Distributed and Streaming Linear Programming in Low Dimensions

Sepehr Assadi*
Princeton University
Princeton, NJ, USA
sassadi@princeton.edu

Nikolai Karpov†
Indiana University Bloomington
Bloomington, IN, USA
nkarpov@iu.edu

Qin Zhang†
Indiana University Bloomington
Bloomington, IN, USA
qzhangcs@indiana.edu

Abstract

We study linear programming and general LP-type problems in several big data (streaming and distributed) models. We mainly focus on low dimensional problems in which the number of constraints is much larger than the number of variables. Low dimensional LP-type problems appear frequently in various machine learning tasks such as robust regression, support vector machines, and core vector machines. As supporting large-scale machine learning queries in database systems has become an important direction for database research, obtaining efficient algorithms for low dimensional LP-type problems on massive datasets is of great value. In this paper we give both upper and lower bounds for LP-type problems in distributed and streaming models. Our bounds are almost tight when the dimensionality of the problem is a fixed constant.

ACM Reference Format:
Sepehr Assadi, Nikolai Karpov, and Qin Zhang. 2019. Distributed and Streaming Linear Programming in Low Dimensions. In *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '19), June 30–July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3294052.3319697

*Supported in part by the Simons foundation Algorithms and Geometry collaboration. Majority of the work done while the author was a graduate student at University of Pennsylvania.
†Supported in part by NSF CCF-1525024, IIS-1633215 and CCF-1844234.

PODS '19, June 30–July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6227-6/19/06…$15.00
https://doi.org/10.1145/3294052.3319697

1 Introduction

As machine learning becomes pervasive, how to effectively support machine learning tasks in database systems has become an imminent question. In a recent paper [31], Makrynioti *et al.* observed that many machine learning problems can be expressed by *linear programs* (LP). They designed a level of abstraction called *SolverBlox* on top of a declarative language *LogiQL*[1] as a framework for expressing linear program formulations. The query in the format of SolverBlox will then be translated to a format supported by an LP solver for computing the solution. In this paper we consider the algorithmic side of this research direction, that is, we focus on the design of efficient LP solvers for large-scale datasets. In particular, we propose algorithms for linear programming in three popular "big data" models, namely, the *coordinator* model [38], the *streaming* model [2, 37], and *massively parallel computation* (MPC) [5, 23, 29]. We also provide almost matching lower bounds when the dimensionality of the linear program is a fixed constant.

In the rest of the introduction we will start with the definition of the problem and the description of the computation models, and then present our results and discuss previous work.

Problem Definition. The basic linear programming problem can be described as follows: we have a set of d variables $(x_1, \ldots, x_d)$ and a set of n linear constraints each of which (indexed by j) is in the form of $\sum_{i=1}^{d} a_i^j x_i \leq b^j$, where a_i^j, b^j are coefficients and d is the *dimension* of the problem. We also have an objective function $\sum_{i=1}^{d} c_i x_i$. The goal is to find an assignment for variables that minimizes the objective function while satisfying all the constraints.

Linear programming is a special case of a more general problem called *LP-type problem* [32], which we will discuss in details in Section 2.1. Besides linear programming, LP-type problems also include several other important problems in machine learning, such as *Linear Support Vector Machines (SVM)* [7], which is widely used in classification and regression analysis [10, 19, 22]), and *Core Vector Machines* [42], which is used to speed up general SVM computation (or,

[1] An extended version of Datalog [3].

Linear SVM augmented by the kernel trick [7]). We will give the formal definitions of these problems in Section 4. The algorithms we propose in this paper work for general LP-type problems.

In this paper we are interested in the scenario when the dimension of the linear program (and LP-type problem in general) is small compared to the number of constraints. Various examples of linear programming and LP-type problems in machine learning are of this type: SVMs and regression problems (in particular, least absolute error regression that can be modeled by linear programming) are often over-constrained; in the problems of Chebyshev approximation and linear separability, the number of variables are typically small.

Computational Models. We study linear programming and LP-type problems in the following big data models.

- *The (multi-pass) streaming model.* In this model, we have a single machine which can make linear scans of the input data sequence. The task is to compute some function defined on the input data sequence. The goal is to minimize the *memory space usage* and the *number of passes* needed. This model captures data that cannot fit the memory, and on which sequential scan is much more efficient than random access.
- *The coordinator model.* In this model, we have k sites and a central coordinator. Each site is connected by a two-way communication channel with the coordinator. The input is initially partitioned among the k sites. The task is for the sites and coordinator to jointly compute some function defined on the union of the k datasets. The computation proceeds in rounds: At the beginning of each round, the coordinator sends a message to each site, and then each site replies with a message back to the coordinator. At the end of the computation, the coordinator outputs the answer. The goal is to minimize the *total bits of the communication* and the *rounds of the computation*. This model fits data that is inherently distributed or cannot fit the storage of a single machine
- *Massively parallel computation (MPC).* In this model, we have k machines interconnected in a network that allows communication between any pairs of machines. Similar to the coordinator model, the input is partitioned among the k machines, and the task is for them to compute some function defined on the union of the k datasets. The computation is again in terms of rounds. At each round, the machines communicate with each other over the network by sending and receiving messages. The message sent by a machine at each round is a function of its input data and all messages it has received in previous rounds. Our goal is to minimize the *number of rounds of the computation*, and the *maximum bits of information sent or received by a machine at any round* (often called the *load* in the literature). MPC has already become the model of choice for studying parallel computation in computer clusters.

Description of the input. Since we are dealing with low-dimensional problems, we assume that the memory on each site/machine in each model is at least proportional to d, the dimension of the problem, but is significantly smaller than n, the number of constraints. As a result, the input is presented by giving the constraints one by one to the algorithm in the streaming model, or partitioning them across different sites/machines in the coordinator and MPC models.

1.1 Our Contributions and Related Work

In the following, we present our results for linear programming in the three big data models described above, and postpone the specifics of their generalization to LP-type problems to later sections. Our main upper bound result is the following.

Result 1. We give the following *polynomial time* algorithms for d-dimensional linear programming with n constraints. For any integer $r \geq 1$ and parameter $\delta \in (0, 1)$:

- *Streaming:* An $O(d \cdot r)$-pass streaming algorithm with $O(n^{1/r}) \cdot \text{poly}(d, \log n)$ space.
- *Coordinator:* An $O(d \cdot r)$-round distributed algorithm with $O(n^{1/r} + k) \cdot \text{poly}(d, \log n)$ total communication.
- *MPC:* An $O(d/\delta^2)$-round algorithm with $O(n^{\delta}) \cdot \text{poly}(d, \log n)$ load per machine.

Our algorithms are randomized and output the correct answer with probability $1 - 1/n^c$ for any desired constant $c \geq 1$.

By Result 1 for $r = \log n$ and $\delta = 1/\sqrt{\log n}$, we obtain linear programming algorithms that use $O(d \log n)$ passes or rounds, and have space, communication, or load requirements in each model that is almost independent of the number of constraints. For low-dimensional instances, this results in a dramatic saving compared to direct implementations of standard LP algorithms in these models.

Previously, Chan and Chen [13] proposed an $O(r^{d-1})$-pass streaming algorithm for linear programming that uses $O(n^{1/r}) \cdot \text{poly}(d, \log n)$ space. Result 1 improves upon this result by achieving an exponentially smaller pass-complexity in terms of d.

In the coordinator model, Daumé et al. [26] gave an algorithm using $O(r^{d+O(1)} \cdot k \cdot n^{1/r})$ communication based on an adaptation of the algorithm of [13]. The round-complexity and communication cost of this algorithm again depends exponentially on d.

In the MPC model, very recently Tao [41] gave a $d^{O(\log(1/\delta))}$-round MPC algorithm with load $O(n^{\delta})$ when $d = \text{polylog}(n)$

(for any $\delta \in (0,1)$). This algorithm is then used as a building block for an interesting database application called *entity matching with linear classification*. The round complexity of our MPC algorithm in Result 1 improves that of [41] by an exponential factor.

To summarize, Result 1 *exponentially* improves upon the pass/round complexities of the state-of-the-art, while using the same or smaller space, communication, or load, in the considered big data models.

We complement our algorithms by giving almost *tight* lower bounds for any fixed dimension (even $d = 2$) in the streaming and coordinator model.

Result 2. We give the following lower bounds for 2-dimensional linear programming with n constraints. For any integer $r \geq 1$:

- *Streaming:* Any r-pass algorithm requires $\Omega(n^{1/2r})$ space.
- *Coordinator:* Any r-round algorithm requires $\Omega(n^{1/2r})$ communication even when number of sites is only $k = 2$.

Our lower bounds hold even for randomized algorithms that output the correct answer with probability at least $2/3$.

A few remarks about Result 2: Firstly, it is easy to see that linear programming in one dimension in the models we consider is a trivial task. Result 2 thus proves the lower bound for the smallest non-trivial dimension. We note that unlike Result 1 that worked in all the three models, Result 2 does *not* prove any lower bound for MPC algorithms. Proving lower bounds for MPC algorithms is considered to be a challenging task as it has serious implications for long standing open problems in complexity theory [39]. Hence, no *unconditional* lower bounds are known so far in the literature for *any* MPC problem and Result 2 is of no exception.

Prior to our work, Chan and Chen [13] gave a lower bound for 2-dimensional linear programming for a *restricted* family of *deterministic* streaming algorithms in the *decision tree* model (the only permitted operation of these streaming algorithms is testing the sign of a function evaluated at the coefficients of a subset of stored hyperplanes). Their lower bound states that this type of algorithms require $\Omega(n^{1/r})$ space to compute the solution in r passes. Our lower bound in Result 2 is much stronger in that it proves a similar pass-space tradeoff for *all* streaming algorithms (even randomized). Finally, Guha and McGregor [24] showed that there is a fixed dimensional optimization problem for which any r-pass streaming algorithm requires $\Omega(n^{1/r})$ space. However, it is not clear how to adapt their proof to linear programming since their optimization problem involves quadratic constraints [33].

Further Related Work. Special cases of linear programming have been studied previously in the big data models. In particular, Ahn and Guha gave multi-pass streaming algorithms for $(1+\varepsilon)$-approximation of *packing* LPs [1] and Indyk *et al.* [27] gave similar algorithms for *covering* LPs (see also [4]). These results focus on high-dimensional linear programs (non-constant d) and only packing/covering LPs, and are hence quite different from our approach in this paper.

Unlike the case for big data models, low-dimensional linear programming has been studied extensively in the RAM model since the 1980s. Megiddo [34] gave an algorithm for d-dimensional linear programming with time complexity $O(2^{2^d} n)$, which is linear in terms of the number of constraints n. This bound was consequently improved by a series of papers [8, 12, 15–17, 20, 21, 28, 32].

2 Preliminaries

Notations. For integers $1 \leq a \leq b$, we define $[a]$ as the set $\{1, \ldots, a\}$, $[a : b] := \{a, a+1, \ldots, b\}$, and $(a : b] := [a : b] \setminus \{a\}$ (we define $[a : b)$ and $(a : b)$ analogously). We use capital letters for sets and random variables and calligraphic letters for set families. We use the notation $\widetilde{O}(f)$ to denote a function of the form $O(f \cdot \text{polylog}(f))$.

Throughout the paper, we say an event happens "with high probability" if its probability can be lower bounded by $1 - 1/n^c$ for any desired constant $c \geq 1$ (n is the number of constraints).

We use the following standard variant of Chernoff bound.

Proposition 2.1 (Chernoff bound). *Suppose $X_1, \ldots, X_t$ are t independent random variables taking value in $[0,1]$ and $X := \sum_{i=1}^{t} X_i$. Then, for any $\varepsilon > 0$,*

$$\mathbf{Pr}\left(|X - \mathbb{E}[X]| > \varepsilon \cdot \mathbb{E}[X]\right) \leq 2 \cdot \exp\left(\frac{-\varepsilon^2 \cdot \mathbb{E}[X]}{3}\right).$$

2.1 LP-type Problems

We consider a generalization of linear programming referred to as LP-type problems[2]. An LP-type problem consists of a pair (S, f), where S is a finite set of elements, and $f : 2^S \to R$ is a set function with a range R which is assumed to have a total order. The function f satisfies two properties:

- *Monotonicity*: for any two sets $X \subseteq Y \subseteq S$, $f(X) \leq f(Y) \leq f(S)$.
- *Locality*: for any two sets $X \subseteq Y \subseteq S$, and any elements $e \in S$, if $f(X) = f(Y) = f(X \cup \{e\})$, then $f(Y) = f(Y \cup \{e\})$.

For an LP-type problem (S, f), we call a set $B \subseteq S$ a *basis* of S if $f(B) = f(S)$, and for all $B' \subset B$ we have $f(B') < f(B)$. The goal is to compute a basis $B_S \subseteq S$ such that $f(B_S) = f(S)$. We

[2] LP-type problems is also known as *abstract linear programming* [6].

say an element $e \in S$ *violates* $X \subseteq S$ if $f(X \cup \{e\}) > f(X)$. It helps to think of an LP-type problem (S, f) as an optimization problem in which elements of S are the constraints, and $f(A)$ computes the best *feasible* solution on the set of constraints A. In the case when the optimal solution is not unique, we just break the tie arbitrarily. Computing $f(B_S) = f(S)$ hence amounts to computing the optimal solution subject to all the constraints (we will make this connection explicit in the context of linear programming and other problems in Section 4).

Combinatorial Dimension. Note that an LP-type problem may have several bases which are of different sizes. We define the *combinatorial dimension* of an LP-type problem to be the *maximum* cardinality of a basis for S, denoted by $\nu_{S,f}$ (ν for short when S and f are clear from the context).

2.2 ε-Nets and VC Dimension

We now define another important notion that we use in designing our algorithms.

VC Dimension. A set-system is a tuple $(\mathcal{H}, U)$ consists of a universe U and a set family $\mathcal{H} \subseteq 2^U$. Let $C \subseteq U$ be a set. Define the intersection between a set family and a set to be the set family

$$\mathcal{H} \cap C := \{H \cap C \mid H \in \mathcal{H}\}.$$

We say that a set C is *shattered* by $\mathcal{H}$ if $\mathcal{H} \cap C$ contains all the subsets of C, i.e., $|\mathcal{H} \cap C| = 2^{|C|}$. The *VC dimension* of set-system $(\mathcal{H}, U)$, denoted by $\lambda_{\mathcal{H}}$ (or λ for short when $\mathcal{H}$ is clear in the context), is then the cardinality of the *largest* set C that is shattered by $\mathcal{H}$.

ε-Net. Given a set-system $(\mathcal{X}, U)$, and a weight function $w : \mathcal{X} \to \mathbb{R}$, for any $\mathcal{Y} \subseteq \mathcal{X}$, let $w(\mathcal{Y}) := \sum_{Y \in \mathcal{Y}} w(Y)$. We say a set $\mathcal{N} \subseteq \mathcal{X}$ is an ε-net of $\mathcal{X}$ with respect to w for a parameter $\varepsilon \in (0, 1)$, iff for any point $u \in U$ such that $\sum_{X \in \mathcal{X}: u \notin X} w(X) \geq \varepsilon \cdot w(\mathcal{X})$, it holds that $\{X \in \mathcal{N} \mid u \notin X\} \neq \emptyset$.

The notion of ε-net is well-studied in the literature (particularly in the computational geometry community [9, 25, 36]), and has been used in the algorithm design for many problems. We use the following simple randomized construction of ε-net for designing a distributed version of Clarkson's algorithm for LP-type problems.

Lemma 2.2 [25]). *For any set-system $(\mathcal{X}, U)$ of VC dimension λ, any weight function $w : \mathcal{X} \to \mathbb{R}$, and $\varepsilon \in (0, 1)$, a set family $\mathcal{N} \subseteq \mathcal{X}$ obtained by randomly sampling*

$$m_{\varepsilon,\lambda,\delta} = \max\left(\frac{8\lambda}{\varepsilon}\log\frac{8\lambda}{\varepsilon}, \frac{4}{\varepsilon}\log\frac{2}{\delta}\right) \tag{1}$$

sets with probability proportional to their weights is an ε-net of $\mathcal{X}$ with probability at least $1 - \delta$.

3 Algorithms

In this section we present our algorithms for Result 1. We will work with a special class of LP-type problems that contains the most natural LP-type problems that we are aware of, including linear programming, Linear SVMs, and Core SVMs mentioned earlier. In particular, we require the LP-type problem $(\mathcal{S}, f)$ to satisfy the following properties:

(P1) Each constraint $X \in \mathcal{S}$ is associated with a set of elements $S_X \subseteq R$ (R is the range of f).
(P2) For any $\mathcal{A} \subseteq \mathcal{S}$, $f(\mathcal{A})$ is the minimal element of $\bigcap_{X \in \mathcal{A}} X$.

It is useful to think of R as the set of feasible solutions. For example, in the case of linear programming, $R = \mathbb{R}^d$ with the natural ordering induced by scalar product with the vector c in the objective function. Each constraint (inequality) $X \in \mathcal{S}$ corresponds to the subset of points S_X which satisfy the constraint, and $f(\mathcal{A})$ is equal to the point which satisfies all constraints in $\mathcal{A}$ and has a minimal scalar product with c. For convenience, we use X and S_X interchangeably.

For this special class of LP-type problems, we define the VC dimension of the problem $(\mathcal{S}, f)$ as the VC dimension of the set system $(\mathcal{S}, R)$.

In the following, we first give a general meta-algorithm for solving LP-type problems with Properties (P1) and (P2), and then show how to implement this meta-algorithm efficiently in each model.

3.1 The Meta Algorithm for LP-Type Problems

Our meta-algorithm follows Clarkson's algorithm [16] for linear programming, but we use a different sampling procedure (by using ε-net) which enables us to work with general LP-type problems with bounded VC dimension; it also significantly simplifies the analysis and facilitates the implementation of our algorithm in the big data models we consider. We further use a different weight increase rate after each iteration, which is essential for reducing the number of passes in the streaming, and the number of rounds in the coordinator and MPC models.

The algorithm proceeds in iterations. We maintain a weight function $w : \mathcal{S} \to \mathbb{R}$ throughout the algorithm which is initialized by setting $w(S) = 1$ for all $S \in \mathcal{S}$. In each iteration, we first sample a set family $\mathcal{N}$ of $m := m_{\varepsilon,\lambda_{\mathcal{S}},\frac{2}{3}}$ sets from $\mathcal{S}$ with probability proportional to their weights so as to obtain an ε-net $\mathcal{N}$ of $\mathcal{S}$ (according to Lemma 2.2). We then compute a basis $\mathcal{B}$ of $\mathcal{N}$, and the set $\mathcal{V}$ of constraints which violate the basis $\mathcal{B}$. If $w(\mathcal{V}) \leq \varepsilon \cdot w(\mathcal{S})$, then we say this iteration "succeeds", and update the weights of all sets $S \in \mathcal{V}$ by setting $w(S) \leftarrow (n^{1/r}) \cdot w(S)$. Otherwise, we say this iteration

ALGORITHM 1: A Meta-Algorithm for LP-Type Problems

Input: An LP-type problem $(\mathcal{S}, f)$ satisfying Properties (P1) and (P2) and integer $r \leq \ln n$.
Output: $f(\mathcal{S})$.

1 Let $\varepsilon := \frac{1}{10 \cdot \nu_{\mathcal{S},f} \cdot n^{1/r}}$, and λ as the VC dimension of the LP-type problem $(\mathcal{S}, f)$.
2 Set $w(S) = 1$ for every $S \in \mathcal{S}$.
3 **repeat**
4 Sample a family $\mathcal{N} \subseteq \mathcal{S}$ of size $m := m_{\varepsilon,\lambda,\frac{2}{3}}$ by picking each set in $\mathcal{S}$ with probability proportional to w for the parameter $m_{\varepsilon,\lambda,\frac{2}{3}}$ in Lemma 2.2.
5 Compute a basis $\mathcal{B}$ of $\mathcal{N}$.
6 Let $\mathcal{V} = \{S \in \mathcal{S} \mid f(\mathcal{B} \cup \{S\}) > f(\mathcal{B})\}$ be the family of sets in $\mathcal{S}$ that violate $\mathcal{B}$.
7 **if** $w(\mathcal{V}) \leq \varepsilon \cdot w(\mathcal{S})$ **then**
8 Set $w(S) = (n^{1/r}) \cdot w(S)$ for every set $S \in \mathcal{V}$.
9 **end**
10 **until** $\mathcal{V} = \emptyset$;
11 **return** $f(\mathcal{B})$.

"fails", and continue to the next one without modifying the weights. A pseudo-code is provided in Algorithm 1.

In the following, we first establish the correctness of the meta-algorithm and then bound the number of iterations it needs.

Lemma 3.1. *When Algorithm 1 stops, it correctly computes* $f(\mathcal{S})$.

Proof. At the end of the algorithm, we have $\mathcal{V} = \emptyset$. This means that for any $S \in \mathcal{S} \setminus \mathcal{B}$, we have $f(\mathcal{B} \cup \{S\}) = f(\mathcal{B})$ by the monotonicity property of f. By the locality property and induction we obtain that $f(\mathcal{B}) = f(\mathcal{B} \cup (\mathcal{S} \setminus \mathcal{B})) = f(\mathcal{S})$, finalizing the proof. ∎

We now bound the number of iterations. We say that an iteration of Algorithm 1 (at Lines 4 to 8) is *successful* iff $w(\mathcal{V}) \leq \varepsilon \cdot w(\mathcal{S})$ in this iteration.

Claim 3.2. *Each iteration of Algorithm 1 is successful with probability at least* $2/3$.

Proof. Since the VC dimension of $(\mathcal{S}, R)$ is λ, by Lemma 2.2, with probability at least $2/3$, the family $\mathcal{N}$ sampled in Line 4 is an ε-net for $(\mathcal{S}, R)$ with respect to the weight function w. In the following, we condition on this event.

Let $x := f(\mathcal{B})$. By Property (P2) of the LP-type problems we consider, we know that x is the minimal element in the intersection of all sets in $\mathcal{B}$ according to the ordering of R. For any set $S \in \mathcal{S}$ to violate $\mathcal{B}$, we need to have $x \notin S$; otherwise $f(\mathcal{B} \cup \{S\}) = x$ which is in contradiction with $f(\mathcal{B} \cup \{S\}) > f(\mathcal{B})$. Recall that $\mathcal{V}$ is the family of all sets in $\mathcal{S}$ that violate $\mathcal{B}$. Suppose towards a contradiction that $w(\mathcal{V}) > \varepsilon \cdot w(\mathcal{S})$. Since none of the sets in $\mathcal{V}$ contain x, and $\mathcal{N}$ is an ε-net, by definition there is a set $S' \in \mathcal{N}$ where S' does not contain x. But this is in contradiction with $\mathcal{B}$ being a basis. To see this, if $f(\mathcal{B}) = f(\mathcal{N})$, then x belongs to all sets in $\mathcal{N}$, and consequently it should also be in S'. We thus have $w(\mathcal{V}) \leq \varepsilon \cdot w(\mathcal{S})$, finalizing the proof. ∎

Lemma 3.3. *The number of iterations in Algorithm 1 is* $O(\nu \cdot r)$ *with probability at least* $1 - e^{-\Omega(\nu \cdot r)}$, *where* ν *denotes the combinatorial dimension of* $(\mathcal{S}, f)$.

Proof. Recall that the weight function $w(\cdot)$ is updated only when an iteration is successful, and each iteration succeeds with probability at least $2/3$ by Claim 3.2. By Chernoff bound (Proposition 2.1), we have that if the algorithm terminates in t iterations, then with probability at least $1 - e^{-\Omega(t)}$, at least $t/2$ of these iterations are successful.

We now focus on successful iterations. Let $w_i(\cdot)$ be the weight function $w(\cdot)$ after the i-th successful iteration. Initially, for any $S \in \mathcal{S}$ we have $w_0(S) = 1$ (and thus $w_0(\mathcal{S}) = n$). We claim that for any integer $t \geq 1$, if Algorithm 1 reaches the t-th successful iteration, then

$$n^{t/\nu r} \leq w_t(\mathcal{S}) \leq e^{t/10\nu} \cdot n. \tag{2}$$

We establish Eq (2) in the following two claims.

Claim 3.4. *For any integer* $t \geq 1$, *we have* $n^{t/\nu r} \leq w_t(\mathcal{S})$.

Proof. Fix an arbitrary basis $\mathcal{B}^* = \{B_1, \ldots, B_k\}$ of $\mathcal{S}$ for some $k \leq \nu$ (recall that by definition, ν is size of the largest basis). Since $\mathcal{B}^* \subseteq \mathcal{S}$, we have $w_t(\mathcal{B}^*) \leq w_t(\mathcal{S})$ for any $t > 0$. We thus only need to show $n^{t/\nu r} \leq w_t(\mathcal{B}^*)$.

The first observation is that in any iteration, if $\mathcal{V} \neq \emptyset$ then we must have $\mathcal{V} \cap \mathcal{B}^* \neq \emptyset$. Indeed, if $\mathcal{V} \cap \mathcal{B}^* = \emptyset$, then $f(\mathcal{B}) = f(\mathcal{B} \cup \mathcal{B}^*) = f(\mathcal{S})$, where the first equality is by the locality property of f and induction, and the second equality holds since $\mathcal{B}^*$ is a basis for $\mathcal{S}$. However, this is in contradiction with the fact that $\mathcal{V} \neq \emptyset$.

Let us now define $\mathcal{B}_i$ as the basis of the ε-net computed in the i-th successful iteration. For any $j \in [k]$, let a_j be the number of iterations i such that $B_j \in \mathcal{B}^*$ violates $\mathcal{B}_i$. That is,

$$a_j = \left|\{i \in [t] \mid f(\mathcal{B}_i) < f(\mathcal{B}_i \cup \{B_j\})\}\right|.$$

Since $\mathcal{V} \cap \mathcal{B}^* \neq \emptyset$ in each of the first t successful iterations, there must exist at least one B_j which violates $\mathcal{B}_i$ for each $j \in [t]$. We thus have $\sum_{j=1}^{k} a_j \geq t$. Moreover, by the weight update rule of the algorithm, we can write the weight of $\mathcal{B}^*$ as $w_t(\mathcal{B}^*) = \sum_{j=1}^{k} \left(n^{1/r}\right)^{a_j}$. By combining these and Jensen's inequality we have

$$w_t(\mathcal{B}^*) \geq k \left(n^{1/r}\right)^{\sum_{j=1}^{k} a_j/k} \geq \left(n^{1/r}\right)^{t/k} \geq n^{t/\nu r},$$

since $k \le \nu$. This concludes the proof of Claim 3.4. ∎

Claim 3.5. *For any integer $t \ge 1$, we have*

$$w_t(\mathcal{S}) \le e^{t/10\nu} \cdot n\,.$$

Proof. For any iteration $t \ge 1$, the weight update procedure at Line 8 of Algorithm 1 gives

$$w_{t+1}(\mathcal{S}) = w_t(\mathcal{S}) + (n^{1/r}-1)\cdot w_t(\mathcal{V}) \le w_t(\mathcal{S}) + (n^{1/r})\cdot w_t(\mathcal{V}). \tag{3}$$

Moreover, by the condition at Line 7 of the algorithm, we have,

$$w_t(\mathcal{V}) \le \varepsilon \cdot w_t(\mathcal{S}) = \frac{1}{10\nu \cdot n^{1/r}} \cdot w_t(\mathcal{S}), \tag{4}$$

by the choice of ε in the algorithm. Combining (3) and (4) we have

$$w_t(\mathcal{S}) \le \left(1 + \frac{1}{10\nu}\right)^t \cdot w_0(\mathcal{S}) \le e^{t/10\nu} \cdot n. \quad ∎$$

We get back to the analysis of the number of iterations. By Eq (2) we have $n^{t/\nu r} \le e^{t/10\nu} n$, hence, $\frac{t}{\nu} \le \frac{10r \ln n}{10 \ln n - r}$. Since $r \le \ln n$, we have $\frac{t}{\nu} \le \frac{10}{9} r$. Therefore the number of successful iterations cannot exceed $\frac{10}{9}\nu r$, and hence the total number of iterations is bounded by $\frac{20}{9}\nu r$ with probability $1 - e^{-\Omega(\nu r)}$. ∎

Remark 3.6. We can easily turn our Las-Vegas algorithm in this section (Algorithm 1) into a Monte-Carlo algorithm by the following modifications: First we pick an ε-net of size $m_{\varepsilon,\lambda_{\mathcal{S}},1/(n\nu)}$, and second, the algorithm return "FAIL" whenever $w(\mathcal{V}) > \varepsilon w(\mathcal{S})$, which will not happen in the first $O(\nu r) = O(\nu \log n)$ iterations with probability at least $1 - \nu \log n \cdot 1/(n\nu) \ge 1 - o(1)$.

3.2 Implementation in the Streaming Model

Starting from this section, we show how to implement Algorithm 1 in the three big data models considered in the paper. We start with the streaming algorithm. In the multi-pass streaming model the elements of $\mathcal{S}$ arrive one by one, and $f(\cdot)$ is known to the algorithm at the beginning. We allow the algorithm to make multiple linear scans of the input.

The main challenge in the streaming implementation of Algorithm 1 is that we cannot afford to store the weights of all elements in $\mathcal{S}$ which are needed in the ε-net sampling. To resolve this issue, we instead store the set of bases computed at all the *successful* iterations – these are the only iterations that we change the weight function – in a collection $\mathcal{B}$, using which we can compute the weight of each element of $\mathcal{S}$ *on the fly*. In particular, the weight of a set $S_i \in \mathcal{S}$ in iteration j of the algorithm, namely, $w_j(S_i)$, is computed as $w_j(S_i) := (n^{1/r})^{a_i}$ where $a_i := |\{\mathcal{B} \in \mathcal{B} \mid f(\mathcal{B} \cup \{S_i\}) > f(\mathcal{B})\}|$. It is immediate to verify that this indeed implements the same weight function in Algorithm 1. It is also easy to see that having access to these weights, we can sample each set with probability proportional to its weight using the weighted version of reservoir sampling [14], and hence implement each iteration of Algorithm 1 in one pass over the stream.

The rest of Algorithm 1 can be implemented in the streaming model in a straightforward way. Let $T_b(m)$ be the time complexity of computing a basis for a set of size m, and $T_v(t,b)$ be the time complexity of finding all elements in a set $\mathcal{T} \subseteq \mathcal{S}$ of size t which violate a set $\mathcal{B}$ of size b, i.e., all $S \in \mathcal{S}$ such that $f(\mathcal{B} \cup S) > f(\mathcal{B})$. This allows us to prove the following theorem.

Theorem 1. *Suppose $(\mathcal{S}, f)$ is an LP-type problem with combinatorial dimension ν, VC dimension λ, and bit-complexity $\mathtt{bit}(\mathcal{S})$ for each element of $\mathcal{S}$. For any integer $r \le \ln n$, we can compute $f(\mathcal{S})$ with high probability in the streaming model, using $O(\nu r)$ passes, and $\widetilde{O}(\lambda n^{1/r} \cdot \nu + \nu^2) \cdot \mathtt{bit}(\mathcal{S})$ space. The total running time of the algorithm is also $O(\nu r \cdot T_v(n,\nu) + \nu r \cdot T_b(\lambda n^{1/r} \cdot \nu))$.*

Proof. The correctness of the algorithm follows from Lemma 3.1. As each iteration of Algorithm 1 can be implemented in one pass, the total number of passes needed by our streaming algorithm is $O(\nu r)$ with high probability by Lemma 3.3.

Recall that the size of each ε-net $\mathcal{N}$ sampled in Algorithm 1 is $m = m_{\varepsilon,\lambda,\frac{2}{3}} = \widetilde{O}\left(\lambda \nu n^{1/r}\right)$, by the choice of ε in the algorithm and $m_{\varepsilon,\lambda,\frac{2}{3}}$ in Lemma 2.2. The space needed by the algorithm to store $\mathcal{N}$ in each iteration is $O(m) \cdot \mathtt{bit}(\mathcal{S})$, which is equal to $\widetilde{O}\left(\lambda \nu n^{1/r}\right) \cdot \mathtt{bit}(\mathcal{S})$ bits. We also need to store all bases in successful iterations, which requires $O(\nu \cdot r) \cdot O(\nu) \cdot \mathtt{bit}(\mathcal{S}) = \widetilde{O}(\nu^2) \cdot \mathtt{bit}(\mathcal{S})$ (since $r = O(\log n)$) as each basis requires $O(\nu) \cdot \mathtt{bit}(\mathcal{S})$ bits to represent and there are total of $O(\nu r)$ such bases.

Each pass of the algorithm involves performing a violation test over the n elements of $\mathcal{S}$, which takes $O(T_v(n,\nu))$ time. And computing a basis of m elements which takes $O(T_b(m))$ times. The running time follows by multiplying these numbers by the number of passes, and by the choice of m. ∎

3.3 Implementation in the Coordinator Model

Recall that in the coordinator model the input set $\mathcal{S}$ is arbitrarily partitioned among k sites $P_1, \ldots, P_k$ such that for any $i \in [k]$, the site P_i receives the elements $\mathcal{S}_i$. The k sites and the coordinator want to jointly compute $f(\mathcal{S}) = f(\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k)$ via communication. The function f is a public knowledge, that is, all parties know how to evaluate the

function $f(\mathcal{T})$ for any $\mathcal{T} \in 2^{\mathcal{S}}$ assuming $\mathcal{T}$ resides entirely on that machine.

Similar to the streaming model, the main step here is also the implementation of the ε-net sampling procedure in Algorithm 1. We show in Appendix A.1 how this step can be performed efficiently, leading to the following theorem.

THEOREM 2. *Suppose $(\mathcal{S}, f)$ is an LP-type problem with combinatorial dimension ν, VC dimension λ, and bit-complexity $\texttt{bit}(\mathcal{S})$ for each element of $\mathcal{S}$. For any integer $r \leq \ln n$, we can compute $f(\mathcal{S})$ with high probability in the coordinator model with $k \geq 2$ machines, using $O(\nu r)$ rounds, and $\widetilde{O}(\lambda n^{1/r} \cdot \nu^2 + k \cdot \nu^2) \cdot \texttt{bit}(\mathcal{S})$ communication in total. The local computation time of the coordinator is $O(\nu r \cdot (T_b(\lambda n^{1/r} \cdot \nu) + k\nu))$ and the local computation time of the i-th site is $O(\nu r \cdot T_v(n_i, \nu))$ where $n_i := |\mathcal{S}_i|$.*

3.4 Implementation in the MPC Model

The implementation of Algorithm 1 in the MPC model can be done similarly as that in the coordinator model, by choosing one of the machines to play the role of coordinator. The only problem is that when the number of machines is large, the machines cannot simply send all the messages to the coordinator directly, as it will blow up the load in the coordinator. In Appendix A.2 we show how to get around this using standard primitives in MPC model. We have the following theorem.

THEOREM 3. *Suppose $(\mathcal{S}, f)$ is an LP-type problem with combinatorial dimension ν, VC dimension λ, and bit-complexity $\texttt{bit}(\mathcal{S})$ for each element of $\mathcal{S}$. For any $\delta \in (0, 1)$, we can compute $f(\mathcal{S})$ with high probability in the MPC model using $O(\nu/\delta^2)$ rounds with $\widetilde{O}(\lambda n^{\delta} \cdot \nu^2) \cdot \texttt{bit}(\mathcal{S})$ load per machine.*

4 Examples and Applications

We now give examples of the application of our algorithms for general LP-type problems. We will discuss several fundamental optimization problems in machine learning, namely, linear programming, Linear SVM, and Core SVM. Recall that when implementing our meta algorithm in each model, we have left two functions $T_v(\cdot)$ (the time needed for performing the violation test) and $T_b(\cdot)$ (the time for computing the basis) unspecified. In this section we will provide concrete bounds for these functions in the context of the concrete problems we study. Throughout this section, we assume that the bit-complexity of each number in the input is $O(\log n)$ bits.

4.1 Linear Programming

A linear program is an optimization problem of the type:

$$\min_{x \in \mathbb{R}^d} \sum_{i=1}^{d} c_i x_i \quad \text{subject to} \quad \sum_{i=1}^{d} a_i^j x_i \leq b^j \quad \text{for all } j \in [n]. \tag{5}$$

A d-dimensional linear program can be modeled as an LP-type problem as follows. Let $\mathcal{S}$ be a set family of size n such that for every constraint in (5), there exists a unique element $S \in \mathcal{S}$ which is the half-space in the d-dimensional Euclidean space $\mathbb{R}^d$ containing the points that satisfy this single constraint. We define the function f over subsets of $\mathcal{S}$ such that for every $\mathcal{A} \subseteq \mathcal{S}$, $f(\mathcal{A})$ is the lexicographically smallest point that minimizes the objective value of LP while satisfying only the constraints in $\mathcal{A}$. The linear program (5) now corresponds to the LP-type problem $(\mathcal{S}, f)$ (we use $\mathcal{S}$ as opposed to our previous notation S, since each element of $\mathcal{S}$ is now itself a subset of $\mathbb{R}^d$, and hence $\mathcal{S}$ forms a set family). We refer the interested readers to [32] for more details on connection between linear programming and LP-type problems.

It is known that the combinatorial dimension ν of this particular LP-type problem $(\mathcal{S}, f)$ is at most $d + 1$ [32]. The VC dimension λ is also at most $d + 1$ [43]. Finding the basis of any given set of constraints and the violating constraints can also be done easily using standard results (see Appendix B.1 for details). We can thus prove the following theorem using Theorems 1, 2, and 3.

THEOREM 4. *We give the following randomized algorithms for d-dimensional linear programming with n constraints. For any $r \geq 1$ and $\delta \in (0, 1)$:*

- Streaming: *An $O(d \cdot r)$-pass algorithm with $\widetilde{O}(d^3 \cdot n^{1/r})$ space in $\widetilde{O}(n) \cdot \text{poly}(d)$ time.*
- Coordinator: *An $O(d \cdot r)$-round algorithm with $\widetilde{O}(d^4 n^{1/r} + d^3 k)$ total communication in which the coordinator and each site $i \in [k]$ spend $\widetilde{O}(n^{1/r} + k) \cdot \text{poly}(d)$ time and $\widetilde{O}(n_i) \cdot \text{poly}(d)$ time, respectively, where n_i is the number of constraints on site i.*
- MPC: *An $O(d/\delta^2)$-round algorithm with $\widetilde{O}(d^3 n^{\delta})$ load per machine and $\widetilde{O}(n) \cdot \text{poly}(d)$ time in total.*

4.2 Linear Support Vector Machine

In Linear Support Vector Machine (SVM) problem [7], we have a set of tuples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that for each index $j \in [n]$, $x_j \in \mathbb{R}^d$ and $y_j \in \{-1, +1\}$. The goal is to compute a hyperplane $u = (u_1, \ldots, u_d)$ which is the outcome of the following quadratic optimization problem [7]:

$$\min_{u \in \mathbb{R}^d} \|u\|_2^2 \quad \text{s.t.} \quad y_j \cdot \langle u, x_j \rangle \geq 1 \quad \text{for all } j \in [n]. \tag{6}$$

From a geometrical point of view, the problem (6) corresponds to finding a hyperplane which separates the set of point $\{x_1, \ldots, x_n\}$ according to their labels with the maximum margin value (if possible); see, e.g., [7] for more information on this fundamental problem.[3] Note that the problem (6) is *not* a linear program. However, one can show that it is an LP-type problem $(\mathcal{S}, f)$ where $\mathcal{S}$ is a set family in $\mathbb{R}^d$ in which every set contains the points that satisfy a particular constraint, and $f(\mathcal{A})$ for $\mathcal{A} \subseteq \mathcal{S}$ computes the optimal solution of (6) given only the constraints to $\mathcal{A}$ [32] (unlike linear programming, the optimal solution to (6) under any set of constraints is unique and hence we do not need the lexicographically first constraint).

The combinatorial dimension of $(\mathcal{S}, f)$ is $\nu \leq d + 1$ [32], and the VC dimension of $(\mathcal{S}, \mathbb{R}^d)$ is $\lambda \leq d + 1$ [43]. One can again use standard results to find a basis of any given set of constraints and all violating constraints of a given basis for this problem (see Appendix B.2). This allows us to prove the following theorem using Theorems 1, 2, and 3.

Theorem 5. *We give the following randomized algorithms for d-dimensional linear support vector machine problem with n constraints. For any $r \geq 1$ and $\delta \in (0, 1)$:*

- Streaming: *An $O(d \cdot r)$-pass algorithm with $\widetilde{O}(d^3 \cdot n^{1/r})$ space in $\widetilde{O}(n) \cdot \text{poly}(d)$ time.*
- Coordinator: *An $O(d \cdot r)$-round algorithm with $\widetilde{O}(d^4 n^{1/r} + d^3 k)$ total communication in which the coordinator and each site $i \in [k]$ spend $\widetilde{O}(n^{3/r} + k) \cdot \text{poly}(d)$ time and $\widetilde{O}(n_i) \cdot \text{poly}(d)$ time, respectively, where n_i is the number of constraints on site i.*
- MPC: *An $O(d/\delta^2)$-round algorithm with $\widetilde{O}(d^3 n^{\delta})$ load per machine and $\widetilde{O}(n + n^{3\delta}) \cdot \text{poly}(d)$ time in total.*

4.3 Core Vector Machine

Tsang at el. [42] proposed *core vector machines* as a way of speeding up kernel methods in SVM training (see [7]). This is achieved by reformulating the original kernel method as an instance of the *minimum enclosing ball* (MEB) problem, defined as follows: Given a set of points $P := \{p_1, \ldots, p_n\}$ in $\mathbb{R}^d$, find a center p and a minimum radius r such that all the points in P are within a d-dimensional sphere of radius r centered at p. MEB can be formulated as the following optimization problem:

$$\min_{r \in \mathbb{R}, p \in \mathbb{R}^d} r \quad \text{subject to} \quad \|p - p_j\|_2 \leq r \quad \text{for all } j \in [n]. \quad (7)$$

This problem is also an LP-type problem $(\mathcal{S}, f)$ formulated similarly to linear programming and Linear SVM [32]. The combinatorial dimension of $(\mathcal{S}, f)$ is $\nu \leq d + 1$ [32] and the VC dimension of $(\mathcal{S}, \mathbb{R}^d)$ is $\lambda \leq d + 1$ [44]. We obtain the following theorem for this problem using Theorems 1, 2, and 3 (see Appendix B.3).

Theorem 6. *We give the following randomized algorithms for d-dimensional core vector machine problem with n constraints. For any integer $r \geq 1$:*

- Streaming: *An $O(d \cdot r)$-pass algorithm with $\widetilde{O}(d^3 \cdot n^{1/r})$ space in $\widetilde{O}(n + n^{3/r}) \cdot \text{poly}(d)$ time.*
- Coordinator: *An $O(d \cdot r)$-round algorithm with $\widetilde{O}(d^4 n^{1/r} + d^3 k)$ total communication in which the coordinator and each site $i \in [k]$ spend $\widetilde{O}(n^{3/r} + k) \cdot \text{poly}(d)$ time and $\widetilde{O}(n_i) \cdot \text{poly}(d)$ time, respectively, where n_i is the number of constraints on site i.*
- MPC: *An $O(d/\delta^2)$-round algorithm with $\widetilde{O}(d^3 n^{\delta})$ load per machine and $\widetilde{O}(n + n^{3\delta}) \cdot \text{poly}(d)$ time in total.*

5 Lower Bounds

In this section we prove information-theoretic lower bounds for linear programming that hold against *any* algorithm. We obtain our lower bounds by establishing the *communication complexity* for 2-dimensional linear programming, and then translating it to lower bounds in the big data models. In the following, we first give some background on communication complexity and then present an intermediate problem, called two-curve intersection problem (TCI), that we consider en route to proving our result for linear programming. We then prove a lower bound for TCI and present its implications for linear programming in the streaming and coordinator models.

5.1 Background

Communication Complexity. We focus on the standard two-party communication complexity model of Yao [45]. In this model, Alice and Bob receive an input $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, respectively. In an r-round protocol, Alice and Bob can communicate up to r messages with each other. In particular, for an even r, Bob first sends a message to Alice, followed by a message from Alice to Bob, and so on, until Bob receives the last message and outputs the answer. For an odd r, the only difference is that Alice starts first and then the players continue like before until Bob outputs the answer.

The communication complexity of a problem $P : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$, denoted by $\text{CC}(P)$, is the minimum worst-case communication cost of any protocol (possibly randomized) that can solve P with probability at least $2/3$. The r-round communication complexity of P, denoted by $\text{CC}^r(P)$, is similarly defined with respect to protocols that are allowed at most r rounds of communication.

Augmented Indexing. In the Augmented Indexing Problem, denoted by Aug-Index_n, Alice is given a binary string

[3] Our algorithm works effectively for the hard-margin Linear SVM. In the case of the soft-margin Linear SVM, the optimization problem can also be formulated in the form of LP-type problem, but the dimension of such formulation is large – proportional to the size of input.

$x \in \{0,1\}^n$, and Bob is given an index $i \in \{0,1\}$ plus the first $i-1$ bits of the string x, i.e., $x_1, \ldots, x_{i-1}$. The goal is for Bob to output the bit x_i. It is well-known that 1-round communication complexity of this problem is $\mathrm{CC}^1(\text{Aug-Index}_n) = \Omega(n)$ (see, e.g. [35]).

Information Theory. Throughout this section, we use bold-face fonts, say $\boldsymbol{A}$, to denote random variables, and normal font, say A, to denote their realizations. For a random variable $\boldsymbol{A}$, $\mathrm{supp}(\boldsymbol{A})$ denotes its support and $\mathrm{dist}(\boldsymbol{A})$ its distribution. We sometimes abuse the notation and use $\boldsymbol{A}$ and $\mathrm{dist}(\boldsymbol{A})$ interchangeably. Furthermore, for a t-tuple $(X_1, \ldots, X_t)$ and any integer $i \in [t]$, we define $X^{<i} := (X_1, \ldots, X_{i-1})$ and $X^{>i} := (X_{i+1}, \ldots, X_t)$.

For random variables $\boldsymbol{A}, \boldsymbol{B}$, let $\mathbb{H}(\boldsymbol{A})$ and $\mathbb{I}(\boldsymbol{A}; \boldsymbol{B})$ denote the Shannon entropy and mutual information, respectively. We use $\mathbb{D}(\boldsymbol{A} \,||\, \boldsymbol{B})$ and $\|\boldsymbol{A} - \boldsymbol{B}\|_{tvd}$ to denote the KL-divergence and total variation distance between $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. Appendix C.1 provides formal definitions and basic properties of these functions.

5.2 The Two-Curve Intersection Problem (TCI)

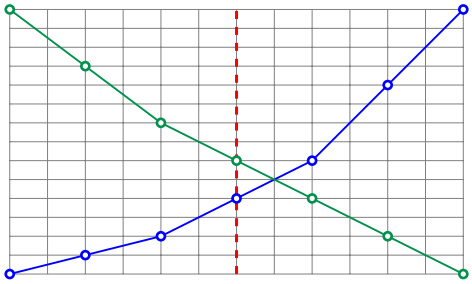
We consider the following problem, whose lower bound implies a lower bound for linear programming in the two-dimensional Euclidean space (as we show shortly).

Alice and Bob are given sequences of n numbers $A := \langle a_1, \ldots, a_n \rangle$ and $B := \langle b_1, \ldots, b_n \rangle$ in $\mathbb{Q}^n$, respectively, such that:

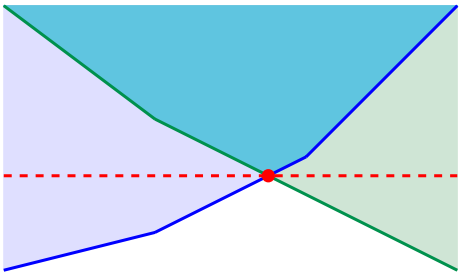
(1) *Monotonicity:* A is monotonically increasing and B is monotonically decreasing.
(2) *Convexity:* For any $i \in [n]$, in A we have $a_i - a_{i-1} \leq a_{i+1} - a_i$ and conversely in B we have $b_i - b_{i-1} \geq b_{i+1} - b_i$.

The goal is to find the *smallest index* $i^* \in [n]$ such that $a_{i^*} \leq b_{i^*}$ but $a_{i^*+1} > b_{i^*+1}$, under the promise that such an index always exists. We can interpret the sequence A as a two-dimensional curve in $\mathbb{R}^2$ that goes through the points $(1, a_1), (2, a_2), \cdots (n, a_n)$ (similarly for B). We refer to this problem as the *two-curve intersection* problem and denote it by TCI_n for sequences of length n (or TCI in general). See Figure 1a for an illustration of this problem.

Connection to 2-Dimensional Linear Programming. We can reduce the two-curve intersection problem to an instance of 2-dimensional linear programming as follows (see Figure 1b). Extend each segment of the curve in Alice's and Bob's input to obtain a line that defines a constraint in which all points above this line are feasible (blue region for Alice and green region for Bob in Figure 1b). The feasible region of this linear program is the set of points in $\mathbb{R}^2$ that lie above both of Alice's and Bob's curve. By minimizing the y-axis on the feasible region, we obtain the first "fractional" point in which Alice's curve goes above Bob's curve, and by rounding down the x-axis of this point, we obtain the index i^* of TCI.



(a) The answer here is $i = 4$ (dashed line).



(b) Linear programming formulation.

Figure 1: An illustration of the two-curve intersection problem and its connection to linear programming.

Geometric Notations. We work in the two-dimensional Euclidean space $\mathbb{R}^2$. We use $p \in \mathbb{R}^2$ to denote a point, and $p.x$ and $p.y$ to denote its x and y coordinates respectively. Throughout, all the points used have rational coordinates (i.e., in $\mathbb{Q}^2$). For two points p_1, p_2 and integers $a \leq b$, we define $\mathsf{LineSegment}(p_1, p_2, a, b)$ as the sequence of $b - a + 1$ numbers $\langle z_a, z_{a+1}, \ldots, z_b \rangle$ such that for all $i \in [a : b]$, (i, z_i) belongs to the unique line in $\mathbb{R}^2$ that passes through the points p_1 and p_2. We use the following elementary geometric facts.

Fact 5.1. *Let* $\langle z_a, \ldots, z_b \rangle := \mathsf{LineSegment}(p_1, p_2, a, b)$.
(1) For every $i \in (a : b]$, $z_i - z_{i-1} := \frac{p_2.y - p_1.y}{p_2.x - p_1.x}$.
(2) For every $i \in [a : b]$, $z_i = \frac{p_2.y - p_1.y}{p_2.x - p_1.x} \cdot (i - p_1.x) + p_1.y = \frac{p_2.y - p_1.y}{p_2.x - p_1.x} \cdot (i - p_2.x) + p_2.y$.

We also define a notion called *step curve*. For a string $X = (x_1, \ldots, x_m) \in \{0,1\}^m$ and a parameter $\alpha \geq 1$, $\mathsf{StepCurve}(X, \alpha)$ is the sequence of $m + 1$ numbers z_i such that $z_0 = 0$ and for all $i \in [m]$, $z_i := z_{i-1} + \alpha + i + x_i$.

5.3 Communication Complexity of TCI

Our goal is to prove the following theorem.

Theorem 7. *For any* $r \geq 1$, $\mathrm{CC}^r(\mathrm{TCI}_n) = \Omega(\frac{1}{r^2} \cdot n^{1/r})$.

The proof of Theorem 7 is based on an inductive argument, following the general *round-elimination* approach in communication complexity (see, e.g. [35, 40]). In this approach, one proves the lower bound for r-round problems by showing that a "too good" r-round protocol will imply a too good $(r-1)$-round protocol, by reducing the r-round problem to *multiple* instances of the $(r-1)$-round problem. Following this argument inductively, we will end up with a protocol using only 1 round. We then directly prove that such a too good 1-round protocol *cannot* exist.

5.3.1 Base Case: One-Round Protocols As a warm-up, we first prove Theorem 7 for $r = 1$, i.e., 1-round protocols.

Lemma 5.2. $\mathrm{CC}^1(\mathsf{TCI}_n) = \Omega(n)$.

Proof. We prove this lemma using a reduction from the Augmented Indexing Problem on a universe of size $n-1$. Given an instance of $\mathsf{Aug\text{-}Index}_{n-1}$ with input $x \in \{0,1\}^{n-1}$ to Alice and $i^* \in [n-1]$ plus $x_1, \ldots, x_{i^*-1}$ to Bob, the players construct the following instance of TCI_n (without any communication):

(1) Alice creates $A := \langle a_1, \ldots, a_n \rangle := \mathsf{StepCurve}(x, 0)$.
(2) Bob creates

$$B := \langle b_1, \ldots, b_n \rangle = \mathsf{LineSegment}(p_1, p_2, 1, n),$$

where $p_1 := (n, 1)$ and $p_2 := (i^*, a_{i^*} + i^* + 1)$.

The players then run the protocol for TCI_n on this instance and Bob outputs $x_{i^*} = 1$ (in answer to the Aug-Index instance) iff i^* is returned by the protocol for TCI_n as the answer on this instance.

Correctness of the reduction. We first verify that the sequences A and B constructed by Alice and Bob satisfy the promise of the TCI_n input. By Fact 5.1, B is both monotone and convex. It is also easy to see that A is monotonically increasing: for all $i \geq 2$, $a_i \geq a_{i-1} + i \geq a_{i-1}$. Finally, to verify the convexity of A, notice that $a_i - a_{i-1} = i + x_{i-1} \leq i + 1$ while $a_{i+1} - a_i = i + 1 + x_i \geq i + 1$.

We now prove the correctness of the output in the reduction. Suppose first that $x_{i^*} = 0$. In this case, $a_{i^*+1} = a_{i^*} + i^* + 1 + x_{i^*} = a_{i^*} + i^* + 1 = b_{i^*}$ by definition. On the other hand, $a_{i^*+2} > b_i$ for all $i > i^*$ as $b_i < b_{i^*}$ and $a_{i^*+2} > a_{i^*+1} = b_{i^*}$. As a result, the correct index in TCI_n is $i^* + 1$. Now suppose $x_{i^*} = 1$. In this case, $a_{i^*+1} > b_{i^*}$ while $a_{i^*} < b_{i^*}$. As such, the correct index in TCI_n is i^*, finalizing the proof of the correctness of the reduction.

Communication cost of the reduction. The instance of TCI_n can be created with no communication. As such,

$$\mathrm{CC}^1(\mathsf{TCI}_n) \geq \mathrm{CC}^1(\mathsf{Aug\text{-}Index}_{n-1}) = \Omega(n). \qquad \blacksquare$$

5.3.2 General Lower Bound: The Outline We now switch to the main part of the argument in which we prove Theorem 7 for all integers $r \geq 1$. In this section we outline our high level approach. In this section, we will oversimplify many details, and the discussions will be informal for the sake of intuition.

We design a family of distribution $\mathcal{D}_1, \mathcal{D}_2, \ldots$, where $\mathcal{D}_r$ is hard distribution for r-round protocols. Distribution $\mathcal{D}_1$ is the distribution of hard instances obtained in Lemma 5.2 (from the hard distribution of Aug-Index). Each instance I in the distribution $\mathcal{D}_r$ is then constructed roughly as follows: we sample $n^{1/r}$ instances from the distribution $\mathcal{D}_{r-1}$ each over $n^{(r-1)/r}$ points. Let us call these instances $I_1, \ldots, I_{n^{1/r}}$. We *embed* these instances inside I so that the following two properties are satisfied: (i) the answer to TCI_n on instance I is the same as the answer to $\mathsf{TCI}_{n^{(r-1)/r}}$ on instance $I_{z^\star}$ for some $z^\star \in [n^{1/r}]$ chosen uniformly at random, and (ii) the first player to speaks (namely Alice for odd r and Bob for even r) is *oblivious* to the identity of $z^\star$.

The proof of the communication lower bound then goes as follows. Using information-theoretic arguments, we can argue that if the first message of the protocol is of size $o(n^{1/r})$, then it only reveals $o(1)$ bits of information about an "average" embedded instance I_i for $i \in [n_r]$ of $\mathcal{D}_{r-1}$. In particular, since the sender of the first message is oblivious to the identity of the $z^\star$ (by property (ii)), the first message only reveals $o(1)$ bits of information about the instance $I_{z^\star}$. This effectively means that the distribution of the instance $I_{z^\star}$ is essentially the same as $\mathcal{D}_{r-1}$ even after the first round. However, by property (i), the players now need to solve the instance $I_{z^\star}$ on $n^{(r-1)/r}$ elements sampled from distribution $\mathcal{D}_{r-1}$ in $r-1$ rounds. By induction, this requires $\Omega(\left(n^{(r-1)/r}\right)^{1/r-1}) = \Omega(n^{1/r})$ bits, which implies the desired lower bound for r-round protocols.

The outline above is arguably the most straightforward application of round-elimination (see, e.g. the tree-pointer-jumping problem in [11]). Unfortunately however, this approach does not work directly in our application. In particular, in the discussion above, we left the specifics of how the $(r-1)$-round instances $I_1, \ldots, I_{n^{1/r}}$ are embedded together to form I. For the above information-theoretic arguments to work, these instances need to be sampled *independently* of each other. On the other hand, for us to be able to embed them together in a valid instance of TCI, we need to ensure that they collectively preserve monotonicity and convexity properties of TCI. This requires *correlating* the instances $I_1, \ldots, I_{n^{1/r}}$, impeding the use of previous information-theoretic argument.

We get around this challenge by carefully "revealing extra information" about the inputs of the players to each other (similar to the reduction from Aug-Index in Lemma 5.2),

which allows to "control" the correlation between different instances $I_1, \ldots, I_{n^{1/r}}$ in terms of these revealed information. We then show that even with this extra information, the two properties above for embedded instances continue to hold, and at the same time, we have enough independence in the instances to make the information-theoretic arguments outlined above work.

We comment that this construction of hard instances of TCI and the proof of the corresponding communication lower bound is one of the main technical contributions of this paper.

5.3.3 General Lower Bound: The Hard Input Distribution We use an integer $N \geq 1$ as a parameter in defining all other parameters of our hard distribution. In particular, for r-round instances, $n_r = N^r$ is the number of points given to Alice and Bob, and $m_r := N$ is the number of $(r-1)$-round instances "embedded" inside the r-round instance. We also define the following two *operators* on instances that are used in our lower bound construction (their roles will become more evident once we give the proper definition of the hard distribution).

- **Slope-Shift Operator:** In any instance I of our hard distribution $\mathcal{D}_r$, the input to Alice is constructed using several (potentially different) StepCurve functions. By applying slope-shift operator on instance I with parameter α, we increase the second parameter in every application of StepCurve in constructing Alice's input by an *additive* factor of α. As a result, any segment in Alice's input constructed with StepCurve$(*, \beta)$ becomes StepCurve$(*, \alpha+\beta)$. We ensure that the operator also changes the slope of Bob's input by α.
- **Origin-Shift Operator:** By applying the origin-shift operator with point $p_A \in \mathbb{R}^2$ from Alice's side in an instance I of $\mathcal{D}_r$, we shift *all* points in the instance I along the same line so that the *left-most* point of Alice's input will be on the point p_A. Similarly, by applying the origin-shift operator with $p_B \in \mathbb{R}^2$ from Bob's side, we shift *all* points along the same line so that the *right-most* point of Bob's input will be on p_B. This operator clearly does not change the slope of any line segment in players' inputs.

We are now ready to describe our hard input distribution.

Distribution $\mathcal{D}_r$: The Hard Distribution for r-round Protocols of TCI. We define the procedure Instance that given a parameter r, construct an instance of TCI.

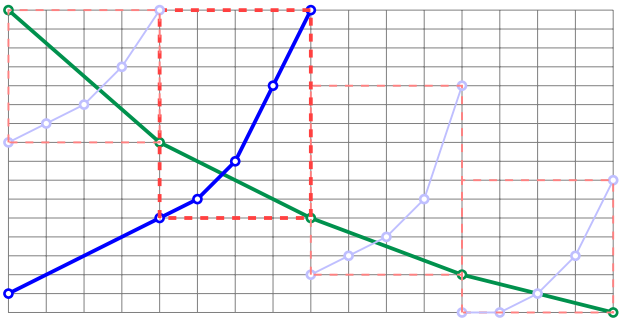
Instance(r).
(1) If $r = 1$, sample (A, B) from the distribution of Lemma 5.2; otherwise, define
$(A, B) :=$ EvenInstance(r) for even r and
$(A, B) :=$ OddInstance(r) for odd r.
(2) Return the points (A, B) as the r-round instance.

We now define the EvenInstance procedure inside Instance.

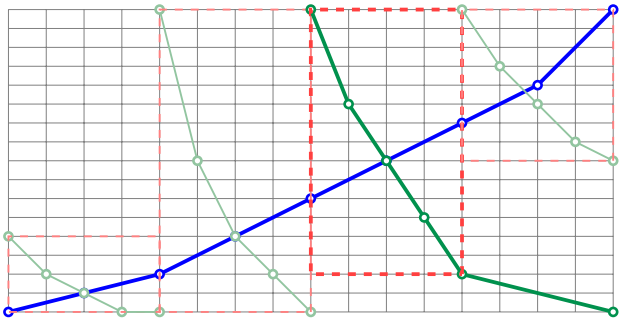
EvenInstance(r).
(1) Sample m_r instances (C_i, D_i) *independently* from Instance$(r-1)$.
(2) For $i = m_r$ down to 1 do:
(a) Let p_B^{i+1} be the left-most point of Bob's input in (C_{i+1}, D_{i+1}) (define $p_B^{m_r+1} := (n_r, 0)$). Apply the origin-shift operator with point p_B^{i+1} from Bob's side on the instance (C_i, D_i).
(b) Let α_r^{i+1} be the largest slope of any segment in (C_{i+1}, D_{i+1}). Apply the slope-shift operator with slope α_r^{i+1} on (C_i, D_i).
(3) Sample $z_r^\star \in [m_r]$ uniformly at random.
(4) Define $A := (A_1, \ldots, A_{m_r})$ where $A_{z_r^\star} = C_{z_r^\star}$; the remaining A_i's for $i \neq z_r^\star$ are constructed by extending the curve in $A_{z_r^\star}$ on both its endpoints along straight lines.
(5) Define $B := (B_1, \ldots, B_{m_r})$ where $B_i = D_i$ for all $i \in [m_r]$.

We refer to instances (C_i, D_i) as *sub-instances*. Several remarks are in order about these sub-instances. Firstly, even though they were originally sampled independently, by applying the origin-shit and slope-shift operators, we have correlated these instances. In particular, each instance (C^i, D^i) depends on instances (C^j, D^j) for $j > i$. Moreover, note that *not* all the points in these instances appear in the final instance (A, B). In particular, we only use the points in $C_{z_r^\star}$ to define $A_{z_r^\star}$; the remaining points in $A \setminus A_{z_r^\star}$ are obtained differently from $C_1, \ldots, C_{m_r}$ (the points in B are however identical to the points in $D_1, \ldots, D_{m_r}$). Nevertheless, the remaining instances still play a marginal role in the definition of the players' inputs because these points define the starting point and starting slope of each sub-instance. In the following, we refer to (A, B) as the *actual* input of Alice and Bob, and refer to the points in $(C_1, D_1), \ldots, (C_{m_r}, D_{m_r})$ that are not part of (A, B) as *fooling* inputs. Figure 2a gives an illustration of EvenInstance.

We use the term sub-instance for both (A_i, B_i) and (C_i, D_i) pairs. For any $i \in [m_r]$, we use $A_i := (a_{i,1}, \ldots, a_{i,n_{r-1}})$ and $B_i := (b_{i,1}, \ldots, b_{i,n_{r-1}})$ to denote the points in sub-instance (A_i, B_i). The following proposition ensures that instances sampled by EvenInstance do not violate the monotonicity



(a) An Illustration of EvenInstance.



(b) An Illustration of OddInstance.

Figure 2: An illustration of EvenInstance **and** OddInstance**. Thick blue and green curves denote the actual inputs of Alice and Bob. Similarly, light blue and green curves denote the fooling inputs. Each red dashed rectangle denotes one sub-instance (thick ones show the special sub-instance).**

and convexity properties of TCI across sub-instances (proof appears in Appendix C.2).

Proposition 5.3. *For* (A, B) *sampled from* EvenInstance*, assuming each sub-instance* (A_i, B_i) *satisfies monotonicity and convexity of* TCI*, then* (A, B) *also satisfies monotonicity and convexity.*

We refer to the instance $(A_{z_r^\star}, B_{z_r^\star}) = (C_{z_r^\star}, D_{z_r^\star})$ as the *special* sub-instance of (A, B). The next proposition signifies the role of the special sub-instance in EvenInstance (see Appendix C.2).

Proposition 5.4. *For instances* (A, B) *sampled from* EvenInstance*, the answer to* $\mathsf{TCI}(A, B)$ *is the same as the answer to* $\mathsf{TCI}(C_{z_r^\star}, D_{z_r^\star})$.

The definition of OddInstance procedure is similar to EvenInstance by switching the role of Alice and Bob. Due to space constraints, we postpone this description to Appendix C.3 (the odd-round analogues of the lemmas and claims also appear in Appendix C.3).

Actual vs Fooling Inputs. As we already observed in the proof of Lemma 5.2, providing the players with extra information about the input of the other player (i.e., giving Bob the first i^* points in Alice's input) facilitates the proof of the lower bound. This is also the case for our hard instances for $r > 1$ round protocols. In the following observations, we state several properties of this extra information which is crucial for our information-theoretic lower bound for TCI (the observations for odd rounds appear in C.3).

Observation 5.5. In EvenInstance, there is a one-to-one mapping between $(A_{z_r^\star}, B_{z_r^\star})$ and the original $(C_{z_r^\star}, D_{z_r^\star})$ (before applying any operator), assuming we are given $(C^{>z_r^\star}, D^{>z_r^\star})$.

The reason behind Observation 5.5 is simply the operators applied to each (C_i, D_i) are functions of $(C^{>i}, D^{>i})$ in EvenInstance and the special sub-instance is just a "copy" of $(C_{z_r^\star}, D_{z_r^\star})$.

Observation 5.5 implies that if players have access to $(C^{>z_r^\star}, D^{>z_r^\star})$ in EvenInstance as an extra input, then they can determine the original distribution of their special sub-instance. This is the main reason that we provide the players with this extra input in our reduction.

Observation 5.6. In EvenInstance, the index $z_r^\star \in [m_r]$ is chosen independently of B and $(C_1, D_1), \ldots, (C_{m_r}, D_{m_r})$.

This observation follows directly from the construction of the instances. Observation 5.6 implies that *even given the extra input*, the player that sends the first message is oblivious to the identity of the special sub-instance.

5.3.4 General Lower Bound: The Communication Complexity

We prove Theorem 7 by induction on the number of rounds, with Lemma 5.2 forming the base of the induction. We have the following lemma.

Lemma 5.7. *For any* $r \geq 1$ *and any* $(1/3)$*-error protocol* π_r *for instances of* TCI *sampled from the distribution* $\mathcal{D}_r$*, the communication cost of* π_r *is* $\Omega(N/r^2)$.

From now on we fix a *deterministic* protocol π_r for TCI on $\mathcal{D}_r$; we later use Yao's minimax principle [46] to extend the lower bound to randomized protocols. We use $\boldsymbol{\Pi}$ to denote the random variable for messages communicated in the protocol, and write $\boldsymbol{\Pi} = (\boldsymbol{\Pi}_1, \ldots, \boldsymbol{\Pi}_r)$, where $\boldsymbol{\Pi}_\ell$ denotes the message communicated in round ℓ. We further use $\boldsymbol{Z}$ to denote the index $z_r^\star$, which corresponds to the index of the special sub-instance. Let $(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{m_r})$ and $(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_{m_r})$ denote the random variables for the points A and B and their partitioning into sub-instances respectively, and $(\boldsymbol{C}_1, \ldots, \boldsymbol{C}_{m_r})$ and $(\boldsymbol{D}_1, \ldots, \boldsymbol{D}_{m_r})$ for C and D.

We start with the following lemma that formalizes our intuition that players cannot reveal information about the special sub-instance in their first round. We first consider

even-round protocols (see Appendix C.3 for the lemma for odd-round protocols). .

Lemma 5.8. *For any even integer r, and any r-round protocol π_r with worst-case message length ℓ on instances of $\mathcal{D}_r$,*

$$\mathbb{I}((\boldsymbol{A}_Z, \boldsymbol{B}_Z); \boldsymbol{\Pi}_1 \mid \boldsymbol{C}^{>Z}, \boldsymbol{D}^{>Z}, Z) \le \ell/N.$$

Proof. We start by expanding the LHS:

$$\begin{aligned}
&\mathbb{I}((\boldsymbol{A}_Z, \boldsymbol{B}_Z); \boldsymbol{\Pi}_1 \mid \boldsymbol{C}^{>Z}, \boldsymbol{D}^{>Z}, Z) \\
&\quad= \mathop{\mathbb{E}}_{z\in[m_r]}\left[\mathbb{I}((\boldsymbol{A}_z, \boldsymbol{B}_z); \boldsymbol{\Pi}_1 \mid \boldsymbol{C}^{>z}, \boldsymbol{D}^{>z}, Z = z)\right] \\
&\qquad\text{(definition of conditional mutual information)} \\
&\quad= \frac{1}{m_r}\sum_{z=1}^{m_r} \mathbb{I}((\boldsymbol{A}_z, \boldsymbol{B}_z); \boldsymbol{\Pi}_1 \mid \boldsymbol{C}^{>z}, \boldsymbol{D}^{>z}, Z = z) \\
&\qquad\text{(distribution of } Z \text{ is uniform over } [m_r]) \\
&\quad= \frac{1}{m_r}\sum_{z=1}^{m_r} \mathbb{I}((\boldsymbol{C}_z, \boldsymbol{D}_z); \boldsymbol{\Pi}_1 \mid \boldsymbol{C}^{>z}, \boldsymbol{D}^{>z}, Z = z) \\
&\qquad\text{(by Observation 5.5)} \\
&\quad= \frac{1}{m_r}\sum_{z=1}^{m_r} \mathbb{I}((\boldsymbol{C}_z, \boldsymbol{D}_z); \boldsymbol{\Pi}_1 \mid \boldsymbol{C}^{>z}, \boldsymbol{D}^{>z}),
\end{aligned}$$

where the last equality is due to the fact that the joint distribution of all random variables $(\boldsymbol{C}_z, \boldsymbol{D}_z), \boldsymbol{\Pi}_r, \boldsymbol{C}^{>z}, \boldsymbol{D}^{>z}$ is independent of the event $Z = z$. Indeed, for even r, the message $\boldsymbol{\Pi}_1$ sent by Bob is a function of $(B_1, \ldots, B_{m_r})$ and $(C_1, D_1), \ldots, (C_{m_r}, D_{m_r})$, and by Observation 5.6, these random variables are all independent of $z_r^\star$. As such, removing the conditioning on the event $Z = z$ does not change the distribution of variables above. Finally,

$$\begin{aligned}
&\frac{1}{m_r}\sum_{z=1}^{m_r} \mathbb{I}((\boldsymbol{C}_z, \boldsymbol{D}_z); \boldsymbol{\Pi}_r \mid \boldsymbol{C}^{>z}, \boldsymbol{D}^{>z}) \\
&\quad= \frac{1}{m_r}\cdot \mathbb{I}(\boldsymbol{C}_1, \ldots, \boldsymbol{C}_{m_r}, \boldsymbol{D}_1, \ldots, \boldsymbol{D}_{m_r}; \boldsymbol{\Pi}_1) \\
&\qquad\text{(chain rule of mutual information)} \\
&\quad\le \frac{1}{m_r}\cdot \mathbb{H}(\boldsymbol{\Pi}_1) \le \frac{\ell}{m_r}. \qquad (\mathbb{H}(\boldsymbol{\Pi}_1) \le \ell)
\end{aligned}$$

The lemma follows by noting that $m_r = N$. ∎

To continue, we need the following definition.

> Distribution μ_e for even r: For an assignment $(\Pi_1, C^{>z}, D^{>z}, z)$ (denoted by E for short) to $(\boldsymbol{\Pi}_1, \boldsymbol{C}^{>Z}, \boldsymbol{D}^{>Z}, \boldsymbol{Z})$, we define $\mu_e(E)$ as the distribution of $(\boldsymbol{A}_z, \boldsymbol{B}_z)$ in $\mathcal{D}_r$ conditioned on $\boldsymbol{\Pi}_r = \Pi_r, Z = z$, $\boldsymbol{C}^{>z} = C^{>z}, \boldsymbol{D}^{>z} = D^{>z}$.

Using Lemma 5.8, we have the following claim (see Appendix C.3 for the claim for odd-round protocols).

Claim 5.9. *For any even integer r and any r-round protocol π_r with worst-case message length $o(N/r^2)$,*

$$\mathop{\mathbb{E}}_{E=(\Pi_1, C^{>z}, D^{>z}, z)}\left[\|\mu_e(E) - \mathcal{D}_{r-1}\|_{tvd}\right] = o(1/r).$$

Proof. By the connection between mutual information and KL-divergence (Fact C.1), we have,

$$\begin{aligned}
&\mathbb{I}((\boldsymbol{A}_Z, \boldsymbol{B}_Z); \boldsymbol{\Pi}_1 \mid \boldsymbol{C}^{>Z}, \boldsymbol{D}^{>Z}, Z) = \\
&\mathop{\mathbb{E}}_{E=(\Pi_1, C^{>z}, D^{>z}, z)}[\mathbb{D}(\mathrm{dist}(\boldsymbol{A}_Z, \boldsymbol{B}_Z \mid E \setminus \Pi_1) \,||\, \mathrm{dist}(\boldsymbol{A}_Z, \boldsymbol{B}_Z \mid E))] \\
&= \mathop{\mathbb{E}}_{E=(\Pi_1, C^{>z}, D^{>z}, z)}[\mathbb{D}(\mathcal{D}_{r-1} \,||\, \mu_e(E))];
\end{aligned}$$

Conditioned on $Z = z$, distribution of $(\boldsymbol{A}_Z, \boldsymbol{B}_Z)$ is the same as the original distribution of $(\boldsymbol{C}_Z, \boldsymbol{D}_Z)$ by Observation 5.5. Furthermore,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{E=(\Pi_1, C^{>z}, D^{>z}, z)}[\mathbb{D}(\mathcal{D}_{r-1} \,||\, \mu_e(E))] \\
&\quad\ge \mathop{\mathbb{E}}_{E=(\Pi_r, C^{>z}, D^{>z}, z)}\left[2\cdot\|\mathcal{D}_{r-1} - \mu_e(E)\|_{tvd}^2\right] \\
&\qquad\text{(Pinsker's inequality (Fact C.3))} \\
&\quad\ge 2\cdot\left(\mathop{\mathbb{E}}_{E=(\Pi_r, C^{>z}, D^{>z}, z)}\left[\|\mathcal{D}_{r-1} - \mu_e(E)\|_{tvd}\right]\right)^2. \\
&\qquad\text{(Jensen's inequality)}
\end{aligned}$$

By Lemma 5.8, $\mathbb{I}((\boldsymbol{A}_Z, \boldsymbol{B}_Z); \boldsymbol{\Pi}_r \mid \boldsymbol{C}^{>Z}, \boldsymbol{D}^{>Z}, Z) = o(1/r^2)$, implying that,

$$\mathop{\mathbb{E}}_{E=(\Pi_r, C^{>z}, D^{>z}, z)}\left[\|\mathcal{D}_{r-1} - \mu_e(E)\|_{tvd}\right] = o(1/r).$$

This finalizes the proof. ∎

Define the recursive function $\delta(k) = \delta(k-1) - o(1/r)$ with base case $\delta(1) = 1/4$ (here r is the number of rounds).

Lemma 5.10. *Any deterministic $\delta(r)$-error protocol π_r on $\mathcal{D}_r$ requires $\Omega(N/r^2)$ communication.*

Proof. The proof is by induction on the number of rounds. The base case for $r = 1$ follows from Lemma 5.2. We now prove the induction step.

Suppose the lemma holds for all integers up to $r-1$, we prove it for r-round protocols. Given a r-round protocol π_r for $\mathcal{D}_r$ that violates the induction hypothesis, we construct a $(r-1)$-round protocol π_{r-1} for $\mathcal{D}_{r-1}$ that also violates the induction hypothesis, a contradiction. The protocol π_{r-1} is constructed in two steps: we first construct a randomized protocol π' from π_r, and then fix the randomness of the protocol to achieve a deterministic protocol.

We now describe π'. For simplicity, we only give the protocol for even choices of r; the extension to odd values is straightforward. Given an instance $(A, B) \sim \mathcal{D}_{r-1}$, protocol π' works as follows:

(1) Using public randomness, the players sample $(\Pi_1, C^{>z}, D^{>z}, z_r^\star)$ from the distribution $\mathcal{D}_r$.
(2) Bob samples remaining coordinates (C_j, D_j) for $j < z_r^\star$ using private randomness from distribution
$$\mathcal{D}_r \mid (\Pi_1, C^{>z}, D^{>z}, z_r^\star).$$
(3) Alice sets $A_{z_r^\star} = A$ and Bob sets $B_{z_r^\star} = B$ by applying the appropriate slope-shift and origin-shift operators based on $C^{>z}, D^{>z}$ which is known to both Alice and Bob.
(4) The players then fill the rest of their input in $(A_1, \ldots, A_{m_r})$ and $(B_1, \ldots, B_{m_r})$; Bob knows all of $(C_1, D_1), \ldots, (C_{m_r}, D_{m_r})$ and can perform the needed slope-shift and origin-shift operators, and Alice simply needs to extend $A_{z_r^\star}$ across straight lines.
(5) The players run π_r on these new points from the second round onwards, assuming that the first communicated message was Π_1. They output the index returned by π_r.

Communication cost of π' is clearly at most as the communication cost of π_r. We now prove the correctness of π'.

Claim 5.11. *Assuming π_r is a $\delta(r)$-error protocol for $\mathcal{D}_r$, π' will be a $(\delta(r) + o(1/r))$-error protocol for $\mathcal{D}_{r-1}$.*

Proof. We have,

$$\Pr_{\mathcal{D}_{r-1}}(\pi' \text{ errs}) = \mathop{\mathbb{E}}_{E=(\Pi_1, C^{>z}, D^{>z}, z_r^\star)} \left[\Pr_{\mathcal{D}_{r-1}}(\pi_r \text{ errs} \mid E) \right]$$
(by Proposition 5.4)
$$\leq \mathop{\mathbb{E}}_{E=(\Pi_1, C^{>z}, D^{>z}, z_r^\star)} \left[\Pr_{\mu_e(E)}(\pi_r \text{ errs}) + \|\mu_e(E) - \mathcal{D}_{r-1}\|_{tvd} \right]$$
(by Fact C.2)
$$\leq \delta(r) + o(1/r^2),$$
(π_r is a $(\delta(r))$-error protocol, and by Claim 5.9)

finalizing the proof. ∎

We are now ready to complete the proof of Lemma 5.10. By Claim 5.11, π' is a $(\delta(r)+o(1/r))$-error protocol for $\mathcal{D}_{r-1}$. π' is a randomized protocol. However, by an averaging argument, we can fix the randomness of π' to obtain a deterministic $(\delta(r) + o(1/r))$-error protocol for $\mathcal{D}_{r-1}$ with the same communication cost $o(N/r^2)$. As $\delta(r) + o(1/r) = \delta(r-1)$, this contradicts the induction hypothesis. We thus have that the communication cost of π_r is $\Omega(N/r^2)$, proving the induction step. This concludes the proof. ∎

Lemma 5.7 now follows immediately from Lemma 5.10 as $\delta(r) = 1/4 + \sum_{k=1}^{r} o(1/r) = 1/4 + o(1) < 1/3$, and by the easy direction of Yao's minimax principle [46].

5.3.5 Proof of Theorem 7

Proof of Theorem 7. By Lemma 5.7, any (1/3)-error r-round protocol for TCI_n requires $\Omega(N/r^2)$ communication on instances of $\mathcal{D}_r$. In these instances, $n = N^r$ by the construction of $\mathcal{D}_r$. Plugging in $N = n^{1/r}$, we obtain $\mathrm{CC}^r(\mathsf{TCI}_n) = \Omega(\frac{1}{r^2} \cdot n^{1/r})$.

We conclude this proof by making the following remark: A r-round instance of our problem consists of at most N^{r-1} applications of StepCurve, each having a larger slope than the previous one by an additive factor of N. As a result, the largest slope using in our construction is $N^{O(r)}$. This implies that the bit-complexity of the numbers we use is bounded by $\log(N^{O(r)}) = O(\log n)$. ∎

As a corollary of Theorem 7, using the connection between two-curve intersection problem and linear programming outlined in Section 5.2, we obtain the following.

Corollary 8. *For any integer $r \geq 1$, any two-player r-round protocol for 2-dimensional linear programming with n constraints requires $\Omega(\frac{1}{r^2} \cdot n^{1/r})$ communication.*

5.4 Lower Bounds for Linear Programming in Big Data Models

We now give some straightforward applications of our communication complexity lower bound for linear programming to streaming and coordinator models, and formalize Result 2.

The Streaming Model. It is well-known that communication complexity lower bounds imply space lower bounds on the space complexity of streaming algorithms (see, e.g. [2, 24]). Using this connection in conjunction with Corollary 8, we have, to establish the following theorem.

Theorem 9. *For any integer $r \geq 1$, any streaming algorithm that makes r passes over the constraints of a 2-dimensional linear program with n constraints and finds the optimal solution with probability at least 2/3 requires $\Omega(\frac{1}{r^3} \cdot n^{1/2r})$ space.*

The Coordinator Model. Any r-round distributed protocol implies a $2r$-round protocol in our communication model. Hence,

Theorem 10. *For any integer $r \geq 1$, any r-round algorithm that finds the optimal solution of a 2-dimensional linear program with n constraints partitioned across $k \geq 2$ sites in the coordinator model with probability at least 2/3 requires $\Omega(\frac{1}{r^2} \cdot n^{1/2r})$ communication.*

Acknowledgments

Qin Zhang would like to thank Yufei Tao for introducing the problem (as well as the two-curve intersection problem as a way of proving a lower bound for linear programming).

References

[1] Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *ICALP 2011*, pages 526–538, 2011.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[3] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and implementation of the logicblox system. In *SIGMOD*, pages 1371–1382, 2015.

[4] Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *STOC*, pages 698–711, 2016.

[5] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *J. ACM*, 64(6):40:1–40:58, 2017.

[6] Robert G Bland and Michel Las Vergnas. Orientability of matroids. *Journal of Combinatorial Theory, Series B*, 24(1):94–123, 1978.

[7] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, pages 144–152, 1992.

[8] Hervé Brönnimann, Bernard Chazelle, and Jiří Matoušek. Product range spaces, sensitive sampling, and derandomization. *SIAM J. Comput.*, 28(5):1552–1575, 1999.

[9] Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[10] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

[11] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *STOC*, pages 641–650, 2008.

[12] Timothy M. Chan. Improved deterministic algorithms for linear programming in low dimensions. In *SODA*, pages 1213–1219, 2016.

[13] Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.

[14] M. T. Chao. A general purpose unequal probability sampling plan. *Biometrika*, 69:653–656, 1982.

[15] Kenneth L. Clarkson. Linear programming in $o(n3^{d^2})$ time. *Inf. Process. Lett.*, 22(1):21–24, 1986.

[16] Kenneth L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, 1995.

[17] Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.

[18] Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006.

[19] David J. Crisp and Christopher J. C. Burges. A geometric interpretation of v-svm classifiers. In *NIPS*, pages 244–250, 1999.

[20] Martin E. Dyer. On a multidimensional search technique and its application to the euclidean one-centre problem. *SIAM J. Comput.*, 15(3):725–738, 1986.

[21] Martin E. Dyer and Alan M. Frieze. A randomized algorithm for fixed-dimensional linear programming. *Math. Program.*, 44(1-3):203–212, 1989.

[22] Bernd Gärtner and Martin Jaggi. Coresets for polytope distance. In *SOCG*, pages 33–42, 2009.

[23] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *ISAAC*, pages 374–383, 2011.

[24] Sudipto Guha and Andrew McGregor. Tight lower bounds for multi-pass stream computation via pass elimination. In *ICALP*, pages 760–772, 2008.

[25] David Haussler and Emo Welzl. epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

[26] Hal Daumé III, Jeff M. Phillips, Avishek Saha, and Suresh Venkatasubramanian. Efficient protocols for distributed classification and optimization. In *ALT*, pages 154–168, 2012.

[27] Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Jonathan Ullman, Ali Vakilian, and Anak Yodpinyanee. Fractional set cover in the streaming model. In *APPROX/RANDOM 2017*, pages 12:1–12:20, 2017.

[28] Gil Kalai. A subexponential randomized simplex algorithm (extended abstract). In *STOC*, pages 475–482, 1992.

[29] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA*, pages 938–948, 2010.

[30] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in õ(vrank) iterations and faster algorithms for maximum flow. In *FOCS*, pages 424–433, 2014.

[31] Makrynioti, Nantia and Vasiloglou, Nikolaos and Pasalic, Emir and Vassalos, Vasilis. Data Science with Linear Programming. http://delbp.github.io/DeLBP-2017/papers/DeLBP-2017_paper_2CR.pdf, 2017.

[32] Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996.

[33] Andrew McGregor. private communication.

[34] Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, 1984.

[35] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *J. Comput. Syst. Sci.*, 57(1):37–49, 1998.

[36] Ketan Mulmuley. *Computational geometry - an introduction through randomized algorithms*. Prentice Hall, 1994.

[37] J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

[38] Jeff M. Phillips, Elad Verbin, and Qin Zhang. Lower bounds for number-in-hand multiparty communication complexity, made easy. In *SODA*, pages 486–501, 2012.

[39] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits: (on lower bounds for modern parallel computation). In *SPAA*, pages 1–12, 2016.

[40] Pranab Sen and Srinivasan Venkatesh. Lower bounds for predecessor searching in the cell probe model. *J. Comput. Syst. Sci.*, 74(3):364–385, 2008.

[41] Yufei Tao. Massively parallel entity matching with linear classification in low dimensional space. In *ICDT*, pages 20:1–20:19, 2018.

[42] Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[43] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[44] Roberta S. Wenocur and Richard M. Dudley. Some special vapnik-chervonenkis classes. *Discrete Mathematics*, 33(3):313–318, 1981.

[45] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *STOC*, pages 209–213, 1979.

[46] Andrew Chi-Chih Yao. Lower bounds by probabilistic arguments (extended abstract). In *FOCS*, pages 420–428, 1983.

[47] Yinyu Ye and Edison Tse. An extension of karmarkar's projective algorithm for convex quadratic programming. *Math. Program.*, 44(1-3):157–179, 1989.

A Missing Details from Section 3

A.1 Implementation in the Coordinator Model

The main step is to perform the sampling of ε-net which is captured by the following lemma.

Lemma A.1. *The coordinator can sample a subset $\mathcal{N} \subseteq \mathcal{S}$ of size m according to the weight function $w : \mathcal{S} \to \mathbb{R}$ using 2 rounds and $O(m \cdot \mathtt{bit}(\mathcal{S}) + k(\ell/r + 1)\log n)$ bits of communication, where ℓ is the number of times the weight function $w(\cdot)$ has been updated when simulating Algorithm 1 in the coordinator model.*

Proof. The sampling algorithm is as follows. In the first round each site P_i sends $w(\mathcal{S}_i)$ to the coordinator. Note that $w(\mathcal{S}_i)$ for any $i \in [k]$ can be described in $\log(1 + n^{1/r})^{\ell} = O(\ell/r \cdot \log n)$ bits.

In the second round the coordinator generates m i.i.d. random numbers $x_1, \ldots, x_m$ from $[k]$ from the distribution $\mathbf{Pr}[i \text{ is sampled}] = \frac{w(\mathcal{S}_i)}{w(\mathcal{S})}$, and sends the i-th site the number $y_i = \left|\{j \mid x_j = i\}\right|$. After obtaining y_i, site P_i samples y_i elements from its local set $\mathcal{S}_i$ according to the distribution $\mathbf{Pr}[S \text{ is sampled}] = \frac{w(S)}{w(\mathcal{S}_i)}$, and sends the sampled elements to the coordinator. Note that $y_i \leq m \leq n$ for any $i \in [m]$, and thus the communication cost of this round is bounded by $O(k) \cdot O(\ell/r \cdot \log n) + O(m) \cdot \mathtt{bit}(\mathcal{S})$ bits.

Finally, the sampling is indeed with respect to the weight function $w(\cdot)$, since $\mathbf{Pr}[S \text{ is sampled}] = \frac{w(\mathcal{S}_i)}{w(\mathcal{S})} \cdot \frac{w(S)}{w(\mathcal{S}_i)} = \frac{w(S)}{w(\mathcal{S})}$. This concludes the proof. ∎

In order to implement Algorithm 1, each site should also be able to determine the set of violating elements in its input. This can be done easily by asking the coordinator to share the basis computed in each iteration with every site. The proof of Theorem 2 follows directly from that of Theorem 1 by plugging in Lemma A.1.

A.2 Implementation in the MPC Model

Our general strategy is to simulate our implementation of the meta-algorithm for the coordinator model in the MPC model for $r = 1/\delta$ round protocols. The main challenge in implementing this is that once we require the load of roughly n^{δ} per machine, we need to start with $k = n^{1-\delta}$ machines to begin with to fit the whole input across all machines. This means that the number of sites in the simulation is k. But then, if all these machines need to send even one bit to the designated coordinator machine (or vice versa), this requires a load of $n^{1-\delta}$ on the coordinator machine which is prohibitively large for any $\delta < 1/2$.

In order to fix this, we are going to use the by now standard approach of [23]. There are only two steps that the coordinator and the machines need to communicate with each other: (1) when the machines need to send a sample of the ε-net, and (2) when the coordinator needs to send the basis to the machines. The latter can be done easily in $O(1/\delta)$ MPC rounds on machines of memory $O(n^{\delta})$: the coordinator first shares this information with n^{δ} other machines in one round; each of these machines next shares this information with another set of n^{δ} machines (unique to each original machine). In $O(1/\delta)$ rounds all the $n^{1-\delta}$ machines would receive this information (see [23] for more details).

To handle the part when the machines need to send the ε-net $\mathcal{N}$ to the coordinator, we do as follows. Recall that the size of $\mathcal{N}$ is at most $\tilde{O}(\lambda n^{\delta} \nu^2)$, and thus it will fit the memory of the coordinator. However, we first need to sample this according to the correct distribution. In order to do this, we use our approach for implementing the streaming algorithm. Since by the previous part we managed to share the basis computed in each iteration with every machine, as in the case of streaming algorithms, the machines can compute the weights of every constraint they have. The total weight of the constraints can also be computed in $O(1/\delta)$ rounds using the sort and search method of [23]. As a result, each machine can locally perform the sampling of $\mathcal{N}$ and send this information to the coordinator.

To summarize, we can implement each round of the algorithm in the coordinator model for $r = 1/\delta$ in $O(1/\delta)$ MPC rounds, hence proving Theorem 3.

B Missing Details from Section 4

B.1 Linear Programming

Let $\mathrm{T}_{LP}(m, t)$ denotes the time needed to solve a linear program with $\Theta(m)$ constraints and $\Theta(t)$ variables.

Proposition B.1. *For any linear program with n constraints and dimension d:*

- *The time needed to compute a basis of $m \geq d$ given constraints is $T_b(m) = O(d \cdot \mathrm{T}_{LP}(m, d))$.*
- *The time needed to compute all constraints that violate a given basis of size $b = O(d)$ among t constraints is $T_v(t, b) = O(t \cdot d + d \cdot \mathrm{T}_{LP}(d, d))$.*

Proof. To find a basis $\mathcal{B}$ of a set $\mathcal{N}$ of m constraints, we first solve the LP only given the constraints in $\mathcal{N}$ to obtain a point $x^* = (x_1^*, \ldots, x_d^*)$ with optimal value c^*. Recall that in our mapping of LP to an LP-type problem, we need to find a lexicographically smallest optimal solution on constraints in $\mathcal{N}$, which may not be the point x^* even though the objective value is still c^*. We thus write another linear program:

$$\min_{x \in \mathbb{R}^d} x_1$$

$$\text{s.t.} \quad \sum_{i=1}^{d} c_i x_i = c^* \quad \text{and} \quad \sum_{i=1}^{d} a_i^j x_i \leq b^j \quad \text{for all } j \in \mathcal{N}.$$

This allows us to find an optimal solution to the LP with the minimum value of x_1. Repeating this procedure for d iterations and for i-th iteration fixing $x_1, .., x_{i-1}$ computed so far, and finding the minimum value for x_i, allows us to find the lexicographically smallest optimal solution. These LPs all are d-dimensional with $\Theta(m)$ constraints, and hence can be solved in $O(d \cdot \mathrm{T}_{LP}(m, d))$ time in total, finalizing the first part.

A basis of size b in a linear program consists of b constraints of the LP that are all tight by the assignment of the variables. Hence, given the basis, we only need to solve the linear program on a system of b linear inequalities to determine a value of x^* that is tight for all the constraints in the basis. This can be done in $O(d\mathrm{T}_{LP}(d, d))$ time (as we do before). After this, we can simply check the d-dimensional vector x^* against all the t constraints and add each one as a violating set if x^* does not satisfy the constraint in $O(t \cdot d)$ time, finalizing the second part. ∎

Plugging in the currently best known bound for linear programming $\mathrm{T}_{LP}(m, d) = \widetilde{O}(\sqrt{d}\left(dm + d^{2.373}\right))$ ([30], Proposition B.1), and the aforementioned bounds on $\nu, \lambda = O(d)$, we can prove Theorem 4 using Theorems 1, 2, and 3.

B.2 Linear Support Vector Machine

In the following, let $\mathrm{T}_{SVM}(m, d)$ denote the time needed to solve an instance of Linear SVM problem with m constraints and d variables. We show how to implement the basis computation and violation test for Linear SVM in the following proposition.

Proposition B.2. *For any Linear SVM problem with n constraints and dimension d:*

- *The time needed to compute a basis of $m \geq d$ given constraints is $T_b(m) = O(\mathrm{T}_{SVM}(m, d))$.*
- *The time needed to compute all constraints that violate a given basis of size b among t constraints is $T_v(t, b) = O(t \cdot d + \mathrm{T}_{SVM}(d, d))$.*

Proof. To find a basis $\mathcal{B}$ of a set $\mathcal{N}$ of m constraints, we simply need to solve another instance of Linear SVM, i.e., (6), only on the given constraints. This can be done in $O(\mathrm{T}_{SVM}(m, d))$ by definition. The second part can also be solved by solving a linear equation exactly as in the case in Proposition B.1. ∎

Plugging in the currently best known bound for Linear SVM $\mathrm{T}_{SVM}(m, d) = O((m + d)^3)$ ([47], Proposition B.2), and the aforementioned bounds on $\nu, \lambda = O(d)$, we can prove Theorem 5 using Theorems 1, 2, and 3.

B.3 Core Vector Machine

Let $\mathrm{T}_{MEB}(m, d)$ denote the time needed to solve an instance of MEB problem with m constraints and d variables. The following proposition show how to implement the basis computation and violation test for MEB (the proof is identical to Proposition B.2 and is hence omitted).

Proposition B.3. *For any Linear SVM problem with n constraints and dimension d:*

- *The time needed to compute a basis of $m \geq d$ given constraints is $T_b(m) = O(\mathrm{T}_{MEB}(m, d))$.*
- *The time needed to compute all constraints that violate a given basis of size b among t constraints is $T_v(t, b) = O(t \cdot d + \mathrm{T}_{MEB}(d, d))$.*

As MEB can be cast as a convex quadratic program, we have $\mathrm{T}_{MEB}(m, d) = O((m + d)^3)$ by [47] as before. Hence, Theorems 1, 2, and 3 imply the following result.

C Missing Details from Section 5

C.1 Background on Information Theory

Our proof relies on basic concepts from information theory, which we review briefly here. For a broader introduction, we refer the interested reader to the excellent text by Cover and Thomas [18].

Entropy and Mutual Information. The Shannon entropy is defined as $\mathbb{H}(\boldsymbol{A}) := \sum_{A \in \mathrm{supp}(\boldsymbol{A})} \mathbf{Pr}\left(\boldsymbol{A} = A\right) \cdot \log\left(\frac{1}{\mathbf{Pr}(\boldsymbol{A}=A)}\right)$. The conditional entropy of $\boldsymbol{A}$ on random variable $\boldsymbol{B}$ is defined as $\mathbb{H}(\boldsymbol{A} \mid \boldsymbol{B}) := \mathbb{E}_{B \sim \boldsymbol{B}}\left[\mathbb{H}(\boldsymbol{A} \mid \boldsymbol{B} = B)\right]$. The (conditional) mutual information between $\boldsymbol{A}$ and $\boldsymbol{B}$ is $\mathbb{I}(\boldsymbol{A}; \boldsymbol{B} \mid \boldsymbol{C}) := \mathbb{H}(\boldsymbol{A} \mid \boldsymbol{C}) - \mathbb{H}(\boldsymbol{A} \mid \boldsymbol{B}, \boldsymbol{C})$. We shall use the following basic properties of entropy and mutual information throughout.

Measures of Distance Between Distributions. For two distributions μ and ν, the *Kullback-Leibler divergence* between μ and ν is denoted by $\mathbb{D}(\mu \mid\mid \nu)$ and defined as:

$$\mathbb{D}(\mu \mid\mid \nu) := \mathop{\mathbb{E}}_{a \sim \mu}\left[\log \frac{\mathbf{Pr}_{\mu}(a)}{\mathbf{Pr}_{\nu}(a)}\right]. \tag{8}$$

We have the following relation between mutual information and KL-divergence.

Fact C.1. *For random variables $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$,*

$$\mathbb{I}(\boldsymbol{A}; \boldsymbol{B} \mid \boldsymbol{C}) =$$

$$\mathop{\mathbb{E}}_{(b,c) \sim (\boldsymbol{B}, \boldsymbol{C})}\left[\mathbb{D}(\mathrm{dist}(\boldsymbol{A} \mid \boldsymbol{C} = c) \mid\mid \mathrm{dist}(\boldsymbol{A} \mid \boldsymbol{B} = b, \boldsymbol{C} = c))\right].$$

We denote the total variation distance between two distributions μ and ν on the same support Ω by $\|\mu - \nu\|_{tvd}$, defined as:

$$\|\mu - \nu\|_{tvd} := \max_{\Omega' \subseteq \Omega}\left(\mu(\Omega') - \nu(\Omega')\right) = \frac{1}{2} \cdot \sum_{x \in \Omega} |\mu(x) - \nu(x)|. \tag{9}$$

We use the following basic properties of total variation distance.

Fact C.2. *Suppose μ and ν are two distributions for $\mathcal{E}$, then,* $\mathbf{Pr}_\mu(\mathcal{E}) \le \mathbf{Pr}_\nu(\mathcal{E}) + \|\mu - \nu\|_{tvd}$.

The following Pinskers' inequality bounds the total variation distance between two distributions based on their KL-divergence,

Fact C.3 (Pinsker's inequality). *For any distributions μ and ν, $\|\mu - \nu\|_{tvd} \le \sqrt{\frac{1}{2} \cdot \mathbb{D}(\mu \,||\, \nu)}$.*

C.2 Missing Proofs of Section 5.3

Proposition (Restatement of Proposition 5.3). *For (A, B) sampled from* EvenInstance*, assuming each sub-instance (A_i, B_i) satisfies monotonicity and convexity of* TCI*, then (A, B) also satisfies monotonicity and convexity.*

Proof. For B_i's, the monotonicity and convexity follow from the origin-shift operator and slope-shift operator, respectively. For A_i's, different sub-instances are obtained by extending two line segments in $A_{z_r^\star}$, and hence A trivially satisfies the properties. ∎

Proposition (Restatement of Proposition 5.4). *For instances (A, B) sampled from* EvenInstance*, the answer to* TCI(A, B) *is the same as the answer to* TCI$(C_{z_r^\star}, D_{z_r^\star})$.

Proof. Since $(A_{z_r^\star}, B_{z_r^\star}) = (C_{z_r^\star}, D_{z_r^\star})$, and $(C_{z_r^\star}, D_{z_r^\star})$ form a valid instance of TCI, clearly A and B also only cross each other between the points in $(A_{z_r^\star}, B_{z_r^\star})$. ∎

C.3 OddInstance and Odd-Round Protocols

We now turn to the definition of the OddInstance procedure inside Instance.

OddInstance(r).

(1) Sample m_r instances (C_i, D_i) *independently* from Instance$(r-1)$.
(2) For $i = 1$ to m_r do:
   (a) Let p_A^{i-1} be the right-most point of Alice's input in (C_{i-1}, D_{i-1}) (define $p_A^0 := (0, 0)$). Apply the origin-shift operator with point p_A^{i-1} from Alice's side on instance (C_i, D_i).
   (b) Let α_r^{i-1} be the largest slope of any segment in (C_{i-1}, D_{i-1}). Apply the slope-shift operator with slope α_r^{i-1} on (C_i, D_i).
(3) Sample $z_r^\star \in [m_r]$ uniformly at random.
(4) Define $A := (C_1, \ldots, C_{m_r})$.
(5) Define $B := (B_1, \ldots, B_{m_r})$ where $B_{z_r^\star} := D_{z_r^\star}$; the remaining B_i's for $i \neq z_r^\star$ are constructed by extending the curve in $B_{z_r^\star}$ on both its endpoints along straight lines.

Similar to EvenInstance, instances of OddInstance also consists of m_r sub-instances among which $(C_{z_r^\star}, D_{z_r^\star})$ is called the *special* sub-instance. Figure 2b gives an illustration of instances sampled by OddInstance.

Analogues of Even-Round Results. We list the following results as odd-round analogues of the bounds in the paper for even-round protocols. Their proofs all follow immediately from the proof of their even-round counter part.

The following two properties are analogous to Propositions 5.3 and 5.4 for EvenInstance.

Proposition C.4. *For (A, B) sampled from* OddInstance*, assuming each sub-instance (A_i, B_i) satisfies monotonicity and convexity of* TCI*, then (A, B) also satisfies monotonicity and convexity.*

Proposition C.5. *For instances (A, B) sampled from* OddInstance*, the answer to* TCI(A, B) *is the same as the answer to* TCI$(C_{z_r^\star}, D_{z_r^\star})$.

The following two observations are analogous to Observations 5.5 and 5.6 for EvenInstance.

Observation C.6. In OddInstance, there is a one-to-one mapping between $(A_{z_r^\star}, B_{z_r^\star})$ and the original sub-instance $(C_{z_r^\star}, D_{z_r^\star})$ (before applying any operator), assuming we are given $(C^{<z_r^\star}, D^{<z_r^\star})$.

Observation C.7. In OddInstance, the index $z_r^\star \in [m_r]$ is chosen independently of $A = (A_1, \ldots, A_{m_r})$ and

$$(C_1, D_1), \ldots, (C_{m_r}, D_{m_r}).$$

The following lemma for odd-round protocols is analogous to Lemma 5.8 for even-round ones (but note the change in the order of conditioning).

Lemma C.8. *For any odd integer r and any r-round protocol π_r with worst-case message length ℓ on instances of $\mathcal{D}_r$,*

$$\mathbb{I}((\boldsymbol{A}_Z, \boldsymbol{B}_Z); \boldsymbol{\Pi}_1 \mid \boldsymbol{C}^{<Z}, \boldsymbol{D}^{<Z}, Z) \le \ell / N.$$

The following definition is analogous to definition of μ_e.

Distribution μ_o for odd r: For an assignment

$$(\Pi_r, C^{<z}, D^{<z}, z)$$

(denoted by O for short) to $(\boldsymbol{\Pi}_1, \boldsymbol{C}^{<Z}, \boldsymbol{D}^{<Z}, \boldsymbol{Z})$, we define $\mu_o(O)$ as the distribution of $(\boldsymbol{A}_z, \boldsymbol{B}_z)$ in $\mathcal{D}_r$ conditioned on $\boldsymbol{\Pi}_r = \Pi_r, \boldsymbol{Z} = z, \boldsymbol{C}^{<z} = C^{<z}, \boldsymbol{D}^{<z} = D^{<z}$.

The following claim for odd-round protocols is analogous to Claim 5.9 for even-round ones (again note the change in the order of conditioning and the distribution).

Claim C.9. *For any odd integer r and any r-round protocol π_r with worst-case message length $o(N/r^2)$,*

$$\mathop{\mathbb{E}}_{O=(\Pi_1, C^{<z}, D^{<z}, z)} \left[\|\mu_o(O) - \mathcal{D}_{r-1}\|_{tvd} \right] = o(1/r).$$