there are many results, as discussed in Section 1.2. One such is the Zarrabi-Zadeh result, which is among many that utilize an ε-kernel [53]. While these algorithms are single pass, they require space complexity of $(1/\varepsilon)^{O(d)}$. Thus, we present a large increase in space efficiency over them. We also present a lower bound on the space complexity for any one-pass algorithm for the MEB problem of $(1/\varepsilon)^{\Omega(d)}$ in Section 4, further motivating our multi-pass approach. While the result by Bâdoiu and Clarkson has space complexity polynomial in d, it is also polynomial in $1/\varepsilon$ in passes and space [9]. Thus, in the high accuracy regime, we present an exponential improvement over their result.

Table 1 Result Comparisons for Various LP-type Problems.

Problem	Ref.	Pass Complexity	Space Complexity (words)
LP-Type Problems	[8]	$O(d \log n)$	$\text{poly}(d, \log n)$
	Ours	$O\left(d^2 \log(1/\varepsilon)\right)$	$\text{poly}(d, \log(1/\varepsilon))$
LP	[34]	$\tilde{O}\left(\sqrt{d} \log(1/\varepsilon)\right)$	$\tilde{O}\left(d^2\right)$
	Ours	$O\left(d^2 \log(1/\varepsilon)\right)$	$\text{poly}(d, \log(1/\varepsilon))$
MEB	[53]	1	$O\left((1/\varepsilon)^{\frac{d-1}{2}} \log(1/\varepsilon)\right)$
	[9]	$\lceil 2/\varepsilon \rceil$	$O(d/\varepsilon)$
	Ours	$O\left(d^2 \log(1/\varepsilon)\right)$	$\text{poly}(d, \log(1/\varepsilon))$
Linear Classification	[23]	$\tilde{O}(1/\varepsilon)$	$\tilde{O}\left(1/\varepsilon^2\right)$
	Ours	$O\left(d^2 \log(1/\varepsilon)\right)$	$\text{poly}(d, \log(1/\varepsilon))$
Bounded SDP	[43]	$O\left(\sqrt{d} \log(1/\varepsilon)\right)$	$O\left(n^2 + d^2\right)$
	Ours	$O\left(d^4 \log(1/\varepsilon)\right)$	$\text{poly}(d, \log(1/\varepsilon))$
	[26]	$O\left(\frac{1}{\varepsilon^2} \log n \left(\begin{smallmatrix} F^2(n+\log d)+\frac{1}{\varepsilon} S \log d \\ +\min\left(\frac{1}{\varepsilon^2} S \log d, d^2\right) \frac{1}{\varepsilon^2} \log d \end{smallmatrix}\right)\right)$ **time complexity**	
	Ours	$\tilde{O}\left(d^2 S \log\left(\frac{1}{\varepsilon}\right)\left(n + d^{4.7437}\right)\right)$ **time complexity**	

Another approach to the $(1+\varepsilon)$-MEB problem is by using the ellipsoid algorithm with a similar metric ε-net construction. While this algorithm also can achieve $\text{poly}(d, \log(1/\varepsilon))$ pass and space complexity, we present two main improvements over using this approach. Most importantly, as our algorithm is in the multiple linear sketch model, we are able to apply our result to streams with addition and removal of points, and to models where the input is distributed. As the ellipsoid algorithm is not in the multiple linear sketch model, it cannot, for example, find the MEB of the symmetric difference of pointsets from two machines. Further, the bulk of our algorithm utilizes ℓ_0 samplers and estimators and simple arithmetic, which can allow our algorithm to be practical to implement.

A strong result to linear programs is by [34]. They use the interior point method for solving linear programs in order to get their result. There are two significant differences of our result to theirs. Firstly, as with the ellipsoid method, while this result might present an improvement in the multipass streaming model, our algorithm is able to handle point additions and deletions, as well as distributed inputs. Secondly, their tilde notation hides logarithmic dependence on n. Our result completely removes this dependence in both the pass and space complexities, showing that n is not an inherent parameter in this problem.

Our improvements over the work by [8] is similar. By using linear sketches we are able to extend their results to models with point additions and deletions such as the turnstile model, and additionally deal with duplicate points due to our usage of ℓ_0 samplers and estimators.

As we present in Section A.7.1, our algorithm can be used to solve the linear classification problem. Note that our result finds an exact classifier. We compare our result to the result by [23]. In the high accuracy regime, this represents an exponential improvement over their result in pass and space complexity.

Another application of our framework is for additive ε-approximation bounded SDP problems. As we present in Section A.8.1, our formulation can be used to solve for the saddle point problem within the unit simplex. For this problem, there is a result by [26]

in the number of arithmetic operations. We achieve an exponential improvement in $1/\varepsilon$ dependence over their result, significant in the high accuracy regime. Another result is by [43] who achieve a significant result for SDPs with high dimensionality. In the high accuracy regime with a large number of constraints $n \gg 1/\varepsilon$, this presents a loss in number of passes polynomial in d, while achieving a significant gain in space complexity, as our result does not depend on n at all.

1.2 Related Work

Approximation algorithms for LP-type problems in big data models have been well-studied, where the trade-off between approximation ratio, number of passes, and space complexity has given rise to different techniques being used for different regimes.

For the MEB problem, there is the folklore result of a 1-pass algorithm, where one stores the first point and a running furthest point from it, giving a 2-approximation. There is also a classic 1-pass 3/2-approximation algorithm by Zarrabi-Zadeh and Chan, that uses $O(d)$ space complexity [54]. Allowing $O(d)$ passes, an algorithm by Agarwal and Sharathkumar, with further analysis by Chan and Pathak, achieves a 1.22-approximation with $O\left(\left(d/\varepsilon^3 \log\left(1/\varepsilon\right)\right)\right)$ space complexity [2, 19]. For the $(1+\varepsilon)$-approximation problem, much work is done in the single-pass model. These algorithms are based on storing extremal points, called an ε-kernel, and achieve a space complexity of $(1/\varepsilon)^{O(d)}$ [1, 18, 3, 53]. There is also a body of work on bounding the size of these ε-kernels, as they are useful to achieve low-space representations of large data [32, 10, 52, 22]. [7] presents a lower bound of $(1/\varepsilon)^{\Omega(d)}$ on the size of an ε-kernel. In the multi-pass model, there is the classic result by Bâdoiu and Clarkson [9]. This algorithm is widely used in big data models, for its linear dependence on d/ε for space and $1/\varepsilon$ for pass complexity, and simplicity [38, 45].

Linear programs are frequently studied in big data models, with a major focus on exact multi-pass LP results. A significant recent result is by [8], who in the multi-pass model achieve a result with $O(d \log n)$ passes and poly $(d, \log n)$ space usage. Their result is also applicable to other LP-type problems as well as parallel computation models. Their algorithm builds on Clarkson's sampling based method [21], and utilizes reservoir sampling to sample a μ-net [20]. We build on the ideas presented by [8], to solve LP-type problem approximations. There is also recent work on multipass approximations for LPs, with a result by [34] who achieve an $\tilde{O}\left(\sqrt{d}\log\left(1/\varepsilon\right)\right)$ pass $\tilde{O}\left(d^2\right)$ space algorithm for solving $(1-\varepsilon)$-approximation LPs. For packing LPs [4] gives a $(1+\varepsilon)$-approximation algorithm, and for covering LPs [28] gives a $(1+\varepsilon)$-approximation algorithm. The linear classification problem is also well studied, as a fundamental machine learning problem. Its classic result is the mistake bound of the perceptron algorithm [39]. More recently, a result by [23] achieves $O\left(1/\varepsilon^2 \log n\right)$ iterations to find a linear classifier.

There is a lot of study on approximate semidefinite programs, focusing on different regimes. [26] give a result on solving bounded SDP saddle point problems with certain assumptions about the problem input. We use similar assumptions to compare our results to theirs. For the regime with a low number of constraints and a large dimension, [43] achieves an $O\left(\sqrt{d}\log d/\varepsilon\right)$ pass algorithm with $\tilde{O}\left(n^2+d^2\right)$ space complexity.

2 Preliminaries

Multiple Linear Sketch Model. In the multiple linear sketch model, an algorithm is able to create fresh linear sketches of the data through each iteration of a multiple iteration model, such as the multipass streaming or multipass strict turnstile models, based on information

from the previous iterations. This model has been used implicitly for many adaptive data stream algorithms, such as finding heavy hitters, sparse recovery, and compressed sensing [29, 40, 37].

ℓ_0 Estimator.

▶ **Theorem 3** (Theorem 10 from [31]). *There is a 1-pass linear sketch which given additive and subtractive updates to an underlying vector $\boldsymbol{v}$ in $\{0, 1, \ldots, \text{poly}(n)\}^N$, gives a $(1 \pm \zeta)$-approximation to the number of non-zero coordinates of v, denoted $\ell_0(v)$, with error probability δ, and using $O(\frac{\log N \log \delta^{-1}}{\zeta^2})$ words of memory.*

ℓ_0 Sampler.

▶ **Theorem 4** (Theorem 2 from [30]). *There is a 1-pass linear sketch which given additive and subtractive updates to an underlying vector $\boldsymbol{v}$ in $\{0, 1, \ldots, \text{poly}(n)\}^N$, outputs a coordinate i in the set of non-zero coordinates of $\boldsymbol{v}$, where for each non-zero coordinate j in $\boldsymbol{v}$, we have $i = j$ with probability $\frac{1}{\ell_0(v)} \pm \delta$, using $O(\log N \log \delta^{-1})$ words of memory.*

▶ Note. Notice that for the ℓ_0 estimator and sampler, the space complexity in words is logarithmic in the dimensionality of the underlying vector which in our case is $N \ll n$. While the entries of the vector can be polynomial in n, the number of words required to store the estimator and sampler are independent of n.

Combinatorial Dimension. The combinatorial dimension of a problem is the maximum cardinality of a basis for the problem, denoted ν. For LP-type problems, we consider a basis as the minimum number of points required to define a solution to the problem.

VC Dimension. Given a set X and a collection of subsets of X $\mathcal{R}$, known as a *set system* $(X, \mathcal{R})$, and $Y \subseteq X$, the projection of $\mathcal{R}$ on Y is defined as $\mathcal{R}|_Y := \{Y \cap R \mid R \in \mathcal{R}\}$. The VC dimension of $(X, \mathcal{R})$ is the minimum integer λ where $|\mathcal{R}|_Y| < 2^{|Y|}$ for any finite subset Y such that $|Y| > \lambda$ [47].

Metric ε-net. Given a metric space (M, d), a metric ε-net is a set of points $E \subseteq M$ such that for any point $\boldsymbol{p} \in M$, there exists a point $\boldsymbol{e} \in E$ for which $d(\boldsymbol{p}, \boldsymbol{e}) \leq \varepsilon$. For large metric spaces, i.e. when $M = \{-\Delta, -\Delta + 1, \ldots, \Delta\}^d$, this formulation requires too many points. In this case, we give an alternate formulation. In the case where M is large, given a center point $\boldsymbol{p}_c$, which we consider to be the origin, and a corresponding norm $\|\boldsymbol{q}\| = d(\boldsymbol{q}, \boldsymbol{p}_c)$ for each point $\boldsymbol{q} \in M$, a metric ε-net is a set of points where for any point $\boldsymbol{q} \in M$, there exists a point $\boldsymbol{e} \in E$ for which $d(\boldsymbol{q}, \|\boldsymbol{q}\| \boldsymbol{e}) \leq \varepsilon \|\boldsymbol{q}\|$.

ϵ-net. Given a set system $(X, \mathcal{R})$, a weight function $w : X \to \mathbb{R}$, and a parameter $\epsilon \in [0, 1]$, a set $N \subseteq X$ is an ϵ-net with respect to $w(\cdot)$ if $N \cap R \neq \emptyset$ for all sets $R \in \mathcal{R}$ such that $w(R) \geq \epsilon \cdot w(X)$ [27].

▶ **Lemma 5** (Corollary 3.8 from [27]). *For any set system $(X, \mathcal{R})$ of VC dimension $d < \infty$, finite $A \subseteq X$, and $\epsilon, \delta > 0$, if N is the set of distinct elements of A obtained by*

$$m \geq \max\left(\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{8d}{\epsilon}\log\frac{8d}{\epsilon}\right)$$

random independent draws from A, then N is an ϵ-net for $\mathcal{R}$ with probability $1 - \delta$.

While this construction in its original presentation by Haussler and Welzl [27] applies to the trivial weight function $w(x) = 1$, by drawing with probability proportional to an element's weight, an ϵ-net with respect to $w(\cdot)$ can be obtained for any weight function [8].

▶ Note. Metric ε-nets are a special case of ϵ-nets, but we prefer to distinguish them for clarity. Further, to not cause confusion between metric ε-nets and ϵ-nets, we refer to the latter as μ-nets.

3 Algorithm

In this section, we present our algorithm for Theorem 1, in the multiple linear sketch model. It is inspired by the algorithm presented in [8], which in turn builds on the ideas of Clarkson in [21]. For a constrained optimization problem such as Linear Programming of small dimensionality, Clarkson's presented idea is to repeatedly randomly sample a subset of the constraints using a weighted sampling method. This allows the algorithm to solve a smaller instance of the problem instead, which allows for smaller space usage. The key idea is that each iteration of the algorithm updates the weights multiplicatively which allows convergence into better samples, ending with a sample containing a basis. [8] utilizes this procedure, with the additional usage of a μ-net in their sampling procedure allowing for an extension to general LP-type problems of small dimensionality. They also build their weight function by storing solutions from previous passes, thus keeping their storage need small.

These previous algorithms are exact, and thus require pass and space complexities dependent on the number of points in the original stream, n. We remove this dependence on n by employing the use of linear sketches, where we snap the space of input points to a smaller space of discrete points formed by a metric ε-net. This allows us to compute ε-approximation solutions to the problems while using much lower storage complexity.

As we utilize the ideas of [8] to solve LP-type problems, we retain the needed properties for LP-type problems from their result, presented here as properties 1 and 2. Further, we add a third property that must be fulfilled, property 3. We stress that most natural LP-type problems follow properties 1 and 2, and that property 3 follows naturally for most if not all ε-approximation solutions to these problems. Our algorithm requires an LP-type problem (S, f) to satisfy the following:

(P1) Each element $\boldsymbol{x} \in S$ is associated with a set of elements $S_{\boldsymbol{x}} \subseteq R$ where R is the range of f.

(P2) For all $P \subseteq S$, $f(P)$ is the minimal element of $\bigcap_{\boldsymbol{x} \in P} S_{\boldsymbol{x}}$.

(P3) For all $P, Q \subseteq S$ where Q is the set of points resultant of snapping all the points $\boldsymbol{p} \in P$ to a metric ε-net, a $(1 + O(\varepsilon))$-approximation to $f(Q)$ is a $(1 + O(\varepsilon))$-approximation to $f(P)$.

As an example, consider the MEB problem where S is the set of all d-dimensional points, R is the set of all d-dimensional balls, and f is the function that returns the MEB of its input. For properties 1 and 2, each point is associated with the set of all balls that enclose it. For any set of points, the intersection of these sets of balls are precisely the set of balls that enclose them, and thus their MEB is the minimal element of this set. Property 3 follows intuitively because "unsnapping" a set of points from a metric ε-net can only increase the size of their MEB by a small amount. A formal proof of this property for the MEB problem is given in Appendix A.5.

Our algorithm is as such. First, we define the metric ε-net we use. We start by taking any point $\boldsymbol{p}_c$ as our "origin". For example, in the multipass streaming model, this can be the first point. We create a lattice covering the unit d-cube around $\boldsymbol{p}_c$ (which contains the

unit d-sphere) by allowing, for each d dimensions, the values $-1 + i\varepsilon/\sqrt{d}$ where i ranges in $\left[0, 2\sqrt{d}/\varepsilon\right]$. Thus, any point that is scaled to the unit d-sphere will be at most $\varepsilon/2$ away from a point in our metric ε-net. The size of this metric ε-net is $\left(1 + \frac{2\sqrt{d}}{\varepsilon}\right)^d$.

The crux of our algorithm is therefore to use this metric ε-net to create a linear sketch of the input. Given any point in the input $\boldsymbol{p} \in P$, we get the point $\frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}$ and snap it to its closest metric ε-net point. Now, we scale it back up, but we round down the norm to $(1+\varepsilon)^l$ for some $l \in \mathbb{N}$. This has the effect of discretizing our point space along the metric ε-net directions, so that our linear sketch has size bounded by $\left\lceil \log_{1+\varepsilon}\left(\max_{\boldsymbol{p}\in P} \|\boldsymbol{p}\|\right)\right\rceil \left(1 + \frac{2\sqrt{d}}{\varepsilon}\right)^d$. In some problems, we are presented input points already inside the unit d-cube, or a d-cube of bounded size. In these cases, we only need to snap the input points to the closest metric ε-net point. We denote the size of our net as N and maintain that it is small compared to the size of the input, n. We stress that we never have to store this net, as the net points are immediately fed into a linear sketch which projects the set of net points to a sketching dimension which is polylogarithmic in the number of net points.

▶ Note. For the sake of clarity, we will refer to the input points $\boldsymbol{p} \in P$ as original points, and these new points $\boldsymbol{q} \in Q$ as snapped points.

Before we explore the algorithm proper, we note a side effect of creating this sketch. Shrinking the point space means that distinct points in the original input can now map to the same point. As our algorithm includes weighted sampling over the snapped points, we must treat these as one point and not two overlapping points. In order to achieve this, we utilize ℓ_0 estimators and samplers, as defined in [31] and [30] respectively, which allow us to consider estimates on points and sample from them without running into the problem of overlapping. We note that these ℓ_0 estimators and samplers are linear sketches, and such are able to be used in the models present in our algorithm.

Our algorithm then proceeds in iterations of weighted sampling, in a similar fashion to [8]. Throughout, we maintain a weight function $w : Q \to \mathbb{R}$. In order to save space, we maintain this weight function not explicitly, but by storing previous successful solutions, as in [8]. Each snapped point starts at weight 1. In each iteration, we sample a μ-net with respect to $w(\cdot)$ using Lemma 5. As we explain previously, this sampling cannot be done directly from the snapped points Q because of overlaps, so we utilize ℓ_0 estimators and samplers.

A key point is that each snapped point can increase in weight at most once in each iteration, so we have a number of discrete weight classes. So, in order to sample a point with accordance to its weight, we use the following procedure. At each iteration, we create estimators and samplers for each possible weight class. On iteration t, this means t copies. Now, we can estimate the total weight of each weight class, and randomly select a weight class accordingly. Then, since each point in that class have equal weight, we can use our estimator for that weight class to uniformly sample a snapped point. This provides us a weighted sampling procedure.

Given a sample B, we calculate its solution $f(B)$, and find the set V of points that violate $f(B)$ (for example, in the MEB problem, violators are points that are outside of the ball). Two sets of ℓ_0 estimators are now used to calculate estimated weights of the snapped points Q and the violators V, overestimating the weight of Q and underestimating the weight of V. If the estimated weight of V is at most μ times the estimated weight of S, then we denote this iteration as successful, and multiply the weight of each violator by $N^{1/s}$. The intuition behind this is that as our algorithm tries to find a basis for the problem, samples from successful iterations can be thought of as good representations of Q, and their violators

are more likely to be points in that basis, and should thus be given a higher weight. The algorithm finishes when there are no violators, at which point it has successfully found $f(Q)$. We can then return a $(1 + O(\varepsilon))$-approximation to $f(P)$ using property 3.

This sampling procedure is similar to [8], with the major difference being our usage of ℓ_0 samplers and estimators. This means that we sample according to an approximate weight function rather than the "true" weight function. We use a TV distance argument and the fact that we appropriately over/underestimate the weights to argue that we retain the bounds of [8]. We show that our approximate weight function is sufficiently close to the "true" weight function, enough that a sample of m elements according to the approximate weights will still retain the same properties. Additionally, we account for the error probabilities of the estimators and samplers to show that our probability of being a μ-net is the same as in the direct sampling of [8]. We also show that for actually successful iterations, meaning when the weight of V is at most μ times the weight of S, we maintain this property in our estimated weights. This is because for the second sets of estimators we create, we over/underestimate the weights in order to satisfy this one-sided bound. Thus, we retain the success criteria with high probability, again matching the bounds of [8]. Such, we bound the total number of iterations of our algorithm at $O(\nu s)$ where ν is the combinatorial dimension of the problem, and $s \in [1, \log N]$ is our parameter.

3.1 Centering the Net with Deletions

As our algorithm handles models with point deletions, we must make sure to construct our metric ε-net around a non-deleted point and find an approximation for the distance of the furthest non-deleted point from it, to define the size of our net. To do this, we use ℓ_0 samplers, which work in the presence of deletions. We first sample any non-deleted point $\boldsymbol{p}_c$, which becomes the center of our net. Now, by construction of P, we have an upper and lower bound on the distance of any point from $\boldsymbol{p}_c$. We simply binary search over the powers of 2 in this range and sample a non-deleted point with norm larger than the current power of 2 in each iteration. By the end of the binary search, which takes $\log\log(\Delta d)$ iterations, we end up with a 2-approximation for δ, considering only non-deleted points. Alternatively, one can find this using 2 iterations. First, we similarly sample a non-deleted input point to become $\boldsymbol{p}_c$. Second, we use an ℓ_0 sampler for each power of 2, and in one pass over the points add them to the appropriate sampler. This way, instead of doing the steps of a binary search, we perform them all at once. While this negates the extra pass complexity, it adds a space complexity in words logarithmic in Δ, since we need to store that many samplers. We will choose the $\log\log(\Delta d)$ iteration solution. We explain this procedure in detail in Section A.2.

3.2 Sampling from Servers

When adapting our sampling procedure to distributed models, one cannot take a sample of m points by asking machines to sample m points each, as this would lead to a load of $m \cdot k$ points. Further, since we are doing weighted sampling, machines need to sample in accordance with other machines' weights. In order to fix these problems, we employ the following protocol, which we describe for the coordinator model, noting that it can be implemented in the parallel computation model by having one machine assume the coordinator role. Each machine sends the coordinator the total weight of their subset. The coordinator then calculates how many of the m sample points B each machine shall sample, and sends them this number. Now, the machines can sample and send only their part of the sample, meaning m points are sent *in total* now, instead of per machine.

3.3 Approximation Guarantees for Various Problems

In these big data models, we are able to solve many useful low dimensional LP-type problems. We show how to apply our algorithm in three separate ways for the problems. For the MEB problem, we give a multiplicative $(1+O(\varepsilon))$-approximation to the optimal ball. For the Linear SVM problem, we show how our algorithm can be presented as a Monte Carlo randomized algorithm. We achieve a one-sided error where if a point set is inseparable, we always output as such, and if a point set is separable, we output a $(1+O(\varepsilon))$-approximation for the optimal separating hyperplane with high probability. For Linear Programming problems, instead of a multiplicative approximation, we give an additive ε-approximation to bounded LP problems. Full complexity results for the models are presented in their respective subsections of Section A. and an overview is shown in Theorem 1.

We can solve SDP problems up to additive ε-approximation. For SDP problems, notice that by taking the vectorization $\boldsymbol{x}$ of a solution X, we can rewrite a given SDP problem as an LP in d^2 dimensions. We must also define the positive semidefinite constraint $X \succeq 0$. This is equivalent to constraining $\boldsymbol{y}^T X \boldsymbol{y} \geq 0$ for all vectors $\boldsymbol{y}$ where $\|y\| = 1$. Notice that this requires an infinite number of constraints. To solve this, we create another lattice along with our original metric ε-net. Using this lattice, we can instead use the constraints $\boldsymbol{z}^T X \boldsymbol{z} \geq 0$ for vectors $\boldsymbol{z}$ on the lattice. Just like property 3, we show that satisfying these constraints satisfies the original semidefinite constraints up to ε. Thus, we are able to successfully reduce the SDP problem into an LP, which we can solve up to additive ε-approximation, while retaining a small number of constraints.

Being able to solve bounded LP problems up to additive ε-approximation allows us to use our algorithm for many important related problems. For example, we are able to solve the linear classification problem by finding a classifier within additive ε of an optimal classifier for a set of points. Since in the problem, the optimal classifier has a separation of at least ε, this means that our found classifier is guaranteed to be a separator for the original points.

4 Lower Bound

In this section, we motivate the usage of multipass algorithms for achieving subexponential space complexity in $(1+\varepsilon)$-approximations for LP-type problems in the high accuracy regime, where $d < (1/\varepsilon)^{0.999}$. We establish these lower bounds for the MEB and linear SVM problems by analyzing the communication complexity with reductions from the Indexing problem.

It is a well-known result that a lower bound on the 1-round communication complexity for a problem gives a lower bound on the space complexity of 1-pass streaming algorithms, via a reduction of Alice running the streaming algorithm on her points and sending the state of the algorithm to Bob, who finishes running the algorithm on his points [6]. We thus use the standard two party communication model for our lower bounds. A problem in this model is a function $P : \mathcal{A} \times \mathcal{B} \to \mathcal{C}$. Alice receives an input $A \in \mathcal{A}$ and Bob receives an input $B \in \mathcal{B}$. In an r-round protocol, Alice and Bob communicate up to r messages with each other, alternating sender and receiver. In particular, if r is even then Bob sends the first message. If r is odd then Alice sends the first message. After r rounds of communication, Bob outputs some $C \in \mathcal{C}$. The goal is for Bob to output $P(A, B)$. The r-round communication complexity of a problem P is the minimum worst-case cost over all protocols in which Bob correctly outputs $P(A, B)$ with probability at least $2/3$. In this model, cost is measured by the total number of bits sent between Alice and Bob throughout the r messages.

We use the Indexing problem for our reductions. In it, Alice is given a binary string $b \in \{0,1\}^{[n]}$ and Bob is given an index $i \in [n]$. The goal is for Bob to output the i^{th} bit of b. It is well-known that the 1-round randomized communication complexity of the Indexing problem is $\Omega(n)$.

For the MEB problem, given an instance of the Indexing problem, Alice transforms her bitstring b into a set of points P, one point $\boldsymbol{p}_j \in \mathbb{R}^d$ for each $b_j = 1$. Separately, Bob transforms his index i into a point $\boldsymbol{q} \in \mathbb{R}^d$. We then show that the MEB of $P \cup \{\boldsymbol{q}\}$ has radius 2 if $b_i = 1$, and radius at most $2 - \Omega(\varepsilon)$ if $b_i = 0$, where ε satisfies $n = (1/\varepsilon)^{\lfloor d/4 \rfloor}$. This gives us the following result.

▶ **Theorem 6.** *For $d < (1/\varepsilon)^{0.999}$, any 1-pass streaming algorithm which yields a $(1+\varepsilon)$-approximation for the MEB problem requires $(1/\varepsilon)^{\Omega(d)}$ words of space.*

A full proof of this lower bound is provided in Appendix B.

For the linear SVM problem, we show similar proofs, where Alice transforms her bitstring into points labeled -1 and Bob transforms his index into a point labeled $+1$. We can then show similar properties about the separating hyperplane, giving us the following results.

▶ **Theorem 7.** *For $d < (1/\varepsilon)^{0.999}$, any 1-pass streaming algorithm which yields a $(1+\varepsilon)$-approximation for the linear SVM problem requires $(1/\varepsilon)^{\Omega(d)}$ words of space.*

▶ **Theorem 8.** *For $d < (1/\varepsilon)^{0.999}$, any 1-pass streaming algorithm which determines if a set of binary labeled points is γ-separable requires $(1/\gamma)^{\Omega(d)}$ words of space.*

References

1. Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, July 2004. doi:10.1145/1008731.1008736.
2. Pankaj K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1481–1489, USA, 2010. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973075.120.
3. Pankaj K. Agarwal and Hai Yu. A space-optimal data-stream algorithm for coresets in the plane. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, SCG '07, pages 1–10, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1247069.1247071.
4. Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation*, 222:59–79, 2013. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011). doi:10.1016/j.ic.2012.10.006.
5. Yuqing Ai, Wei Hu, Yi Li, and David P. Woodruff. New characterizations in turnstile streams with applications. In *Proceedings of the 31st Conference on Computational Complexity*, CCC '16, Dagstuhl, DEU, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
6. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.
7. Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Near-Optimal epsilon-Kernel Construction and Related Problems. In Boris Aronov and Matthew J. Katz, editors, *33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2017.10.

8 Sepehr Assadi, Nikolai Karpov, and Qin Zhang. Distributed and streaming linear programming in low dimensions. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '19, pages 236–253, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3294052.3319697.

9 Mihai Bădoiu and Kenneth L. Clarkson. Smaller core-sets for balls. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 801–802, USA, 2003. Society for Industrial and Applied Mathematics.

10 Mihai Bădoiu and Kenneth L. Clarkson. Optimal core-sets for balls. *Computational Geometry*, 40(1):14–22, 2008. doi:10.1016/j.comgeo.2007.04.002.

11 A. Blum, A. Frieze, R. Kannan, and S. Vempala. A polynomial-time algorithm for learning noisy linear threshold functions. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 330–338, 1996. doi:10.1109/SFCS.1996.548492.

12 Yaroslav Bulatov, Sachin Jambawalikar, Piyush Kumar, and Saurabh Sethia. Hand recognition using geometric classifiers. In David Zhang and Anil K. Jain, editors, *Biometric Authentication*, pages 753–759, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-25948-0_102.

13 Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998. doi:10.1023/A:1009715923555.

14 Tom Bylander. Learning linear threshold functions in the presence of classification noise. In *Proceedings of the Seventh Annual Conference on Computational Learning Theory*, COLT '94, pages 340–347, New York, NY, USA, 1994. Association for Computing Machinery. doi:10.1145/180139.181176.

15 Emmanuel Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, June 2012. doi:10.1145/2184319.2184343.

16 Emmanuel J. Candes and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010. doi:10.1109/TIT.2010.2044061.

17 Jair Cervantes, Farid Garcia-Lamont, Lisbeth Rodríguez-Mazahua, and Asdrubal Lopez. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215, 2020. doi:10.1016/j.neucom.2019.10.118.

18 Timothy M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computational Geometry*, 35(1):20–35, 2006. Special Issue on the 20th ACM Symposium on Computational Geometry. doi:10.1016/j.comgeo.2005.10.002.

19 Timothy M. Chan and Vinayak Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 195–206, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-22300-6_17.

20 M. T. Chao. A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656, 1982. URL: http://www.jstor.org/stable/2336002.

21 Kenneth L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, March 1995. doi:10.1145/201019.201036.

22 Kenneth L. Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Trans. Algorithms*, 6(4), September 2010. doi:10.1145/1824777.1824783.

23 Kenneth L. Clarkson, Elad Hazan, and David P. Woodruff. Sublinear optimization for machine learning. *J. ACM*, 59(5), November 2012. doi:10.1145/2371656.2371658.

24 Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '03, pages 296–306, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/773153.773182.

25 John Dunagan and Santosh Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Mathematical Programming*, 114(1):101–114, 2008. doi:10.1007/s10107-007-0095-7.

26 Dan Garber and Elad Hazan. Sublinear time algorithms for approximate semidefinite programming. *Mathematical Programming*, 158(1):329–361, 2016. doi:10.1007/s10107-015-0932-z.

27 David Haussler and Emo Welzl. Epsilon-nets and simplex range queries. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, pages 61–71, New York, NY, USA, 1986. Association for Computing Machinery. doi:10.1145/10515.10522.

28 Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Jonathan Ullman, Ali Vakilian, and Anak Yodpinyanee. Fractional Set Cover in the Streaming Model. In Klaus Jansen, José D. P. Rolim, David P. Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*, volume 81 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:20, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.12.

29 Piotr Indyk, Eric Price, and David P. Woodruff. On the power of adaptivity in sparse recovery. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science—FOCS 2011*, pages 285–294. IEEE Computer Soc., Los Alamitos, CA, 2011. doi:10.1109/FOCS.2011.83.

30 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 49–58, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1989284.1989289.

31 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1807085.1807094.

32 Piyush Kumar, Joseph S. B. Mitchell, and E. Alper Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *ACM J. Exp. Algorithmics*, 8:1.1–es, December 2004. doi:10.1145/996546.996548.

33 Yi Li, Huy L. Nguyen, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 174–183, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591812.

34 S. Cliff Liu, Zhao Song, Hengjie Zhang, Lichen Zhang, and Tianyi Zhou. Space-Efficient Interior Point Method, with Applications to Linear Programming and Maximum Weight Bipartite Matching. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 88:1–88:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2023.88.

35 Nantia Makrynioti, Nikolaos Vasiloglou, Emir Pasalic, and Vasilis Vassalos. Modelling machine learning algorithms on relational data with datalog. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, DEEM'18, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209889.3209893.

36 J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4):498–516, 1996. doi:10.1007/BF01940877.

37 Vasileios Nakos, Xiaofei Shi, David P. Woodruff, and Hongyang Zhang. Improved algorithms for adaptive compressed sensing. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 90, 14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

38 Frank Nielsen and Richard Nock. Approximating smallest enclosing balls. In Antonio Laganá, Marina L. Gavrilova, Vipin Kumar, Youngsong Mun, C. J. Kenneth Tan, and Osvaldo Gervasi, editors, *Computational Science and Its Applications – ICCSA 2004*, pages 147–157, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-24767-8_16.

39 A. B. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, New York, NY, USA, 1962. Polytechnic Institute of Brooklyn.

40 Eric Price and David P. Woodruff. Lower bounds for adaptive sparse recovery. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 652–663. SIAM, Philadelphia, PA, 2012.

41 Benjamin Recht. A simpler approach to matrix completion. *J. Mach. Learn. Res.*, 12(null):3413–3430, December 2011. doi:10.5555/1953048.2185803.

42 Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. In Alain Finkel and Matthias Jantzen, editors, *STACS 92*, pages 567–579, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

43 Zhao Song, Mingquan Ye, and Lichen Zhang. Streaming semidefinite programs: $O(\sqrt{n})$ passes, small space and fast runtime, 2023. arXiv:2309.05135.

44 Xiaoming Sun, David P. Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. *ACM Trans. Algorithms*, 17(4), October 2021. doi:10.1145/3470566.

45 Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. *J. Mach. Learn. Res.*, 6:363–392, December 2005. URL: https://jmlr.org/papers/v6/tsang05a.html.

46 Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 859–869, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451108.

47 V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971. doi:10.1137/1116025.

48 V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In Vladimir Vovk, Harris Papadopoulos, and Alexander Gammerman, editors, *Measures of Complexity: Festschrift for Alexey Chervonenkis*, pages 11–30. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-21852-6_3.

49 David P. Woodruff. Sketching as a tool for numerical linear algebra. *Found. Trends Theor. Comput. Sci.*, 10(1–2):1–157, October 2014. doi:10.1561/0400000060.

50 Eric Xing, Michael Jordan, Stuart J Russell, and Andrew Ng. Distance metric learning with application to clustering with side-information. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002. URL: https://proceedings.neurips.cc/paper_files/paper/2002/file/c3e4035af2a1cde9f21e1ae1951ac80b-Paper.pdf.

51 Yinyu Ye and Edison Tse. An extension of Karmarkar's projective algorithm for convex quadratic programming. *Mathematical Programming*, 44(1):157–179, 1989. doi:10.1007/BF01587086.

52 E. Alper Yildirim. Two algorithms for the minimum enclosing ball problem. *SIAM J. on Optimization*, 19(3):1368–1391, November 2008. doi:10.1137/070690419.

53 Hamid Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. *Algorithmica*, 60(1):46–59, 2011. doi:10.1007/s00453-010-9392-2.

54 Hamid Zarrabi-Zadeh and Timothy M. Chan. A simple streaming algorithm for minimum enclosing balls. In *Proceedings of the 18th Annual Canadian Conference on Computational Geometry, CCCG 2006, August 14-16, 2006, Queen's University, Ontario, Canada*, 2006. URL: http://www.cs.queensu.ca/cccg/papers/cccg36.pdf.

A Application Specifics

In this section, we present in depth how our algorithms is applied to various big data models, and the specifics of the LP-type problems we solve, achieving the bounds in Theorem 1 and Table 1.

A.1 Application in the Multipass Streaming Model

The first model we consider is the multipass streaming model. In this model, our input is presented as a stream of points P, and our algorithm is allowed to make multiple linear scans of this stream, while maintaining a small space complexity.

Our algorithm is well suited for handling a large stream of data, as it creates a linear sketch of the data points. Therefore, all of our sampling and estimation can easily be done with the aid of our ℓ_0 samplers and ℓ_0 estimators. The major consideration we have to make is in how we calculate and store the weights of each snapped point, because we use that to decide which estimators and samplers to use for each point. We clearly cannot store the weight of each snapped point, as that would take up space linear in the number of snapped points. However, one can notice that the weight of a snapped point changes only when it is a violator of a successful iteration, at which point it is multiplied by $N^{1/s}$. Thus, it suffices to instead store the solution $f(B)$ for each successful iteration. Now, the weight of a snapped point is simply $N^{v/s}$, where v is the number of stored solutions $f(B)$ it violates, which can be calculated on the fly efficiently.

Given this, we can calculate the number of passes needed by our algorithm. First, we see that each iteration of our algorithm requires two passes over the stream. As seen in the last paragraph, getting the weight of each snapped point while creating our sample can be done in one pass. Further, after calculating a solution $f(B)$, checking for violators can also be done in one pass. Finally, we require one initial pass over the data to set our origin point as well as to get the value of N.

The space complexity of our algorithm comes from the four quantities it stores, those being the ℓ_0 estimators, ℓ_0 samplers, previous solutions $f(B)$, and the current sample B. Note that we do not actually need to store the metric ε-net points, as with our construction being a lattice, we can quickly access the closest metric ε-net point to any point $\boldsymbol{p}_i \in P$.

In each iteration we have t $\left(1 \pm \frac{1}{m^{3/2}}\right)$-approximation ℓ_0 estimators with $\frac{1}{\text{poly}(N)}$ error. Each of these estimators has a space usage of $O(m^3 \log^2 N)$. Additionally, we have $2t$ $\left(1 \pm \frac{1}{4}\right)$-approximation ℓ_0 estimators with $\frac{1}{\text{poly}(N)}$ error. Each of these estimators has a space usage of $O(\log^2 N)$. Finally, we have t ℓ_0 samplers with with $\frac{1}{\text{poly}(N)}$ error, each of which has a space usage of $O(\log(N))$. Given that t is bounded by $O(\nu s)$, and that m is bounded by $O\left(\nu\lambda N^{1/s} \log\left(\nu\lambda N^{1/s}\right)\right)$, we have a total storage from all of these of $O\left(\nu^4 s\lambda^3 N^{3/s}\right) \cdot \text{polylog}\left(\nu, \lambda, N\right)$ words. The t previous solutions can be stored in $O(\nu s) \cdot S_f$ space, where S_f is the number of words needed to store a solution $f(B)$. Finally, a sample of m points can be stored in $O\left(\nu\lambda N^{1/s} \log\left(\nu\lambda N^{1/s}\right)\right)$ words. Combining all of these, the total space complexity for our algorithm is

$$O\left(\nu^4 s\lambda^3 N^{3/s}\right) \cdot \text{polylog}\left(\nu, \lambda, N\right) + O(\nu s) \cdot S_f + O\left(\nu\lambda N^{1/s} \log\left(\nu\lambda N^{1/s}\right)\right).$$

We are mostly interested in the multi-pass case where $s = \log N$. First, note that in this case $N^{3/s}$ becomes constant. Further, we have that $\log N = \log\left(\left\lceil \log_{1+\varepsilon} r_m \right\rceil\right) + d \log\left(1 + \frac{2\sqrt{d}}{\varepsilon}\right)$. The first term diminishes quickly, and since in the high accuracy regime $d < (1/\varepsilon)^{0.999}$, we obtain a space complexity of

$$O\left(\nu^4\lambda^3 d \log\left(\frac{1}{\varepsilon}\right)\right) \cdot \text{polylog}\left(\nu, \lambda, d, \log\left(\frac{1}{\varepsilon}\right)\right) + O\left(\nu d \log\left(\frac{1}{\varepsilon}\right)\right) \cdot S_f + O\left(\nu\lambda \log\left(\nu\lambda\right)\right).$$

The time complexity of our algorithm depends on three operations we perform in each iteration. The first operation is to create our sample B. This can be done in $O(m)$ time per iteration since each point can be sampled from our ℓ_0 samplers in linear time. The second operation is to calculate the solution for our sample, $f(B)$, which is done once per iteration. The third operation is to check if a snapped point is a violator, i.e. to calculate if $f(B \cup \{\boldsymbol{q}\}) > f(B)$. This operation is done for all points each iteration. For a given LP-type problem, we can denote the time bounds for these as T_B and T_V. Together, this gives us a total time complexity over $O(\nu s)$ iterations of $O(\nu s (T_V \cdot n + m + T_B))$. In order for our algorithm to be efficient in time complexity, we want T_V to be constant time, and T_B to be polynomial in d, and logarithmic in $1/\varepsilon$.

All together, we achieve the following result for the multipass streaming model.

▶ **Theorem 9.** *For any $s \in [1, d \log(1/\varepsilon)]$, there exists a randomized algorithm to compute a $(1 + O(\varepsilon))$-approximation solution to an LP-type problem in the multipass streaming model, that takes $O(\nu s)$ passes over the input, with*

$$O\left(\nu^4 s \lambda^3 N^{3/s}\right) \cdot \text{polylog}(\nu, \lambda, N) + O(\nu s) \cdot S_f + O\left(\nu \lambda N^{1/s} \log\left(\nu \lambda N^{1/s}\right)\right)$$

space complexity in words and $O(\nu s (T_V \cdot n + m + T_B))$ time complexity, where S_f is the number of words needed to store a solution $f(B)$.

A.2 Application in the Strict Turnstile Model

In the strict turnstile model, there is an underlying vector $\boldsymbol{v}$ where the i^{th} element of the vector corresponds to a point $\boldsymbol{p}_i \in P$. This vector is initialized to $\mathbf{0}$. The input is then presented a stream of additive updates to the coordinates of $\boldsymbol{v}$, presented as $\boldsymbol{v} \leftarrow \boldsymbol{v} + \boldsymbol{e}_i$ or $\boldsymbol{v} \leftarrow \boldsymbol{v} - \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is the i^{th} standard unit vector. At the end of the stream, we are guaranteed that $\boldsymbol{v}$ is itemwise nonnegative. This stream can also be thought of as operations insert($\boldsymbol{p}_i$) and delete($\boldsymbol{p}_i$), with the guarantee that at the end of the stream, no point $\boldsymbol{p}_i$ has negative copies. Our algorithm is allowed to make multiple linear scans of this stream, while maintaining a small space complexity.

The application of our algorithm to this model is similar to the multipass streaming model, as ℓ_0 samplers and estimators work in the presence of point insertions and deletions.

The only additional hurdle presented by the strict turnstile model is that we cannot simply pick the first point in the data stream as our origin and center of our metric ε-net and get the maximum distance point in order to define N. This is because we need to make sure that the points we use here are not ones that will end up at zero copies at the end of the input stream. Instead, we present the following scheme. We first use an ℓ_0 sampler in order to sample a distinct non-deleted point $\boldsymbol{p}$, which we can then think of as our origin, and set up our metric ε-net around. We also notice that it suffices to get an $O(1)$-approximation of the largest norm of a (shifted) non-deleted point. Further, note that as each input point is in $\{-\Delta, -\Delta + 1, \ldots, \Delta\}^d$, this norm is upper bounded by $O\left(\Delta\sqrt{d}\right)$ and lower bounded by 1. We perform a binary search over the powers of 2 in this range. Since there are $O(\log(\Delta d))$ powers of 2 in the range, the binary search will take $O(\log\log(\Delta d))$ iterations. In each iteration of this search, we will use an ℓ_0 sampler in order to sample a distinct non-deleted point with norm above the current power of 2. When we find the largest such power of 2 where there exists a non-deleted point, we will have a 2-approximation of the largest norm from our origin point $\boldsymbol{p}_1$, and this can set our N. With this additional work over the multpass streaming model, we get the following result.

▶ **Theorem 10.** *For any $s \in [1, d\log(1/\varepsilon)]$, there exists a randomized algorithm to compute a $(1+O(\varepsilon))$-approximation solution to an LP-type problem in the multipass streaming model, that takes $O(\nu s + \log\log(\Delta d))$ passes over the input, with*

$$O\left(\nu^4 s\lambda^3 N^{3/s}\right)\cdot \text{polylog}\left(\nu,\lambda,N\right) + O(\nu s)\cdot S_f + O\left(\nu\lambda N^{1/s}\log\left(\nu\lambda N^{1/s}\right)\right)$$

space complexity in words and $O\left(\nu s\left(T_V\cdot n + m + T_B\right) + \log\log(\Delta d)\cdot n\right)$ time complexity, where S_f is the number of words needed to store a solution $f(B)$.

A.3 Application in the Coordinator Model

In the coordinator model, there are k distributed machines and a separate coordinator machine, which is connected to each machine via a two-way communication channel. The input set P is distributed among the machines, with each machine i getting a part P_i, and the goal is to jointly compute the solution $f(P)$ for the LP-type problem. Communication between the machines proceeds in rounds of messages. The final computation is done by the coordinator. In the coordinator model, our algorithm aims to minimize the number of communication rounds and maximum bits of information sent or received in any round.

The major implementation detail of the model is how to sample m points in the distributed setting and denote success of iterations. We will achieve this by the coordinator calculating what share of the m point sample each machine should be responsible for, and then building up an m point sample from the samples of each machine.

Our algorithm can do both of these in each round with three rounds of communication between the coordinator and the machines. First, at the start of each round, if the last round was deemed successful, the coordinator sends each machine the solution $f(B)$, for each machine to store and use in their weight calculations. The machines send the coordinator the weight of their subset of snapped points, $w(Q_i) = \sum_{\boldsymbol{q}_j\in Q_i} w(\boldsymbol{q}_j)$. Now, the coordinator generates m i.i.d. numbers $x_1,\ldots,x_m$ where each number $i \in k$ has probability of being picked according to the weight share of machine i, i.e. $\mathbf{Pr}\left[i \text{ is generated}\right] = \frac{w(Q_i)}{w(Q)}$ where $w(Q) = \sum_{i\in[k]} w(Q_i)$ is calculated by the coordinator. Starting the second round of communication, the coordinator sends each machine i a number $y_i = |\{l \mid x_l = i\}|$. Each machine then uses its ℓ_0 samplers to sample y_i points from its subset Q_i, according to the weight share of each snapped point $\boldsymbol{q}_j$, i.e. $\mathbf{Pr}\left[\boldsymbol{q}_j \text{ is sampled}\right] = \frac{w(\boldsymbol{q}_j)}{w(Q_i)}$, and sends this sample to the coordinator. This sampling system ensures that in total, $\mathbf{Pr}\left[\boldsymbol{q}_j \text{ is sampled}\right] = \frac{w(\boldsymbol{q}_j)}{w(Q_i)}\cdot\frac{w(Q_i)}{w(Q)} = \frac{w(\boldsymbol{q}_j)}{w(Q)}$, and thus the sample is correctly weighted. The coordinator then combines each machine's sample to create the total sample B, and calculates $f(B)$. Finally, the coordinator sends $f(B)$ to all machines, which they use to calculate their subset's violators. They send back the weight of their violator set, $w(V_i)$. If all of these $w(V_i)$'s are 0, then the algorithm is done. If not, the coordinator can calculate whether the iteration is successful, and continue to the next.

To complete our analysis, notice that we can pick a point as our origin and get the furthest distance from it in two rounds.

To calculate the maximum load over a round of communication, we look at the messages being sent. The start of the algorithm has each message representable as a point, so the maximum load is k. Then, in each iteration, we have four different messages to account for. A solution is S_f words in size, so sending it has load $k\cdot S_f$. Each weight being sent is bounded by $w(Q)$, which itself is upper bounded by $N\cdot\left(N^{1/s}\right)^t$ on the t^{th} iteration. Since we have $O(\nu s)$ iterations, we get that the weights are bounded by N^ν, which can fit into 1 word. Thus, we can bound the load of sending weights at k words. Each number y_i is bounded by m, and can thus fit into 1 word. Finally, the samples being sent by each machine have in total m points, thus the load is m words.

All together, we achieve the following result for the coordinator model.

▶ **Theorem 11.** *For any $s \in [1, d\log(1/\varepsilon)]$, there exists a randomized algorithm to compute a $(1+O(\varepsilon))$-approximation solution to an LP-type problem in the coordinator model, that takes $O(\nu s)$ rounds of communication, with*

$$O\left(\nu\lambda N^{1/s}\log\left(\nu\lambda N^{1/s}\right)+k\cdot S_f\right)$$

load in words, where S_f is the number of words needed to store a solution $f(B)$. The local computation time of the coordinator is $O(\nu s(m+T_B+k))$, and the local computation time of each machine i is $O(\nu s(n_i T_V))$ where $n_i = |P_i|$.

A.4 Application in the Parallel Computation Model

In the parallel computation model, there are k distributed machines, which are each connected via two-way communication channels. The input set P is distributed among the machines, with each machine i getting a part P_i, and the goal is to jointly compute the solution $f(P)$ for the LP-type problem. Communication between the machines proceeds in rounds, where each round the machines can communicate with each other in messages. In the coordinator model, our algorithm aims to minimize the number of communication rounds and maximum bits of information sent or received in any round.

Our procedure for running our algorithm in the parallel computation model is simply to run it as our algorithm for the coordinator model, assigning one of the machines to act as the coordinator. Thus, we can achieve the same bounds on the rounds of communication and maximum load. For the computation times, one machine will have a computation time on the order of the coordinator time added to the machine time, and other machines will have computation times same as in the coordinator model.

A.5 Solving the MEB Problem

Given n points $\boldsymbol{p}_i \in \{-\Delta, -\Delta+1, \ldots, \Delta\}^d$, the objective of the MEB problem is to find a d-dimensional ball $(\boldsymbol{c}, r) \in \mathbb{R}^d \times \mathbb{R}$ with center c and radius r that encloses all of the input points, i.e. where for all points $\boldsymbol{p}_i$, $d(\boldsymbol{c}, \boldsymbol{p}_i) \leq r$, and which has the smallest such radius r.

The MEB problem is an LP-type problem where S is the set of d-dimensional points, and $f(\cdot)$ is the function that maps from a set of points to their MEB. The combinatorial dimension of the MEB problem is $\nu = d+1$. Similarly, the VC dimension of the MEB set system is $\lambda = d+1$ [48]. A solution to the MEB problem is a ball $(\boldsymbol{c}, r)$ and can thus be stored in 2 words of space. Further, violator checks can easily be done on a stored solution by checking for a point $\boldsymbol{q}$, whether $d(\boldsymbol{c}, \boldsymbol{q}) \leq r$, giving us $T_V = O(1)$ for the MEB problem. The MEB of a sample of m points can be found in $O\left((m+d)^3\right)$ [51], which gives us T_B for the MEB problem. Therefore, we can achieve the following result for the MEB problem, using the results from Theorems 9, 10, and 11.

▶ **Theorem 12.** *For any $s \in [1, d\log(1/\varepsilon)]$, there exists a randomized algorithm to compute a $(1+4\varepsilon)$-approximation solution to the MEB problem with high probability in the following models, where $N = \lceil \log_{1+\varepsilon} r_m \rceil \left(1+\frac{2\sqrt{d}}{\varepsilon}\right)^d$.*

- *Multipass Streaming: An $O(ds)$ pass algorithm with*

  $$O\left(d^7 s N^{3/s}\right)\cdot \mathrm{polylog}(d,N) + O\left(ds + d^2 N^{1/s}\log\left(dN^{1/s}\right)\right)$$

  space complexity in words and $O\left(ds\left(n+(m+d)^3\right)\right)$ time complexity.

- *Strict Turnstile: An $O(ds + \log\log(\Delta d))$ pass algorithm with the same space and time complexity as the multipass streaming model.*
- *Coordinator/Parallel Computation: An algorithm with $O(ds)$ rounds of communication and*

  $$O\left(d^2 N^{1/s} \log\left(dN^{1/s}\right) + k\right)$$

  load in words. The local computation time of the coordinator is $O\left(ds\left((m+d)^3 + k\right)\right)$, and the local computation time of each machine i is $O(dsn_i)$ where $n_i = |P_i|$.

A.6 Solving the Linear SVM Problem

A Monte Carlo application of our algorithm is solving the Linear SVM Problem. In the problem, we are given n points $\boldsymbol{p}_i = (\boldsymbol{x}_i, y_i)$ where $\boldsymbol{x}_i \in \left\{-1, -1 + \frac{1}{\Delta}, \ldots, 1\right\}^d$ and $y_i \in \{-1, +1\}$. We are also given some $\gamma > 0$ for which the points are either inseparable or γ-separable. The objective of the problem is to determine either that the points are inseparable, or to compute a d-dimensional hyperplane $(\boldsymbol{u}, b)$ where b is the bias term, which separates the points with the largest margin, i.e. to solve the quadratic optimization problem

$$\min_{\boldsymbol{u} \in \mathbb{R}^d} \|\boldsymbol{u}\|^2 \quad \text{subject to} \quad y_i\left(\boldsymbol{u}^T \boldsymbol{x}_i - b\right) \geq 1 \quad \text{for all} \quad i \in [n]\,. \tag{1}$$

The linear SVM problem is an LP-type problem where S is the set of tuples of d-dimensional points and ± 1 y values, and $f(\cdot)$ is the function that maps from a set of points to their optimal separating hyperplane, or to the value infeasible. The combinatorial dimension of the linear SVM problem is $\nu = d + 1$ for a separable set, and $\nu = d + 2$ for an inseparable set. Further, the VC dimension of the linear SVM set system is $\lambda = d + 1$ [48].

However, there is an additional hurdle to overcome. If the points are too close together, running our algorithm with a metric ε-net might not retain separability. Therefore, we instead create a metric $\frac{\varepsilon\gamma}{2}$-net centered at the origin, where $\varepsilon < \frac{1}{2}$. This means that we have snapped points $\boldsymbol{q}_i = (\boldsymbol{z}_i, y_i)$ where $\boldsymbol{z}_j = \boldsymbol{x}_i + \boldsymbol{e}_i$ for some metric net point $\boldsymbol{e}_i$ with $\|\boldsymbol{e}_i\| \leq \frac{\varepsilon\gamma}{2}$. Our algorithm then finds the hyperplane that solves the quadratic optimization problem

$$\min_{\boldsymbol{u} \in \mathbb{R}^d} \|\boldsymbol{u}\|^2 \quad \text{subject to} \quad y_i\left(\boldsymbol{u}^T \boldsymbol{z}_i - b\right) \geq 1 \quad \text{for all} \quad i \in [N]\,. \tag{2}$$

We run our algorithm as a Monte Carlo algorithm, where if we do not find a solution in the first $O(ds)$ iterations, we return that the point set is not separable. This gives us an algorithm with a one-sided error. That is, if a point set is inseparable, we will always output as such. If it is separable however, we will output a $(1 + O(\varepsilon))$-approximation for the optimal separating hyperplane with high probability.

A solution to the linear SVM problem is a hyperplane $(\boldsymbol{u}, b)$ and can thus be stored in 2 words of space. Further, violator checks can easily be done on a stored solution by checking for a point $\boldsymbol{q} = (\boldsymbol{z}, y)$, whether $y\left(\boldsymbol{u}^T \boldsymbol{z} - b\right) \geq 1$, giving us $T_V = O(1)$ for the linear SVM problem. The optimal separating hyperplane of a sample of m points can be found in $O\left((m+d)^3\right)$ [51], which gives us T_B for the linear SVM problem. Therefore, we can achieve the following result for the linear SVM problem, using the results from Theorems 9, 10, and 11.

▶ **Theorem 13.** *For any $s \in [1, d\log(1/\varepsilon)]$, there exists a randomized algorithm to compute a $(1 + 2\varepsilon)$-approximation solution to the linear SVM problem with high probability in the following models, where $N = \left(1 + \frac{4\sqrt{d}}{\varepsilon\gamma}\right)^d$.*

- *Multipass Streaming/Strict Turnstile: An $O(ds)$ pass algorithm with*
  $$O\left(d^7 s N^{3/s}\right) \cdot \text{polylog}\,(d, N) + O\left(ds + d^2 N^{1/s} \log\left(dN^{1/s}\right)\right)$$
  space complexity in words and $O\left(ds\left(n + (m+d)^3\right)\right)$ time complexity.
- *Coordinator/Parallel Computation: An algorithm with $O(ds)$ rounds of communication and*
  $$O\left(d^2 N^{1/s} \log\left(dN^{1/s}\right) + k\right)$$
  load in words. The local computation time of the coordinator is $O\left(ds\left((m+d)^3 + k\right)\right)$, and the local computation time of each machine i is $O(dsn_i)$ where $n_i = |P_i|$.

A.7 Solving Bounded Linear Programming Problems up to Additive Epsilon Error

So far, we have dealt with applications with multiplicative error, i.e. where our solution is within $(1 + O(\varepsilon))$ times the optimal solution. We can also use our algorithm to solve problems up to additive error, such as for bounded linear programs. Linear programs in general are optimization problems of the form

$$\max_{x \in \mathbb{R}^d} \boldsymbol{c}^T \boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{a}_i^T \boldsymbol{x} \leq b_i \quad \text{for all} \quad i \in [n] \tag{3}$$

where the input is the objective vector $\boldsymbol{c}$ as well as n input constraints $\boldsymbol{p}_i = (\boldsymbol{a}_i, b_i)$.

We will work with a certain class of linear programs, where the input points, as well as the solution vector $\boldsymbol{x}$ is bounded, i.e., for all constraints i, $\|\boldsymbol{a}_i\|, |b_i| \in O(1)$, as well as $\|\boldsymbol{c}\|, \|\boldsymbol{x}\| \in O(1)$. Given this, we can run our algorithm by snapping each constraint point $\boldsymbol{p}_i$ to a metric $O(\varepsilon)$-net to get $\boldsymbol{a}'_i = \boldsymbol{a}_i + \boldsymbol{e}_i$ where $\|\boldsymbol{e}_i\| \leq O(\varepsilon)$ and $b'_i = b_i + f_i$ where $|f_i| \leq O(\varepsilon)$. As a note, this construction means that we require no additional passes in the turnstile model to create our net, similar to the linear SVM problem. This snapping then gives us the LP

$$\max_{\boldsymbol{x} \in \mathbb{R}^d} \boldsymbol{c}^T \boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{a}_i'^T \boldsymbol{x} \leq b'_i \quad \text{for all} \quad i \in [N]\,. \tag{4}$$

The optimal solution to LP (4), $\boldsymbol{x}$, then gives an additive ε-approximation solution to LP (3) as well.

Linear programs are the eponymous LP-type program, where S is the set of constraints of the LP and $f(\cdot)$ is the function that maps the set of constraints to their optimal solution, or to the value infeasible. An important note is that there might be multiple optimal solutions for a set of constraints, in which case any tie-breaking system may be used (a common one is to take the lexicographically smallest optimal point). The combinatorial dimension of linear programming is $\nu = d$, and the VC dimension is $\lambda = d + 1$ [36, 48]. A solution to an LP is a point $\boldsymbol{x}$ and can thus be stored in 1 word. Further, violator checks can easily be done on a stored solution by checking whether it satisfies a given constraint, giving us $T_V = O(1)$ for LP problems. The optimal solution to an LP with d variables and m constraints can be found in $\tilde{O}\left(md + d^{2.5}\right)$ [46], which gives us T_B for LP problems. Therefore, we can achieve the following result for bounded LP problems, using the results from Theorems 9, 10, and 11.

▶ **Theorem 14.** *For any $s \in [1, d \log(1/\varepsilon)]$, there exists a randomized algorithm to compute an additive ε-approximation solution to bounded LP problems with high probability in the following models, where $N \in O\left(\frac{\sqrt{d}}{\varepsilon}\right)^d$.*

- *Multipass Streaming/Strict Turnstile: An $O(ds)$ pass algorithm with*
  $$O\left(d^7 s N^{3/s}\right) \cdot \text{polylog}\,(d, N) + O\left(ds + d^2 N^{1/s} \log\left(dN^{1/s}\right)\right)$$
  space complexity in words and $\tilde{O}\left(ds\left(n + md + d^{2.5}\right)\right)$ time complexity.

- *Coordinator/Parallel Computation: An algorithm with $O(ds)$ rounds of communication and*
  $$O\left(d^2 N^{1/s} \log\left(dN^{1/s}\right) + k\right)$$
  load in words. The local computation time of the coordinator is $\tilde{O}\left(ds\left(md + d^{2.5} + k\right)\right)$, and the local computation time of each machine i is $O(dsn_i)$ where $n_i = |P_i|$.

A.7.1 Solving the Linear Classification Problem

One example of a bounded LP is the linear classification problem. In the problem, we are given n labeled examples $\boldsymbol{p}_i$, comprising of a bounded d-dimensional feature vector $\boldsymbol{x}_i \in \left\{-1, -1 + \frac{1}{\Delta}, \ldots, 1\right\}^d$ and a corresponding label $y_i \in \{-1, +1\}$. The goal of the problem is to find a separating hyperplane $\boldsymbol{u}$, which can be thought of as a normal vector $\boldsymbol{u} \in \mathbb{R}^d$ where $\|\boldsymbol{u}\| = 1$, such that $y_i\left(\boldsymbol{x}_i^T \boldsymbol{u}\right) \geq 0$ for all points $\boldsymbol{p}_i$. We will consider the related approximate optimization problem, where we assume that there is an optimal classifier $\boldsymbol{u}^*$ such that $y_i\left(\boldsymbol{x}_i^T \boldsymbol{u}^*\right) \geq \varepsilon$ for all points $\boldsymbol{p}_i$. We thus want to find a classifier that is within additive ε of the separation of this optimal classifier.

This problem can be written as a bounded LP as such. First, notice that we have two types of constraints, $\boldsymbol{x}_i^T \boldsymbol{u} \leq 0$ for any point $\boldsymbol{p}_i$ with $y_i = -1$, and $\boldsymbol{x}_i^T u \geq 0$ for any point $\boldsymbol{p}_i$ with $y_i = +1$. For simplicity, we will actually consider the negation of any such example, so we will actually consider points $\boldsymbol{p}_i' = (\boldsymbol{x}_i', y_i')$ where $\boldsymbol{x}_i' = -\boldsymbol{x}_i$ and $y_i' = -y_i$ if $y_i = +1$, and $\boldsymbol{x}_i' = \boldsymbol{x}_i$ and $y_i' = y_i$ otherwise. This allows us to only have constraints of the type $\boldsymbol{x}_i'^T \boldsymbol{u} \leq 0$ for all $i \in [n]$. Our objective is to maximize the separation of $\boldsymbol{u}$, which is $min_{i \in [n]} \boldsymbol{x}_i'^T \boldsymbol{u}$. While this objective isn't linear as is, it can be made linear by utilizing an additional variable representing the separation, and n additional constraints.

Thus, we can use our framework for bounded LPs in order to solve this problem with separation that is within additive ε of the optimal hyperplane's separation. Very importantly, since the optimal hyperplane's separations is ε, our hyperplane will in fact be a separating hyperplane for the original points, meaning that our solution fully satisfies the original constraints P. Thus, we are able to solve the linear classification problem with an additive ε-approximation of the largest separation, in the bounds given in Theorem 14.

A.8 Solving Bounded Semidefinite Programming Problems up to Additive Epsilon Error

Another additive ε-approximation application of our algorithm is in solving bounded semidefinite programming problems. These problems are optimization problems of the form

$$\max_{X \in \mathbb{R}^{d \times d}} \langle C, X \rangle_F \quad \text{subject to} \quad \begin{array}{c} \langle A^{(i)}, X \rangle_F \leq b^{(i)} \quad \text{for all} \quad i \in [n] \\ X \succeq 0 \end{array} \tag{5}$$

where $\langle \cdot, \cdot \rangle_F$ is the Frobenius inner product, i.e. $\langle A, B \rangle_F = \sum_{i,j} A_{ij} B_{ij}$, and $X \succeq 0$ denotes X as a positive semidefinite matrix. The input will be the objective matrix C as well as n input constraints $\boldsymbol{p}_i = \left(A^{(i)}, b^{(i)}\right)$.

We will work with a certain bounded class of SDP problems. Namely, we will have the following boundedness assumptions.

- X has unit trace, i.e. $\mathrm{Tr}(X) = 1$,
- $\|C\|_F \leq 1$ where $\|\cdot\|_F$ is the Frobenius norm.

Further, for all constraints i

- $\left\|A^{(i)}\right\|_2 \leq 1$ where $\|\cdot\|_2$ is the spectral norm,

- $\left\|A^{(i)}\right\|_F \leq F$,
- The number of nonzero entries of $A^{(i)}$ is bounded by S,
- $\left|b^{(i)}\right| \leq 1$.

▶ **Definition 15.** *These bounds and later calculations use several definitions of norms. For an $n \times m$ matrix A, we use the following definitions of norm.*

- *Entry-wise* 1*-norm:* $\|A\| = \sum_{i\in[n],j\in[m]} A_{ij}$.
- *Spectral norm:* $\|A\|_2 = \max_{\boldsymbol{v}\in\mathbb{R}^m,\|\boldsymbol{v}\|\neq 0} \frac{\|A\boldsymbol{v}\|}{\|\boldsymbol{v}\|}$.
- *Frobenius norm:* $\|A\|_F = \sqrt{\sum_{i\in[n],j\in[m]} A_{ij}^2}$.
- *Schatten* 1*-norm:* $\|A\|_* = \sum_{i=1}^{\min(n,m)} \sigma_i(A)$*, where* $\sigma_i(A)$ *are the singular values of* A.

Our algorithm solves these problems as bounded LP problems with the solution vector $\boldsymbol{x} \in \mathbb{R}^{d^2}$ where $\boldsymbol{x} = \mathrm{Vec}(X)$ is the vectorization of X. This gives us the natural objective function $\max_{\boldsymbol{x}\in\mathbb{R}^{d^2}} \boldsymbol{c}^T\boldsymbol{x}$ where $\boldsymbol{c} = \mathrm{Vec}(C)$. For the constraints of these LPs, we consider the two different type of SDP constraints separately. First, we have constraints of the type $\langle A^{(i)}, X\rangle_F \leq b^{(i)}$, which we can represent as $\boldsymbol{a}^{(i)T}\boldsymbol{x} \leq b^{(i)}$ where $\boldsymbol{a}^{(i)} = \mathrm{Vec}(A^{(i)})$. We also have the constraint that X must be positive semidefinite, i.e. $X \succeq 0$. This is equivalent to the constraints $\boldsymbol{y}^T X\boldsymbol{y} \geq 0$ for all vectors $\boldsymbol{y} \in \mathbb{R}^d$ where $\|\boldsymbol{y}\| = 1$. We can represent each of these as constraints $(\boldsymbol{y}\otimes\boldsymbol{y})^T x$. The problem however is that we would need an infinite amount of these constraints, as there are infinite $\boldsymbol{y}$ vectors to consider.

Thus, we again utilize metric ε-nets. For each $A^{(i)}$, we note that a naive lattice where each coordinate $A^{(i)}_{jk}$ is snapped to the nearest multiple of $\frac{\varepsilon}{d}$ would suffice, however we can achieve a tighter bound because of the assumption that $A^{(i)}$ has at most S nonzero entries. We can therefore create a net for each of the $\binom{d^2}{S}$ possible combinations, where each net size is exponential in S rather than d^2. Now, for any $A^{(i)}$, its nonzero coordinates must be fully captured by at least one of these nets, and so we can deterministically choose one and snap it to a point in that net. We can further snap to the nearest multiple of $\frac{\varepsilon}{\min(d,S)}$ in order to decrease the size of each net. Finally, notice that for each coordinate of $A^{(i)}$, $\left|A^{(i)}_{jk}\right| \leq \left\|A^{(i)}\right\|_2 \leq 1$. Thus each net is of size $\left(\frac{2\min(d,S)}{\varepsilon} + 1\right)^S$, and therefore we have a total size of $\binom{d^2}{S} \cdot O\left(\left(\frac{\min(d,S)}{\varepsilon}\right)^S\right)$ for the nets. For each b_i, we can snap it to the nearest multiple of ε, giving a net of size $O\left(\frac{1}{\varepsilon}\right)$. In total, this means that in our snapped LP, we have $\binom{d^2}{S} \cdot O\left(\left(\frac{\min(d,S)}{\varepsilon}\right)^S \frac{1}{\varepsilon}\right)$ constraints of the form $\langle A'^{(i)}, X\rangle_F \leq b'^{(i)}$.

For the positive semidefinite constraints, we create a lattice where each coordinate is a multiple of $\frac{\varepsilon}{d\sqrt{d}}$. This gives us a metric ε-net of size $O\left(\left(\frac{d\sqrt{d}}{\varepsilon}\right)^d\right)$, and so we can reduce the infinite constraint space into that many constraints of the form $\boldsymbol{z}^T X\boldsymbol{z} \geq 0$ where $\boldsymbol{z} \in \mathbb{R}^d$ are points on the metric ε-net. We can call the original (infinitely numerous) constraints Y, and the new constraints Z. Because our algorithm now solves an LP, our combinatorial dimension is $\nu = d^2$, and our VC dimension is $\lambda = d^2 + 1$.

Thus, our algorithm can be utilized to solve additive ε-approximation SDP problems. Notice that a solution to an SDP is a point $(X + \frac{3\varepsilon}{d} I)$ and can thus be stored in 1 word of space. Further, violator checks can easily be done on a stored solution by checking whether it satisfies a given constraint, giving us $T_V = O(1)$ for SDP problems. For the bounds of the problem, notice that we convert an SDP problem into an LP problem on d^2 dimensions. Thus, we can use the T_B for LPs with d^2 variables and m constraints. Therefore, we can achieve the following result for bounded SDP problems, using the results from Theorems 9, 10, and 11.

▶ **Theorem 16.** *For any $s \in [1, d^2 \log(1/\varepsilon)]$, there exists a randomized algorithm to compute an additive ε-approximation solution to bounded SDP problems with high probability in the following models, where $N \in \binom{d^2}{S} \cdot O\left(\left(\frac{\min(d,S)}{\varepsilon}\right)^S \frac{1}{\varepsilon}\right) + O\left(\left(\frac{d\sqrt{d}}{\varepsilon}\right)^d\right)$.*

- *Multipass Streaming/Strict Turnstile: An $O\left(d^2 s\right)$ pass algorithm with*
  $$O\left(d^{14} s N^{3/s}\right) \cdot \text{polylog}\left(d, N\right) + O\left(d^2 s + d^4 N^{1/s} \log\left(d N^{1/s}\right)\right)$$
  space complexity in words and $\tilde{O}\left(d^2 s\left(n + md^2 + d^5\right)\right)$ time complexity.
- *Coordinator/Parallel Computation: An algorithm with $O\left(d^2 s\right)$ rounds of communication and*
  $$O\left(d^4 N^{1/s} \log\left(d N^{1/s}\right) + k\right)$$
  load in words. The local computation time of the coordinator is $O\left(d^2 s\left(md^2 + d^5 + k\right)\right)$, and the local computation time of each machine i is $O\left(d^2 s n_i\right)$ where $n_i = |P_i|$.

A.8.1 Solving SDP Saddle Point Problems in the Unit Simplex

One example of a bounded SDP where our algorithm can be used to achieve a useful result is the saddle point problem

$$\max_X \min_{\boldsymbol{p} \in \Delta_n} \sum_{i \in [n]} p_i \left(\langle A^{(i)}, X \rangle_F - b^{(i)}\right) \tag{6}$$

where for all $i \in [n]$, $A^{(i)} \in \mathbb{R}^{d \times d}$ are symmetric and $b^{(i)} \in \mathbb{R}$. $X \in \mathbb{R}^{d \times d}$ is a positive semidefinite matrix of trace 1, and $\Delta_n = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} \geq 0, \|\boldsymbol{x}\|_1 = 1\}$ is the $n-1$ dimensional unit simplex. In the case where the optimal solution to Equation 6 is nonnegative, then solving it up to additive ε error is equivalent to finding an X that satisfies all constraints $\langle A^{(i)}, X \rangle_F \geq b^{(i)}$ up to additive ε error. The optimization version of this is maximizing a margin σ where $\langle A^{(i)}, X \rangle_F \geq b^{(i)} + \sigma$.

Notice that the constraints of the problem directly fit into our framework for bounded SDP problems, and the margin σ can be represented as an aditional variable in our LP formulation. The maximization objective $\max \sigma$ now requires a 1-hot vector c, which fits into our boundedness constraint. Thus, we can use our general framework for bounded SDP problems in order to solve this problem up to an additive ε-approximation, in the bounds given in Theorem 16.

B Lower Bound Proof for the MEB Problem

▶ **Theorem 6.** *For $d < (1/\varepsilon)^{0.999}$, any 1-pass streaming algorithm which yields a $(1+\varepsilon)$-approximation for the MEB problem requires $(1/\varepsilon)^{\Omega(d)}$ words of space.*

Proof. We first provide the proof for $d = 2$, and then extended to higher dimensions.

With $d = 2$, let $\varepsilon = \left(\frac{1}{n}\right)^2$ and Alice's points $\boldsymbol{p}^{(1)}, \boldsymbol{p}^{(2)}, \ldots, \boldsymbol{p}^{(n)} \in \mathbb{R}^2$ be equally spaced points on the unit circle, where $\boldsymbol{p}^{(j)} = \left(\cos\left(2\pi j\sqrt{\varepsilon}\right), \sin\left(2\pi j\sqrt{\varepsilon}\right)\right)$. For each $j \in [n]$, if $b_j = 1$, Alice adds $\boldsymbol{p}^{(j)}$ to her portion of the input stream to the MEB problem. Bob adds to his portion of the input stream the point $\boldsymbol{q} \in \mathbb{R}^2$, given by $\boldsymbol{q} = \left(3\cos\left(\pi + 2\pi i\sqrt{\varepsilon}\right), 3\sin\left(\pi + 2\pi i\sqrt{\varepsilon}\right)\right)$, such that $\boldsymbol{q}$ is the point opposite $\boldsymbol{p}^{(i)}$ through the origin such that $\left\|\boldsymbol{p}^{(i)} - \boldsymbol{q}\right\| = 4$.

If $b_i = 1$, then both $\boldsymbol{p}^{(i)}$ and $\boldsymbol{q}$ are part of the input stream to Alice and Bob's instance of MEB. In this case, it is easy to see that the MEB is that which has a diameter from $\boldsymbol{p}^{(i)}$ to $\boldsymbol{q}$, meaning that the MEB has radius 2. However, if $b_i = 0$, then $\boldsymbol{p}^{(i)}$ is not part of the input to the MEB instance. One can now show that $\left\|\boldsymbol{p}^{(j)} - \boldsymbol{q}\right\| \leq 4 - \Omega(\varepsilon)$ for all other j where $b_j = 1$, giving the desired result.

We now extend this idea to an arbitrary dimension d satisfying $d < (1/\varepsilon)^{0.999}$. Without loss of generality, we assume d is even. If it is not, we can simply disregard one of the dimensions. Let $\varepsilon = \left(\frac{1}{n}\right)^{4/d}$. As in the 2 dimensional case, we define Alice's n points $\boldsymbol{p}^{(1)}, \boldsymbol{p}^{(2)}, \ldots, \boldsymbol{p}^{(n)} \in \mathbb{R}^d$ where each point $\boldsymbol{p}^{(j)}$ is given by a sequence $\{j_k\}_{k=1}^{d/2}$ in $\{1, 2, \ldots, \frac{1}{\sqrt{\varepsilon}}\}$, so that the ℓ-th coordinate of $\boldsymbol{p}^{(j)}$ is given by

$$\boldsymbol{p}_\ell^{(j)} = \begin{cases} \sqrt{\frac{2}{d}} \cos\left(2\pi j_{\frac{\ell+1}{2}} \sqrt{\varepsilon}\right) & \text{if } \ell \text{ is odd,} \\ \sqrt{\frac{2}{d}} \sin\left(2\pi j_{\frac{\ell}{2}} \sqrt{\varepsilon}\right) & \text{if } \ell \text{ is even.} \end{cases}$$

Informally, we pair up the axes of d-dimensional space, and choose the points on the unit circle which when projected onto the plane of a pair of axes will resemble the 2-dimensional case. As in the 2-dimensional case, we can show that if $b_i = 1$, then the MEB has radius 2, and if $b_i = 0$ then the MEB has radius at most $2 - O(\varepsilon)$. ◀