Streaming Algorithms for Robust Distinct Elements

Di Chen[*]
Hong Kong Univ. of Science and Technology
Clear Water Bay, Kowloon
Hong Kong
dchenad@cse.ust.hk

Qin Zhang[†]
Indiana University Bloomington
Bloomington, IN 47401
United States
qzhangcs@indiana.edu

ABSTRACT

We study the problem of estimating distinct elements in the data stream model, which has a central role in traffic monitoring, query optimization, data mining and data integration. Different from all previous work, we study the problem in the *noisy* data setting, where two different looking items in the stream may reference the same entity (determined by a distance function and a threshold value), and the goal is to estimate the number of distinct *entities* in the stream. In this paper, we formalize the problem of robust distinct elements, and develop space and time-efficient streaming algorithms for datasets in the Euclidean space, using a novel technique we call bucket sampling. We also extend our algorithmic framework to other metric spaces by establishing a connection between bucket sampling and the theory of locality sensitive hashing. Moreover, we formally prove that our algorithms are still effective under small distinct elements ambiguity. Our experiments demonstrate the practicality of our algorithms.

1. INTRODUCTION

Estimating the number of distinct elements is a fundamental problem in the streaming model [18, 2], where a single machine is observing a sequence of data items of unknown size arriving over time, and it would like to compute the number of distinct elements of this data sequence by a single left-to-right scan using a memory space that is much smaller than the size of the input sequence. The distinct elements problem was first studied by Flajolet and Martin [18] in 1985, and has attracted a significant attention in the past three decades, due to its central importance in traffic monitoring, query optimization, data mining and integration. In this paper we try to attack a more difficult question:

[*]Work supported in part by HKRGC grants GRF-621413 and GRF-16211614. Part of the work was done when the author was visiting Indiana University Bloomington.
[†]Work supported in part by NSF CCF-1525024, and IU's Office of the Vice Provost for Research through the FRSP.

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3531-7/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2882903.2882915

> *What if the dataset is noisy? More precisely, if two different looking data items may reference to the same entity, can we effectively and efficiently estimate the number of distinct entities in the streaming model?*

Here we define an entity in a broad sense: it can be an atomic real-world entity; it can also be a topic (e.g., images can be partitioned into cats, dogs, cars, etc.). In general, given a pairwise distance function between items, and a threshold value α, we say two items reference to the same entity if their distance is no more than α; we call these two items *near-duplicates*. One can view the distinct elements problem in the noise-free case as setting $\alpha = 0$.

Motivation. Our problem is motivated by the fact that real-world datasets are inherently noisy. A few examples:

- Images of the same content uploaded to Flickr may look differently due to different formats, compressions, rescales, photoshop edits, etc.
- Twitter messages are re-distributed with small edits.
- Queries of the same meaning are sent to Google under different keywords combinations.

In this paper we study the distinct elements problem on such noisy datasets in the streaming model. The streaming model is relevant since (1) items (images, messages, query keywords) typically come in a streaming fashion; and (2) a single scan of the dataset using a small working memory is much faster than randomly accessing data in the disk.

We note that one cannot run the streaming algorithms designed for noise-free datasets in the noisy setting since an algorithm for noise-free datasets will treat 100 near-duplicated items as 100 different elements, and then the resulting number of distinct elements will be 100 instead of 1.

The universal needs of handling noisy datasets have fueled an extensive research activity in the past few decades. The line of research that is closest to ours is called data *de-duplication* (or *entity resolution*, *record linkage*, etc.). We refer readers to [25, 14, 21, 12] for introductions to this subject. Unfortunately, in the streaming model it is impossible to perform entity de-duplication thoroughly, simply because we cannot store the whole dataset, even spending only 1 bit for each item in the stream.

One may wonder if we can adapt the previous techniques for noise-free streaming data. Indeed, if we can design a *magic* hash function h which can hash items referencing to the same entity (different entities) to the same element (different elements), then we can simply adopt any existing algorithm for distinct elements in the hashing space. However, such a magic hash function is unlikely to be constructed and stored in a small space (say, logarithmic in terms of the stream length), simply due to the fact that the number of such mappings is exponentially large.

One who is familiar with locality sensitive hashing (LSH) [22, 3] may think if LSH functions can be used as magic hash functions. We notice that LSHs can only guarantee that similar items are mapped into the same bucket with a certain probability, while not-so-similar items may also be mapped into the same bucket with a certain probability. One can use the "AND-OR" trick[1] to sharpen these probabilities, but the "OR" part cannot be implemented in the streaming setting, because unlike "AND" it does not create a well-defined hash value for each point. However, the theory of LSH is still very useful to our problem, and we will discuss it in detail in Section 4.

Similarly, all linear sketching algorithms in the streaming literature cannot be used in the noisy data setting, simply because items referencing to the same entity may be mapped into different coordinates of the sketching vectors.

One who is familiar with clustering may wonder if this problem has any connection with the *streaming k-center* problem (e.g., [20]), in which we want to cluster items to k groups so that the maximum group diameter is minimized. Indeed, an algorithm called `Baseline` that we used for comparison in the experiments can be thought as a streaming algorithm for k-center. The issue of formulating our robust distinct elements problem as a clustering problem is that in our case, the number of clusters, that is the "k", can potentially be linear in the number of distinct entities, and thus the space usage will be at least linear in the worst case.

Problem Definition. Now we formally define our robust distinct elements (F_0) problem.

Definition 1 (Robust F_0). *Let $d(\cdot,\cdot)$ be a distance function, and α be a threshold value. The robust F_0 of a dataset S, denoted by $F_0(S, \alpha)$ in this paper, is the size of the* minimum-cardinality *partition $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$ of S so that for each pair of points p, q that belong to the same group G_i, $d(p, q) \leq \alpha$; in other words, the diameter of each group G_i is at most α.*

In the analysis part of this paper we assume that α is given, which can often be obtained by domain knowledge or deduced from a small sample of labelled items. In the experiments (Section 5) we will also give a method to handle the case where we do not know the value of α.

When clear from the context, we omit α and/or S and simply write F_0 , $F_0(S)$, or $F_0(\alpha)$. Note that the definition of robust F_0 is *monotonic*, that is, $F_0(A) \leq F_0(B)$ if $A \subseteq B$ and $F_0(\alpha_0) \leq F_0(\alpha_1)$ if $\alpha_0 \geq \alpha_1$.

We next introduce a notion of *well-shaped* datasets.

Definition 2 ((α, β)-sparsity and separation ratio). *Let $S \subseteq U$ be a dataset, $d : U \times U \to [0, \infty)$ be a distance function, and α be a threshold value. We say S is (α, β)-*sparse *for some $\beta \geq \alpha$ if for any pair of items $u, v \in S$, we have*

$$\textit{either} \quad d(u, v) \leq \alpha, \ \textit{or} \quad d(u, v) > \beta. \tag{1}$$

We call $\max_\beta \beta/\alpha$ *be the* separation ratio *of S, where β runs over all values that (1) holds.*

[1]http://en.wikipedia.org/wiki/Locality-sensitive_hashing

Intuitively, if S is (α, β)-sparse, then all pairs of items that belong to the same group have distance at most α (consistent with the definition of robust F_0), and all pairs that belong to different groups have distance at least β.

Definition 3 (Well-shaped dataset). *If S is (α, β)-sparse with $\beta/\alpha \geq 2$, then items in S can naturally be partitioned into a set of groups such that the* intra-*group distance (diameter) is at most α, and the* inter-*group distance is more than β. We call such datasets* well-shaped.

To see the natural partition, note that for any three points u, v, w, if $d(u, v) \leq \alpha$ and $d(v, w) \leq \alpha$, by the triangle inequality we must have $d(u, w) \leq 2\alpha \leq \beta$. Now since S is (α, β)-sparse, we must have $d(u, w) \leq \alpha$ (by (1)).

It is also easy to see that if S is well-shaped, then the natural partition achieves the minimum, and it can be computed easily offline: we just group items within distance α in a greedy fashion. However, for a general dataset, computing $F_0(S)$ as an optimization problem is difficult even in the RAM model, not mentioning the more restrictive streaming model. To handle general datasets we introduce a notion we call *F_0-ambiguity*, which characterizes how far a dataset is from well-shapedness. This parameter will be used in our F_0 approximation guarantees for general datasets (Section 3.1).

Definition 4 (F_0-ambiguity). *Let $S \subseteq U$ be a dataset, and let $d : U \times U \to [0, \infty)$ be a distance function. The F_0-ambiguity of S is the minimum δ ($\delta \in [0, 1)$) such that there exists $T \subseteq S$ satisfying the followings:*

- *$S \backslash T$ is well-shaped.*
- *$F_0(S \backslash T) \geq (1 - \delta) \cdot F_0(S)$,*

We use $\delta(S)$ (or simply δ when there is no confusion) to denote the F_0-ambiguity of S.

Note that if $\delta = 0$, then S is well-shaped. Conversely, if S is not well-shaped, then $\delta > 0$. Also note that T is just used for defining how *ambiguous* the dataset is; we are not dropping items in T – the final robust F_0 still takes items in T into account.

We comment in advance that our algorithms are designed to handle datasets with *small* F_0-ambiguities. We believe that it is hard to handle datasets with high F_0-ambiguities in the streaming model since a comprehensive entity-resolution (in particular, identifying two point clouds connected by a bridge point as a single group) cannot be done using a small memory space. For applications mentioned in the motivation, we expect that the F_0-ambiguities of the datasets are small given carefully chosen distance functions and threshold α's. We will show how to automatically determine good α's in the Euclidean metric in Section 5.

Our Contributions. We have made the following contributions in this paper.

1. We have given a first study of the robust F_0 problem in the streaming model, and have obtained *provable* theoretical guarantees. Our space upper bounds for well-shaped datasets in $O(1)$-dimensional Euclidean spaces even match (up to a logarithmic factor) the lower bound in the noise-free setting. This upper bound is obtained using a new technique called *bucket sampling* which may be of independent interest. These are presented in Section 2.1.

2. We have *quantitatively* shown that our algorithms also work well for general datasets in the Euclidean space with small F_0-ambiguity. This is presented in Section 3.

3. We have proposed an algorithm framework for the robust F_0 problem in a general metric space, and established a connection between our bucket sampling framework and the theory of locality sensitive hashing (LSH). More precisely, if a metric space admits an efficient LSH scheme satisfying an additional natural property, then we immediately obtain a streaming algorithm for estimating robust F_0 in that metric space. This is presented in Section 4.

4. We have implemented our algorithms for the Euclidean metric and run them on image data. The experiments have demonstrated the effectiveness of our robust F_0 algorithms in accuracy, space usage and running time. This is presented in Section 5.

Related Work. The distinct elements problem in the noise-free setting has been studied extensively in the streaming literature, due to its numerous applications in network traffic monitoring [15], query planning [27], data mining in graph databases [26], data integration [8] and data warehousing [1]. The first streaming algorithm to $(1+\epsilon)$-approximate distinct elements was proposed by Flajolet and Martin [18], followed by a long line of research in the past several decades ([4, 5, 13, 17, 19], etc.), culminated in an optimal algorithm with $O(1/\epsilon^2 + \log u)$ bits (u is the item universe size) by Kane et al. [24]. However, all these algorithms cannot be used to handle noisy datasets simply because they will consider each item in a group as a different element. As a toy example, if the dataset contains 10000 near-duplicates belonging to the same group, these algorithms will give values close to 10000 which is far away from the true robust F_0 which is 1.

As far as we have concerned, the distinct element problem has not been studied in the noisy streaming data setting. Very recently, statistical estimations for noisy well-shaped datasets have been studied in the distributed setting for several basic problems, including distinct elements, ℓ_0-sampling, frequency moments, heavy hitters and empirical entropy [30], in the general metric space, but the algorithms in [30] cannot be applied to the streaming setting since all of them need a "second look" at the dataset. On the other hand, our streaming algorithms can be trivially translated to algorithms for distributed data: k parties process their local datasets using the streaming algorithm in turn following a fixed order, and then send their memory configurations to their successors; the last party outputs the answer. In particular, by such a translation we can obtain a distributed robust F_0 algorithm with communication cost of $O(k/\epsilon^2)$ words for datasets in the Euclidean space, improving the generic algorithm in [30] by a factor of $1/\epsilon \cdot \text{poly} \log m$ ($m = |S|$ is the length of the stream).

Preliminaries. We use $[t]$ to denote $\{1, \ldots, t\}$. Let $U = [u]$ be the item universe. We say n' is a $(1+\epsilon)$-approximation of n if $n' \in [(1-\epsilon)n, (1+\epsilon)n]$. All log's are base of 2.

We will need a few mathematical tools including Markov Inequality, Chebyshev's inequality and Chernoff bound. Due to the space constraints we leave them in Appendix C

We summarize the main notations in this paper in Table 1.

Symbol	Description
m	length of the stream
n	number of groups, that is, robust F_0
U	item universe
S	set of items in the stream, $\|S\| = m$
$\mathcal{G}/G$	set of groups / a group in $\mathcal{G}$
C	a grid cell in the Euclidean space;
	also denote the set of points in the cell
$\mathcal{C}$	set of non-empty grid cells;
$\mathcal{G}_C$	set of groups intersecting cell C
$w(G)$	number of non-empty cells group G intersects
δ	F_0-ambiguity
ϵ	multiplicative approximation ratio
α	threshold of group diameter
cell(p)	cell where point p is located

Table 1: List of notations

2. WELL-SHAPED DATASETS IN THE EUCLIDEAN SPACE

In this section we consider the case where data items are points in the Euclidean space, which is a very useful metric since many objects, such as documents and images, can be mapped into vectors/points in the Euclidean spaces.

We will focus on well-shaped datasets, that is, those with separation ratio $\beta/\alpha > 2$. The general datasets will be studied in Section 3. W.l.o.g., we assume that $\alpha = 1$, since we can always rescale all distances between points in $S \subset \mathbb{R}^2$ by a factor of α.

2.1 Constant Dimensional Euclidean Spaces

In this section we consider points in constant dimensional Euclidean spaces. We present our algorithms in the 2D case, but they can be literally carried over to any $O(1)$-dimensional spaces.

For a well-shaped dataset $S = \{p_1, \ldots, p_m\}$, let $\mathcal{G} = \{G_1, \ldots, G_{F_0}\}$ be the natural minimum-cardinality group partition of S (we of course do not know F_0 which is the objective that we are going to compute). We post a random grid $\mathbb{G}$ of side length $1/\sqrt{2}$ on $\mathbb{R}^2$, and call a grid cell simply a *cell*. Note that such a random grid can be specified by a pair of two offsets $(\Delta_x, \Delta_y) \in [0, 1/\sqrt{2})^2$ from the origin, and thus can be stored in $O(1)$ words. We assume all points have x, y-coordinates in the range $[-\text{MAX}, \text{MAX}]$, and thus we only need to consider the portion of the grid $\mathbb{G}$ in this range, and consequently $|\mathbb{G}|$ is finite and can be computed in advance. For simplicity, we assume that points do not fall onto the boundary of cells, because this probability will be zero if the grid is random.

Let $\mathcal{C}$ be the collection of non-empty cells, that is, $\mathcal{C} = \{C \in \mathbb{G} \mid C \cap S \neq \emptyset\}$. See Figure 1 for an illustration. In this section $d(p, q)$ denotes the Euclidean distance between points p, q. For a cell C and a point p, let $d(p, C) = \min_{q \in C} d(p, q)$; define $d(p, C) = 0$ if $p \in C$. For a $p \in S$, let cell(p) be the cell where p is located.

2.1.1 *The Algorithm*

The Idea. We start by giving some general ideas of our algorithm. For each cell C in the grid $\mathbb{G}$, let $\mathcal{G}_C$ be the set of groups intersecting C, that is,

$$\mathcal{G}_C = \{G \in \mathcal{G} \mid G \cap C \neq \emptyset\}.$$

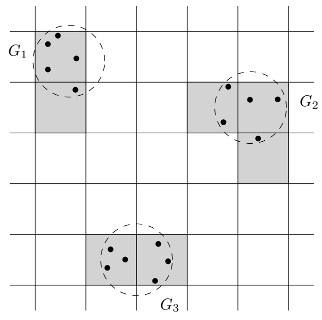

Figure 1: A well-shaped point set. Non-empty cells $\mathcal{C}$ are in gray.

Note that if the dataset is $(1, \beta)$-sparse with $\beta \geq 2$ and $\mathbb{G}$ has side length $1/\sqrt{2}$, then $|\mathcal{G}_C| = 1$ for each non-empty cell $C \in \mathcal{C}$. Let G_C denote this unique intersecting group. Taking Figure 1 for example, the G_C for the left top two cells is G_1; the G_C for the bottom two cells is G_3; and the G_C for the right three cells is G_2.

We define the weight of a group G to be the number of non-empty cells $C \in \mathcal{C}$ it intersects, that is,

$$w(G) = |\{C \in \mathcal{C} \mid C \cap G \neq \emptyset\}|,$$

and define the weight of a non-empty cell $C \in \mathcal{C}$ to be

$$w(C) = \frac{1}{w(G_C)}. \tag{2}$$

For example, in Figure 1, the weights of the left top two cells 1/2; the weights of the bottom two cells are 1/2; and the weights of the right three cells is 1/3. It is easy to see that the sum of weights of all cells is F_0:

$$\begin{aligned} \sum_{C \in \mathcal{C}} w(C) &= \sum_{G \in \mathcal{G}} \sum_{C : C \cap G \neq \emptyset} w(C) \\ &= \sum_{G \in \mathcal{G}} \left(w(G_C) \cdot \frac{1}{w(G_C)} \right) \\ &= F_0. \end{aligned} \tag{3}$$

To obtain a small-space $(1+\epsilon)$-approximation of F_0, a natural idea is to sample a sufficiently large subset of cells $\mathcal{C}' \subseteq \mathcal{C}$, compute their weights, and then add up all weights and rescale to obtain an approximation of F_0.[2] However, at first glance it seems difficult to implement this idea in the streaming model, for the following reasons.

1. The weight of each cell C is determined by how many cells the group G_C intersects, and thus we need to know the information about C's non-empty neighboring cells which intersect group G_C; call them *important* neighboring cells. This suggests us to fix the sampled cells before the stream of points come, since otherwise we risk discarding the information of important neighboring cells when sampling a cell at a later stage of the one-pass scan. Note that we cannot store all the cells due to the small working space constraint.

2. On the other hand, most grid cells will be empty at the end of the streaming process. If we sample cells at the very beginning, then most sampled cells will be empty at the end thus are not in the set $\mathcal{C}$, and consequently cannot be used to approximate F_0. In other words, to get enough non-empty sampled cells at the end we need to sample a much larger number of cells, and thus need a much larger working space.

These two observations seem to contradict each other. That is, the first suggests to sample cells at the beginning of the streaming process, while the second suggests to sample a cell until we know it is not empty (i.e., until an input point in S falls into that cell). To handle this dilemma, we propose a method we call *bucket sampling*, which we will explain in the rest of this section.

Bucket Sampling. Our key idea is to sample *collections* of cells in the grid, and only keep information about non-empty and neighboring non-empty cells for each sampled collection, which can be done at the runtime when an input point in S falls into a cell. We can use a hash function to sample collections of cells at the beginning of the streaming process, and then we do not need to explicitly maintain the IDs of each cell in a sampled collection (which cannot be done in a small space), but just use the hash function to test their "memberships" (i.e., whether they are in a sampled collection) when needed. Since we have employed a hash function to solve the dilemma, we call each collection a bucket, and this sampling process bucket sampling.

We maintain a random hash function $h : \mathbb{G} \to [R]$, where R is a carefully chosen value such that

$$\mathbf{Pr}[|\{C \in \mathcal{C} \mid h(C) = 1\}| \in [1000/\epsilon^2, 20000/\epsilon^2]] \geq 0.99.$$

Note that the cardinality of $\mathcal{C}$ will increase during the streaming process, therefore we need to update(double) the value of R once in a while (see Line 10 in Algorithm 1). Every time R doubles, we also need to update h. At the beginning we set $h(x) = 1$ for all x, and when R is updated we reset $h(x) = (ax + b \bmod \kappa \bmod R) + 1$, where κ is an arbitrary but fixed prime in $[|\mathbb{G}|, 2|\mathbb{G}|]$, and a, b are randomly chosen from $\{0, \ldots, \kappa - 1\}$ conditioned on $a \neq 0$.

We now choose the set of sampled cells to be $\mathcal{C}' = \{C \in \mathcal{C} \mid h(C) = 1\}$. Thus by our choice of R,

$$\mathbf{Pr}\left[|\mathcal{C}'| \in [1000/\epsilon^2, 20000/\epsilon^2]\right] \geq 0.99. \tag{4}$$

To maintain the value R, we can use any off-the-shelf streaming algorithm for distinct elements on *noise-free* datasets to maintain a number z which is a $(1+\epsilon)$-approximation to the number of non-empty cells $|\mathcal{C}|$, and double R when $z \geq 6000R/\epsilon^2$. Algorithm 1 shows how to maintain the value R, the hash function h, and the set of sampled cells $\mathcal{C}'$. In this paper in the theory part we use the algorithm by Kane et al. [24] to maintain z, denoted as `CountNonEmptyCells`(p), where p is a streamed-in point. While in the experiments we will use the algorithm by Bar-Yossef et al. [4] which is simpler to implement, though its theoretical performance is a bit worse. The formal analysis to show (4) will be given in Section 2.1.2.

Point Storage. During the streaming process we will store a set of center points, denoted by Γ initialized to be $\emptyset$. At

[2]We comment that up to this point, at the high level, our idea is similar to Algorithm 2 in [30] in the distributed model, but there is an important difference which is used to save the space usage by a factor of $\max_{G \in \mathcal{G}} |G|$: we operate on *cells* instead of *individual items*. Our algorithm after this point will be very different from that in [30], mainly because in the streaming model we are not allowed to scan the input for a second time.

Algorithm 1 Maintain the Set of Sampled Cells $\mathcal{C}'$

```
1: $z$ is a global variable which is a $(1+\epsilon)$-approximation of $|\mathcal{C}|$
2: $h : \mathbb{G} \to [R]$ is the global random hash function with $R$ initialized to be 1.
3: $\kappa$ is an arbitrary but fixed prime between $|\mathbb{G}|$ and $2|\mathbb{G}|$. $a$ is randomly sampled from $\{1, \ldots, \kappa - 1\}$ and $b$ is randomly sampled from $\{0, 1, \ldots, \kappa - 1\}$.
4: procedure MaintainSampledCells($p$)
5:     if $h(\text{cell}(p)) = 1$ then
6:         $\mathcal{C}' := \mathcal{C}' \cup \{\text{cell}(p)\}$
7:     end if
8:     $z :=$ CountNonEmptyCells($p$)
9:     if $z \geq 6000R/\epsilon^2$ then
10:        $R := 2R$                          ▷ Double the range of $h$
11:        set $h(x) = (ax + b \bmod p \bmod R) + 1$
12:        $\mathcal{C}' := \{C \in \mathcal{C}' \mid h(C) = 1\}$ and discard stored point centers (Algorithm 2) in cells that are no longer in $\mathcal{C}'$    ▷ Re-sample each cell in $\mathcal{C}$ with probability 1/2
13:    end if
14: end procedure
```

Algorithm 2 Store Point Centers for Sampled Cells $\mathcal{C}'$

```
1: procedure StoreCenter($p$)
2:     if $\exists C \in \mathbb{G}$ s.t. $h(C) = 1 \wedge d(p, C) \leq 1$ then
3:         if ($\nexists q \in \Gamma$ s.t. cell($p$) = cell($q$)) then
4:             insert $p$ to $\Gamma$              ▷ Keep a new center
5:         end if
6:     end if
7: end procedure
```

the end of the streaming process we will use Γ to recover the weight of each sampled cell. The algorithm for processing and storing each input point is described in Algorithm 2. In words, it maintains all streamed-in points p that are in or within distance 1 to a cell in $\{C \in \mathbb{G} \mid h(C) = 1\}$, given that no other point in cell(p) has already been stored in Γ.

We illustrate the "sample and store" procedure in Figure 2. Cells with red heavy strokes are sampled cells in $\mathcal{C}$; points in red are stored point centers.

We comment that in Algorithm 2 it could be the case that a stored point p is within a distance of 1 to a sampled cell $C \in \mathbb{G}$ that is not in $\mathcal{C}$; in other words, C will be empty at the end of the streaming process and thus will be ignored. In such a case p may not be used to compute the weights for any sampled $C \in \mathcal{C}'$ at the end (in Algorithm 3), and is thus *wasted*. However, in the analysis we will show that such an overhead will not affect the asymptotic storage cost.

Post-processing and the Full Algorithm. At the end of the streaming process, we use Algorithm 3 to compute the weight for each sampled cell $C \in \mathcal{C}'$. We simply find all groups that intersect C using the stored point centers, and then for each such group compute its weight. The weight of the sampled cell will be the sum of the inverse of these group weights. Note that in the setting considered in this section each cell only intersects at most one group, and thus the *while* iteration at Line 7 will only run once. In Section 2.2 when we consider higher dimensional Euclidean spaces, a cell may intersect multiple groups.

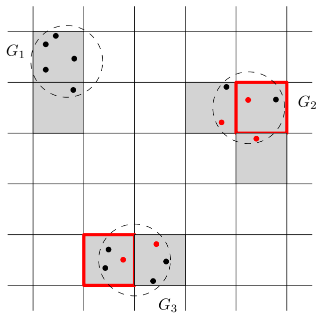

Figure 2: Cells with red stroke are sampled cells in $\mathcal{C}$; points in red are stored point centers.

Algorithm 3 Computing Weights of Sampled Cells

```
1: $\Gamma$ is the set of points stored by calling StoreCenter( )
2: procedure ComputeWeight( )
3:     $W := 0$
4:     $\Gamma' = \Gamma$                              ▷ Backup of $\Gamma$
5:     for each $C \in \mathcal{C}'$ do
6:         $w := 0$
7:         while $\exists p \in \Gamma$ s.t. $d(p, C) \leq 1$ do
8:             pick any such $p \in \Gamma$  ▷ Consider a group in $G$ intersecting $C$
9:             $K := \{\text{cell}(p)\}$
10:            for each $q \in \Gamma$ s.t. $d(q, p) \leq 1$ do
11:                if cell($q$) $\notin K$ then
12:                    $K := K \cup \{\text{cell}(q)\}$   ▷ $K$ is the set of cells the considered group $G$ intersects
13:                end if
14:                delete $q$ from $\Gamma$
15:            end for
16:            $w := w + \frac{1}{|K|}$  ▷ Compute the weight of cell $C$
17:            delete $p$ from $\Gamma$
18:        end while
19:        $W := W + w$            ▷ Compute total weights of sampled cells
20:        $\Gamma := \Gamma'$                             ▷ Restore $\Gamma$
21:    end for
22:    return $W$
23: end procedure
```

We describe the final one-pass streaming algorithm in Algorithm 4.

2.1.2 The Analysis

We now show the correctness of our algorithm and analyze its efficiency. We need the following lemma in [24] to maintain the value z at Line 8 in Algorithm 1.

Lemma 1 ([24]). *There is an algorithm that computes a $(1 + \epsilon)$-approximation of $|\mathcal{C}|$ with probability 0.99, using $O(1/\epsilon^2 + \log u)$ bits space and $O(1)$ processing time per item.*

Correctness. Let $F_0 = F_0(S, 1)$ be the number of groups in S, which is the value we are going to compute. First, notice that

$$1 \leq |\mathcal{C}|/F_0 \leq 9. \tag{5}$$

The left hand side inequality is due to the fact that each non-empty cell intersects at most one group in our setting.

Algorithm 4 Computing Robust F_0 in $\mathbb{R}^2$

```
1: $\mathcal{C}'$ is the set of sampled cells, maintained by MaintainSampledCells( )
2: $z$ is a global variable which is a $(1+\epsilon)$-approximation of $|\mathcal{C}|$
3: Streamed input points $p_1, \ldots, p_m$
4: while a new input point $p_i$ comes do
5:     MaintainSampledCells($p_i$)
6:     StoreCenter($p_i$)
7: end while
8: $w :=$ ComputeWeight( )                ▷ Post-processing
9: Output $\frac{z}{|\mathcal{C}'|} \cdot w$     ▷ Rescale to compute robust $F_0(S)$
```

The right hand side inequality holds since the side length of the grid is $1/\sqrt{2}$, while the diameter of each group is at most 1, as a consequence each group intersects at most 9 grid cells.

Second, we show that Inequality (4) holds. Recall that at Line 9, 10 in Algorithm 1, R is maintained such that

$$\frac{3000R}{\epsilon^2} \le z \le \frac{6000R}{\epsilon^2},$$

where z is a $(1+\epsilon)$-approximation of $|\mathcal{C}|$. Therefore

$$\frac{|\mathcal{C}|}{R} \in \left[\frac{2000}{\epsilon^2}, \frac{12000}{\epsilon^2}\right]. \tag{6}$$

Write $\mathcal{C}$ as $\{C_1, \ldots, C_{|\mathcal{C}|}\}$. For each $C_i \in \mathcal{C}$, define $Y_i = 1$ if $h(C_i) = 1$, and $Y_i = 0$ otherwise. Let $Y = \sum_{i \in [|\mathcal{C}|]} Y_i$, thus $Y = |\mathcal{C}'|$. We have $\mathbf{E}[Y] = \frac{|\mathcal{C}|}{R}$. By a Chernoff bound,

$$\begin{aligned}\mathbf{Pr}\left[|\mathcal{C}'| \in \left[\frac{|\mathcal{C}|}{2R}, \frac{3\,|\mathcal{C}|}{2R}\right]\right] &= \mathbf{Pr}\left[|Y - \mathbf{E}[Y]| \le \frac{\mathbf{E}[Y]}{2}\right] \\ &\ge 1 - 2e^{-\frac{|\mathcal{C}|}{12R}} \ge 1 - 2e^{-\frac{2000}{12\epsilon^2}} \\ &\ge 0.99.\end{aligned}$$

Consequently,

$$\begin{aligned}\mathbf{Pr}\left[|\mathcal{C}'| \in \left[\frac{1000}{\epsilon^2}, \frac{20000}{\epsilon^2}\right]\right] &\overset{\text{by (6)}}{\ge} \mathbf{Pr}\left[|\mathcal{C}'| \in \left[\frac{|\mathcal{C}|}{2R}, \frac{3\,|\mathcal{C}|}{2R}\right]\right] \\ &\ge 0.99,\end{aligned}$$

proving Inequality (4).

Now let $t = |\mathcal{C}'|$ be the number of sampled cells. Define X_i ($i \in [t]$) be the random variable representing the weight of the i-th sampled cell $C_i \in \mathcal{C}'$, and let $X = \frac{1}{t}\sum_{i \in [t]} X_i$. Since each cell in $\mathcal{C}'$ is sampled from the set of non-empty cells $\mathcal{C}$ independently uniformly at random, we have

$$\mathbf{E}[X_i] = \frac{F_0}{|\mathcal{C}|} \quad \text{for each } i \in [t],$$

$$\text{and} \qquad \mathbf{E}[X] = \mathbf{E}\left[\frac{1}{t}\sum_{i \in [t]} X_i\right] = \frac{F_0}{|\mathcal{C}|}. \tag{7}$$

We next compute their variances. For each $i \in [t]$.

$$\begin{aligned}\mathbf{Var}[X_i] &= \mathbf{E}[X_i^2] - \mathbf{E}^2[X_i] \le \mathbf{E}[X_i^2] \\ &= \frac{1}{|\mathcal{C}|}\sum_{C \in \mathcal{C}} (w(C))^2 \\ &\le \frac{1}{|\mathcal{C}|} \cdot F_0 \overset{\text{by (7)}}{=} \mathbf{E}[X],\end{aligned}$$

where the last inequality is by the fact that $w(C) \le 1$ for any $C \in \mathcal{C}$, and $\sum_{C \in \mathcal{C}} w(C) = F_0$ (by (3)).

Since X_i ($i \in [t]$) are i.i.d. samples, we have

$$\mathbf{Var}(X) = \frac{1}{t^2}\sum_{i \in [t]} \mathbf{Var}(X_i) \le \frac{\mathbf{E}[X]}{t}. \tag{8}$$

Finally, by a Chebyshev's inequality, we have

$$\mathbf{Pr}[|X - \mathbf{E}[X]| > \epsilon \mathbf{E}[X]] \le \frac{\mathbf{E}[X]}{t \cdot \epsilon^2 \mathbf{E}^2[X]} \le 0.01, \tag{9}$$

where the last inequality is due to $\mathbf{E}[X] = \frac{F_0}{|\mathcal{C}|} \ge 1/9$ (by (5)) and $t = |\mathcal{C}'| \ge 1000/\epsilon^2$ (by (4)). Therefore, $X \cdot |\mathcal{C}|$ is a $(1+\epsilon)$-approximation of F_0 with probability at least 0.99. Finally, in Algorithm 4 we have $w = X \cdot t$, and z is a $(1+\epsilon)$-approximation of $|\mathcal{C}|$, thus zw/t is a $(1+\epsilon)^2 \le (1+3\epsilon)$-approximation of F_0. Note that the constant 3 in the approximation ratio can be removed by setting $\epsilon' = 3\epsilon$ and adjusting relevant constants in the analysis.

To sum up, the final success probability is $1 - 0.01 - 0.01 - 0.01 = 0.97$, where the first error term 0.01 is introduced by (4), the second error term 0.01 is introduced by (9), and the third error term 0.01 is introduced by Lemma 1.

Space and Time Complexities. Now we analyse the performance of Algorithm 4.

For the space usage, the dominating cost is the storing of point centers in Γ. Note that each cell C will have at most one point center, which is stored only if $h(C) = 1$ or $h(C') = 1$ where C' is any of C's neighboring cells. Thus each $C \in \mathcal{C}$ will have a point center stored with probability at most $9/R = O(1/(\epsilon^2\,|\mathcal{C}|))$. Therefore the total stored point centers can be bounded by $O(1/\epsilon^2)$ with probability $1 - o(1)$ by a Chernoff bound. The costs for storing other random variables in Algorithm 4 and its subroutines, the grid and the hash functions are negligible.

For each new input point p, storing p using Algorithm 2 can be done in $O(1)$ time, since we can locate the cell $h(p)$ and then search/check its neighboring sampled cells in constant time. By Lemma 1, `MaintainSampledCells`(p) can be done in $O(1)$ time. Therefore the total processing time per item is $O(1)$. At the end, Algorithm 4 calls `ComputeWeight`() to post-process the stored information and then return the answer. It is easy to see that `ComputeWeight`() can be done in $O(1/\epsilon^2)$ time since each stored center point and each sampled cell will be considered at most $O(1)$ times.

We can in fact avoid a separate post-processing step by maintaining the weights of each cell and their sum during the streaming process. This will not change the $O(1)$ processing time per item since each item will only affect the weights of an $O(1)$ number of neighboring cells.

As mentioned, our algorithm and analysis can be literally carried to any $O(1)$-dimensional space (by appropriately adjusting constant parameters). We sum up this section with the following theorem. Note that our space upper bound even matches (up to a logarithmic factor) the $\Omega(1/\epsilon^2)$ bits space streaming lower bound for distinct elements in the noise-free data setting [23].

Theorem 1. *There is a streaming algorithm that given a well-shaped dataset S in the $O(1)$-dimensional Euclidean space, outputs a $(1+\epsilon)$-approximation to robust $F_0(S)$ with probability* 0.96*, using $O(1/\epsilon^2)$ words of space and $O(1)$ processing time per item.*

2.2 High Dimensional Euclidean Spaces

In this section we consider points in the d-dimensional Euclidean space for $d > 2$.

Following the ideas in Section 2.1.1, we employ a d-dimensional grid but with side length d to partition $\mathbb{R}^d$ to cells. We first discuss the case when $\beta \geq d^{3/2}$, which guarantees that each cell intersects at most one group of points. We will show that in this case our algorithm for the 2D case (with the grid size replaced by d) can still produce a $(1+\epsilon)$-approximation to $F_0(S)$. We then extend it to general well-shaped datasets (i.e., $(1,\beta)$-sparse with $\beta \geq 2$).

2.2.1 *$(1,\beta)$-sparse Datasets with $\beta \geq d^{3/2}$*

Correctness. Compared with the 2D case discussed in Section 2.1, the main difference in the d-dimensional case with a random grid of side length d is that each group can intersect up to 2^d grid cells in the worse case. Note that we cannot simply use 2^d to replace the constant "9" in Equation (5) in the 2D case, because the number of sampled cells $|\mathcal{C}'|$ and the final space usage will be proportional to this intersection value, and will consequently be exponential in d. Fortunately, we can show that *on average* each group will intersect $\Theta(1)$ cells, which is enough for keeping the correctness of Algorithm 4.

LEMMA 2. *Let S be a $(1,\beta)$-sparse dataset in the d-dimensional Euclidean space with $\beta \geq d^{3/2}$, let $\mathbb{G}$ be a random grid of side length d, let $\mathcal{C} = \{C \in \mathbb{G} \mid C \cap S \neq \emptyset\}$, and let $\mathcal{G} = \{G_1, \ldots, G_{F_0}\}$ be the natural minimum-cardinality group partition of S. Then $1 \leq |\mathcal{C}|/F_0 \leq 300$ with probability* 0.99.

Due to the space constraints, we delay the technical proof to Appendix B.1.

It is easy to see that if we use Lemma 2 to replace Inequality (5), the rest of correctness proof in Section 2.1.2 still holds (by appropriately adjusting constant parameters).

Space and Time Complexities. We next analyze the space and time costs of our algorithm in the d-dimensional Euclidean space. For a group $G \in \mathcal{G}$, let $\mathsf{B}(G)$ be a ball of diameter 3 that contains all unit balls centered at some point $q \in G$. Let ν be the probability that $\mathsf{B}(G)$ is cut by the boundaries of grid cells, which can be bounded by $\nu \leq 3/d$. Let N_G be the number of grid cells group G intersects; let M_G be the number of grid cells that group G adjoin, i.e.,

$$M_G = |\{C \mid \exists p \in G \text{ s.t. } d(p,C) \leq 1\}|.$$

Obviously $N_G \leq M_G$. Similar to (23) (in Appendix B.1), by replacing μ with ν, we can show that

$$\mathbf{E}[M_G] = O(1). \tag{10}$$

Note that a non-empty cell $C \in \mathcal{C}$ will have a point stored only if at least one cell in M_{G_C} has been sampled. We thus can upper bound the total number of point centers stored in Γ by

$$Z = \sum_{G \in \mathcal{G}} (N_G \cdot \chi(\text{one of cells in } M_G \text{ is sampled})),$$

where $\chi(A) = 1$ if event A holds, and $\chi(A) = 0$ otherwise.

We now bound the expectation of Z.

$$\begin{aligned}
\mathbf{E}[Z] &= \sum_{G \in \mathcal{G}} \mathbf{E}[N_G \cdot \chi(\text{one of cells in } M_G \text{ is sampled})] \\
&\leq \sum_{G \in \mathcal{G}} \mathbf{E}[M_G \cdot \chi(\text{one of cells in } M_G \text{ is sampled})] \\
&\leq \sum_{G \in \mathcal{G}} \sum_{i=0}^{d} 2^i \binom{d}{i} \nu^i (1-\nu)^{d-i} \cdot \frac{2^i}{R} \\
&= \frac{F_0}{R} \cdot (4\nu + (1-\nu))^d \\
&\leq O\left(\frac{1}{\epsilon^2}\right) \cdot (1 + 9/d)^d = O\left(\frac{1}{\epsilon^2}\right),
\end{aligned}$$

where the first inequality is due to the fact that $N_G \leq M_G$ (by definitions); in the second inequality, $2^i/R$ is the probability that one of cells in M_G is sampled (recall that R is the range size of the random hash function used in the bucket sampling; see Section 2.1.1); the last inequality is due to $|\mathcal{C}| \geq F_0$ and Equation (6) (constants need to be adjusted for the high dimensional case). Therefore by a Markov inequality, $Z = O(1/\epsilon^2)$ with probability 0.99.

We next consider the processing time per item. Compared with the 2D case, the difference is that in `StoreCenter`(p), in the worst case one needs to test up to 2^d adjacent cells to see if any of them are sampled. Fortunately by (10) we can upper bound the total processing time by

$$T = \sum_{G \in \mathcal{G}} (|G| \cdot M_G).$$

Taking the expectation,

$$\begin{aligned}
\mathbf{E}[T] &= \sum_{G \in \mathcal{G}} (|G| \cdot \mathbf{E}[M_G]) \\
&\overset{\text{by (10)}}{\leq} O(1) \cdot \sum_{G \in \mathcal{G}} |G| = O(m).
\end{aligned}$$

Therefore $T = O(m)$ with probability 0.99 by a Markov inequality. Thus on average, for each inserted point we only need to check $O(1)$ adjacent cells.

Finally, the cost of the post-processing step can again be amortized into each insertion step. We thus arrive at the following theorem.

THEOREM 2. *There is a streaming algorithm that given a $(1,\beta)$-sparse dataset S with $\beta \geq d^{3/2}$ in the d-dimensional Euclidean space, outputs a $(1+\epsilon)$-approximation to robust $F_0(S)$ with probability* 0.9, *using $O(d/\epsilon^2)$ words of space and amortized $O(d)$ processing time per item.*

2.2.2 *Well-shaped Datasets*

For general well-shaped datasets in $\mathbb{R}^d$ $(d > 2)$ and a grid with side length d, it is possible that a grid cell intersects multiple groups of points. While the high level idea of sampling cells and add up their weights still apply, we need to modify the definition of the weight of a non-empty cell $C \in \mathcal{C}$ (Equation (2)) to be

$$w(C) = \textstyle\sum_{G \in \mathcal{G} : G \cap C \neq \emptyset} 1/w(G).$$

Algorithms presented in Section 2.1 can still be used for this case. However, once a cell intersects multiple groups, its weight can no longer be bounded by a constant. Thus it is not clear how to use rigorous theoretical analysis to determine the least number of cells we need to sample in order to obtain a $(1+\epsilon)$-approximation of $F_0(S)$, since the variance of the weights of cells can be very high. We leave this to future work.

3. GENERAL DATASETS IN CONSTANT DIMENSIONAL EUCLIDEAN SPACES

In this section we consider a general dataset which is possibly not well-shaped. That is, it has an F_0-ambiguity $\delta > 0$. We can rigorously show that Algorithm 4 still performs well if δ is small. Ideally, we would also like to have an algorithm to estimate the F_0-ambiguity δ of a dataset, so that we can filter out those inputs that have a large δ, on which Algorithm 4 may not perform well; in other words, we can declare that the value returned by Algorithm 4 may not be very accurate due to the fact that the dataset is "too noisy" (call such a dataset a *bad* input). Unfortunately, we noticed that one needs $\Omega(m)$ (recall that $m = |S|$ is the length of the stream) space to differentiate whether $\delta = 0$ or $1/2$.

3.1 Approximation Guarantees under Small F_0-ambiguity

In this section we prove the following theorem, which shows that Algorithm 4 still performs well when F_0-ambiguity is small.

THEOREM 3. *There is a streaming algorithm that given a dataset S with F_0-ambiguity δ in the $O(1)$-dimensional Euclidean space, outputs a $(1 + \Theta(\delta + \epsilon))$-approximation to robust $F_0(S)$ with probability* 0.9, *using $O(1/\epsilon^2)$ words of space and $O(1)$ processing time per item.*

The proof for this theorem is very technical, and we delay it to Appendix A due to the space constraints. The idea of the proof is as follows. Let T be the set of "outliers" defined in Definition 4, such that S/T is well-shaped and $F_0(S\backslash T) \geq (1 - \delta) \cdot F_0(S)$. If the F_0-ambiguity δ is small, then we can think the optimal grouping is the natural grouping for $S\backslash T$ plus some balls[3] of diameter α covering points in T. Note that in $O(1)$-dimensional Euclidean space, a ball of diameter 2α can be covered by $O(1)$ balls of diameter α. Thus in the optimal solution the balls that are used to cover T are *evenly spread* across the (natural) groups formed by $S\backslash T$. Since our algorithms use *near-uniform* group samplings (the probability of sampling each group differs by at most a constant), if δ is small then only a small fraction of our sampled groups will be relevant to points in T. Consequently, the outliers T will not affect much of our estimation of $F_0(S\backslash T)$, which is close to $F_0(S)$ by the definition of T.

3.2 Estimating F_0-ambiguity is Theoretically Hard

The problem of estimating the F_0-ambiguity is closely related to the problem of computing the diameter of a point set, that is, given a set of points S in the Euclidean space, try to find $\max_{p,q\in S} d(p, q)$. It has been shown that to compute the diameter exactly in the streaming model, we need $\Omega(|S|) = \Omega(m)$ space [16].

The connection between computing the diameter of the point set and estimating its F_0 ambiguity is as follows. Again rescale $\alpha = 1$. If the diameter of the dataset is 1, then all the points will be in one group, that is, (robust) $F_0 = 1$; otherwise if the diameter is $1 + \iota$ for an arbitrarily small $\iota > 0$, then F_0 will be at least 2. By the hardness of the diameter problem we cannot differentiate whether $F_0 = 1$ or 2 unless using $\Omega(m)$ space, equivalently, we cannot differentiate whether $\delta = 0$ or $1/2$ without using $\Omega(m)$ space.

[3]This is not very precise since a group of diameter 1 may not be covered by a ball of diameter 1, but they are close.

4. A SOLUTION FRAMEWORK FOR GENERAL METRIC SPACE

So far we have discussed datasets in the Euclidean space. In this section we show that our algorithmic framework in Section 2 can also be used for datasets in other metric spaces.

Recall our high level ideas for datasets in the Euclidean space in Section 2. First, we use a random grid to partition the item universe U to cells so that each group of items only adjacent to a small number of grid cells. Second, we reduce the problem of approximating robust $F_0(S)$ to sampling cells and adding up their weights (and rescaling at the end). Third, to solve the dilemma on the timing of the samplings we employ a technique we call bucket sampling.

The observation is that the second and third steps can be applied to any metric spaces, while in the first step the random grid can be thought as a "hash function", which is generalized to the following concept.

DEFINITION 5 (SMART HASH FUNCTION). *We say a hash function h is ρ-*smart *on an (α, β)-sparse ($\beta \geq 2\alpha$) dataset S and its natural minimum-cardinality group partition if it satisfies the followings:*

- *(Low "image distance"). Each group is adjacent to ρ cells on average, where we say a group G is adjacent to a hash cell C if there exists a pair of items $p, q \in S$ such that $p \in G, h(q) = C$ and $d(p, q) \leq \alpha$. We call ρ the* image distance *of the smart hash function.*
- *(No false-positive). Items from different groups will be hashed into disjoint buckets.*

Obviously, the smaller the ρ, the better the performance of the smart hash function. When $\rho = 1$, we get a magic hash function, which is however unlikely to exist (see our discussion in the introduction).

4.1 Locality Sensitive Hashing as Smart Hash Functions

As we have shown, a random grid can be used as a smart hash function for the Euclidean space. In fact, a random grid can be seen as a *locality sensitive hash* (LSH) function, whose general form is as follows:

DEFINITION 6 (LSH). *(see, e.g., [22, 3]) Let U be the item universe, and $d(\cdot, \cdot)$ be a distance function. We say a hash family $\mathcal{H}$ is (ℓ, u, p_1, p_2)-sensitive if for any two items $p, q \in U$,*

1. *if $d(p, q) \leq \ell$ then $\mathbf{Pr}_{h\in_r\mathcal{H}}[h(p) = h(q)] \geq p_1$,*
2. *if $d(p, q) \geq u$ then $\mathbf{Pr}_{h\in_r\mathcal{H}}[h(p) = h(q)] \leq p_2$,*

where $h \in_r \mathcal{H}$ means picking h randomly from $\mathcal{H}$.

Under this definition, the random grid we used in Section 2.1 (in $\mathbb{R}^2$, of side length $1/\sqrt{2}$) can be thought as a $\left(\ell, u, \frac{1}{\sqrt{2}} - \ell, \frac{(1-u)^2}{2}\right)$-sensitive LSH. In this section we show that a locality sensitive hash function for *any* metric space, if satisfies an additional natural property, can be used as a smart hash function for that space, and thus can be used to compute robust F_0.

Definition 7 (Concentrated Hash Function). *Let $S \subseteq U$ be a dataset and let $\mathcal{G} = \{G_1, \ldots, G_{F_0}\}$ be the natural minimum-cardinality partition of S. A hash function h is called η-concentrated on S if for any $G \in \mathcal{G}$,*

$$|\{h(x) \mid \exists y \in G \text{ s.t. } d(x,y) \le \alpha\}| \le \eta.$$

We say an LSH family that is η-concentrated on S if for any $h \in \mathcal{H}$, h is η-concentrated on S.

In fact, many popular LSHs are of low concentration, as we will see shortly.

Definition 8 (k-fold Hash Function). *Let $\mathcal{H}$ be a hash family. We say $\mathcal{F}$ is a k-fold hash family of $\mathcal{H}$ if*

$$\mathcal{F} = \{F = (h_1, \ldots, h_k) \mid h_i \in \mathcal{H} \text{ for any } i \in [k]\},$$

For any $x, y \in U$, we define $F(x) = F(y)$ if and only if $h_i(x) = h_i(y)$ for any $i \in [k]$.

The following lemma reveals the connection between LSHs and smart hash functions. The proof can be found in Appendix B.2.

Lemma 3. *Let $\beta \ge 2\alpha$. Let $S \subseteq U$ be an (α, β)-sparse dataset consisting of m items. Let $\mathcal{G}$ be the natural minimum-cardinality partition of S. Let $\mathcal{H}$ be a $(2\alpha, \beta, p_1, p_2)$-sensitive LSH family that is η-concentrated on S. Let $\mathcal{F}$ be a k-fold hash family of $\mathcal{H}$ and let $f \in_r \mathcal{F}$. Then f is $100(\eta(1-p_1)+p_1)^k$-smart on S with probability at least $(0.99 - m^2 p_2^k)$.*

We next show two examples on how use Lemma 3 together with some well-known LSHs to construct smart hash functions with low image distances for data in corresponding metric spaces.

Gaussian LSH for Euclidean Metric. For a point set $S \subseteq \mathbb{R}^d$ ($|S| = m$) in the Euclidean space that is $(\frac{1}{2}, \beta)$-sparse (rescale α to $\frac{1}{2}$ for technical convenience), we can use the Gaussian LSH [10]. Let γ a parameter. Define Gaussian LSH family $\mathcal{H}_G$ be the set of functions $h_{\vec{a},b}(\vec{x}) : \mathbb{R}^d \to \mathbb{N}$ mapping a d-dimensional vector $\vec{x}$ onto the set of integers, where $\vec{a}$ is a vector chosen from the unit sphere $\mathcal{S}^{d-1}$, and b is a scale parameter in the range $[0, \gamma)$. More precisely,

$$h_{\vec{a},b}(x) = \left\lfloor \frac{\vec{x} \cdot \vec{a} + b}{\gamma} \right\rfloor.$$

It was shown in [10] that $\mathcal{H}_G$ is $(1, \beta, p(1), p(\beta))$-sensitive where

$$p(x) = 1 - 2\Phi\left(-\frac{\gamma}{x}\right) - \frac{2}{\sqrt{2\pi}\gamma/x}\left(1 - e^{-\frac{\gamma^2}{2x^2}}\right),$$

where $\Phi(\cdot)$ is the cumulative distribution function for the standard Gaussian distribution $\mathcal{N}(0,1)$. Now we choose $\beta = \log m$ and $\gamma = \log m$, and then

- $\eta = 2$ since each group, when projecting to a line by an $h \in \mathcal{H}$, has a diameter at most 1, and thus can adjacent at most 2 cells;
- $p_1 = p(1) \ge 1 - 2/\log m$;
- $p_2 = p(\beta) = 1 - 2\Phi(-1) - \frac{2}{\sqrt{2\pi}}(1 - e^{-1/2}) < 0.4$.

Finally let $\mathcal{F}$ be a k-fold hash family of $\mathcal{H}$ with $k = 2\log m$. By Lemma 3 we have that for an $f \in_r \mathcal{F}$, f is $\Theta(1)$-smart on S with probability at least 0.98.

To plug-in the algorithm framework in Section 2, we need to define for a point p and a hash bucket C the distance $d(p, C)$ so that we can test $d(p, C) \le \alpha$ (in Section 2 w.l.o.g. we have assumed $\alpha = 1$). Note that LSH buckets *partitions* the item universe. In the case of Gaussian LSH for Euclidean Metric, the set of the points in $\mathbb{R}^d$ (denoted by $S(C)$) hashed into the same bucket C is the intersection of $2k = O(\log m)$ half-spaces. We define $d(p, C) = \min_{q \in S(C)} d(p, q)$, which can be computed by measuring the Euclidean distances between p and the boundaries of the $2k$ half-spaces.

The following theorem summarizes the performance of Gaussian LSH on estimating robust F_0 in the Euclidean metric. The analysis is very similar to that for high dimensional Euclidean space in Section 2.2, and is omitted here.

Theorem 4. *There is a streaming algorithm that given a dataset S with separation ratio $\Omega(\log m)$ in the d-dimensional Euclidean space, outputs a $(1+\epsilon)$-approximation to robust $F_0(S)$ with probability 0.9, using $O(\log m/\epsilon^2)$ words of space and amortized $O(d \log m)$ processing time per item.*

Random Projection LSH for Cosine Metric. The cosine distance function is defined as follows: given two vectors $\vec{a}, \vec{b} \in \mathbb{R}^d$, the cosine distance is defined to be $d(\vec{a}, \vec{b}) = 1 - \langle \vec{a}, \vec{b} \rangle / (\|\vec{a}\| \, \|\vec{b}\|)$. This distance function is used extensively in information retrieval and text mining, in particular, in comparing the similarity between two documents [28].

For a set of vectors S ($|S| = m$) in $\mathbb{R}^d$ under the cosine distance, we can use the random projection LSH [9]. Define random projection LSH family $\mathcal{H}_P$ be the set of functions $h_{\vec{a}}(\vec{x}) : \mathbb{R}^d \to \{+1, -1\}$ mapping a d-dimensional vector $\vec{x}$ onto signs $\{+1, -1\}$, where $\vec{a}$ is a vector chosen from the unit sphere $\mathcal{S}^{d-1}$. Concretely,

$$h_{\vec{a}}(\vec{x}) = \begin{cases} +1 & \text{if } \langle \vec{a}, \vec{x} \rangle > 0 \\ -1 & \text{otherwise,} \end{cases} \tag{11}$$

It is shown in [9] that $\mathcal{H}_P$ is $(2\alpha, \beta, 1 - 2\alpha/\pi, 1 - \beta/\pi)$-sensitive. Obviously $\eta = 2$ since the range of each hash function in $\mathcal{H}_P$ only consists of two numbers $\{+1, -1\}$.

Let $\mathcal{F}$ be a k-fold hash family of $\mathcal{H}$ with $k = \frac{3 \log m}{\log(1 - \pi/\beta)}$. By Lemma 3, for any parameter α, β with $\alpha \le \frac{1}{\log m}$ and $\Omega(1) \le \beta < \pi$, we have that for an $f \in_r \mathcal{F}$, f is $\Theta(1)$-smart on S with probability at least 0.98.

Similar to Gaussian LSH, the set of the vectors/points in $\mathbb{R}^d$ hashed into the same Random Projection LSH bucket is the intersection of $k = O(\log m)$ half-spaces. Thus $d(p, C)$ can again be computed by measuring the cosine distances between p and the boundaries of the k half-spaces.

We have the following theorem for random projection LSH for the cosine metric. The analysis is again similar as before and is omitted.

Theorem 5. *Let α, β be parameters such that $\alpha \le \frac{1}{\log m}$ and $\Omega(1) \le \beta < \pi$. There is a streaming algorithm that given an (α, β)-sparse dataset S in $\mathbb{R}^d$ under the cosine distance, outputs a $(1+\epsilon)$-approximation to robust $F_0(S)$ with probability 0.9, using $O(\log m/\epsilon^2)$ words of space and amortized $O(d \log m)$ processing time per item.*

At the end of this section, we would like to remark that not all LSH functions can be used as smart hash functions with a small image distance. We delay the details of this remark to Appendix B.3 due to the space constraints.

5. EXPERIMENTS

We will refer to our algorithm as `Sketch` in this section. Our theoretical analysis has shown that `Sketch` has good accuracy guarantees, and is both space and time efficient. We now show how these theoretical expectations interact with practice.

We note that our theoretical analysis focuses on a fixed, known α (i.e. group diameter). We explore in this section how one can deal with unknown α, and propose a heuristic that reliably complements `Sketch` under realistic circumstances.

5.1 The Setup

Datasets. We derive our test data from a set of images, denoted by `Images`, which contains 4 million images taken from ImageNet [11]. These images serve as the *ground truth*.

To study the scalability of `Sketch`. We take subsets `Images` with sizes $4,000,000$ (whole set), $512,600$, $100,000$ and $10,000$, denoted by `I4m`, `I500k`, `I100k` and `I10k` respectively. The sizes are chosen to differ by magnitudes, to contrast the effects of the resource usage.

For each image in `I500k`, `I100k` and `I10k`, we introduce an average of 100 perturbed duplicates, producing test datasets `I500k100x`, `I100k100x` and `I10k100x`, which contain approximately 50 million, 10 million and 1 million images, respectively. For `I500k`, we also generate another dataset `I500k10x`, where an average of 10 duplicates are generated for each ground truth image.

To test our algorithms on datasets with low duplication ratios, we introduce an average of 2 perturbed duplicates to `I4m`, `I500k`, with number of duplications for each image following the *power-law* distribution, this gives datasets `I4m2x` and `I500k2x`.

Finally, we map each image to a signature in a feature space. Each image is mapped to a point, i.e. a *signature* of the image, in $\mathbb{R}^5$, $\mathbb{R}^{10}$ or $\mathbb{R}^{20}$. Our final test datasets are

- `I500k100x5d`, `I100k100x5d`, `I10k100x5d` for basic tests.
- `I4m2x5d`; `I500k2x5d` for low duplication ratio tests.
- `I4m2x10d`; `I4m2x20d` for high dimensionality tests.

The duplicates are generated through resizing images in the ground truth set; each image is randomly resized to between 300 and 500 in width and/or length. Resizing introduces interpolation errors, which are usually small but can sometimes be large, serving as a realistic source of noise.

To map an image to a point in $\mathbb{R}^d$, we first convert an image into grayscale, and then take the color histogram with d buckets as a d-dimensional vector, and rescale so that its ℓ_1 norm is $100,000$. The final rescaling effectively normalizes each signature to that of an image with $100,000$ pixels. We note that our datasets are not well-shaped (i.e., they have non-zero F_0-ambiguities), which is expected for real datasets.

Parameters and resources. We use the grid as our smart hash function, and the ℓ_2 norm as the distance function.

We use the distinct elements algorithm in [4] (for the noise-free data setting) to approximate the number of non-empty cells for `Sketch`. The storage space is fixed at 500 words, which is accurate enough for our purpose. This space cost is negligible compared with the other parts of our sketching algorithm, and thus is neglected for simplicity.

We run our experiments on a desktop PC with 8GB of RAM and a 4-core 3.40GHz Intel i7 CPU. Single-threaded running times are recorded.

5.2 Reference Algorithms

We compare `Sketch` with two alternative methods. The first one is `Baseline`, corresponding to a non-sampling version of our sketching algorithm (thus needs at least *linear* space in the worst case). In `Baseline`, we scan through points in S in an arbitrary order, during which we maintain a set $\hat{\Gamma}$ of balls with radius α, initialized to $\emptyset$. When a point p is not covered by the union of balls in $\hat{\Gamma}$, we add to $\hat{\Gamma}$ the ball with radius α centered at p. At the end, we output $|\hat{\Gamma}|$.

It is easy to see that `Baseline` stores up to F_0 balls, outputs $F_0(S, \alpha)$ if S is well-shaped, and otherwise outputs a value between $(1 - \delta(S))F_0$ and F_0 (recall that $\delta(S)$ is the F_0-ambiguity of the dataset S). `Baseline` is used to demonstrate the validity of our data model, measure the accuracy of `Sketch`, and serve as a baseline for the space usage. Note that in the worst case it will use a linear amount of space. In some sense it represents the clustering-based algorithms and the full space de-duplication algorithms (but can be done much faster in one scan).

The second one, `CellCount`, simply counts the number of non-empty cells in the grid. As the grid can be considered an LSH, this method corresponds to grouping elements simply by LSH. We show that `CellCount` is not suitable for estimating robust F_0 by contrasting it to `Sketch` and `Baseline`.

As mentioned in the related work, all the state-of-the-art streaming algorithms for distinct elements for noise-free datasets are just infeasible to handle noisy datasets – they will simply output values that are approximations to the total number of items instead of groups; for example, on `I100k100x5d` their outputs will be values close to 10 million, which is far-away from the ground truth $100,000$. We thus do not compare them in our experiments.

5.3 Finding Group Diameters

We first consider the case where α is not known a priori. Our idea is to run `Sketch` on parallel guesses α' for α, and make use of the assumption that the dataset has a good separation ratio.

Recall that once we fix S, $F_0(\alpha')$ decreases monotonically as α' increases. In particular, if $\alpha' < \alpha$, then $F_0(\alpha') \geq F_0(\alpha)$, as ground truth groups will be broken down into smaller groups; on the other hand if $\alpha' > \beta$, we will have $F_0(\alpha') \leq F_0(\alpha)$, because small groups may be merged into a few larger groups. However, if the dataset is (α, β)-sparse with a reasonably large separation ratio β/α, then there will be a range of values of $\alpha' \in [\alpha, \beta]$ for which $F_0(\alpha') = F_0(\alpha)$.

Of course, real-world datasets rarely have large well-defined separation ratios; it could be the case that some groups are too close to each other, though the overall distribution is spatially sparse. In the case when the separation ratio is not large, we can still expect there is a region where the change in $F_0(\alpha')$ is very small as we vary α', unless in extreme cases.

Suppose that our trials use $\alpha' = \alpha_0, \alpha_1, \ldots, \alpha_l$ where α_i's form a geometric sequence, finding the appropriate value for α reduces to searching for a region with a low gradient in

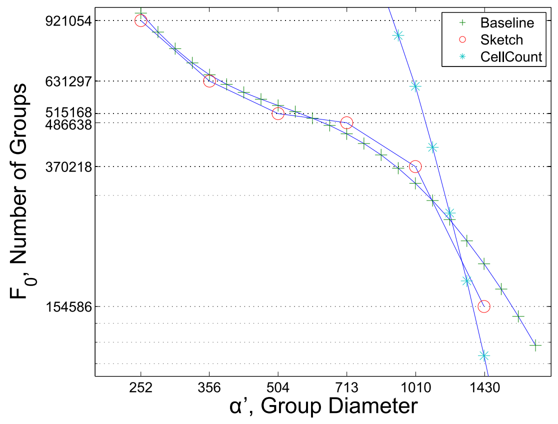

Figure 3: $F_0(\alpha')$ against α' on dataset `I500k100x5d`, with log-log plots.

the log-log plot of F_0 against α. Our heuristic simply finds an i ($0 < i \le l$) that minimizes the absolute value of the slope of the corresponding piece-wise linear plot, that is:

$$\left| \frac{\log F_0(\alpha_i) - \log F_0(\alpha_{i-1})}{\log \alpha_i - \log \alpha_{i-1}} \right| . \qquad (12)$$

This is equivalent to finding two adjacent values α' that make the least percentage change in $F(\alpha')$. We then take $\tilde{F}_0 = (F_0(\alpha_i) + F_0(\alpha_{i-1}))/2$ as the estimator for the true F_0. We call this the *saddle point heuristic*. We will see that, given an algorithm that computes $F_0(\alpha')$ accurately, this allows us to identify appropriate ranges for α' which is close to or contains α, and the corresponding $F_0 = F_0(\alpha)$.

For `Baseline`, to demonstrate that this heuristic is viable, we try values of α' in smaller steps in factors of $2^{\frac{1}{8}}$ to build a fine-grading graph. For `Sketch`, since we need to reduce the space usage, we have to narrow down the search range for a good α'. Our idea is to determine the smallest and largest values of α' that we are going to try, denoted by $\alpha_{\min}$ and $\alpha_{\max}$, which can be done by sampling a small number of labelled data points. We then step from $\alpha_{\min}$ to $\alpha_{\max}$ by increasing α' by factors of $\sqrt{2}$.

5.4 Results - Unknown Group Diameters

Figure 3 shows a log-log plot for `Baseline`, `Sketch` and `CellCount` on dataset `I500k100x5d`, with $F_0(\alpha')$ against α'.

First of all, it can be seen that `Sketch`, with larger steps, interpolates `Baseline` well.

For `Baseline`, one can see that the gradient decreases as one approaches from both sides near the interval $504 \le \alpha' \le 713$, where $F_0(\alpha')$ is very close to the ground truth. This is in accordance to the intuition of the heuristic. In particular, the segment $504 \le \alpha' \le 599$ minimizes the slope (Equation (12)). Similarly, for `Sketch`, the slope is the smallest for the segment $504 \le \alpha' \le 713$, and the F_0 values at the endpoints are both close to the ground truth 512,600.

It is easy to see that no such thing can be said for `CellCount`. The output is wildly high for lower cell lengths, rapidly drops to very low values in the graph, encompassing a large range of guesses at F_0 without hinting at any plausible way to detect the ground truth graphically. This suggests modelling robust F_0 directly using LSH is not a good idea.

	Full-info	`Baseline`	`Sketch`
Space (million points)	51	3.2	0.43
Average Error	–	4.8%	7.3%

Table 2: Unknown α; saddle point heuristic on 6 α' values for both `Baseline` and `Sketch`; over 10 runs.

Samples:	200	400	800	1,600
Space (pts):	1,500	3,000	6,000	12,000
`I500k100x5d`	27.3%	19.7%	11.6%	9.3%
`I500k10x5d`	25.8%	21.1%	10.4%	8.8%
`I100k100x5d`	22.6%	18.2%	8.9%	7.5%
`I10k100x5d`	17.1%	14.5%	8.2%	7.6%

Table 3: Average error over 20 runs; single sketch; known α. *Samples* refers to number of sampled buckets; *space* denotes the total number of points stored.

We note that the cost for the unknown α case is high even for a non-streaming algorithm, since we still have to guess different α' values in parallel. In Figure 3, we used small steps in `Baseline` to figure out the best α, during which we also had to allocate space for each run of `Baseline`.

We compare `Sketch` to two alternatives: one is `Baseline` but we use the same 6 α' values as that for `Sketch`; the other is simply storing the entire dataset and process offline. For `Baseline` we randomize the order of points in the data streams, so the answer may vary due to the F_0-ambiguity.

Table 2 summarizes the space usage and accuracy. `Sketch` uses 6 sketches each with a space bound of 12,000 points, and 6 values of α' are tried in parallel. The total space is thus 432,000 (points). The corresponding space usage for `Baseline` is 3,200,000. The space saving of `Sketch` over `Baseline` is about a magnitude, at a small cost of accuracy. We expect the space savings to grow as the size of the dataset grows further.

5.5 Results - Known Group Diameters

We next consider the case where α is known, e.g. using domain knowledge.

Table 3 shows average error percentages over 20 trials for datasets `I500k100x5d`, `I500k10x5d`, `I100k100x5d` and `I10k100x5d`, run against different space allowances (in terms of number of points stored), using α' determined by the saddle point heuristic. Table 4 summarizes the error percentages by taking the median of outputs of 6 parallel sketches, again average over 20 trials. All results are compared with the corresponding exact values $F_0(\alpha')$ outputted by `Baseline`.

We notice that the error percentages correlate relatively weakly with the numbers of duplicates and the numbers of groups. In contrast to magnitude differences in data size, the error percentages display much smaller changes.

We next look at the effects of using different numbers of duplicates. The observation is that the datasets `I500k10x5d` and `I500k100x5d` do not require significantly different space budgets to achieve similar accuracies, despite the magnitude difference in the number of duplicates. This shows that the choice of the length of our grid is good at limiting the number of cells that a group can intersect, and hence bounding the variance of the estimator.

Table 5 summarizes the average processing times for various datasets, for `Sketch` under different space budgets, and

Space (pts)	9,000	18,000	36,000	72,000
I500k100x5d	22.8%	10.6%	8.3%	6.6%
I500k10x5d	15.8%	9.2%	6.7%	5.7%
I100k100x5d	12.7%	8.4%	5.5%	3.5%
I10k100x5d	10.8%	8.8%	3.6%	3.0%

Table 4: Average error over 20 runs; median output of 6 sketches; known α.

Samples:	200	400	800	1,600	
Space (pts):	1,500	3,000	6,000	12,000	Baseline
I500k100x5d	0.45	0.47	0.49	0.46	1.45
I500k10x5d	0.42	0.50	0.52	0.46	1.42
I100k100x5d	0.48	0.44	0.48	0.53	1.38
I10k100x5d	0.42	0.48	0.51	0.50	1.35

Table 5: Average processing time (seconds) per $10,000$ pts

for Baseline. We observe that in Sketch the average time to process each point is almost a constant, which is consistent with our theoretical analysis. Baseline is a bit slower, but the time is still linear in the number of points.

5.6 Datasets with Low Duplication Ratios and/or High Dimensionality

In fact, in all of our experiments above, for the sake of a "stress test", the ambiguity we injected into the raw dataset Images is on the high end – the average number of duplicates we generated is between 10 to 100, while for many real-world datasets this ratio can be below 2. For example, the duplication ratios of the document datasets in the Internet studied by Broder [7] are between 1.4 to 1.8. Similar phenomenon has been observed by Wang et al. [29] on Internet images.

We thus also test our algorithms in datasets with low duplication ratios. In datasets I500k2x5d and I4m2x5d, the number of duplications of images follow a *power-law* distribution scaled to an expectation of 2 (to better model the real-world image duplications).

Table 6 shows that under the same ground truth $F_0 = 512,600$, the performance of Sketch becomes better when the duplication ratio becomes smaller. The error does not increase much when we increase the robust F_0 to 4 million while keeping the same sketch size.

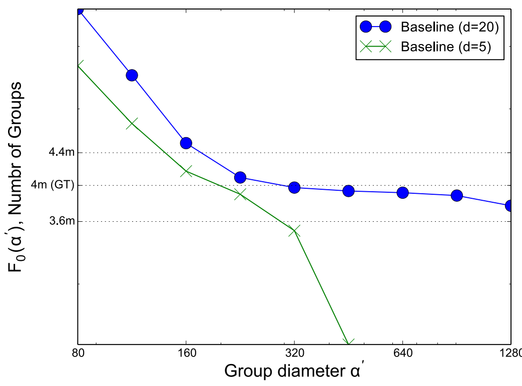

Figure 4: $F_0(\alpha')$ against α' on dataset I4m2x20d and I4m2x5d, with log-log plots. GT stands for *ground truth*.

No. pts	9,000	18,000	36,000	72,000
I500k100x5d	22.8%	10.6%	8.3%	6.6%
I500k10x5d	15.8%	9.2%	6.7%	5.7%
I500k2x5d	5.2%	3.0%	2.8%	2.2%
I4m2x5d	6.0%	3.5%	3.3%	2.4%

Table 6: Vary duplication ratio; average error over 20 runs; median output of 6 sketches; known α.

No. pts	9,000	18,000	36,000	72,000	144,000
I4m2x5d	6.0%	3.5%	3.3%	2.4%	1.7%
I4m2x10d	5.8%	4.2%	3.4%	2.6%	1.5%
I4m2x20d	6.4%	4.4%	3.6%	2.0%	1.3%

Table 7: Vary dimensionality; average error over 20 runs; median output of 6 sketches; known α.

Finally we test how the dimensionality of the feature space affects the performance of Sketch. Table 7 shows the performance of Sketch when the dimensionality varies. One can see that under the same number of sampled points, the performance of Sketch is roughly the same on the three datasets. The reason is that when mapping images to points in $\mathbb{R}^5$, the dataset I4m2x5d has already exhibit a good separation ratio; see the α-curve for I4m2x5d in Figure 4. Further increasing the dimensionality indeed gives better separation ratio; see the α-curve for I4m2x20d in Figure 4. However, it does not give much advantage in terms of accuracy. The main reason is that the variance generated in the sampling step of Sketch is larger in higher dimensions. On the other hand, points in higher dimensions cost more storage space and processing time (note that datasets in Table 7 are compared under the same number of samples points, not the total space). To conclude, our experiments suggest that we should use the least necessary number of features that exhibit a good separation ratio.

6. CONCLUSION

In this paper we present a set of streaming algorithms for computing the number of distinct elements for noisy datasets. We have shown that our algorithm for well-shaped noisy datasets in constant dimensional Euclidean spaces is theoretically optimal. We have also rigorously proved that our algorithms still work well on datasets with small F_0-ambiguities, and proposed a method to detect datasets with high F_0-ambiguities. Furthermore, we have proposed an algorithm framework for datasets in a general metric space using the theory of locality sensitive hashing. The effectiveness of our algorithms have been verified experimentally on image data (or rather, point sets in the Euclidean space). Future work includes designing algorithms for computing robust F_0 for datasets in high dimensions with provable guarantees and extending our experiments to other metrics and similarity functions.

7. REFERENCES

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD*, pages 574–576, 1999.

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

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Comm. of the ACM*, 51(1):117, 2008.

[4] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[5] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 199–210. ACM, 2007.

[6] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[7] A. Z. Broder. Identifying and filtering near-duplicate documents. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, COM '00, pages 1–10, London, UK, UK, 2000. Springer-Verlag.

[8] P. Brown, P. J. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, and Y. Sismanis. Toward automated large-scale information integration and discovery. In *Data Management in a Connected World*, pages 161–180, 2005.

[9] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[10] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, pages 253–262, 2004.

[11] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255, 2009.

[12] X. L. Dong and F. Naumann. Data fusion: resolving data conflicts for integration. *Proceedings of the VLDB Endowment*, 2(2):1654–1655, 2009.

[13] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *Algorithms-ESA 2003*, pages 605–617. Springer, 2003.

[14] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[15] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, Oct. 2006.

[16] J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2004.

[17] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *DMTCS Proceedings*, (1), 2008.

[18] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[19] S. Ganguly. Counting distinct items over update streams. *Theoretical Computer Science*, 378(3):211–222, 2007.

[20] S. Guha. Tight results for clustering and summarizing data streams. In *ICDT*, pages 268–275, 2009.

[21] T. N. Herzog, F. J. Scheuren, and W. E. Winkler. *Data quality and record linkage techniques*, volume 1. Springer, 2007.

[22] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[23] P. Indyk and D. P. Woodruff. Tight lower bounds for the distinct elements problem. In *FOCS*, pages 283–288, 2003.

[24] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS*, pages 41–52, 2010.

[25] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, pages 802–803. ACM, 2006.

[26] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. Anf: a fast and scalable tool for data mining in massive graphs. In *SIGKDD*, pages 81–90, 2002.

[27] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.

[28] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[29] X. Wang, L. Zhang, and C. Liu. Duplicate discovery on 2 billion internet images. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2013, Portland, OR, USA, June 23-28, 2013*, pages 429–436, 2013.

[30] Q. Zhang. Communication-efficient computation on distributed noisy datasets. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 313–322, 2015.

APPENDIX

A. PROOF OF THEOREM 3

In this section we prove Theorem 3. We first consider the 2-dimensional Euclidean space.

Given a dataset S with F_0-ambiguity δ, let $n_1 = F_0(S)$ and $n_2 = F_0(S \backslash T)$ where $T \subseteq S$ is the corresponding subset such that $S \backslash T$ is well-shaped. By the monotonicity of the robust $F_0(\cdot)$ function and the second item of Definition 4,

$$(1-\delta)n_1 \leq n_2 \leq n_1. \qquad (13)$$

Let $\mathcal{G} = \{G_1, \ldots, G_{n_1}\}$ be the minimum-cardinality partition of S. We will show that if we run Algorithm 4 (call it $\mathcal{A}$) on S, then with a good probability the following holds:

$$(1 - O(\delta + \epsilon))n_1 \leq \mathcal{A}(S) \leq (1 + O(\delta + \epsilon))n_1. \qquad (14)$$

In other words, $\mathcal{A}(S)$ produces a $(1+O(\delta+\epsilon))$-approximation to $F_0(S)$.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the sets of non-empty cells corresponding to S and $S \backslash T$ respectively. Obviously $\mathcal{C}_2 \subseteq \mathcal{C}_1$. By the second item of Definition 4, and the fact that a group intersects at most 9 grid cells under the settings in Section 2.1, we have

$$|\mathcal{C}_1 \backslash \mathcal{C}_2| \le 9\delta\, |\mathcal{C}_1|, \tag{15}$$

We now run $\mathcal{A}$ on S, and let $\mathcal{C}'_1 \subseteq \mathcal{C}_1$ be the whole set of sampled items. Let $\mathcal{C}'_2 \subseteq \mathcal{C}'_1$ be the set of cells sampled from $\mathcal{C}_2$. Since $\mathcal{C}'_1$ is sampled uniformly at random from $\mathcal{C}_1$, by (15) and a Chernoff bound, we have with probability $1-o(1)$ that

$$\left|\mathcal{C}'_1 \backslash \mathcal{C}'_2\right| \le 10\delta \left|\mathcal{C}'_1\right|. \tag{16}$$

Moreover, observe that

$$\forall C \in \mathcal{C}_1, w(C) = \Theta(1). \tag{17}$$

This follows from the simple geometry fact that each cell can intersect at most $O(1)$ unit-diameter groups under the condition that $F_0(S)$ is defined on the group partition with the minimum size, since otherwise some groups can be merged to produce a group partition of a smaller size.

By (15), (16) and (17), we have with probability $1-o(1)$,

$$\begin{aligned} & (1-O(\delta))\frac{|\mathcal{C}_1|}{|\mathcal{C}'_1|} \cdot \sum_{C \in \mathcal{C}'_1} w(C) \\ \le \quad & \frac{|\mathcal{C}_2|}{|\mathcal{C}'_2|} \cdot \sum_{C \in \mathcal{C}'_2} w(C) \\ \le \quad & (1+O(\delta))\frac{|\mathcal{C}_1|}{|\mathcal{C}'_1|} \cdot \sum_{C \in \mathcal{C}'_1} w(C). \end{aligned} \tag{18}$$

The idea to finish the proof is the following: Suppose we run Algorithm (4) on $S \backslash T$ only, since $\mathcal{C}'_2$ is a uniformly random sample on $\mathcal{C}_2$, we know by the analysis in Section 2.1 that with probability 0.96,

$$\frac{|\mathcal{C}_2|}{|\mathcal{C}'_2|} \sum_{C \in \mathcal{C}'_2} w(C) \in [(1-\epsilon)n_2, (1+\epsilon)n_2]. \tag{19}$$

And then by (13), (18) and (19), it holds with probability $1-o(1)-0.04 > 0.95$ that

$$(1-O(\delta+\epsilon))n_1 \le \frac{|\mathcal{C}_1|}{|\mathcal{C}'_1|} \sum_{C \in \mathcal{C}'_1} w(C) \le (1+O(\delta+\epsilon))n_1, \tag{20}$$

proving (14).

The cavity of the above argument is that (19) does not necessarily hold since points from T may interfere the value of $w(C)$ for a cell $C \in \mathcal{C}'_2$. Fortunately, we can show that such an interference is negligible.

For a cell $C \in \mathcal{C}_2$, we say it is *contaminated* if there exists a cell $C' \in \mathcal{C}_1 \backslash \mathcal{C}_2$ and a group $G \in \mathcal{G}$ such that both C and C' intersect with G. Let $\mathcal{C}_3 \subseteq \mathcal{C}_2$ denote the set of contaminated cells, and let $\mathcal{C}'_3 \subseteq \mathcal{C}'_2$ be the set of sampled contaminated cells. Note that $w(C)$ will only decrease if cell C is contaminated. Now observe that each cell in $\mathcal{C}_1 \backslash \mathcal{C}_2$ can contaminate at most $O(1)$ cells in $\mathcal{C}_2$ in the Euclidean plane. Together with (16) and a Chernoff bound, we have with probability $1-o(1)$ that

$$\left|\mathcal{C}'_3\right| \le O(\delta) \cdot \left|\mathcal{C}'_2\right|. \tag{21}$$

Therefore by (17), (21) and the analysis in Section 2.1, it holds with probability $1-o(1)-0.04$ that,

$$\begin{aligned} \frac{|\mathcal{C}_2|}{|\mathcal{C}'_2|} \sum_{C \in \mathcal{C}'_2} w(C) \quad & \in \quad \left[(1-\epsilon)n_2\left(1-O\left(\frac{|\mathcal{C}'_3|}{|\mathcal{C}'_2|}\right)\right), (1+\epsilon)n_2\right] \\ & \subseteq \quad [(1-O(\epsilon+\delta))n_2, (1+\epsilon)n_2]. \end{aligned} \tag{22}$$

Finally replacing (19) with (22), we can show that (20) still holds, and thus (14) also holds. Therefore, the probability that (14) holds is $1-o(1)-o(1)-0.04 > 0.95$, where the two $o(1)$ error terms are introduced by Chernoff bounds (for deducting (16) and (21)), and the 0.04 error term is introduced by using the analysis in Section 2.1.

The above analysis can be literally carried to any constant dimensional Euclidean space (one only needs to slightly adjust some constant parameters).

B. OTHER MISSING PROOFS

B.1 Proof for Lemma 2

Proof: First, note that if $\beta \ge d^{3/2}$, then each cell can intersect at most one group. Thus $|\mathcal{C}| \ge F_0$.

Second, since the grid $\mathbb{G}$ is random and each group has diameter at most 1, the probability that the convex hull $\mathsf{CH}(G)$ of a group of points G is *cut* by the boundaries of grid cells in each dimension is at most $\mu = 1/d$. If $\mathsf{CH}(G)$ is cut by grid boundaries in i dimensions, then G intersects (at most) 2^i grid cells. Therefore the expectation of the number of cells each group G intersects, denoted by N_G, can be bounded by

$$\begin{aligned} \mathbf{E}[N_G] \quad & \le \quad \sum_{i=0}^{d} 2^i \binom{d}{i} \mu^i (1-\mu)^{d-i} \\ & = \quad (2\mu + (1-\mu))^d \\ & = \quad (1+1/d)^d < 3. \end{aligned} \tag{23}$$

By the linearty of expectation, $\mathbf{E}[\mathcal{C}] \le \sum_{G \in \mathcal{G}} \mathbf{E}[N_G] \le 3F_0$. Therefore $|\mathcal{C}| \le 300F_0$ with probability at least 0.99 by a Markov inequality.

□

B.2 Proof for Lemma 3

We investigate the two items in the definition of a smart hash function (Definition 5). Let f be a random hash function sampled from $\mathcal{F}$. For each pair $x, y \in S$ that are in different groups, the probability that $f(x) \ne f(y)$ is at least $1-p_2^k$ by the definitions of LSH and k-fold hash function. By a union bound, with probability $1-m^2 p_2^k$, we have for all pairs $x, y \in S$ that are in different groups, $f(x) \ne f(y)$.

Let $f(x) = (h_1(x), \ldots, h_k(x))$ where $h_i \in_r \mathcal{H}$. We next consider each group $G \in \mathcal{G}$. We say G is *cut* by h_i if there exist two items $x \in G$ and $y \in U$ such that $d(x,y) \le \alpha$ but $h_i(x) \ne h_i(y)$. Thus if G is cut by j hash functions, then the cardinality of the $\{f(x) \mid \exists y \in G \ s.t.\ d(x,y) \le \alpha\}$ can be bounded by η^j since $\mathcal{H}$ is η-concentrated. The probability that G is cut by h_i can be bounded by p_1 since $\mathcal{H}$ is $(2\alpha, \beta, p_1, p_2)$-sensitive and Diameter$(G) \le \alpha$ (S is (α, β)-sparse). We thus can upper bound

$$\mathbf{E}[\rho] \le \sum_{j=0}^{k} \eta^j \binom{k}{j} (1-p_1)^j p_1^{k-j},$$

which is $(\eta(1-p_1)+p_1)^k$. By a Markov inequality $\rho \le 100(\eta(1-p_1)+p_1)^k$ with probability 0.99.

B.3 A Remark on LSH and Smart Hash Function

We note that not all LSH functions can be used as smart hash functions with a small image distance. We use the MinHash [6] for example.

Given a set $A = \{a_1, \ldots, a_t\} \subseteq U$, the MinHash is defined to be

$$h(A) = \min\{g(a) \mid a \in A\},$$

where $g : U \to \mathbb{N}$ is a random hash function. MinHash is an $(\alpha, \beta, 1-\alpha, 1-\beta)$-sensitive LSH for the Jaccard distance function $J(\cdot, \cdot)$, which is defined as follows: Given two sets $A, B \subseteq U$,

$$\mathrm{JDist}(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}.$$

For any $x \in U \backslash A$, let $A_x = A \cup \{x\}$. It is easy to see that even for the small threshold $\alpha = 1/(t+1)$, A_x $(x \in U \backslash A)$ are in the same group, but $|h(A_\ell) \mid \ell \in [t]| \approx |U|/t$ which could be very large.

On the other hand, in practice, when given any query point, one often chooses to only search a much smaller number of buckets than $|U|/t$ for near neighbors, giving a *de facto* bound on the 'image distance', and the techniques in this paper may still be applied.

C. MATHEMATICAL TOOLS

We will need the following mathematical tools.

LEMMA 4 (MARKOV INEQUALITY). *Let $X \geq 0$ be a random variable. Then for all $a > 0$, $\mathbf{Pr}[X \geq a] \leq \frac{\mathbf{E}[X]}{a}$.*

LEMMA 5 (CHEBYSHEV'S INEQUALITY). *Let $X \geq 0$ be a random variable. Then for all $a > 0$,*

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq a] \leq \frac{\mathbf{Var}[X]}{a^2}.$$

LEMMA 6 (CHERNOFF BOUND). *Let $X_1, \ldots, X_k \in [0, 1]$ be independent random variables, and let $X = \sum_{i=1}^k X_i$.*

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq \epsilon \mathbf{E}[X]] \leq 2e^{-\frac{\epsilon^2 \mathbf{E}[X]}{3}}.$$