Fast Sketch-based Recovery of Correlation Outliers*

Graham Cormode[1] and Jacques Dark[2]

1 **Department of Computer Science, University of Warwick, Coventry, UK**
g.cormode@warwick.ac.uk
2 **Department of Computer Science, University of Warwick, Coventry, UK**
j.dark@warwick.ac.uk

Abstract

Many data sources can be interpreted as time-series, and a key problem is to identify which pairs out of a large collection of signals are highly correlated. We expect that there will be few, large, interesting correlations, while most signal pairs do not have any strong correlation. We abstract this as the problem of identifying the highly correlated pairs in a collection of n mostly pairwise uncorrelated random variables, where observations of the variables arrives as a stream. Dimensionality reduction can remove dependence on the number of observations, but further techniques are required to tame the quadratic (in n) cost of a search through all possible pairs.

We develop a new algorithm for rapidly finding large correlations based on sketch techniques with an added twist: we quickly generate sketches of random combinations of signals, and use these in concert with ideas from coding theory to decode the identity of correlated pairs. We prove correctness and compare performance and effectiveness with the best LSH (locality sensitive hashing) based approach.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems

Keywords and phrases correlation, sketching, streaming, dimensionality reduction

Digital Object Identifier 10.4230/LIPIcs.ICDT.2018.13

1 Introduction

One of the most basic tasks in data analysis is to identify correlations between data sources, modeled as random variables. Discovered correlations are used to remove unnecessary features, to build predictive models, and to identify unexpected behaviors and dependencies. In this paper, we consider the most common measure of correlation: the Pearson product-moment correlation coefficient, which describes the linear relationship between a pair of random variables. This measure is simple to state and interpret: it is computed as the (sample) covariance of the two variables, divided by the product of the corresponding standard deviations. It ranges from -1 (strong negative correlation) through 0 (no correlation) to $+1$ (strong positive correlation). Hence, we are typically interested only in attribute pairs with correlation close to 1 in (absolute) magnitude.

For large numbers of variables, it can quickly become infeasible to compute the correlations of all of the quadratically many pairs. However, our observation is that most correlations are uninteresting: for many kinds of data, we expect that most pairs of variables would *not*

* The work of GC is supported by European Research Council grant ERC-2014-CoG 647557 and a Royal Society Wolfson Research Merit Award; JD is supported by a Microsoft Research PhD Scholarship (MRL 2014-038).

display any (strong) correlation. For example, if we consider the activity profiles of users of a large web service, then we do not expect many pairs to be strongly correlated (there may be weak correlations due to similar time-of-day and day-of-week behavior)—any strong correlation between a pair would be unusual, indicating potentially nefarious activity worthy of further investigation. We model this by assuming that the number of correlated pairs is asymptotically smaller than the quadratically many possible pairs.

With this in mind, we can ask the following questions: given a stream of observation data, can we identify all correlation outliers (unusually large correlation coefficients, defined by being greater in magnitude than some parameter ϕ) with query time cost sub-quadratic in the number of variables and sub-linear in the number of observations?

This can be accomplished using a combination of a Fast Johnson-Lindenstrauss Transform (FJLT, to compress the rows of the input matrix) and Locality Sensitive Hashing (LSH, to efficiently find the outlier pairs). However, for small ϕ, the query time of this strategy looks like $n^{2-\Theta(\phi)}$, even as we shrink all the non-outlier correlations down to 0. Valiant [19] showed how to improve this to $n^{2-\Theta(\sqrt{\phi})}$, but we would like to remove the dependence on ϕ from the exponent of n.

This paper describes an algorithm which takes sketches of the rows and uses fast matrix multiplication to quickly transform them into an approximation of a sketch of the correlation matrix. We then remove the 1's along the diagonal, and use a heavy hitters recovery technique to pull out the outliers.

We then provide analysis for this algorithm, showing that for constant Frobenius norm[1] of the non-outlier non-diagonal correlations, this query process can be performed in time[2] $\tilde{O}(\phi^{-2}n^{5/3})$, assymptotically better than LSH for small enough ϕ. However, this comes at the cost of requiring much larger sketches of the input matrix rows.

2 Preliminaries

2.1 Models

We treat the observation data as defining an $n \times p$ matrix of reals, $\mathbf{M}$. Here, n denotes the number of attributes, while p indexes the different observations. Hence, each of the p columns represents an independent observation of some n-dimensional random variable. We label the columns (observations) as $\mathbf{x^{(i)}}$ for $i \in [p]$. For this data, we can apply standard definitions of covariance and correlation.

▶ **Definition 1.** Recall that:

- The *sample mean* is given by $\overline{\mathbf{x}} = \frac{1}{p}\sum_{i=1}^{p}\mathbf{x^{(i)}}$.
- The *sample covariance* is given by $\mathbf{V} = \frac{1}{p-1}(\mathbf{M} - \overline{\mathbf{x}}\mathbf{e}^T)(\mathbf{M} - \overline{\mathbf{x}}\mathbf{e}^T)^T$, where $\mathbf{e}$ is the p-dimensional vector with entries all ones.
- The *sample correlation* is given by $\mathbf{C} = \mathbf{\Sigma}^{-\frac{1}{2}}\mathbf{V}\mathbf{\Sigma}^{-\frac{1}{2}}$, where $\mathbf{\Sigma}$ is the diagonal matrix consisting of the diagonal entries of $\mathbf{V}$.

Essentially, the covariance is found by shifting the rows of $\mathbf{M}$ to have mean 0 and then taking inner products between them normalized by a factor of $\frac{1}{p-1}$. The definition of the correlations is similar, but further normalized by diving out the standard deviations.

---

[1] For matrix $\mathbf{A}$, the Frobenius norm is $\|\mathbf{A}\|_F = (\sum_{i,j}\mathbf{A}_{i,j}^2)^{\frac{1}{2}}$.

[2] Using the convention that $\tilde{O}(\cdot)$ is the $O(\cdot)$ cost with log factors suppressed.

It will be useful for our analysis to have notations for the rows of $\mathbf{M}$ with the shift normalization applied.

▶ **Definition 2.** For each row vector $\mathbf{y^{(i)}}$:

- Let the standardized row vector $\hat{\mathbf{y}}^{(\mathbf{i})}$ be given by $\hat{\mathbf{y}}^{(\mathbf{i})} = \frac{\hat{\mathbf{y}}^{(\mathbf{i})} - \bar{\mathbf{x}}_\mathbf{i}\mathbf{e}^T}{\|\hat{\mathbf{y}}^{(\mathbf{i})} - \bar{\mathbf{x}}_\mathbf{i}\mathbf{e}^T\|_2}$.

The observation matrix $\mathbf{M}$ is input as a stream of m updates $\langle u_1, u_2, \cdots u_m \rangle$ arriving one at a time. Starting from the zero matrix $\mathbf{M^{(0)}} = \mathbf{0}$, each update u_s describes a change to be made to $\mathbf{M^{(s-1)}}$ in order to determine $\mathbf{M^{(s)}}$. By the end of the stream, we have $\mathbf{M^{(m)}} = \mathbf{M}$. The format of the updates depends on the exact choice of stream model—we will consider three variants: row-wise permutation, column-wise permutation, and turnstile.

Row-Wise Permutation Stream (RPS) In this model, the updates are simply a list of the entries of $\mathbf{M}$, one row at a time. With each step from $\mathbf{M^{(s-1)}}$ to $\mathbf{M^{(s)}}$, one entry is changed from 0 to its final value. Entries in the same row arrive contiguously, so each row is filled out one after the other. Without loss of generality, we can assume that rows arrive in index order, so that $\mathbf{M}_{i,j} \leftarrow u_{(i-1)p+j}$ Since each entry is set exactly once, the stream has length $m = np$. The arrival of each new row corresponds to adding a new attribute to the data set.

Column-Wise Permutation Stream (CPS) This model works the same as the row-wise version, but with entries arriving as contiguous columns. Again, $m = np$ but now $\mathbf{M}_{i,j} \leftarrow u_{(j-1)n+i}$. The arrival of a new column corresponds to adding a new observation (e.g. from a new time step).

Turnstile Stream (TS) The turnstile model is the most general that we consider. Here updates are of the form $u_t = (\alpha, i, j)$ indicating that the $(i, j)^{\text{th}}$ entry should be incremented by $\alpha \in \mathbb{R}$. That is, $\mathbf{M}^{(s)}_{i,j} \leftarrow \mathbf{M}^{(s-1)}_{i,j} + \alpha$, while all other entries remain the same. Changes happen in any order, and entries can change any number of times as long as the correct state is reached by the end of the stream. Hence, the stream length m is arbitrary.

Both RPS and CPS are then special cases of this model. TS represents the situation where each of the observed values needs to be aggregated from a variety of sources. For example: suppose the entries in our observation matrix represent the number of requests for a specific resource (indexed by rows) at a specific site (indexed by columns) in a distributed system. Then, the number of requests at each node will need to be accumulated to produce the actual observation data.

2.2 Problem Statement

In what we term the *correlation outliers* problem, we are given a stream describing $\mathbf{M}$ (according to one of the three models), and three parameters: k, ϕ, and R. We make use of the following concepts:

▶ **Definition 3.** For the sample correlation matrix $\mathbf{C}$ (of $\mathbf{M}$):

- Let $\textsc{Large}_\phi \subset [n]^2$ refer to the set of index pairs of off-diagonal entries of $\mathbf{C}$ which have magnitude at least ϕ.
- Let $\mathbf{C}_{-k}$ refer to the matrix obtained by taking $\mathbf{C}$, removing all the diagonal entries, and removing the k largest magnitude off-diagonal entries (replacing them with 0's).

The problem is then to maintain a summary of the stream so that all index pairs contained in $\textsc{Large}_\phi$ can be retrieved with high probability ($o(\frac{1}{n})$ chance of failure), provided that

$|\textsc{Large}_\phi| \leq k$ and $\|\mathbf{C}_{-k}\|_F \leq R$. Since the full input can be trivially maintained in $O(np)$ space, we seek solutions with space cost that is $o(np)$. Further, the summary should be quick to update (taking polylogarithmic time) and, at the end of the stream, the query routine should run in time $o(n^2)$.

Parameter Regimes of Interest We argue that the assumption that k, the number of highly correlated pairs, is $o(n^2)$ is a reasonable one: otherwise, simply reporting all the correlated pairs would take quadratic time, and naive exhaustive solutions would suffice. Prior work has made various assumptions to limit the scale and quantity of the correlations. In particular, Valiant's 'light bulb problem' [20] considers the case when all vectors are chosen uniformly at random from the Boolean hypercube, except for one correlated pair. In this setting, the expected correlation of the uncorrelated pairs is 0, and the observed values are bounded by $O(p^{-1/2})$.

Our problem description similarly models the underlying correlation matrix as having k "large" pairs with correlation magnitude $\geq \phi$ and all other (off-diagonal) correlations have a Frobenius weight of at most R. Our results provide the most interesting bounds when we take R to be polynomially small as a function of the number of vectors n. We argue that this is consistent with analogous problems, such as in compressed sensing, sparse Fourier transform, and coding theory. In these settings, it is common to study the case when the target vector is sparse, i.e. outside of the k non-zero values, the data is exactly zero [17]; noisy with zero mean [5]; or asymptotically decaying polynomially [8] All of these cases fall within our model of constant R as long as p is some moderately high degree polynomial of n.

2.3 Our Contributions

We describe an algorithm which answers the correlation outliers problem in the turnstile streaming model. We analyze its space and time costs, and show that they meet the desiderata above. Our algorithm stores a separate sketch of each row of $\mathbf{M}$ (described in Section 2.5). Comparing these directly would still take time $\Theta(n^2)$ to perform an all-pairs comparison. Instead, we achieve an improved query time with the following three ideas:

- By randomly assigning variables into Π groups, and linearly combining the (sketched) information of all variables in the same group we can go from having to consider n^2 pairs of variables to Π^2 pairs of groups. This can be seen as a second level of sketching. This has previously been used in the offline setting for Valiant's Boolean correlation outliers algorithm [19].
- Error correcting codes are composed with the grouping step (including/excluding variables from groups based on code bits) in a way that allows us to recover the identities of large entries in a particular group pair using the decoder. This has previously been used for heavy hitters problems on sketches (see: [9, 16, 14]).
- Fast matrix multiplication algorithms allow us to quickly generate batches of sketch estimates of inner products. This speeds up the evaluation of the inner products between pairs of groups. Then checking whether the results of these computations exceeds a given threshold produces the strings of bits for the decoder.

Several other offline algorithms can be similarly implemented under this sketch-and-search strategy, notably Locality Sensitive Hashing [10] and the approximate Closest Pair algorithm of Valiant [19]. We compare these with out method below.

We note that the LSH and Closest Pair algorithm consider slightly different input assumptions from us which makes the comparison a little tricky. While in this paper we

consider a bound R on the total Frobenius weight of the non-outlier pairs, these other algorithms consider an input with non-outlier pairs having a correlation magnitude smaller than a flat threshold ϕ_1.

Since we are motivated by the case when R is constant, we include in the table the costs for these other two algorithms in the case when ϕ_1 tends to 0. For details of this as well as minor modifications for the streaming setting see Section 2.4.

Our main result is stated in full in Theorem 20. Using $\theta = 2/3$ we get:

Technique	Models	Sketch Size	Query Space	Query Time
Full Search	All	$\tilde{O}(\phi^{-2}n)$	$\tilde{O}(\phi^{-2}n)$	$\tilde{O}(\phi^{-2}n^2)$
LSH	All	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(kn^{2-\Theta(\phi)})$
Valiant	All	$\tilde{O}(n)$	$\tilde{O}(n^{2-\Theta(\sqrt{\phi})})$	$\tilde{O}(kn^{2-\Theta(\sqrt{\phi})})$
Our Approach	All	$\tilde{O}(n^{5/3}(k^2 + \frac{R^2}{\phi^2}))$	$\tilde{O}(n^{5/3}(k^2 + \frac{R^2}{\phi^2}))$	$\tilde{O}(\phi^{-2}n^{5/3}(k^2 + \frac{R^2}{\phi^2}))$

This space usage is $o(np)$ and subquadratic in n for $p \in \Omega(n^{2/3+\epsilon})$ and constant k, R, ϕ.

2.4 Related Work

Locality Sensitive Hashing. Asking for high correlation is equivalent to looking for small Euclidean distance between the standardized (normalized and centered) row vectors. A correlation of ϕ corresponds to a distance on the sphere of $\sqrt{2-2\phi}$. Hence, this problem can be solved using Euclidean Locality Sensitive Hashing (LSH). Negative correlation outlier pairs can be found by simply considering every row and its negation.

The LSH framework is parameterised by $c > 1$: the ratio $\frac{d_1}{d_2}$ between d_1, the smallest distance between "dissimilar" items and d_2, the largest distance between "similar" items. To compare the efficiency of different families of hash functions, we talk about their sensitivity ρ as a function of c. This is given by $\rho = \frac{\log 1/p_1}{\log 1/p_2}$ where p_1 and p_2 are the collision probabilities of the similar and dissimilar pairs respectively. The best known Euclidean LSH algorithms have $\rho = \frac{1}{c^2} + o(1)$ (data independent, [10]) and $\rho = \frac{1}{2c^2-1} + o(1)$ (data dependent, [4]).

For n input vectors of d-dimensions, the framework can be used to find all similar pairs with high probability in $\tilde{O}(nd)$ space and $\tilde{O}(n^{1+\rho}d)$ time. Notice that we can save space compared with the normal operation of the framework, since we can check for similar pairs and scrap the results of each hash function before moving on to the next. Assuming outlier correlations are greater than ϕ_0, and non-outlier correlations are smaller than ϕ_1, we can use a Fast Johnson-Lindenstrauss Transformation to compress input rows to length $O(\epsilon^{-2}\log n)$, distorting the pairwise distances by at most $(1 \pm \epsilon)$. Then we can do LSH with $c^2 = \left(\frac{1-\epsilon}{1+\epsilon}\right)^2 \left(\frac{1-\phi_0}{1-\phi_1}\right)$. This gives us a space cost of $\tilde{O}(n)$ and a time cost of $\tilde{O}(n^{2-\Theta(\phi_0)})$, even if ϵ goes to 1 and ϕ_1 goes to 0.

Compressed Matrix Multiplication. Pagh [16] considered the problem of efficiently computing sparse or approximate matrix products. The key idea is that by choosing a particular structure for the sketching functions, it is possible to quickly compute a sketch of the outer product $\mathbf{x}\mathbf{y}^T$, from sketches of $\mathbf{x}$ and $\mathbf{y}$, through the use of FFTs (in time $O(b \log b)$ for length b sketches). Since a matrix product $\mathbf{A}\mathbf{B}^T$ can be decomposed into a sum of such outer products between corresponding columns, this allows for efficient computation of matrix products from sketches of columns.

As the algorithm only requires access to matched columns of $\mathbf{A}$ and $\mathbf{B}$ one at a time, in the special case of $\mathbf{A} = \mathbf{B}$ this approach can be used in the CPS model to build a sketch of $\mathbf{A}\mathbf{A}^T$. In particular, we can build a sketch of the covariance matrix $\mathbf{V}$ in this streaming model, from input observation matrix $\mathbf{M}$, with update time cost $O(b \log b)$ ($O(1)$ amortized,

since n dominates $b \log b$) and space usage $O(b)$. To recover dominant entries from these sketches, Pagh describes an approach (building on [9, 16, 14]) that uses $O(\log^2 n)$ sketches of sub-matrices of $\mathbf{AB}$, along with error correcting codes, to discover the identity of a small number of entries which dominate the Frobenius norm of the product, with high probability. This process runs in $O(b \log^2 n)$ time and space. Putting these pieces together provides a solution to a *covariance outliers* version of our problem in the CPS model.

Unfortunately, this approach cannot be adapted directly to the *correlation outliers* problem. Large correlations between low variance signals would be drowned out by the contribution from high variance signals that are much more weakly correlated. To apply this technique, we would need to record the whole of $\mathbf{M}$ (perhaps feasible for small np), or perform two passes over the stream—using the first to determine the variances, and then using the covariance solution on the rescaled inputs with the second pass. Instead, we will adapt the recovery process to work on different kinds of sketches.

Boolean Vectors. Valiant [19] showed how to quickly find a single correlated pair among Boolean vectors in time $O(n^{\frac{5-\omega}{4-\omega}+\epsilon} + nd) \subset O(n^{1.62} + nd)$, where $\omega < 2.4$ is the exponent of matrix multiplication. They then used this along with embeddings into the Hamming space to develop a Euclidean space approximate closest pair algorithm capable of finding a $(1+\epsilon)$ approximation to the closest pair in time $O(n^{2-\Theta(\sqrt{\epsilon})})$.

This algorithm can be adapted to solve our streaming problem in a similar manner to LSH. We keep random hyperplane projections of each of the input vectors as sketches, and then using their signs for the algorithm at query time. This takes only $\tilde{O}(n)$ space and k repetitions can be used to find k outlier pairs. Again, using the model where outliers have correlation magnitude above ϕ_0 and non-outliers below ϕ_1, we can choose $\epsilon = \sqrt{\frac{1-\phi_1}{1-\phi_0}} - 1$. This looks like $\Theta(\phi_0)$ even as ϕ_1 goes to 0.

Karppa et al. [13] improved on this work, giving a faster algorithm for Boolean vector outlier pairs, tending towards $\tilde{O}(n^{\frac{2\omega}{3}}) \subset \tilde{O}(n^{1.6})$ as the non-outlier correlations tend to 0.

Our approach uses several of the same ideas, such as Cartesian grouping and signed aggregation of the rows, but as we are interested in a slightly different problem (with vanishing rather than fixed-small-threshold non-outliers) we do not need to rely on the nice concentration properties for Boolean vectors, and can apply these ideas directly on to the Euclidean vectors - producing a fast and simple algorithm.

2.5 Sketches of Vectors

Our results make use of *sketches* of vectors. These can be thought of as random projections from the original high-dimensional space down to a lower dimensional space, such that geometric properties of the vectors are (approximately) preserved. In particular, given vectors $\mathbf{x}$ and $\mathbf{y}$, sketches exist that can estimate:

$$\text{Squared Euclidean length } \|\mathbf{x}\|_2^2 \text{ up to error } \epsilon\|\mathbf{x}\|_2^2. \tag{1}$$

$$\text{Inner product } \langle \mathbf{x}, \mathbf{y} \rangle \text{ up to error } \epsilon\|\mathbf{x}\|_2\|\mathbf{y}\|_2. \tag{2}$$

Many results for such sketches are known, from the earliest (non-constructive) results based on the Johnson-Lindenstrauss lemma [11], the tug-of-war sketches due to Alon, Matthias, Szegedy and Gibbons [3, 2], and several more [1, 15, 12]. For concreteness, we will adopt the so-called (fast) AMS sketches (explained in [7]). These create a sketch of size $O(\epsilon^{-2} \log 1/\delta)$

so that any query obtains the above claimed ϵ guarantee with probability at least $1 - \delta$, where the probability is over the random choices used to determine the random projection.

The AMS sketching procedure maps (linearly and randomly) the space of p-dimensional vectors to the space of $d \times b$ matrices. Each row of the output sketch is obtained by pre-multiplying the input vector by a diagonal matrix whose entries are Rademacher (uniformly random ± 1), and then pre-multiplying by a $b \times p$ sparse matrix where each column has a single 1, with 0 everywhere else[3]. This process generates one row of the sketch, and is repeated independently d times to generate all rows. The stated (ϵ, δ) guarantee can be achieved for $d \in \Theta(\log 1/\delta)$ and $b \in \Theta(\epsilon^{-2})$. As they are a sparse linear transformation of their input, any addition to an entry in the sketched vector can be applied to the sketch in time $O(d) = O(\log 1/\delta)$.

▶ **Definition 4.** Let:

- $\text{AMS}_{\epsilon,\delta}$ refer to a distribution of random linear maps corresponding to fast AMS sketches with the stated (ϵ, δ) norm and inner product approximation guarantees ((1) and (2)).
- (ϵ, δ)-*sketch transformation* $\mathbf{S}$ be a linear map drawn from $\text{AMS}_{\epsilon,\delta}$.
- The symbol $\odot$ represent the binary operation of performing the inner product query between two sketches. So $\mathbf{S}(\mathbf{x}) \odot \mathbf{S}(\mathbf{y}) \approx \langle \mathbf{x}, \mathbf{y} \rangle$, and $\mathbf{S}(\mathbf{x}) \odot \mathbf{S}(\mathbf{x}) \approx \|\mathbf{x}\|_2^2$.

One application of sketches is to estimate the value of a particular index in a vector. This can be achieved as a special case of an inner product query: we use the sketch to estimate $\langle \mathbf{x}, \mathbf{e}_i \rangle$, where $\mathbf{e}_i$ is the vector that is 1 at location i and 0 elsewhere. The guarantee ensures that we obtain an estimate with error at most $\epsilon \|\mathbf{x}\|_2$. This use of sketches is referred to as a Count sketch [6].

3 Algorithm and Analysis

3.1 Algorithm Overview

Our algorithm works in the most general stream model we considered, the turnstile model. At a high level, our algorithm consists of:

- An *initialization* procedure to set up the sketch data structure.
- An *update* procedure to process updates from the stream.
- A *query* procedure to recover the suspected elements of $\text{LARGE}_{\phi,k}$.

Our sketch structure is built on top of a collection of AMS sketches with standard initialization and update procedures, plus some additional variables to keep running totals. We will briefly review these procedures in Section 3.2, as well as discussing some basic properties and routines required for the query algorithm.

The query process itself is based on two main ideas. First, we can take linear combinations of AMS sketches and then perform inner product queries between them in order to estimate certain kinds of linear combinations of entries of $\mathbf{C}$. Further, we can perform batches of such queries quickly using fast matrix multiplication. And secondly, we can utilize error correcting codes to identify the large magnitude entries in these linear combinations of entries even with the error introduce by the AMS sketches.

[3] The construction does not require the entries to be chosen fully independently at random, so it is common to describe the sketch transformations in terms of hash functions drawn from limited independence families. This allows the transform to be stored in polylogarithmic space.

Algorithm 1: UPDATE

Input: TS model update $u_s = (\alpha, i, j)$

1 $\mathbf{r^{(i)}} \leftarrow \mathbf{r^{(i)}} + \alpha \cdot \mathbf{S}(\mathbf{e_j})$
2 $t^{(i)} \leftarrow t^{(i)} + \alpha$

Algorithm 2: STANDARDIZE

1 **for** $i \in [n]$ **do**
2 $\quad \mathbf{r^{(i)}} \leftarrow \mathbf{r^{(i)}} - (t^{(i)}/p) \cdot \mathbf{S}(\mathbf{e})$
3 $\quad \mathbf{r^{(i)}} \leftarrow (\mathbf{r^{(i)}} \odot \mathbf{r^{(i)}})^{-1/2} \cdot \mathbf{r^{(i)}}$

Rather than fast matrix multiplication between combinations of rows, we could have hoped to employ Pagh's compressed matrix multiplication for our second layer of sketching. However, to produce a w-bucket sketch would take time $\frac{w \log w}{\epsilon^2}$ for each row of the AMS sketches. Then to control the variance of the output buckets, we would need $w = n^2\epsilon^2$, giving a time cost of $\Omega(n^2)$ to build the secondary sketch.

The outline and the discussion of the query algorithm is therefore broken up into four parts. In Section 3.3 we describe a "Cartesian sketch" which compresses a matrix by applying a pair of independent transformations (each akin to the Count sketch), one row-wise and one column-wise. Then, in Section 3.4 we show that we can use the error correcting code technique to recover large entries from Cartesian sketches. Further, we show that the recovery technique is robust to additional sources of noise per entry of the sketch. Next, in Section 3.5 we show how the AMS sketches in our structure can be used to build good enough approximations of the Cartesian sketches to satisfy the noise limits. Finally, in Section 3.6 we analyze the overall space and time costs and discuss how to amplify the probability of success. For brevity, full proofs are deferred to the Appendix, and we present informal proofs in the main body to convey the high level ideas.

3.2 Row Sketching

For our data structure we will keep an AMS sketch of each row of the observation matrix along with a running total. The choice of sketch parameters (ϵ, δ) will be made in the final analysis in Section 3.6.

To initialize the structures, we randomly pick an (ϵ, δ)-sketch transformation S, and initialize n sketches $\mathbf{r^{(i)}}$ to all zeros. We also create n counters $t^{(i)}$, initialized to zero. Algorithm 1 shows how to apply a received update in the TS model. We use $\mathbf{e_j}$ to indicate the length p-vector consisting of a 1 in entry j and 0 everywhere else. The update simply updates the ith sketch with index j, and updates the corresponding sum of weights, $t^{(i)}$. Let $\mathbf{y^{(i)}}$ refer to the i^{th} row of $\mathbf{M}$ for $i \in [n]$. By following these procedures we will have $\mathbf{r^{(i)}} = \mathbf{S}(\mathbf{y^{(i)}})$ and $t^{(i)} = \sum_{j \in [p]} \mathbf{y^{(i)}}_j$ at the conclusion of the stream.

An important operation we will need to be able to perform on these row sketches is to *standardize* them. Recalling $\overline{\mathbf{x}}$ and $\mathbf{e}$ from Definition 1, we define:

▶ **Definition 5.** For a given row vector $\mathbf{y^{(i)}}$, the *standardized vector* $\hat{\mathbf{y}}^{(\mathbf{i})}$ is given by:

$$\hat{\mathbf{y}}^{(\mathbf{i})} = (\mathbf{y^{(i)}} - \overline{\mathbf{x}}_i \mathbf{e}^T) / \|\mathbf{y^{(i)}} - \overline{\mathbf{x}}_i \mathbf{e}^T\|_2.$$

If we have a sketch $\mathbf{S}(\mathbf{y^{(i)}})$ we will refer to $\mathbf{S}(\hat{\mathbf{y}}^{(\mathbf{i})})$ as the *standardized sketch*.

In the RPS and CPS models we could keep track of the running sums of α^2 for each row, allowing us to compute the exact rescaling factor required to standardize the sketches. However, in the more general TS setting, the best we can do is an approximation. Algorithm 2 describes the procedure for computing the approximately standardized sketches.

Initially, it may appear that to perform this standardization at query time, we need to spend $\Omega(pd)$ time building the sketch $\mathbf{S}(\mathbf{e})$. However, we can amortize this cost during the

update phase. As long as at least p entries of the final $\mathbf{M}$ are non-zero, then we can build up $\mathbf{S}(\mathbf{e})$ one entry per update by using a single counter to track which entries have been added. In the atypical case that $\mathbf{M}$ is extremely sparse, we will need to add $O(pd)$ to the query time to complete the construction of this sketch.

▶ **Lemma 6.** *After performing the* STANDARDIZE *routine, the inner product query between sketches* $\mathbf{r}^{(\mathbf{i})}$ *and* $\mathbf{r}^{(\mathbf{j})}$ *produces an estimate of* $\mathbf{C}_{i,j}$ *having* 4ϵ *additive error with probability at least* $1 - 3\delta$*, for* $\epsilon < 1/2$*.*

Informal Proof. Each sketch approximates the sketch of a standardized row ($\mathbf{r}^{(\mathbf{i})} \approx \mathbf{S}(\hat{\mathbf{y}}^{(\mathbf{i})})$) and the inner product query between sketches of standardized rows approximates the correlation ($\mathbf{S}(\hat{\mathbf{y}}^{(\mathbf{i})}) \odot \mathbf{S}(\hat{\mathbf{y}}^{(\mathbf{j})}) \approx \mathbf{C}_{i,j}$). To get a small additive error on our estimates, we then just need both sketches to be approximated well and the inner product query between them to give a good results. Each of the three events occurs with probability $(1 - \delta)$.

The correlation between two rows can be expressed as the inner product of the corresponding standardized vectors. The sketches output by STANDARDIZE approximate the true standardized sketches. To get a small additive error on our estimate, we then just need both sketches to be approximated well and the inner product query between them to give a good result. Each of the three occurs with probability $(1 - \delta)$. ◀

3.3 Cartesian Sketches

▶ **Definition 7.** For an $n \times n$ matrix $\mathbf{A}$, we call $\textsc{Cart}(\mathbf{A})$ a $\Pi \times \Pi$ Cartesian sketch of $\mathbf{A}$ if for each $(h, g) \in [\Pi]^2$ we have

$$\textsc{Cart}(\mathbf{A})_{h,g} = \sum_{P_1(x)=h} \sum_{P_2(y)=g} (s_1(x)s_2(y)\mathbf{A}_{x,y}),$$

where s_1 and s_2 are independently selected from a pairwise independent family of random sign functions $[n] \to \{-1, +1\}$, and where P_1 and P_2 are functions $[n] \to [\Pi]$ selected independently and uniformly at random from the set of functions:

$$\{f : [n] \to [\Pi] \text{ s.t. } |f^{-1}(i)| = n/\Pi \text{ for each } i \in [\Pi]\}.$$

From this definition, we can see that a Cartesian sketch transformation is very similar to a pair of independent Count sketch transformations (one performed row-wise, one column-wise). The difference is the use of fully random partitioning functions which produce exactly equal buckets. If we were performing exactly a pair of Count sketches, we would also have P_1 and P_2 expressed as limited independence hash functions. However, the $O(n \log n)$ space needed to store fully random permutations will not impact our asymptotic space usage and makes the subsequent analysis simpler. The entries of the sketches will be referred to as buckets, and the $(i, j)^{\text{th}}$ entry of the original matrix $\mathbf{A}$ is said to be mapped to the $(h, g)^{\text{th}}$ bucket (for a given choice of sketch functions) if $P_1(i) = h$ and $P_2(j) = g$.

▶ **Definition 8.** Let $\mathcal{B}_{h,g} = \{(i, j) \in [n]^2 \text{ s.t. } P_1(i) = h \text{ and } P_2(j) = g\}$, i.e. the set of index pairs mapped to bucket (h, g).

3.4 Recovery Process

Now we will describe how to apply the recovery process to a series of Cartesian sketches of a given matrix. We will describe an algorithm which gives a constant probability of finding any given element of $\textsc{Large}_{\phi,k}$ (Definition 3) and argue that it works.

Algorithm 3: RecoveryStep

Input: Cartesian sketches $\mathbf{L}^{(\mathbf{l})}$ and $\mathbf{R}^{(\mathbf{l})}$ for $l \in [L \log n]$, and Cartesian sketch transformation Cart

Output: Index multiset Ω of suspected large entries

```
Create new empty multiset Ω
for (h, g) ∈ [Π]^2 do
    Create new empty strings I and J
    for l ∈ [L log n] do
        if |L^(l) − Cart(E^(l))| ≥ φ/2 then Append 1 to I else Append 0 to I
        if |R^(l) − Cart(E^(l))| ≥ φ/2 then Append 1 to J else Append 0 to J
    Append (D(I), D(J)) to Ω
return Ω
```

For this procedure, we apply an error correcting code to encode row and column indices (which take values in $[n]$) into a longer binary codeword. We will assume access to some family of functions (over choices of n) with the desired properties to perform the encoding and decoding. For a fixed n, let:

$$\mathcal{E} : [n] \to \{0,1\}^{L \log n} \text{ and } \mathcal{D} : \{0,1\}^{L \log n} \to [n]$$

where $L > 1$ indicates how much bigger the codeword is compared to the input size. Here, we write $\mathcal{E}$ and $\mathcal{D}$ for the encoder and decoder functions (respectively) of a scheme which can recover from up to $\lambda L \log n$ bit flip errors — i.e. an error rate of λ. That is, for any length $L \log n$ binary word $\mathbf{w}$ with at most $\lambda L \log n$ bits set to 1, we have[4] $\mathcal{D}(\mathcal{E}(i) \oplus \mathbf{w}) = i$ for every $i \in [n]$.

Error correcting codes are known to exist for $L \in O(1)$ and $\lambda \in \Omega(1)$, which can be implemented to perform encoding and decoding in $O(\log n)$ time and $O(\text{polylog}\, n)$ space (for example [18]).

▶ **Definition 9.** For each $l \in [L \log n]$ (each bit in the code words), we define a *masking matrix* $\mathbf{E}^{(\mathbf{l})}$. This is a diagonal binary matrix where entry $\mathbf{E}^{(\mathbf{l})}{}_{i,i}$ is the l^{th} bit of the code word $\mathcal{E}(i)$. That is, $\mathbf{E}^{(\mathbf{l})}{}_{i,i} = \mathcal{E}(i)_l$.

These masking matrices can be pre- or post-multiplied with $\mathbf{C}$ to mask rows or columns (respectively) based on bits of their index encodings.

The recovery process is described in Algorithm 3. It takes as input sketches $\mathbf{L}^{(\mathbf{l})} = \text{Cart}(\mathbf{C}\mathbf{E}^{(\mathbf{l})})$ and $\mathbf{R}^{(\mathbf{l})} = \text{Cart}(\mathbf{E}^{(\mathbf{l})}\mathbf{C})$ for each $l \in [L \log n]$, for a randomly selected Cartesian sketch transformation Cart.

To understand why this process should work, consider the special case where $\mathbf{C}$ is 0 on all the non-large, off-diagonal entries. That is, the only non-zero entries are the diagonals (which must be 1) and the entries corresponding to elements of $\text{Large}_{\phi,k}$. In this situation, the only entries contributing to the Cartesian sketches are entries of $\text{Large}_{\phi,k}$ corresponding to unmasked rows and columns. Now, consider what happens in a bucket with a single large entry mapped to it. Whenever the row or column of the large entry is masked, the corresponding bucket value will be 0; and when the row and column are not masked, the

[4] Here $\oplus$ represents the "exclusive-or" bitwise operation between binary words.

bucket value will have magnitude at least ϕ — in particular, greater than $\phi/2$. This means that, in the algorithm, on the outer loop corresponding to this bucket, $\mathcal{I}$ and $\mathcal{J}$ will be exactly the code words corresponding to the row and column indices of the large entry. Hence, the index pair of the large entry is added to Ω. So, isolated large entries will be correctly recovered in this special case, and as long as k is sufficiently smaller than Π we have a good chance of any given large entry being isolated. To formalize this argument, and extend it to the more general case, we define a few different events.

▶ **Definition 10.** For fixed $\mathbf{C}$ and a fixed coding scheme define the following random events, over the random choice of sketch functions:

- Let $\textsc{CorrectDecode}_{h,g}$ be the event that the recovery process returns the "correct" index for bucket (h, g). If there is exactly one of $\textsc{Large}_{\phi,k}$ in the bucket, then the correct result is the index pair of that entry. Otherwise, any returned value is considered correct.
- Let $\textsc{SmallError}_{h,g,l}$ be the event that the non-large entries of $\mathbf{CE^{(1)}} - \mathbf{E^{(1)}}$ and $\mathbf{E^{(1)}C} - \mathbf{E^{(1)}}$ each contribute less than $\phi/4$ to bucket (h, g) of their corresponding sketches. That is, $\textsc{Cart}(\mathbf{E^{(1)}C_{-k}})_{h,g}$ has magnitude smaller than $\phi/4$.

We begin with a proposition explaining the circumstances we are looking for to successfully find a large entry.

▶ **Lemma 11.** *If we have that:* $\mathbb{P}[\textsc{CorrectDecode}_{h,g}] \geq 1 - x$ *for all* $(h, g) \in [\Pi]^2$*, then for any given* $(i, j) \in \textsc{Large}_{\phi,k}$*, we have that* (i, j) *is in the list of index pairs produced by* RecoveryStep *with probability at least* $1 - x - 2k/\Pi$.

From this proposition, we can see that if we can get a lower bound on the probability of $\textsc{CorrectDecode}_{h,g}$ for every $(h, g) \in [\Pi]^2$, then we can get an overall guarantee for the recovery process.

▶ **Lemma 12.** *If we have that:* $\mathbb{P}[\textsc{SmallError}_{h,g,l}] \geq 1 - y$ *for all* $l \in [L \log n]$*, then* $\mathbb{P}[\textsc{CorrectDecode}_{h,g}] \geq 1 - y/\lambda$.

The last piece we need is a lower bound on the probability of $\textsc{SmallError}_{h,g,l}$.

▶ **Lemma 13.** *Recalling Definition 3, we have:*

$$\mathbb{P}[\textsc{SmallError}_{h,g,l}] \geq (1 - 32\|\mathbf{C_{-k}}\|_F^2/(\Pi^2\phi^2)).$$

Now we have all the pieces we need to show that the recovery process works.

▶ **Lemma 14.** *If we have that* $\Pi \geq \max\{18k, 18\|\mathbf{C_{-k}}\|_F/(\phi\lambda^{1/2})\}$*, then the output of* RecoveryStep *will include any fixed index pair in* $\textsc{Large}_{\phi,k}$ *with probability at least* $\frac{2}{3}$.

Observe that we chose the definition of the event $\textsc{SmallError}_{h,g,l}$ to leave room for an additional source of noise of similar size $\phi/4$. We will need this robustness later.

▶ **Corollary 15.** *Lemma 14 holds even when there is additional noise applied to each bucket entry, provided it has magnitude smaller than* $\phi/4$ *with probability at least* $(1 - \lambda/18)$ *on any fixed bucket.*

In the next subsection, we will show that an approximate Cartesian sketch can be constructed from row sketches within these tolerances.

Algorithm 4: APPROXIMATE

Input: Approximately standardized row sketches $\mathbf{r^{(1)}}, \cdots, \mathbf{r^{(n)}}$ and functions P_1, P_2, s_1, s_2 corresponding to a Cartesian sketch transformation CART

Output: $\mathbf{L^{(l)}}$ and $\mathbf{R^{(l)}}$, estimates of CART$(\mathbf{E^{(l)}C})$ and CART$(\mathbf{CE^{(l)}})$ respectively, for $l \in [L \log n]$

for $l \in [L \log n]$ **do**
  Initialize empty matrices $\mathbf{L^{(l)}}$ and $\mathbf{R^{(l)}}$
  for $h \in [\Pi]$ **do**
    Initialize LEFT$[h]$, RIGHT$[h]$, LEFTMASKED$[h]$, RIGHTMASKED$[h]$ as zero sketches $\mathbf{S(0) = 0}$
  for $i \in [n]$ **do**
    LEFT$[P_1(i)] \leftarrow$ LEFT$[P_1(i)] + s_1(i) \cdot \mathbf{r^{(i)}}$
    LEFTMASKED$[P_1(i)] \leftarrow$ LEFTMASKED$[P_1(i)] + \mathbf{E^{(l)}}_{i,i} \cdot s_1(i) \cdot \mathbf{r^{(i)}}$
    RIGHT$[P_2(i)] \leftarrow$ RIGHT$[P_2(i)] + s_2(i) \cdot \mathbf{r^{(i)}}$
    RIGHTMASKED$[P_2(i)] \leftarrow$ RIGHTMASKED$[P_2(i)] + \mathbf{E^{(l)}}_{i,i} \cdot s_2(i) \cdot \mathbf{r^{(i)}}$
  for $(h, g) \in [\Pi]^2$ **do**
    $\mathbf{L^{(l)}}_{h,g} \leftarrow$ LEFTMASKED$[h] \odot$ RIGHT$[g]$
    $\mathbf{R^{(l)}}_{h,g} \leftarrow$ LEFT$[h] \odot$ RIGHTMASKED$[g]$
return $\mathbf{L^{(1)}}, \cdots, \mathbf{L^{(L \log n)}}$ *and* $\mathbf{R^{(1)}}, \cdots, \mathbf{R^{(L \log n)}}$

3.5 Approximation from Row Sketches

We need a way of quickly approximating CART$(\mathbf{E^{(l)}C})$ and CART$(\mathbf{CE^{(l)}})$ for each $l \in [L \log n]$ for a randomly chosen Cartesian sketch transformation CART, from the row sketches described in Section 3.2. This is done by the procedure described in Algorithm 4.

For each $l \in [L \log n]$, the returned $\mathbf{L^{(l)}}$ is our approximate CART$(\mathbf{E^{(l)}C})$ and $\mathbf{R^{(l)}}$ is our approximate CART$(\mathbf{CE^{(l)}})$.

The algorithm works by observing that $\mathbf{C}$ can be approximated from the row sketches by performing all the possible inner product queries between pairs of sketches and placing the results in the corresponding positions of the matrix. We could then apply CART to the result. However, we make the further observation that since CART can be broken up into pieces that look like pre- and post-multiplication by matrices, we can rearrange the order of operation. We can perform the CART sketch first, directly on the row sketches, and then perform the all-pairs inner product query second. This simple change results in the main performance bottle-neck (the all-pairs inner product query) happening on a much smaller matrix, greatly speeding up the entire query process.

We will show that this process produces a good enough approximation of the CART sketch to act as the input to RECOVERYSTEP.

▶ **Lemma 16.** *At the end of* APPROXIMATE*, for any given* $(h, g) \in [\Pi]^2$*, we have:*

$$|\mathbf{L^{(l)}}_{h,g} - \text{CART}(\mathbf{E^{(l)}C})_{h,g}| \leq \epsilon n \Pi^{-1} 207 \lambda^{-1/2},$$
$$\text{and } |\mathbf{R^{(l)}}_{h,g} - \text{CART}(\mathbf{CE^{(l)}})_{h,g}| \leq \epsilon n \Pi^{-1} 207 \lambda^{-1/2},$$

with probability at least $1 - \lambda/27 - \delta(2 + 12n/\Pi)$*, as long as* $\epsilon < 1/2$*.*

To meet the requirements for RECOVERY to work on these approximations, we need to set limits on the choices of ϵ and δ.

Algorithm 5: RECOVER

Input: Row sketches $\mathbf{r^{(1)}}, \cdots, \mathbf{r^{(n)}}$ and totals $t^{(1)}, \cdots, t^{(n)}$
Output: Index set Ω of entries we are confident are large

```
Create empty multiset Ω
for γ ∈ [Γ] do
    Randomly generate functions P_1, P_2, s_1, s_2 for a Cartesian sketch CART
    Run APPROXIMATE, passing it P_1, P_2, s_1, s_2 and the row sketches
    Run RECOVERYSTEP, passing it CART and the result of APPROXIMATE
    Append the result of RECOVERYSTEP to Ω
Remove entries from Ω appearing fewer than Γ/2 times
return Ω (as a set)
```

▶ **Lemma 17.** *If we have that:* $\delta \leq \lambda/(54(2+12n/\Pi))$, *and* $\epsilon \leq \min\{1/2, (\phi\Pi\lambda^{1/2})/(828n)\}$, *then* APPROXIMATE *produces approximations which are within the noise tolerance of* RECOVERYSTEP.

3.6 Analysis of Algorithm

Putting together the previous subsections, we can make the full recovery algorithm. The outline is listed in Algorithm 5.

▶ **Lemma 18.** *If we have that:*

$$\delta \leq \lambda/(54(2+12n/\Pi)),\ \epsilon \leq \min\{1/2, \phi\Pi\lambda^{1/2}/828n\},\ \Pi \geq \max\{18k, 18\|\mathbf{C}_{-\mathbf{k}}\|_F\phi^{-1}\lambda^{-1/2})\},$$

then we can choose a $\Gamma \in O(\log n)$ *such that* RECOVER *returns every element of* $\text{LARGE}_{\phi,k}$ *with probability at least* $1 - n^{-3}$.

Using $\mathcal{M}(x, y)$ to represent the time required to multiply a $x \times y$ matrix by an $y \times x$ matrix, we can bound the time and space costs of the overall algorithm.

▶ **Lemma 19.** RECOVER *can be implemented to run in*

time $\tilde{O}(\Gamma(\Pi^2 + \log(1/\delta)(n\epsilon^{-2} + \mathcal{M}(\Pi, \epsilon^{-2}))))$ *and space* $\tilde{O}(\Pi^2 + n\epsilon^{-2}\log(1/\delta))$.

By setting $\Pi = n^\theta$ we can look for the right trade-off.

▶ **Theorem 20.** *For every* $\theta \in [0, 1]$, *there exists a sketch of size*

$$\tilde{O}\left(n^{2\theta}(k^2 + \frac{R^2}{\phi^2}) + n^{3-2\theta}(\frac{\phi^2}{k^2\phi^2 + R^2})\right)$$

from which we can extract the (up to k*) entries with magnitude at least* ϕ *in time*

$$\tilde{O}\left(n^{2\theta}(k^2 + \frac{R^2}{\phi^2}) + n^{3-2\theta}(\frac{\phi^2}{k^2\phi^2 + R^2}) + \mathcal{M}\left(n^\theta(k + \frac{R}{\phi}), n^{2-2\theta}(\frac{\phi^2}{k^2\phi^2 + R^2})\right)\right)$$

with high probability.

▶ **Corollary 21.** *In particular, for* $\theta = 2/3$, *we can build a sketch of size* $\tilde{O}\left(n^{5/3}(k^2 + \frac{R^2}{\phi^2})\right)$ *with query time* $\tilde{O}\left(n^{5/3}(k^2 + \frac{R^2}{\phi^2})\right)$.

4 Concluding Remarks

We have shown how to guarantee accurate recovery of correlation outliers using a sketch-based method, beating LSH on query time for small correlation outliers and vanishing correlation non-outliers. A key part of our approach is to use sketching and coding ideas repeatedly: as well as using sketching to reduce the initial dimensionality of the data, we use a second "layer" of sketching when we combine subsets of signals, in order to speed up queries over many pairs of sketches at the cost of increased error. Where LSH tries to hash the correlated elements together, we try to separate them and then recover them from the noise. This produces a trade-off between the size of the underlying sketches and the final query time. This general approach could work in other situations where a large number of sub-queries need to be evaluated to search for large values, for example with measures of similarity/distance other than correlation.

Further, as the technique produces a linear intermediate sketch, this approach is easily adapted to recover pairs whose correlation deviates from some expected correlation matrix, or has changed compared with some previous point in time (simply perform the heavy hitters recover on the difference between two intermediate sketches built using the same permutations, signs, and codes).

Future directions would include finding ways to use alternative primitives to fast matrix multiplication (such as fast convolution via FFT, as adopted by Pagh) and trying to combine the advantages of LSH-based methods and heavy-hitters-based methods.

It would also be useful to re-analyze the algorithms of Valiant and Karppa et al. in our model of bounded total weight of the non-outliers. This could allow us to use some of their more powerful ideas to bring down the exponent in our query time cost from 5/3 to less than 1.6 (as they achieve in their Boolean algorithm for all vanishing non-outliers).

Acknowledgements

We thank Milan Vojnovic for several discussions about this work, and Jelani Nelson for his help and guidance in preparing the final version of this paper.

A Detailed Proofs

Proof of Lemma 6. Recalling definition 1, $\mathbf{C}_{i,j}$ can be expressed as $\mathbf{V}_{i,i}^{-1/2}\mathbf{V}_{i,j}\mathbf{V}_{j,j}^{-1/2}$, where each $\mathbf{V}_{h,g}$ is the scaled inner product between standardized rows $\hat{\mathbf{y}}^{(\mathbf{h})}$ and $\hat{\mathbf{y}}^{(\mathbf{g})}$:

$$\mathbf{V}_{h,g} = \frac{1}{p-1}(\mathbf{y}^{(\mathbf{h})} - \overline{\mathbf{x}}_{\mathbf{h}}\mathbf{e}^T)(\mathbf{y}^{(\mathbf{g})} - \overline{\mathbf{x}}_{\mathbf{g}}\mathbf{e}^T)^T.$$

Observe that factors in $V_{h,g}$ involving $p-1$ cancel in the expression for $\mathbf{C}_{i,j}$, so they can be ignored. What remains is the inner product between normalized (to Euclidean norm 1) versions of vectors $\mathbf{y}^{(\mathbf{i})} - \overline{\mathbf{x}}_{\mathbf{i}}\mathbf{e}^T$ and $\mathbf{y}^{(\mathbf{j})} - \overline{\mathbf{x}}_{\mathbf{j}}\mathbf{e}^T$.

Before performing STANDARDIZE, we had each $\mathbf{r}^{(\mathbf{i})} = \mathbf{S}(\mathbf{y}^{(\mathbf{i})})$. We also have that $(t^{(i)}/p) = \overline{\mathbf{x}}_i$. This means that at the end of the routine, each $\mathbf{r}^{(\mathbf{i})}$ is now a sketch of $(z^{(i)})^{-1/2}(\mathbf{y}^{(\mathbf{i})} - \overline{\mathbf{x}}_{\mathbf{i}}\mathbf{e}^T)$, where $(z^{(i)})^{-1/2}$ is the correct normalization factor to within multiplicative error in the range $[(1-2\epsilon)^{1/2}, (1+2\epsilon)^{1/2}]$ with probability at least $1-\delta$.

For the result, we require two such rescaling factors to be within their bounds, and also for the inner product query to succeed. Each of these three events holds with probability at least $1-\delta$, giving an overall probability at least $1-3\delta$ by the union bound.

To determine the overall error, consider that since $\epsilon < 1/2$ and $|\mathbf{C}_{i,j}| \leq 1$,
$(\tilde{\mathbf{y}}^{(\mathbf{h})}(1 \pm 2\epsilon)^{1/2}) \odot (\tilde{\mathbf{y}}^{(\mathbf{g})}(1 \pm 2\epsilon)^{1/2}) \in (1 \pm 2\epsilon)\mathbf{C}_{i,j} + \epsilon(1 \pm 2\epsilon) \subset \mathbf{C}_{i,j} \pm 4\epsilon.$

◀

Proof of Lemma 11. Let $(h, g) = (P_1(i), P_2(j))$ be the bucket (i, j) is mapped to. Since the partition functions are chosen uniformly at random, the chance that none of the other entries mapped to the same bucket are in $\textsc{Large}_{\phi,k}$ is at least $1 - 2k/\Pi$. To see this, observe that in the worst case, all index pairs in $\textsc{Large}_{\phi,k}$ have either the same row or same column index. Then, by the Markov inequality, we have less than $2k/\Pi$ probability that at least one of the remaining $k - 1$ entries in $\textsc{Large}_{\phi,k}$ are mapped into one of the remaining $n/\Pi - 1$ slots in that bucket.

Now, if entry (i, j) turns out to be the only large entry in its bucket, then the event $\textsc{CorrectDecode}_{h,g}$ occurring implies that the index pair recovered from bucket (h, g) will be (i, j). The chance of both occurring is then at least $1 - x - 2k/\Pi$. ◀

Proof of Lemma 12. In the event that bucket (h, g) contains more or less than one large entry, then $\textsc{CorrectDecode}_{h,g}$ automatically holds, so we only need to consider the case of exactly one large entry in the bucket.

Now, consider the case of only one large entry $\mathbf{C}_{i,j}$ being mapped to the bucket. Observe that we can write

$$\textsc{Cart}(\mathbf{E}^{(\mathbf{1})}\mathbf{C} - \mathbf{E}^{(\mathbf{1})})_{h,g} = \textsc{Big} + \textsc{Small},$$

where $\textsc{Big} = {\mathbf{E}^{(\mathbf{1})}}_{i,i} \cdot s_1(i) \cdot s_2(j) \cdot \mathbf{C}_{i,j}$ and $\textsc{Small} = \textsc{Cart}(\mathbf{E}^{(\mathbf{1})}\mathbf{C}_{-\mathbf{k}})_{h,g}$ (see Definition 3).

When the event $\textsc{SmallError}_{h,g,l}$ holds, we have that $|\textsc{Small}| \leq \phi/4$. Also, $|\textsc{Big}|$ is either 0 (when the row of the large entry is masked) or greater than ϕ (when not masked). So, $\textsc{SmallError}_{h,g,l}$ holding means that the l^{th} threshold bit will match the l^{th} bit of the code word for the row index we are trying to recover. An analogous argument applies to $\textsc{Cart}(\mathbf{C}\mathbf{E}^{(\mathbf{1})} - \mathbf{E}^{(\mathbf{1})})_{h,g}$ and the column index.

For the decoder to correctly recover an index from its code word, we need at most a λ fraction of errors. So, we need less than a λ fraction of the events $\textsc{SmallError}_{h,g,l}$ for $l \in [L \log n]$ failing to hold. By Markov's inequality, we can put the chance of more than a λ fraction of failures at less than y/λ. ◀

Proof of Lemma 13. For fixed $(h, g, l) \in [\Pi]^2 \times [L \log n]$, consider the random variable $\textsc{Cart}(\mathbf{E}^{(\mathbf{1})}\mathbf{C}_{-\mathbf{k}})_{h,g}$ (random over the choices of P_1, P_2, s_1, and s_2 that make up $\textsc{Cart}$). This can be broken down into a sum of contributions from each entry of $\mathbf{E}^{(\mathbf{1})}\mathbf{C}_{-\mathbf{k}}$, as follows:

$$\textsc{Cart}(\mathbf{E}^{(\mathbf{1})}\mathbf{C}_{-\mathbf{k}})_{h,g} = \sum_{(i,j)\in[n]^2} \beta_{i,j,l}$$

$$\text{where } \beta_{i,j,l} = \begin{cases} ({\mathbf{E}^{(\mathbf{1})}}_{i,i}) s_1(i) s_2(j) (\mathbf{C}_{-\mathbf{k}})_{i,j} & \text{if } (i,j) \in \mathcal{B}_{h,g} \\ 0 & \text{otherwise,} \end{cases}$$

recalling from definition 8 that $\mathcal{B}_{h,g}$ represents the index pairs mapped to bucket (h, g).

Due to the independently selected pairwise independent random sign functions s_1 and s_2, each term has mean $\mathbb{E}[\beta_{i,j,l}] = 0$ and covariance $\text{Cov}[\beta_{i_1,j_1,l}, \beta_{i_2,j_2,l}] = 0$ (where either $i_1 \neq i_2$ or $j_1 \neq j_2$). This means the variance of the sum (the bucket value) is simply the sum of the variances of the terms.

Each term has variance $\mathrm{Var}\left[\beta_{i,j,l}\right] \leq (\mathbf{E^{(l)}C_{-k}})^2_{i,j}/\Pi^2$. To see this, observe that each term has at most a $1/\Pi^2$ chance of being non-zero (due to the random partition functions). Summing up all the terms gives us

$$\mathrm{Var}\left[\textsc{Cart}(\mathbf{E^{(l)}C_{-k}})_{p,q}\right] = \|\mathbf{E^{(l)}C_{-k}}\|_F^2/\Pi^2 \leq \|\mathbf{C_{-k}}\|_F^2/\Pi^2.$$

Then, by Chebyshev's inequality, we have $\textsc{Cart}(\mathbf{E^{(l)}C_{-k}})_{p,q} \geq \phi/4$, with probability less than $16\|\mathbf{C_{-k}}\|_F^2/(\Pi^2\phi^2)$. An analogous argument applies to $\textsc{Cart}(\mathbf{C_{-k}E^{(l)}})_{h,g}$, giving the result by union bound. ◀

Proof of Lemma 14. Substituting $\Pi \geq 18\|\mathbf{C_{-k}}\|_F/(\phi\lambda^{1/2})$ into Lemma 13 gives us that:

$\mathbb{P}\left[\textsc{SmallError}_{h,g,l}\right] \geq 1 - 8\lambda/81$.

Then by Lemma 11 (with $y = 8\lambda/81$), we get that:

$\mathbb{P}\left[\textsc{CorrectDecode}_{h,g}\right] \geq 1 - 8/81$.

Finally, using the fact that $\Pi \geq 18k$ (from the initial assumptions) along with Lemma 12 (with $x = 8/81$), we have that any fixed $(i,j) \in \textsc{Large}_{\phi,k}$ will be in the output of RecoveryStep with probability at least $1 - 8/81 - 1/9 = 64/81 \geq 2/3$. ◀

Proof of Corollary 15. Observe that the proof of Lemma 12 still works with an additional term of magnitude no more than $\phi/4$. Then, observe that the choice of parameters in Lemma 14 leaves enough slack to condition on an additional event occurring with probability greater than $1 - \lambda/18$ per $\textsc{SmallError}_{h,g,l}$. ◀

Proof of Lemma 16. If we performed the algorithm with the exact vectors instead of AMS sketches, then $\mathbf{L^{(l)}}_{h,g}$ would be exactly $\textsc{Cart}(\mathbf{E^{(l)}C})$. Any difference is due to the inner product approximation error which is smaller than $\epsilon\|\mathbf{h}\|_2\|\mathbf{g}\|_2$ with probability at least $1-\delta$, where $\mathbf{h}$ and $\mathbf{g}$ are the vectors that $\textsc{LeftMasked}[h]$ and $\textsc{Right}[g]$ are sketches of. First consider

$$\mathbf{h} = \sum_{P_1(i)=h} (\mathbf{E^{(l)}}_{i,i} \cdot s_1(i) \cdot \mathcal{R}^{(i)} \cdot \tilde{\mathbf{y}}^{(\mathbf{i})}),$$

where $\mathcal{R}^{(i)}$ is the rescaling error caused by Standardize (see Section 3.2). Recall that each $|\mathcal{R}^{(i)}| \leq 1 + 4\epsilon$ with probability at least $1 - 3\delta$ as long as $\epsilon < 1/2$.

The squared 2-norm $\|\mathbf{h}\|_2^2$ is given by:

$$\sum_{P_1(i)=P_1(j)=h} \mathcal{R}^{(i)}\mathcal{R}^{(j)}\langle(\mathbf{E^{(l)}}_{i,i}s_1(i)\tilde{\mathbf{y}}^{(\mathbf{i})}), (\mathbf{E^{(l)}}_{j,j}s_1(j)\tilde{\mathbf{y}}^{(\mathbf{j})})\rangle$$

For each $i = j$, the corresponding term is equal to $\mathcal{R}^{(i)}\mathcal{R}^{(j)}$, and for each $i < j$ there is a matching equal term with i and j swapped. So, with probability at least $1 - 3n\delta/\Pi$ the norm is at most $n(1+4\epsilon)^2/\Pi$ plus an independent random variable (random over choice of s_1) with mean 0 and variance less than

$$4\|\mathbf{C}\|_F^2(1+4\epsilon)^2/\Pi^2 \leq 4n^2(1+4\epsilon)^2/\Pi^2.$$

So by Chebyshev's inequality and a union bound, $\|\mathbf{h}\|_2^2$ is smaller than $\frac{n}{\Pi}(1+4\epsilon)^2(22\lambda^{-1/2}+1)$ with probability greater than $1 - \lambda/108 - 3n\delta/\Pi$. The same bound applies to $\|\mathbf{g}\|_2^2$, so $\epsilon\|\mathbf{h}\|_2\|\mathbf{g}\|_2 \leq \epsilon(1+4\epsilon)^2 n\Pi^{-1}23\lambda^{-1/2} \leq \epsilon n\Pi^{-1}207\lambda^{-1/2}$ with probability at least $1 - \lambda/54 - \delta(1+6n/\Pi)$.

An analogous argument works for entry $\mathbf{R^{(l)}}_{h,g}$ and $\textsc{Cart}(\mathbf{CE^{(l)}})$. A union bound over the probabilities of failure gives the result. ◀

Proof of Lemma 17. The assumptions imply that:

$$\epsilon n \Pi^{-1} 23 \lambda^{-1/2} \leq 23\phi/828 \leq \phi/4, \text{ and } 1 - \delta(2 + 12n/\Pi) - \lambda/27 \leq 1 - \lambda/18.$$

This tells us exactly that the errors on the approximations according to Lemma 16 are within the bounds allowed by Lemma 15. ◀

Proof of Lemma 18. From Lemmas 14 and 17 we know for $\Gamma = 1$, this algorithm succeeds at finding any one large entry with probability at least $2/3$. By performing $O(\log n)$ independent repetitions and then only considering those index pairs appearing at least half the time, then by the Chernoff bound we can amplify the probability of finding any one of the large entries to $1 - n^{-5}$. There are at most n^2 such pairs, giving the result. ◀

Proof of Lemma 19. RecoveryStep can be implemented to run in time $O(\Pi^2 \text{polylog} n)$ since we have Π^2 iterations of the outer loop, $O(\log n)$ iterations of the inner loop, and all operations taking $O(\text{polylog} n)$ time (coding schemes with such fast decoding algorithms exist).

Approximate can be implemented to run in time $O(\log n \log(1/\delta)(n\epsilon^{-2} + \mathcal{M}(\Pi, \epsilon^{-2})))$ where $\mathcal{M}(\Pi, \epsilon^{-2})$ is the time required to multiply a $\Pi \times \epsilon^{-2}$ matrix by an $\epsilon^{-2} \times \Pi$ matrix. This holds because there are $O(\log n)$ iterations of the outer loop. Then within we have $O(n)$ additions involving sketches of size $O(\epsilon^{-2} \log(1/\delta))$. We also have a series of inner products which can be expressed as a batched all-pair query. This can be performed as a series of $O(\log(1/\delta))$ matrix multiplications.

Putting this together, we get a time cost of $\tilde{O}(\Gamma(\Pi^2 + \log(1/\delta)(n\epsilon^{-2} + \mathcal{M}(\Pi, \epsilon^{-2}))))$. The filtering step adds no extra asymptotic time, since we can filter by sorting the $O(\Pi^2 \log n)$ pairs and then iterating over them counting repetitions to see if any exceed the $\Gamma/2$ threshold.

RecoveryStep uses $O(\Pi^2 \log n + \text{polylog} n)$ space to store a pair of length $O(\log n)$ strings, a multiset of up to Π^2 index pairs, and the input of $O(\log n)$ Π-by-Π sketches. The polylog overhead is used for the encoding scheme.

Approximate uses $O(\Pi^2 \log n + n\epsilon^{-2} \log(1/\delta))$ space to store $O(n)$ sketches and $O(\log n)$ $\Pi \times \Pi$ matrices.

All together we need $\tilde{O}(\Pi^2 + n\epsilon^{-2} \log(1/\delta))$ space, since the multiset contains at most $O(\Pi^2 \log n)$ index pairs. ◀

Proof of Theorem 20. Substitute bounds in Lemma 18 into costs in Lemma 19. ◀

References

1. D. Achlioptas. Database-friendly random projections. In *ACM Principles of Database Systems*, pages 274–281, 2001.
2. N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *ACM Principles of Database Systems*, pages 10–20, 1999.
3. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *ACM Symposium on Theory of Computing*, pages 20–29, 1996.
4. Alexandr Andoni and Ilya P. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. *CoRR*, abs/1501.01062, 2015. URL: `http://arxiv.org/abs/1501.01062`.
5. Emmanuel Candes, Mark Rudelson, Terence Tao, and Roman Vershynin. Error correction via linear programming. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 668–681. IEEE, 2005.
6. M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Procedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2002.
7. Graham Cormode. Sketch techniques for massive data. In Graham Cormode, Minos Garofalakis, Peter Haas, and Chris Jermaine, editors, *Synposes for Massive Data: Samples, Histograms, Wavelets and Sketches*, Foundations and Trends in Databases. NOW publishers, 2011.
8. David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.
9. Anna C Gilbert, Yi Li, Ely Porat, and Martin J Strauss. Approximate sparse recovery: optimizing time and measurements. *SIAM Journal on Computing*, 41(2):436–453, 2012.
10. P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *ACM Symposium on Theory of Computing*, pages 604–613, 1998.
11. W.B. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.
12. Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM*, 61(1):4:1–4:23, 2014.
13. Matti Karppa, Petteri Kaski, and Jukka Kohonen. A faster subquadratic algorithm for finding outlier correlations. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1288–1305, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=2884435.2884525`.
14. Kasper Green Larsen, Jelani Nelson, Huy L Nguyên, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 61–70. IEEE, 2016.
15. P. Li, T. Hastie, and K. W. Church. Nonlinear estimators and tail bounds for dimension reduction in L_1 using cauchy random projections. *Journal of Machine Learning Research (JMLR)*, 2007.
16. Rasmus Pagh. Compressed matrix multiplication. *ACM Trans. Comput. Theory*, 5(3):9:1–9:17, August 2013. URL: `http://doi.acm.org/10.1145/2493252.2493254`, `doi:10.1145/2493252.2493254`.
17. Eric Price and David P. Woodruff. (1 + eps)-approximate sparse recovery. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 295–304, Washington, DC, USA, 2011. IEEE Computer Society. URL: `http://dx.doi.org/10.1109/FOCS.2011.92`, `doi:10.1109/FOCS.2011.92`.
18. D.A. Spielman. Linear-time encodable and decodable error-correcting codes. *Information Theory, IEEE Transactions on*, 42(6):1723–1731, Nov 1996. `doi:10.1109/18.556668`.

19 Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *J. ACM*, 62(2):13:1–13:45, May 2015. URL: `http://doi.acm.org/10.1145/2728167`, `doi:10.1145/2728167`.

20 Leslie Valiant. Functionality in neural nets. In *First Workshop on Computational Learning Theory*, page 28–39, 1988.