Massively Parallel Entity Matching with Linear Classification in Low Dimensional Space

Yufei Tao

Chinese University of Hong Kong, Shatin, Hong Kong
taoyf@cse.cuhk.edu.hk

Abstract

In *entity matching classification*, we are given two sets R and S of objects where whether r and s form a match is known for each pair $(r, s) \in R \times S$. If R and S are subsets of domains $D(R)$ and $D(S)$ respectively, the goal is to discover a *classifier function* $f : D(R) \times D(S) \to \{0, 1\}$ from a certain class satisfying the property that, for every $(r, s) \in R \times S$, $f(r, s) = 1$ if and only if r and s are a match.

Past research is accustomed to running a learning algorithm directly on all the labeled (i.e., match or not) pairs in $R \times S$. This, however, suffers from the drawback that even reading through the input incurs a quadratic cost. We pursue a direction towards removing the quadratic barrier. Denote by T the set of matching pairs in $R \times S$. We propose to accept R, S, and T as the input, and aim to solve the problem with cost proportional to $|R| + |S| + |T|$, thereby achieving a large performance gain in the (typical) scenario where $|T| \ll |R||S|$.

This paper provides evidence on the feasibility of the new direction, by showing how to accomplish the aforementioned purpose for *entity matching with linear classification*, where a classifier is a linear multi-dimensional plane separating the matching and non-matching pairs. We actually do so in the MPC model, echoing the trend of deploying massively parallel computing systems for large-scale learning. As a side product, we obtain new MPC algorithms for three geometric problems: linear programming, batched range counting, and dominance join.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Entity Matching, Linear Programming, Range Counting, Dominance Join, Massively Parallel Computation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2018.20

1 Introduction

Entity matching has garnered considerable attention from the database community (representative works: [7, 13, 15, 22, 21, 23]). Generally speaking, the objective is to determine whether two objects o_1 and o_2 belong to the same entity, e.g., "are the two voice recordings by the same person?". For this purpose, we are given training sets R and S such that, each pair $(r, s) \in R \times S$ carries a *label*, indicating whether r matches s. The goal of learning is to produce a *classifier* of a certain type that correctly decides the matching results for all $(r, s) \in R \times S$. This classifier will then be applied to new, unknown, object pairs $(o_1, o_2) \notin R \times S$.

The literature on entity matching is accustomed to applying a learning algorithm directly on the labeled $R \times S$. This suffers from the drawback that even reading $R \times S$ itself forces a quadratic cost. This issue has been recognized; for instance, [21] stated that "*the resulting quadratic complexity is inefficient for large datasets even on cloud infrastructures.*" In practice, it is commonly dealt with using the so-called "blocking technique" (e.g., [13, 21, 22, 23]), which relies on heuristic rules to divide R and S into *blocks*, such that only objects from the same block can possibly match each other (e.g., a block includes the male recordings, while the female recordings form another block). This technique, however, offers no provable improvement in theory.

We will pursue a different direction towards removing the quadratic pitfall. The fundamental observation is that often times the non-matching pairs in $R \times S$ by far out-number the matching ones. So, why not supply only R, S, and the set T of matching pairs in $R \times S$? Conceptually, nothing is lost because if $(r, s) \in R \times S$ does not appear in T, it must be a non-matching pair. The hope is that we can design algorithms that produce precisely the same learning result, but with cost proportional to $|R| + |S| + |T|$, thus harvesting a significant gain when $|T| \ll |R||S|$. Algorithmically, this is essentially asking: is it possible to carry out the learning *without* enumerating $R \times S$?

This paper will establish theoretical evidence on the feasibility of this direction. We will look at a specific form of entity matching—called *entity matching with linear classification*—where the classifier is a multi-dimensional plane separating matching and non-matching pairs. We will achieve the purpose on coarse-grained parallel computing platforms (e.g., MapReduce [10] and Spark [29]), responding to the need of deploying such platforms for large-scale learning. Our algorithmic endeavor will require us to attack several fundamental geometric problems as well. Next, after introducing the computation model, we will formally define all the relevant problems, and then present our results.

1.1 The Computation Model

We will work under the *massively parallel computation* (MPC) model due to its popularity in the database area [4, 1, 3, 5, 19, 24, 25, 20]. In this model, p machines are interconnected in a network. An algorithm runs in *rounds*, each of which has two phases. In the first phase, every machine carries out computation locally, whereas in the second phase, the machines exchange information over the network. Specifically, a machine u can send messages to arbitrary machines in the second phase, as long as u has prepared those messages in the first phase (of the same round). This means, for example, that u is not allowed to decide what to send based on what is received in this round.

The performance of an algorithm is measured by (i) the amount of network communication, and (ii) The number of rounds executed. The first metric, specifically, is measured in *load*. Formally, the *load of a round* equals $\max_{i=1}^{p} \lambda_i$ where λ_i is the number of words communicated (sending and receiving combined) by the i-th machine in this round. The *load* of the algorithm equals the maximum load of all the rounds executed. In the early development of the model, research concentrated on algorithms that perform an exceedingly small number (e.g., 1 or 2) of rounds [2, 3, 25], because old MapReduce implementations suffered from expensive system-level overhead in executing a round. Such overhead has been considerably reduced in today's systems [14, 29], making it feasible for algorithms to perform more (but preferably $O(1)$) rounds [4, 1, 19, 28, 24, 20]. This is especially true when additional rounds help to reduce the load by significant factors.

Several remarks are in order. By default, at the beginning of an MPC algorithm, the input is distributed on the p machines at the will of an adversary, but in a balanced manner, i.e., each machine stores $O(N/p)$ elements, where N is the input size. Some of our algorithms drop the "balanced" requirement, and instead allow an arbitrarily uneven distribution of the N elements onto the p machines. All the previous work on the MPC model considers the value of p to be less than the input size N by a polynomial factor. Our analysis will assume $p \le N^{0.9}$. The constant 0.9 is not mandatory, and can be replaced by any constant less than 1 by adjusting the constants in our analysis accordingly. We assume that every number (from either the input or intermediate computation) fits in a word. All our algorithms transmit a number always as a whole word, while we allow the transmission of individual bits in proving lower bounds. Regarding CPU calculation, our algorithms will use only comparisons, $+$, $-$, $\cdot$, and $/$. Finally, when we say that a randomized algorithm succeeds "with high probability" (w.h.p.), we mean that its failure probability is $O(1/N^2)$.

1.2 Definition of the Main Problem

We consider entity matching between two sets R, S of objects whose similarity is reflected by absolute coordinate differences in a multi-dimensional space (the dimensions are the extracted features based on which learning is performed). Formally, let R and S each be a set of points in $\mathbb{R}^d$. Denote the coordinate of a point $q \in \mathbb{R}^d$ on dimension $i \in [1, d]$ as $q[i]$. Each pair $(r, s) \in R \times S$ defines a d-dimensional point $q_{(r,s)}$ whose coordinate on dimension $i \in [1, d]$ is $q_{(r,s)}[i] = |r[i] - s[i]|$ that is, $q_{(r,s)}$ captures the similarity of r and s on the d dimensions. Let $Q_{R,S}$ be the (perhaps multi-) set of points thus obtained, namely: $Q_{(R,S)} = \{q_{(r,s)} \mid (r, s) \in R \times S\}$. Each $q_{(r,s)} \in Q_{(R,S)}$ carries a *label* of either 1 or 0, indicating whether r and s are a match. Define:

$$T_{(R,S)} \quad = \quad \{(r, s) \in R \times S \mid \text{label of } q_{(r,s)} = 1\},$$

that is, $T_{(R,S)}$ contains all and only the matches between R and S. We are now ready to clarify the input and output of the *entity matching with linear classification* (EMLC) problem:

- Input: R, S, and $T_{(R,S)}$.
- Output: A linear plane π such that the label-1 points of $Q_{(R,S)}$ fall on one side of π, while the label-0 points on the other. If such a separation plane does not exist, the output is NULL.

1.3 Our Results: The Main and Accompanying Problems

The main result of the paper is:

▶ **Theorem 1.** *Set $N = |R| + |S| + |T_{(R,S)}|$ for the EMLC problem in $\mathbb{R}^d$ with $d = O(\text{polylog}\, N)$. There is a $d^{O(1)}$-round MPC algorithm that solves the problem w.h.p. with load $d^{O(d)} \cdot N\sqrt{\log N}/p$.*

The result is most appealing when the dimensionality d is a fixed constant, in which case our algorithm performs constant rounds with load $O(N\sqrt{\log N}/p)$. The theorem confirms the superiority of accepting R, S, and $T_{(R,S)}$ as the input to EMLC (rather than $Q_{(R,S)}$) when $|T_{(R,S)}| \ll |R||S|$. Indeed, even just "uploading" the labeled $Q_{(R,S)}$ onto the p machines will entail a network cost of $\Omega(|R||S|/p)$ on at least one machine[1]. The algorithm in Theorem 1 combines new MPC solutions to several geometric problems, as explained below.

Linear Programming (LP). Define a *half-space* h in $\mathbb{R}^d$ as the set $\{q \in \mathbb{R}^d \mid \sum_{i=1}^{d} \alpha_i \cdot q[i] \geq \beta\}$, where real numbers $\alpha_1, \alpha_2, ..., \alpha_d$ and β are the *coefficients* of h. The input to LP is a set H of N half-spaces. Let $I(H)$ be the intersection of all the half-spaces in H. The objective is to (i) determine whether $I(H)$ is empty, and (ii) if not, report a point $q \in I(H)$ with the smallest $q[1]$ (i.e., minimize the coordinate on dimension 1).

[1] Remember that $Q_{(R,S)}$ is a set of *labeled* points; and the labeling depends on the underlying application, and does not need to follow any geometric patterns. Therefore, $Q_{(R,S)}$ cannot be computed from R and S.

We are not aware of existing results in the MPC model, but the problem has been well studied in PRAM (see [11, 17] for a collection of results). The focus of those algorithms, however, is to minimize "work"[2], rather than communication. In particular, for constant d, when adapted to finish within constant rounds under MPC, they all have a load of $\Omega(\frac{N/p}{\text{polylog}\, N})$. We will prove:

▶ **Theorem 2.** *Suppose that the dimensionality d is $O(\text{polylog}\, N)$. Fix any constants ϵ, δ satisfying $\delta > \epsilon > 0$ and $p \leq N^{\delta-\epsilon}$. There is a $d^{O(1)}$-round MPC algorithm which solves the LP problem w.h.p. with load $O(N^\delta/p)$. Furthermore, this is true even if the half-spaces are distributed on the p machines arbitrarily at the beginning (i.e., perhaps unevenly).*

The theorem may appear a bit unusual such that it is worth offering a guided tour, assuming $d = O(1)$ for simplicity:

- First, look at the realistic scenario where $p = O(\text{polylog}\, N)$. In this case, the theorem essentially says that LP can be settled in constant rounds with load $O(N^\delta/p)$, for arbitrarily small constant $\delta > 0$ (simply set $\epsilon = \delta/2$).
- Consider now the less realistic scenario where $p = \Theta(N^c)$ for some constant $c > 0$. The theorem tells us that the load is $O(N^{c+\epsilon}/p) = O(N^\epsilon)$ for arbitrarily small constant $\epsilon > 0$ (set $\delta = c + \epsilon$).
- In all circumstances, Theorem 2 asserts that LP can be settled with a load *polynomially* less than N/p. Recall that $\Omega(\frac{N/p}{\log N})$ is a load lower bound for numerous problems under the MPC model, e.g., sorting [16], set intersection [27], equi-joins [19], etc. It is thus interesting to see that a problem as sophisticated as LP requires far less communication.
- The theorem does not require a balanced distribution of the input on the p machines. This is a nice property when the input is produced by a preceding operator (e.g., a join) which may leave a huge number—in the worst case $\Omega(N)$—of half-spaces on one machine.

Batched Range Counting (BRC). In $\mathbb{R}^d$, define R as a set of axis-parallel rectangles, and S as a set of points. The objective of BRC is to report, for every rectangle $r \in R$, the number $|r \cap S|$ (i.e., how many points of S are covered in r).

Set $N = |R| + |S|$. For constant d, Hu et al. [19] recently presented a constant-round algorithm with load $O((N/p) \log^{d-1} p)$. We improve their result to optimality:

▶ **Theorem 3.** *There is an $O(d)$-round deterministic MPC algorithm that solves the BRC problem with load $d^{O(d)} \cdot N/p$. Furthermore, any constant-round deterministic algorithm must entail a load of $\Omega(N/p)$ when $d = 2$.*

Dominance Join (DJ). In $\mathbb{R}^d$, let R and S be two sets of points. The objective of DJ is to report all pairs $(r, s) \in R \times S$ such that s dominates r (namely, $s[i] \geq r[i]$ for all $i \in [1, d]$). An algorithm is *α-out-balanced* if each machine produces at most $\alpha \cdot \text{OUT}/p$ result pairs, where OUT is the total number of result pairs.

Let $N = |R| + |S|$. For constant d, Hu et al. [19] proposed a constant-round algorithm with load $O((N/p) \log^{d-1} p + \sqrt{\text{OUT}/p})$ (there was no discussion on α-out-balance in [19]). We prove:

[2] The *work* of a PRAM algorithm is the product of (i) the number of steps performed and (ii) the number of processors deployed.

▶ **Theorem 4.** *For the DJ problem, there is a $d^{O(d)}$-out-balanced deterministic MPC algorithm that performs $O(d^2)$ rounds, and incurs load $d^{O(d)} \cdot (N + OUT)/p$.*

For constant d, our algorithm does not always improve the load of [19], but it does whenever $\text{OUT} = o(N \log^{d-1} p)$. This is especially useful when OUT is small and has no dependence on d, as is indeed what we rely on to prove a "dimensionality-oblivious" load of $O(N\sqrt{\log N}/p)$ for EMLC under any constant d (see Theorem 1). Adopting the algorithm of [19] instead will introduce a multiplicative factor of $\log^{O(d)} p$.

2 Preliminaries

ϵ-Nets and Its Application to LP. We will review some useful results on LP. Let H be a set of d-dimensional planes. An *ϵ-net* of H is a subset $X \subseteq H$ with the following property: any point $q \in \mathbb{R}^d$ that appears in at least $\epsilon|H|$ half-spaces of H, must be in at least one half-space of X.

▶ **Lemma 5** (Theorem 5.7.3 of [26]). *Consider an integer $r \in [4d, N]$ and a value $\Delta \in (0, 1/2]$ satisfying $r \leq 1/\Delta$. Let X be a set of $O(d \cdot r \log(1/\Delta))$ samples from H taken uniformly at random with replacement. Then, X is a $(1/r)$-net of H with probability at least $1 - O(\Delta)$.*

The next corollary follows directly from the above lemma and the definition of ϵ-net (by considering the complement of each half-space in H and X):

▶ **Corollary 6.** *The following property holds with probability at least $1 - O(\Delta)$ on the sample set X in Lemma 5: any point $q \in \mathbb{R}^d$ inside $I(X)$ (i.e., the intersection of the half-spaces in X). can be outside less than N/r half-spaces in H.*

The next result we review concerns an elegant framework for solving the LP problem. At the beginning, take an arbitrary subset X_0 of H. In general, given a subset X_i of H (for $i \geq 0$), we perform an *iteration* as follows. First, solve the LP problem on X_i. If $I(X_i)$ is empty, stop the algorithm and return empty. Otherwise, let ϕ_i be the solution point found. If $i = d + 1$, return ϕ_{d+1} (for the original LP problem). If $i < d + 1$, collect the set Y_i of half-spaces in H that do *not* contain ϕ_i. Launch the next iteration with $X_{i+1} = X_i \cup Y_i$. The lemma below states an interesting fact:

▶ **Lemma 7** ([8]). *Regardless of X_0, the above algorithm always solves the problem correctly.*

The previous two lemmas constitute the core idea behind a number of LP algorithms (e.g., [8, 6, 17]). The version below is based an original proposition from [8], but includes optimization from [6]. Simply choose X_0 to be a random sample set (with replacement) of size $O(d \cdot \sqrt{N \log N})$. Applying Lemma 5 with $r = \Theta(\sqrt{N/\log N})$ and $\Delta = 1/N^3$, we know that X_0 has the property stated in Corollary 6 with probability at least $1 - 1/N^3$. The fact $X_0 \subseteq X_i$ ($i \geq 1$) implies that every X_i also has this propertiy w.h.p. Hence, w.h.p., $|Y_i| = O(N/r) = O(\sqrt{N \log N})$ holds for every $i \in [1, d]$, which in turn tells us that $|X_i| \leq |X_0| + i \cdot N/r = O(d \cdot \sqrt{N \log N})$ for all $i \in [1, d+1]$.

The above description essentially reduces an LP problem of size N to $d + 1$ smaller problems each with size $O(d \cdot \sqrt{N \log N})$. Standard analysis shows that, when d is a constant, the running time is $O(N)$ w.h.p. in the RAM model.

Broadcasting in MPC. Suppose that the machines carry ids $1, 2, ..., p$. In designing an algorithm, we often need to send information from one or more machines to several machines in a *continuous* id range. The lemma below explains the cost when there is one source machine:

▶ **Lemma 8** ([18]). *If a machine needs to broadcast x words to y machines in a continuous id range, this can be done in constant rounds with load*

– $O(x)$ *if* $x \geq y^c$ *for an arbitrarily small constant* $c > 0$*;*
– $O(y^c)$ *otherwise, where* $c > 0$ *is an arbitrarily small constant.*

This implies the next result for the case when there are multiple source machines:

▶ **Corollary 9.** *Suppose that z machines are holding in total x words of messages. In constant rounds, these x words can be broadcast to y machines in a continuous id range with load $O(x + y^c)$, where $c > 0$ is an arbitrarily small constant (this holds regardless of the value of z).*

Proof. In one round, instruct all the z machines to send their messages to one machine, which requires load $O(x)$. Then, the recipient machine broadcasts all the words to the y machines. The previous lemma indicates that this can be done with constant rounds and load $O(x + y^c)$. ◀

3 Linear Programming in MPC

We will prove Theorem 2 in this section. As before, let H be the input set of N half-spaces in $\mathbb{R}^d$, and $I(H)$ be their intersection. The goal is to find a point $q \in I(H)$ with the smallest coordinate on dimension 1. We will give a $d^{O(1)}$-round MPC algorithm with load $O(N^\epsilon)$ which is $O(N^\delta/p)$, for the selection of δ and ϵ as in Theorem 2. Our algorithm for LP can be thought of as an efficient implementation of the framework described in Section 2. The main idea is to *do away with* the default requirement of MPC that, the input should be distributed onto the machines evenly. Indeed, the requirement does not even allow us to distribute $X_1 = X_0 \cup Y_0$ (as defined in Section 2) for recursion, because this would lead to a load of $\Omega(d\sqrt{N} \log N/p)$. We circumvent this obstacle by sampling in an "in-place" manner (keeping the samples on the original machines), and performing the recursion anyway in spite of an unbalanced input distribution.

3.1 In-Place Sampling

Suppose that there is a set X of elements that are distributed on p machines in an arbitrary manner (not necessarily balanced). We want to take t samples from X uniformly at random with replacement. If an element $e \in X$ is sampled t_e times, we indicate the fact by storing the count t_e on the machine where e is stored. No elements need to be moved across the machines.

The above purpose can be fulfilled in $O(c)$ rounds with load $O(p^{1/c})$, for any constant $c \geq 1$. This holds regardless of $|X|$ and t. We can organize the p machines arbitrarily into a tree where each internal node, except possibly the root, has $\Theta(p^{1/c})$ child nodes. The height of the tree is $O(c)$. The *level* of a node is the number of edges on its path to the root (the root is at level 0).

Let $X(u)$ be the set of elements stored on node (i.e., a machine) u. In the first step, each node (i.e., a machine) u obtains the number—denoted as x_u—of elements of X that are stored in the subtree rooted at u. This is trivial if u is a leaf because x_u is simply $|X(u)|$. Inductively, after a node v at level $i \geq 1$ has acquired x_v, it sends x_v to its parent u (at level $i - 1$). Node u then sums up the numbers from its child nodes, together with $|X_u|$, to obtain x_u. This bottom-up process takes $O(c)$ rounds to finish, and has a load equal to the internal fanout $O(p^{1/c})$.

In the second step, each node u decides the number t_u of samples to be taken from X_u. This can be done in a top-down manner. Suppose, inductively, that a node u at level $i \geq 0$ has been informed that y_u samples should be from the subtree rooted at u (for the base case of induction, u is the root, and $y_u = t$). Then, u generates y_u independent integers uniformly at random from $[1, x_u]$. Remember that, from the previous step, u has already obtained x_v for every child v, and that $|X_u| + \sum_v x_v = x_u$. Hence, from the y_u random integers, u can generate t_u, as well as y_v for each child v. This step also has $O(c)$ rounds and load $O(p^{1/c})$.

Now that every machine u knows t_u, it performs the sampling locally at X_u, which completes the in-place sampling.

3.2 The LP Algorithm

The core of our algorithm can be summarized as:

▶ **Lemma 10.** *Fix a sufficiently large integer N, an integer $d = O(\text{polylog}\, N)$, and a constant satisfying $0 < \epsilon < 1$. Consider that we are given an LP problem of size $n \in [N^\epsilon, N]$.*

Available to us is an algorithm $\mathcal{A}_1$ that is capable of solving any LP problem of a smaller size $m = O(d\sqrt{n}\log N)$ with probability at least $1 - \Delta_{\mathcal{A}_1}$, even if the input is distributed on the p machines arbitrarily. Suppose that $\mathcal{A}_1$ performs $\rho(m)$ rounds, and incurs load $\lambda(m)$.

Then, for the LP problem on size n, there is an $O(d) + (d+1)\cdot\rho(m)$-round algorithm $\mathcal{A}_2$ that, with probability at least $1 - (1/N^3 + O(d\cdot\Delta_{\mathcal{A}_1}))$, solves the problem with load $\max\{O(d + p^\epsilon), \lambda(m)\}$, regardless of how the input is distributed on the machines initially.

Before proving the lemma, let us first explain how it leads to Theorem 2. Here, set N to the input size of the original LP problem, and d to the problem's dimensionality, and take the value of ϵ from Theorem 2. Given an LP problem with input size $n \leq N$, we do the following:

– If $n \leq N^\epsilon/d$, simply send all the half-spaces to one machine, and solve the problem there (with probability 1). Requiring apparently only $\rho(n) = 1$ round and load $\lambda(n) = O(N^\epsilon)$, this serves as the base algorithm $\mathcal{A}_1$ for the recursion Lemma 10 implies.
– Otherwise, apply the algorithm $\mathcal{A}_2$ in Lemma 10, which performs $\rho(n) = O(d) + (d+1)\cdot\rho(m)$ rounds, and requires load $\lambda(n) = \max\{O(d+p^\epsilon), \lambda(m)\}$. Effectively, the algorithm solves $d+1$ smaller problems of size at most m.

To settle the original LP problem, simply run the above algorithm on H. For performance analysis, we need to solve the above recurrences in order to obtain $\rho(N)$ and $\lambda(N)$.

Notice that $m = O(d\sqrt{n}\log N) \leq n^{0.51}$ for N larger than a certain constant, applying $d = O(\text{polylog}\, N)$ and $n \geq N^\epsilon$. We compute $\rho(n)$ by recursively computing $d+1$ copies of $\rho(m)$ until $n \leq N^\epsilon/d$. The recursion tree triggered by $\rho(N)$ has a depth of $O(\log(1/\epsilon))$, and contains $(d+1)^{O(\log(1/\epsilon))} = d^{O(1)}$ nodes. As a result, $\rho(N) = O(d)\cdot d^{O(1)} = d^{O(1)}$. The analysis of $\lambda(N)$ is much simpler: by (i) $d = \text{polylog}(N)$, and (ii) $\lambda(n) = O(N^\epsilon)$ in the base case ($n \leq N^\epsilon$), we know that $\max\{O(d+p^\epsilon), \lambda(m)\}$ is always $\lambda(m)$. Therefore, we have $\lambda(N) = O(N^\epsilon)$.

Our algorithm *fails* if, in any application of Lemma 10, $\mathcal{A}_2$ fails. The above analysis indicates that Lemma 10 is applied $d^{O(1)}$ times in total. Hence, we guarantee that the algorithm fails with probability $d^{O(1)}\cdot O(d/N^3) = o(1/N^2)$, as desired in Theorem 2.

Proof of Lemma 10. It suffices to combine in-place sampling with the ideas reviewed in Section 2. No half-spaces of H need to be moved from one machine to another. For each machine u and any subset $X \subseteq H$, we denote by $X(u)$ the set of half-spaces in X that are stored at u.

To solve the given LP problem of size n, first perform in-place sampling to obtain a sample set X_0 of size $O(d\sqrt{n \log N})$. It follows from Section 3.1 that the sampling takes $O(1/\epsilon) = O(1)$ rounds and incurs load $O(p^\epsilon)$. Applying Lemma 5 with $\Delta = 1/N^3$ and $r = \sqrt{n/\log(1/\Delta)}$, we know that X_0 has the property in Corollary 6 with probability at least $1 - 1/N^3$.

In general, given a subset X_i of H ($i \geq 0$) which is not necessarily evenly distributed on the machines, an *iteration* invokes algorithm $\mathcal{A}_1$ to solve the LP problem on X_i with probability at least $1 - \Delta_{\mathcal{A}_1}$. If $I(X_i)$ is empty, stop the algorithm and return empty. Otherwise, let ϕ_i be the solution point found. If $i = d+1$, return ϕ_i. Now consider $i < d+1$, the machine holding ϕ_i broadcasts the point to all other machines in constant rounds with load $O(d + p^\epsilon)$ (Lemma 8). Each machine u identifies the set $Y_i(u)$ of half-spaces that are in $H(u)$ and do not contain ϕ_i. This, conceptually, defines $Y_i = \bigcup_u Y_i(u)$ and $X_{i+1} = X_i \cup Y_i$. The next iteration is then launched with X_{i+1}.

When X_0 has the property in Corollary 6, by the reasoning mentioned in Section 2, we know that $X_i \leq |X_0| + i \cdot n/r = O(d\sqrt{n \log N})$ for all $i \in [1, d+1]$. Therefore, each iteration requires $O(1) + \rho(m)$ rounds and incurs load $\max\{O(d + p^\epsilon), \lambda(m)\}$. The number of rounds of all iterations is $O(d) + (d+1) \cdot \rho(m)$. The algorithm correctly solves the LP problem with probability at least $1 - (1/N^3 + (d+1)\Delta_{\mathcal{A}_1})$. This completes the proof of Lemma 10.

4 Batched Range Counting

This section will be mainly concerned with:

> **Batched Dominance Counting (BDC).** In $\mathbb{R}^d$, let R and S be two sets of points. The objective is to report, for each point $r \in R$, the number of points $s \in S$ such that r dominates s (namely, $r[i] \geq s[i]$ for all $i \in [1, d]$).

Set $N = |R| + |S|$. We will give an MPC algorithm that solves the above problem in $O(d)$ rounds with load $d^{O(d)} \cdot N/p$. Our technique is bootstrapping in nature. First, design a base algorithm that performs constant rounds and has load $O(p^d + d^2 \cdot N/p)$. This load is acceptable only when p is small, but otherwise can be rather expensive (e.g., for $p = N^{0.9}$). To fix this, we recursively partition the problem, until the number of machines in each subproblem is small enough to apply the base algorithm with an acceptable load. The details will be presented in Sections 4.1 and 4.2. Load $O(N/p)$ is asymptotically optimal for constant d, as shown in Section 4.3 via a reduction from *sparse set disjointness*. These BDC results will then imply Theorem 3, as explained in Section 4.4.

4.1 A Base Algorithm with Load $O(p^d + d^2 \cdot N/p)$

For each dimension $i \in [1, d]$, divide the space $\mathbb{R}^d$ into p *slabs* using $p-1$ planes perpendicular to the dimension, such that each slab has $\Theta(N/p)$ points of the input $R \cup S$. These slabs are said to be *on dimension i*. The slab boundaries of all dimensions together define a grid of p^d cells. Count the number of points of S in each cell. We want to give every machine a copy of all the p^d counts (i.e., one per cell). It is fundamental to do so in constant rounds with load $O(p^d + dN/p)$.[3] A point $s \in S$ dominated by a point $r = (r[1], ..., r[d]) \in R$ must appear

[3] The slab boundaries on each dimension can be decided by sorting, which takes constant rounds and load $O(N/p)$ [16]. Carrying this out for all dimensions simultaneously increases the load by a factor of d, without changing the number of rounds. By Corollary 9, the $d \cdot p^d$ boundaries can be broadcast to all the p' machines in constant rounds with load $O(d \cdot p^d)$. Every machine can now locally calculate the

- (**Cat. 1**) Either in a cell completely covered by $(-\infty, r[1]] \times (-\infty, r[2]] \times \ldots \times (-\infty, r[d]]$;
- (**Cat. 2**) Or in the same slab as r on at least one dimension.

The number of points in the first category can be calculated from the p^d counts already available on the machine where r is stored. It remains to count the pairs (r, s) in the second category.

Since a slab has $O(N/p)$ points, we can send all those points to one machine, which then locally performs the counting of Category 2 for every $r \in R$ in the slab. Specifically, for each $i \in [1, d]$, we send the points of each slab on dimension i to a distinct machine, which can be done in a single round with load $O(dN/p)$. Doing so for all dimensions simultaneously still in one round increases the load to $O(d^2 \cdot N/p)$. A slight complication is that, if r and s are in the same slab on more than one dimension, s may be counted multiple times for r. This can be easily avoided by introducing a consistent de-duplication policy: e.g., count s for r in a slab on dimension i only if they are not in the same slab on any dimension $j < i$. We thus have obtained a constant-round algorithm of load $O(p^d + d^2 \cdot N/p)$.

4.2 An Algorithm with Load $d^{O(d)} \cdot N/p$

Next, we describe an alternative algorithm for BDC. More generally, given an arbitrary $f \in (0, 1]$, we discuss how to deploy $p' = p^f$ machines to tackle a BDC problem of input size $N' = O(p' \cdot N/p)$; this will be referred to as an *f-BDC problem*. The original BDC problem corresponds to $f = 1$. Our algorithm will perform $\rho(f)$ rounds, and incur load $\lambda(f)$, where $\rho(f)$ and $\lambda(f)$ are functions to be resolved in the analysis. Notice that $N'/p' = O(N/p)$.

Base Case $f \le 0.1/d$. Simply invoke the base algorithm of Section 4.1. It finishes in $\rho(f) = O(1)$ rounds, and incurs load $\lambda(f) = O(p'^d + d^2 \cdot N'/p') = O(p^{df} + d^2 \cdot N/p) = O(p^{0.1} + d^2 \cdot N/p) = O(d^2 \cdot N/p)$, where the last step used $p \le N^{0.9}$.

Inductive Case $f > 0.1/d$. Along each dimension, divide the space into $p^{0.1/d}$ slabs each with $\Theta(N'/p^{0.1/d})$ points of $R \cup S$. As in Section 4.1, the slabs of all dimensions together define a grid, which here has $p^{0.1}$ cells. Count the number of points of S in each cell, and give every machine a copy of the $p^{0.1}$ counts (one count per cell). This can be done in constant rounds with load $O(p^{0.1} + dN'/p') = O(dN/p)$.

We now proceed in the same way as in the base algorithm except that Category 2 is now handled by solving $(f - 0.1/d)$-BDC problems recursively. Focusing on an arbitrary dimension, let us assign $\Theta(p'/p^{0.1/d})$ machines to each slab on the dimension. Since a slab has $\Theta(N'/p^{0.1/d})$ points, counting the number of pairs (r, s) of Category 2 within the slab is an $(f - 0.1/d)$-BDC problem. The $(f - 0.1/d)$-BDC problems of all slabs on the dimension can be solved in parallel using $\alpha(f - 0.1/d)$ rounds and load $\lambda(f - 0.1/d)$. Doing so for all d dimensions together increases the load by a factor of d, without changing the number of rounds. This indicates the following recurrences:

$$\begin{aligned} \rho(f) &= O(1) + \rho(f - 0.1/d) \\ \lambda(f) &= d \cdot \lambda(f - 0.1/d) + O(d \cdot N/p) \end{aligned}$$

boundary faces of all cells. We then count the number of points of S in each cell using the sum-by-key algorithm of [19] (for each point in S, its "key" is the id of the cell containing it; this algorithm counts, for every key k, the number of points having k as the key), using constant rounds with load $O(N/p)$. Broadcasting the p^d counters to all the machines can be done in constant rounds and load $O(p^d)$ by Corollary 9.

Resolving them gives $\rho(1) = O(d)$, and $\lambda(1) = d^{O(d)} \cdot N/p$. We thus have obtained an $O(d)$-round BDC algorithm with load $d^{O(d)} \cdot N/p$.

4.3 Lower Bound

We can prove that any constant-round deterministic BDC algorithm must incur a load of $\Omega(N/p)$ when $d = 2$. For this purpose, let us revisit the *sparse set disjointness* problem in communication complexity. Alice and Bob each take a size-k subset of $\{1, 2, ..., U\}$; denote their subsets as X_A and X_B, respectively. They want to determine whether $X_A \cap X_B = \emptyset$ by communicating the least number of bits. It is known that any deterministic algorithm requires sending $\Omega(k \log(2U/k))$ bits between the two parties [27].

Fix $U = k^2$ and set the word length to $\log_2 U$ bits. Suppose that we are given a constant-round MPC algorithm $\mathcal{A}$ that solves the BDC problem with load $o(N/p)$ (measured in words). We can use $\mathcal{A}$ for sparse set disjointness as follows. Alice constructs a 2D point set $R = \{(x, U - x) \mid x \in X_A\}$, and likewise, Bob constructs $S = \{(x, U - x) \mid x \in X_B\}$. It is easy to verify that the BDC problem returns a non-zero count for at least one point in R, if and only if $X_A \cap X_B \neq \emptyset$. We ask Alice and Bob to simulate $\mathcal{A}$, by asking each of them to be in charge of $p/2$ machines. At the beginning, Alice (or Bob) distributes her (or his, resp.) elements on her (or his, resp.) $p/2$ "machines" evenly. They then do the communication as instructed by $\mathcal{A}$. Since each round of $\mathcal{A}$ has load $o(k/p)$, every "machine" from Alice can send $o(k/p)$ words over to Bob, and vice versa. Hence, a round necessitates only $o(k)$ words—namely $o(k \log U)$—bits of communication. As $\mathcal{A}$ finishes in constant rounds, we have obtained an algorithm solving sparse set disjointness with $o(k \log U)$ bits, contradicting the $\Omega(k \log(2U/k)) = \Omega(k \log U)$ lower bound.

4.4 Proving Theorem 3 and Beyond

Batched range counting (BRC) can be reduced to BDC elegantly [12]. For each rectangle $r \in R$, the number $|r \cap S|$ can be aggregated from the "dominance counts" of the 2^d corners of r, where the *dominance count* of a corner q equals how many points in S are dominated by q. Hence, we can convert BRC into 2^d instances of BDC of the same size, solve all of them simultaneously, and then aggregate the 2^d dominance counts for each $r \in R$. This gives an algorithm that settles the BRC problem in $O(d)$ rounds with load $2^d \cdot d^{O(d)} \cdot N/p = d^{O(d)} \cdot N/p$. Finally, the lower bound on BDC clearly applies to BRC. This completes the proof of Theorem 3.

A Semi-Group Remark. Our solution actually settles the more general problem of batched range *sum*. Specifically, let R and S be defined as in BRC. However, each point $s \in S$ is now associated with a *weight* from a semi-group domain. The goal now is to report, for every $r \in R$, the sum of the weights of all the points in $r \cap S$. The proposed algorithm solves this problem in $O(d)$ rounds with load $d^{O(d)} \cdot N/p$, which is optimal for constant d.

5 Dominance Join

This section will establish Theorem 4 on the DJ problem. Recall that we have two sets of points in $\mathbb{R}^d$: R and S. The objective is to find all $(r, s) \in R \times S$ such that s dominates r. Set $N = |R| + |S|$, and denote by OUT the number of pairs in the result. We will solve the problem with an $d^{O(d)}$-out-balanced algorithm that runs in $O(d^2)$ rounds with load $d^{O(d)} \cdot (N + \mathrm{OUT})/p$.

Our solution benefits from the techniques of Section 4 in two ways. First, it deploys BDC (batched dominance counting) as an intermediate operation. Second, following the bootstrapping approach of Section 4, we will first develop a base algorithm for DJ that runs in constant rounds but with load $O(p^d + d^2 \cdot N/p + d \cdot \mathrm{OUT}/p)$, and then attain the target performance guarantees with recursion. Attention, however, should be paid to how we avoid a $\log^{O(d)} p$ term in the load. For this purpose, we abandon the range-tree-styled "canonical decomposition" approach in [19], which is inherently penalized by a multiplicative $O(\log p)$ factor per dimension. Instead, we will show how to use the grid partitioning approach in Section 4 for reporting, by integrating it with new cell-pairing and machine-assignment ideas.

To facilitate discussion, in the following presentation, we will (conceptually) color the points of R in *red*, and those of S in *blue*.

5.1 A Base Algorithm with Load $O(p^d + d^2 \cdot N/p + d \cdot \mathrm{OUT}/p)$

Divide the space into p slabs on each dimension, such that each slab has $\Theta(N/p)$ points of $R \cup S$. Number these slabs as $1, 2, ..., p$ in ascending order (call those numbers *slab ids*). The slab boundaries of all d dimensions define a grid of p^d cells. For each cell in the grid, count the number of red points and number of blue points. This gives a total of $2p^d$ counts. Broadcast a copy of all these counts to the p machines. As in Section 4.1, all the above can be done in constant rounds with load $O(p^d + dN/p)$. We say that a cell c_1 *dominates* another c_2 if the slab id of c_1 is larger than or equal to that of c_2 on every dimension.

Run the algorithm of Section 4.1 to obtain the value of OUT, which requires constant rounds and load $O(p^d + dN/p)$. Recall from Section 4.1 that the set of result pairs (r, s) can be classified into two categories. We slightly rephrase the description of those categories below:

- **(Cat. 1)** The cell containing r dominates the cell containing s;
- **(Cat. 2)** r and s are in the same slab on at least one dimension.

Next, we will explain how to produce the result pairs of each category, in such a way that $O(d \cdot \mathrm{OUT}/p)$ pairs are produced on each machine.

Reporting Category 1. Let $k = O(p^{2d})$ be the number of *dominance cell pairs* (c, c') satisfying (i) $c \neq c'$, and (ii) c dominates c'. Let us order all such pairs lexicographically: $(c_1, c'_1), (c_2, c'_2), ..., (c_k, c'_k)$. For each $i \in [1, k]$, denote by $F(i)$ the set of red points in c_i, and by $G(i)$ the set of blue points in c'_i. We say that (c_i, c'_i) is *heavy* if $|F(i)| \cdot |G(i)| \geq \mathrm{OUT}/p$, or *light* if $0 < |F(i)| \cdot |G(i)| < \mathrm{OUT}/p$. Note that, importantly, a light pair must produce at least one result pair. Next, we will report (r, s) contributed by heavy and light pairs separately.

Dealing with Heavy Pairs. There are at most p heavy dominance cell pairs; denote them as: (c_{i_1}, c'_{i_1}), (c_{i_2}, c'_{i_2}), ..., $(c_{i_{k_1}}, c'_{i_{k_1}})$ where $k_1 \leq p$. For each $j \in [1, k_1]$, assign $p_j = \Theta(\frac{|F(i_j)||G(i_j)|}{\mathrm{OUT}} p)$ machines with continuous ids, making sure $\sum_j p_j = p$. Ask these p_j machines to collect (from other machines) the points of $F(i_j)$ and $G(i_j)$ evenly, so that each machines gets $O(1 + \frac{|F(i_j)|+|G(i_j)|}{p_j})$ points. Doing so for all $j \in [1, k_1]$ in parallel takes constant rounds and entails load

$$O\left(dN/p + d \cdot \max_{j=1}^{k_1} \frac{|F(i_j)| + |G(i_j)|}{p_j}\right) = O\left(dN/p + d \cdot \max_{j=1}^{k_1} \frac{|F(i_j)| + |G(i_j)|}{|F(i_j)||G(i_j)|} \frac{\mathrm{OUT}}{p}\right)$$
$$= O(d(N + \mathrm{OUT})/p)$$

where the first term $O(dN/p)$ is because every machine may send out all of the $O(dN/p)$ points in its local storage. Then, for each $j \in [1, k_1]$, compute $F(i_j) \times G(i_j)$ (i.e., the cartesian product) on the p_j machines assigned. The algorithm of [3] can do this in constant rounds with load $O(d\frac{|F(i_j)|+|G(i_j)|}{p_j} + d\sqrt{\frac{|F(i_j)||G(i_j)|}{p_j}}) = O(d(N + \mathrm{OUT})/p)$. Their algorithm is $O(1)$-out-balanced, with each machine producing $O(\frac{|F(i_j)||G(i_j)|}{p_j}) = O(\mathrm{OUT}/p)$ result pairs.

Dealing with Light Pairs. Let us redefine $(c_{i_1}, c'_{i_1}), (c_{i_2}, c'_{i_2}), ..., (c_{i_{k_2}}, c'_{i_{k_2}})$ as the light pairs in lexicographic order where $k_2 = O(p^{2d})$ is the number of such pairs. Partition the range $[1, k_2]$ into p intervals $I_1, I_2, ..., I_p$ such that $O(\mathrm{OUT}/p)$ result pairs are produced from every I_i, namely: for any $j \in [1,p]$, it holds that $\sum_{x\in I_j} |F(i_x)||G(i_x)| = O(\mathrm{OUT}/p)$. Such a partitioning definitely exists because, by definition of light pair, $|F(i_x)||G(i_x)| < \mathrm{OUT}/p$ for all $x \in [1, k_2]$. Assign one machine to every I_j ($j \in [1,p]$). This machine collects the $F(i_x)$ and $G(i_x)$ for all $x \in I_j$, and then locally produces all the results pairs from them. This requires constant rounds and load

$$\begin{aligned} O\Big(dN/p + d\cdot \max_{j=1}^{p} \sum_{x\in I_j} |F(i_x)| + |G(i_x)|\Big) &= O\Big(dN/p + d\cdot \max_{j=1}^{p} \sum_{x\in I_j} |F(i_x)||G(i_x)|\Big) \\ &= O(d(N + \mathrm{OUT})/p). \end{aligned}$$

Reporting Category 2. Let $\sigma_1, \sigma_2, ..., \sigma_p$ be the slabs of a certain dimension. The objective of *processing* this dimension is to report all result pairs (r, s) such that r and s are both in an identical slab. We achieve the purpose with constant rounds and $O(dN/p)$ load as follows. For $i \in [1,p]$, let R_i and S_i be the set of red and blue points in σ_i, respectively. To each σ_i, use a machine to obtain (in parallel) locally the number OUT_i of result pairs that are produced by R_i and S_i. Then, re-assign $p_i = \Theta(\lceil \frac{\mathrm{OUT}_i}{\mathrm{OUT}} p \rceil)$ machines to σ_i, making sure $\sum_i p_i = p$. For every σ_i, we carry out the following steps in parallel. First, broadcast R_i and S_i to all the p_i assigned machines, which by Corollary 9 can be done in constant rounds with load $O(d(|R_i|+|S_i|)+p_i) = O(dN/p+p)$. Each of the p_i machines then produces $O(\mathrm{OUT}/p)$ distinct result pairs from R_i and S_i. This distinctness requirement can be fulfilled by, e.g., resorting to the lexicographic order of the result pairs.

We process all the dimensions simultaneously using the above strategy. The number of rounds remains unchanged, but the load is increased d times to $O(dp + d^2 \cdot N/p) = O(p^d + d^2 \cdot N/p)$. Each machine now reports at most $O(d \cdot \mathrm{OUT}/p)$ result pairs. The algorithm is therefore $O(d)$-out-balanced. Duplicate reporting can be avoided by the same approach described in Section 4.1. Note that this approach can only decrease the number of result pairs reported on a machine, and therefore, will not affect the $O(d)$-out-balance.

Thus, we have obtained a constant-round $O(d)$-out-balanced algorithm that solves the DJ problem with a load bounded by $O(p^d + d^2 \cdot N/p + d \cdot \mathrm{OUT}/p)$.

5.2 An Algorithm with Load $d^{O(d)} \cdot (N + \mathrm{OUT})/p$

Next, we present an algorithm for the DJ problem that will establish Theorem 4. More generally, given an arbitrary $f \in (0, 1]$, we discuss how to deploy $p' = p^f$ machines to tackle a DJ problem of input size $N' = O(p' \cdot N/p)$ and output size $\mathrm{OUT}' = O(p' \cdot \frac{OUT}{p})$; this will be referred to as an *f-DJ problem.* The original DJ problem corresponds to $f = 1$. Our algorithm will be $\alpha(f)$-out-balanced, perform $\rho(f)$ rounds, and incur load $\lambda(f)$, where $\alpha(f), \rho(f)$, and $\lambda(f)$ are functions that are the target of analysis. Note that $N'/p' = O(N/p)$ and $\mathrm{OUT}'/p' = O(\mathrm{OUT}/p)$. We will enforce the invariant that the value of OUT' is already known (at $f = 1$, this can be ensured by running the BDC algorithm of Section 4).

Base Case $f \le 0.1/d$. Simply invoke the base algorithm. From the previous subsection, we know that the algorithm finishes in $\rho(f) = O(1)$ rounds, and incurs load $\lambda(f) =$

$O(d^2 \cdot N/p + d \cdot \text{OUT}/p)$. The algorithm is $O(d)$-out-balanced, i.e., $\alpha(f) = O(d)$.

Inductive Case $f > 0.1/d$. Impose a grid by dividing $\mathbb{R}^d$ along each dimension into $p^{0.1/d}$ slabs. For each cell, count the number of red points and number of blue points therein. Send a copy of these counts to all machines. All these can be accomplished in constant rounds with load $O(dN/p)$.

The result pairs can be divided into Categories 1 and 2 as explained in Section 5.1. We will report the two categories separately.

Category 1. This category can be produced in the same way as described in the base algorithm (replacing N, p, and OUT with N', p', and OUT$'$ respectively), which performs constant rounds and requires load $O(d(N + \text{OUT})/p)$.

Category 2. The procedure for this category also follows the framework illustrated in the base algorithm. Consider the *processing* of a dimension with slabs $\sigma_1, \sigma_2, ..., \sigma_{p^{0.1/d}}$ (recall that the objective is to find result pairs (r, s) such that r and s are both in some slab). For each $i \in [1, p^{0.1/d}]$, let R_i and S_i be the set of red and blue points in σ_i, respectively.

To compute the number OUT_i of result pairs produced by R_i and S_i, we apply the BDC algorithm of Section 4, by assigning $\Theta(p'/p^{0.1/d})$ machines to σ_i. Run an instance of algorithm in parallel for each slab. All instances finish in $O(d)$ rounds, and incur load $d^{O(d)} \cdot \frac{N'/p^{0.1/d}}{p'/p^{0.1/d}} = d^{O(d)} \cdot N/p$.

For each i, we will compute the result pairs from R_i and P_i with

$$p_i \quad = \quad \Theta\left(\frac{p'}{p^{0.1/d}} + \frac{\text{OUT}_i}{\text{OUT}'} \cdot p'\right)$$

machines, making sure that $\sum_i p_i = p'$. For this purpose, distribute R_i and P_i onto the p_i machines evenly. Doing so for all i simultaneously takes constant rounds and incurs load

$$O\left(\frac{dN'}{p'} + \max_{i=1}^{p^{0.1/d}} \frac{d \cdot (|R_i| + |P_i|)}{p_i}\right) = O\left(\frac{dN'}{p'} + \frac{dN'/p^{0.1/d}}{p'/p^{0.1/d}}\right) = O(dN/p).$$

Now, producing the result pairs from R_i and P_i becomes an $(f - 0.1/d)$-DJ problem. Solving it recursively in parallel for all i. This requires $\rho(f - 0.1/d)$ rounds with load $\lambda(f - 0.1/d)$. Each of the p_i machines produces at most $\alpha(f - 0.1/d) \cdot \text{OUT}_i/p_i = \alpha(f - 0.1/d) \cdot O(\text{OUT}/p)$ result pairs.

We carry out the above processing on all the dimensions simultaneously. This does not affect the number of rounds, but increases the load by d times. Each machine now produces at most $d \cdot \alpha(f - 0.1/d) \cdot O(\text{OUT}/p)$ result pairs.

Summarizing the discussion on Categories 1 and 2 gives the following recurrences:

$$\begin{aligned} \alpha(f) &= d \cdot \alpha(f - 0.1/d) \\ \rho(f) &= O(d) + \rho(f - 0.1/d) \\ \lambda(f) &= d \cdot \lambda(f - 0.1/d) + d^{O(d)} \cdot N/p + O(d \cdot \text{OUT}/p). \end{aligned}$$

To analyze the performance of our algorithm on the initial DJ problem, we need to solve the above recurrences to obtain $\rho(1), \lambda(1)$, and $\alpha(1)$. It is straightforward to verify that $\alpha(f) = d^{O(d)}$ and $\rho(1) = O(d^2)$. To compute $\lambda(f)$, recursively break it into d copies of $\lambda(f - 0.1/d)$ until $f \leq 0.1/d$. The recursion tree triggered by $\lambda(1)$ has a depth $O(d)$ with $d^{O(d)}$ nodes in total. Each leaf contributes $O(d^2 \cdot N/p + d \cdot \text{OUT}/p)$ to $\lambda(f)$, while each internal node contributes $d^{O(d)} N/p + O(d \cdot \text{OUT}/p)$. Hence:

$$\lambda(1) = d^{O(d)} \left(d^{O(d)} \cdot N/p + d \cdot \text{OUT}/p\right) = d^{O(d)} \cdot (N + \text{OUT})/p.$$

This establishes the claim in Theorem 4.

6 Entity Matching with Linear Classification

We are now ready to solve EMLC. Recall that the input includes two sets R, S of points in $\mathbb{R}^d$, and the set $T_{(R,S)}$ of matching pairs in $R \times S$. Every pair $(r, s) \in T_{(R,S)}$ defines a point $q_{(R,S)} \in Q_{(R,S)}$ with label 1, whereas every $(r, s) \in R \times S \setminus T_{(R,S)}$ defines a point $q_{(R,S)} \in Q_{(R,S)}$ with label 0. We want to find a plane π that separates the 1-label points from the 0-label points in $Q_{(R,S)}$, or assert that such a plane does not exist.

Section 6.1 will explain the core of our technique: a reduction from EMLC to linear programming and dominance join. We will first present the reduction without having any computation models in mind. Then, Section 6.2 will give a fast implementation in MPC to establish Theorem 1.

6.1 Reduction to LP and DJ

A plane π can be described as $\{q \in \mathbb{R}^d \mid \sum_{i=1}^{d} \alpha_i \cdot q[i] = \beta\}$ where α_i $(i \in [1, d])$ and β are the coefficients. It defines two half-spaces: $\pi^+ : \{q \in \mathbb{R}^d \mid \sum_{i=1}^{d} \alpha_i \cdot q[i] \geq \beta\}$, and $\pi^- : \{q \in \mathbb{R}^d \mid \sum_{i=1}^{d} \alpha_i \cdot q[i] < \beta\}$. If a separation plane π exists, then π^+ contains all and only either the label-1 points, or the label-0 points. Due to symmetry, it suffices to discuss the former situation.

The space $\mathbb{R}^d$ where the points of $Q_{(R,S)}$ distribute is the *primal* space. We instead look at the corresponding LP problem in the *dual* $(d + 1)$-dimensional space that consists of all the possible $(\alpha_1, \alpha_2, ..., \alpha_d, \beta)$. Every point $q \in Q_{(R,S)}$ defines a half-space $h_q = \{(\alpha_1, \alpha_2, ..., \alpha_d, \beta) \in \mathbb{R}^{d+1} \mid \sum_{i=1}^{d} \alpha_i \cdot q[i] \vee \beta\}$ where the operator $\vee$ is $\geq$ if q has label 1, or $<$ otherwise. This spawns a set H of $|R||S|$ half-spaces. The LP problem on H returns a solution point if and only if π exists for the original EMLC problem. To avoid the quadratic pitfall, however, we cannot afford to materialize H, which precludes directly invoking an LP algorithm on H. Next, we show that materialization is unnecessary.

Set $n = |R| + |S|$; it must hold that $|H| \leq n^2$. Take a random sample set Σ (with replacement) of H with size $|\Sigma| = \Theta(d \cdot n\sqrt{\log n})$. By Lemma 5, Σ is a $(1/r)$-net of H with probability $1 - 1/n^2$ where $r = \Theta(n\sqrt{\log n})$. Then, we carry out the LP algorithmic framework in Section 2, by setting $X_0 = \Sigma$. By reviewing the framework, one can see that it suffices to implement two key steps:

- **(LP Step)** In the i-th $(i \geq 0)$ iteration, solve the LP problem on X_i.
- **(DJ Step)** If a solution point ϕ_i is found in the LP step, find the set Y_i of half-spaces in H that do not contain ϕ_i.

Note that X_i, Y_i, and ϕ_i are as defined in Section 2. The details of each step are clarified below (after which it will become clear why the second step is named as such).

The LP Step. The analysis in Section 2 tells us that $|X_i| = O(|R||S|/r) = O(n^2/r) = O(n\sqrt{\log n})$ when Σ is a $(1/r)$-net. This means that w.h.p. we are able to afford to materialize X_i and apply an LP algorithm on it.

The DJ Step. Divide Y_i into:

- Y^1: the set of half-spaces $h_q \in Y_i$ that are defined by a point $q \in Q_{(R,S)}$ with label 1;
- Y^0: the set of half-spaces $h_q \in Y_i$ that are defined by a point $q \in Q_{(R,S)}$ with label 0.

As all the matching pairs have been given in $T_{(R,S)}$, we can easily produce Y^1 by scanning $T_{(R,S)}$ once: for each $(r, s) \in T_{(R,S)}$, include $h_{q_{(r,s)}}$ into Y^1 if it does not contain ϕ_i.

The computation of Y^0 is less trivial. Let us write out the coordinates of ϕ_i as $(\alpha_1^*, \alpha_2^*, ..., \alpha_d^*, \beta^*)$. The next lemma gives a crucial observation:

▶ **Lemma 11.** *Define $Z(\phi_i)$ as the set of all $(r, s) \in R \times S$ satisfying*

$$\sum_{i=1}^{d} \alpha_i^* \cdot |r[i] - s[i]| \quad \geq \quad \beta^*. \tag{1}$$

Then, $Y^0 = Z(\phi_i) \setminus T_{(R,S)}$.

Proof. Consider an arbitrary non-matching pair $(r, s) \in R \times S \setminus T_{(R,S)}$. Since $q_{(r,s)}$ has label 0, $h_{q_{(r,s)}} = \{(\alpha_1, \alpha_2, ..., \alpha_d, \beta) \in \mathbb{R}^{d+1} \mid \sum_{i=1}^{d} \alpha_i \cdot q_{(r,s)}[i] < \beta\}$. By definition of $q_{(r,s)}$, (1) can be written as $\sum_{i=1}^{d} \alpha_i^* \cdot q_{(r,s)}[i] \geq \beta^*$, which indicates that $h_{q_{(r,s)}}$ does not contain ϕ_i. Therefore, (r, s) satisfies (1) if and only if $q_{(r,s)} \in Y^0$. The correctness of the lemma then follows. ◀

Motivated by the above, we turn our attention to computing $Z(\phi_i)$, after which Y^0 can then be obtained with a set difference with $T_{(R,S)}$. It turns out that, as elaborated below, this can be achieved by solving 2^d instances of DJ, each of which is between two sets of $(d+1)$-dimensional points with sizes $|R|$ and $|S|$, respectively.

Fix an arbitrary point $r \in R$. To compute $T_{(R,S)}$, we want to find all $s \in S$ satisfying (1). There are 2^d different ways to remove the absolute signs in (1) because, on each dimension $i \in [1, d]$, we should independently distinguish $r[i] \geq s[i]$ and $r[i] < s[i]$. Due to symmetry, it suffices to consider

$$r[i] \geq s[i] \text{ for all } i \in [1, d] \tag{2}$$

in which case (1) can be written as:

$$\sum_{i=1}^{d} \alpha_i^* \cdot r[i] \geq \beta^* + \sum_{i=1}^{d} \alpha_i^* \cdot s[i]. \tag{3}$$

Motivated by this, we construct two sets R', S' of $(d+1)$-dimensional points:

$$\begin{aligned} R' &= \left\{ \left(r[1], r[2], ..., r[d], \sum_{i=1}^{d} \alpha_i^* \cdot r[i] \right) \middle| \, r \in R \right\} \\ S' &= \left\{ \left(s[1], s[2], ..., s[d], \beta^* + \sum_{i=1}^{d} \alpha_i^* \cdot s[i] \right) \middle| \, s \in S \right\} \end{aligned}$$

The DJ between the two sets returns all $(r', s') \in R' \times S'$ such that r' dominates s'. Every (r', s') corresponds to a unique pair $(r, s) \in R \times S$ satisfying (2) and (3), and vice versa. Performing all the 2^d DJs will produce the desired $Z(\phi_i)$. It is possible that a pair $(r, s) \in R \times S$ is reported multiple, but apparently at most 2^d times.

A final remark concerns the size of $Z(\phi_i)$. When Σ (i.e., the sample set on H) is a $(1/r)$-net (recall that $r = \Theta(n\sqrt{\log n})$, we know from Section 2 that $|Y_i| = O(d \cdot n\sqrt{\log n})$. As $Y^0 \subseteq Y_i$, Lemma 11 implies that w.h.p. $Z(\phi_i)$ has size at most $O(d \cdot n\sqrt{\log n}) + |T_{(R,S)}|$.

6.2 Proof of Theorem 1

To implement the reduction of Section 6.1, we first need to obtain the sample set Σ. This is equivalent to sampling $t = O(d \cdot n\sqrt{\log n})$ pairs from $R \times S$ with replacement. For this purpose, we will take a size-t sample set Σ_1 of R, a size-t sample set Σ_2 of S, randomly permute both, and then produce Σ by paring the i-th element (after permutation) of Σ_1 with that of Σ_2.

To execute the strategy, first apply the in-place sampling algorithm of Section 3 to obtain Σ_1 and Σ_2, respectively. The fact $p \leq n^{0.9}$ assures that w.h.p. every machine produces $O(t/p)$ samples from Σ_1 and Σ_2. To see why, notice that a machine holds $O(n/p)$ points of $R \cup S$ initially, and hence, produces $\Theta(\frac{n}{p}\frac{t}{n}) = \Theta(t/p)$ samples in expectation. By Chernoff, it produces $O(t/p)$ samples with probability at least $1 - e^{-\Theta(t/p)} = 1 - e^{-\Omega(n^{0.1})}$, which is $1 - o(1/(p \cdot n^2))$. Applying the union bound on p machines gives the desired high probability result. Randomly permuting Σ_1 (similarly for Σ_2) can be done by first associating each sample with an independent random integer in some domain $[1, U]$, and then, sorting all the samples by their random integers. The sorted list is a random permutation of Σ_1, as long as no two samples' random integers collide, which can be ensured with probability at least $1 - 1/n^2$ as long as U is chosen to be n^c for a sufficiently large constant c. Finally, pairing up Σ_1 and Σ_2 can also be achieved with sorting. Therefore, w.h.p. the above algorithm returns Σ in constant rounds with load $O(d \cdot t/p)$.

Implementing the rest of the reduction is a simple matter. Carry out the LP step with Theorem 2, which has $d^{O(1)}$ rounds and requires load $o(|X_i|/p) = o(n\sqrt{\log n}/p)$ w.h.p. Regarding the DJ step, recall that we need to produce Y^1 and Y^0. The former can be obtained in constant rounds with load $O(d \cdot (N + |T_{(R,S)}|)/p)$ by broadcasting ϕ_i to all machines, and then, redistributing Y^1 evenly. To obtain Y^0, we first prepare the 2^d instances of DJ locally on the p machines. These instances can then be solved in parallel with Theorem 4 in $O(d^2)$ rounds and load $d^{O(d)} \cdot (n + \mathrm{OUT})/p$ where $\mathrm{OUT} = |Z(\phi_i)|$, which is $O(d \cdot n\sqrt{\log n}) + |T_{(R,S)}|$ w.h.p. Since the algorithm of Theorem 4 is $d^{O(d)}$-out-balanced, each machine holds $d^{O(d)} \cdot |Z(\phi_i)|/p$ elements of $Z(\phi_i)$. This allows the set difference $Z(\phi_i) \setminus T_{(R,S)}$ to be implemented again by sorting in constant rounds and load $d^{O(d)} \cdot |Z(\phi_i)|/p + O(d \cdot |T_{(R,S)}|/p)$. Putting everything together, we have obtained an algorithm solving the EMLC problem w.h.p. using $d^{O(1)}$ rounds and load $d^{O(d)} \cdot (n\sqrt{\log n} + |T_{(R,S)}|)/p$.

It is worth mentioning that the algorithm of Theorem 3, although not explicitly invoked above, is required in the DJ algorithm of Theorem 4.

7 Conclusions

This paper pursues a new direction to perform entity matching classification with a cost that is sub-quadratic to the number of objects involved in the training. The novelty lies in receiving only the *matching* pairs, as opposed to the traditional approach of accepting all of the matching and non-matching pairs. We have demonstrated the first success of the direction, by proving its feasibility for *entity matching with linear classification* (EMLC) on the massively parallel computation (MPC) model. Specifically, for small d (our results are most appealing for constant d, and could also be interesting for $d = O(\log\log N/\log\log\log N)$), we have designed an MPC algorithm that performs EMLC in a small number of rounds, with a load that is (almost) as low as the communication cost of uploading the input to a parallel system. In doing so, we have also obtained new MPC results on several geometric problems: linear programming, batched range counting, and dominance join.

The limitations of the new direction are still yet to be understood when the classifier functions are more sophisticated. On the positive side, we want to develop algorithms with similar performance guarantees for various forms of classifiers (e.g., decision trees and support vector machines), perhaps leveraging recent advances on sparse matrices such as [9]. On the negative side, it is an intriguing question to ask when the direction actually does *not* work, namely, when it is essentially the best approach to work directly with a labeled cartesian product.

Acknowledgements

This work was partially supported by a direct grant (Project Number: 4055079) from CUHK and by a Faculty Research Award from Google.

References

1 Foto N. Afrati, Manas R. Joglekar, Christopher Re, Semih Salihoglu, and Jeffrey D. Ullman. GYM: A multiround distributed join algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 4:1–4:18, 2017.

2 Foto N. Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D. Ullman. Parallel skyline queries. *Theory Comput. Syst.*, 57(4):1008–1037, 2015.

3 Foto N. Afrati and Jeffrey D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(9):1282–1298, 2011.

4 Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, and Abhinandan Nath. Parallel algorithms for constructing range and nearest-neighbor searching data structures. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 429–440, 2016.

5 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 273–284, 2013.

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

7 Xu Chu, Ihab F. Ilyas, and Paraschos Koutris. Distributed data deduplication. *Proceedings of the VLDB Endowment (PVLDB)*, 9(11):864–875, 2016.

8 Kenneth L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM (JACM)*, 42(2):488–499, 1995.

9 Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 81–90, 2013.

10 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 137–150, 2004.

11 Martin E. Dyer and Sandeep Sen. Fast and optimal parallel multidimensional search in PRAMs with applications to linear programming and related problems. *SIAM Journal of Computing*, 30(5):1443–1461, 2000.

12 Herbert Edelsbrunner and Mark H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters (IPL)*, 14(3):124–127, 1982.

13 Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Inf. Syst.*, 65:137–157, 2017.

14 Wenfei Fan, Jingbo Xu, Yinghui Wu, Wenyuan Yu, Jiaxin Jiang, Zeyu Zheng, Bohan Zhang, Yang Cao, and Chao Tian. Parallelizing sequential graph computations. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 495–510, 2017.

15 Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude W. Shavlik, and Xiaojin Zhu. Corleone: hands-off crowdsourcing for entity matching. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 601–612, 2014.

16 Michael T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal of Computing*, 29(2):416–432, 1999.

17 Michael T. Goodrich and Edgar A. Ramos. Bounded-independence derandomization of geometric partitioning with applications to parallel fixed-dimensional linear programming. *Discrete & Computational Geometry*, 18(4):397–420, 1997.

18 Xiao Hu, Paraschos Koutris, and Ke Yi. The relationships among coarse-grained parallel models. *Technical report, HKUST*, 2016.

19 Xiao Hu, Yufei Tao, and Ke Yi. Output-optimal parallel algorithms for similarity joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 79–90, 2017.

20 Bas Ketsman and Dan Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 417–428, 2017.

21 Lars Kolb, Andreas Thor, and Erhard Rahm. Load balancing for mapreduce-based entity resolution. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 618–629, 2012.

22 Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2):197–210, 2010.

23 Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1):484–493, 2010.

24 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-case optimal algorithms for parallel query processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 8:1–8:18, 2016.

25 Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 223–234, 2011.

26 Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice-Hall, 1994.

27 Mert Saglam and Gábor Tardos. On the communication complexity of sparse set disjointness and exists-equal problems. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 678–687, 2013.

28 Yufei Tao, Wenqing Lin, and Xiaokui Xiao. Minimal MapReduce algorithms. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 529–540, 2013.

29 Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI*, pages 15–28, 2012.