Information and Inference: A Journal of the IMA (2016) Page 1 of 27
doi:10.1093/imaiai/iaw004

Detecting the large entries of a sparse covariance matrix in sub-quadratic time

OFER SHWARTZ AND BOAZ NADLER†
Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel
†Corresponding author. Email: boaz.nadler@weizmann.ac.il

[Received on 12 May 2015; revised on 15 December 2015; accepted on 27 January 2016]

The covariance matrix of a p-dimensional random variable is a fundamental quantity in data analysis. Given n i.i.d. observations, it is typically estimated by the sample covariance matrix, at a computational cost of $O(np^2)$ operations. When n, p are large, this computation may be prohibitively slow. Moreover, in several contemporary applications, the population matrix is approximately sparse, and only its few large entries are of interest. This raises the following question: Assuming approximate sparsity of the covariance matrix, can its large entries be detected much faster, say in sub-quadratic time, without explicitly computing all its p^2 entries? In this paper, we present and theoretically analyze two randomized algorithms that detect the large entries of an approximately sparse sample covariance matrix using only $O(np \text{ poly} \log p)$ operations. Furthermore, assuming sparsity of the population matrix, we derive sufficient conditions on the underlying random variable and on the number of samples n, for the sample covariance matrix to satisfy our approximate sparsity requirements. Finally, we illustrate the performance of our algorithms via several simulations.

Keywords: sparse covariance matrix; sub-quadratic time complexity; multi-scale group testing.

1. Introduction

Let $Z = (Z_1, \ldots, Z_p)$ be a p-dimensional real-valued random variable with a $p \times p$ covariance matrix Σ. Given n i.i.d. samples of Z, denoted by $\{\mathbf{z}_i\}_{i=1}^n$, a fundamental task in statistical inference is to estimate Σ. A standard estimator of Σ is the sample covariance matrix $S = (1/(n-1)) \sum_i (\mathbf{z}_i - \bar{\mathbf{z}})(\mathbf{z}_i - \bar{\mathbf{z}})^\top$, where $\bar{\mathbf{z}}$ is the sample mean. Direct computation of all p^2 entries of S requires $O(np^2)$ operations. In various contemporary data analysis applications, where both p and n are large, this computation may be prohibitively slow and challenging in terms of memory and storage.

In several applications, however, the population covariance matrix is approximately sparse, whereby only its few large entries are of interest, and the remaining entries are either small or even precisely equal to zero. Applications leading to sparse covariance matrices include, among others, gene arrays and biological networks (Butte *et al.*, 2000), social networks, climate data and f-MRI scans. As we are only interested in the large entries, a key question is whether these can be computed significantly faster, possibly by cleverly detecting their locations, without directly computing the entire matrix.

In this paper, we present and theoretically analyze two different randomized algorithms with sub-quadratic time complexity, to detect and subsequently compute the large entries of a sparse sample covariance matrix. First, in Section 3, we present a reduction of this problem to the sparse Fast Fourier Transform (sFFT) (Hassanieh *et al.*, 2012). A solution to our task then directly follows by invoking multiple calls to the recently developed randomized sFFT sub-linear time algorithm. Next, in Section 4 we present a simpler and more direct algorithm, based on the construction of $O(\log p)$ binary random trees. We prove that, under suitable assumptions on the sparsity of the matrix S, both algorithms are guaranteed, with high probability, to locate its large entries. Furthermore, their runtimes are $O(nrp \log^3 p)$

operations for the sFFT-based algorithm, and $O(nrp\log^2 p)$ operations for the tree-based method, where r is a bound on the number of large entries in each row of S. By suitable normalization of the input data, both algorithms can also detect large entries of a sparse sample *correlation matrix*, whose entries are $S_{ij}/\sqrt{S_{ii}S_{jj}}$.

The theoretical analysis of the two algorithms of Sections 3 and 4 relies on the assumption that S is approximately sparse. In reality, in various applications one may only assume that the population matrix Σ is approximately sparse. In Section 5 we provide sufficient conditions on Σ, the underlying random variable Z and the number of samples n that ensure, with high probability, the approximate sparsity of S.

Finally, in Section 6 we empirically compare the tree-based algorithm with other methods to detect large entries of S. In addition, we illustrate on artificially generated data that includes near duplicates the potential applicability of our algorithm for the *Near-Duplicate Detection* problem, common in document corpora analysis (Xiao *et al.*, 2011).

1.1 *Related works*

In the statistics literature, the problem of sparse covariance estimation has received significant attention (see, for example, Bickel & Levina, 2008; Bien & Tibshirani, 2011; Cai & Liu, 2011; Chaudhuri *et al.*, 2007; El Karoui, 2008a). As discussed in Sections 3 and 4, under suitable sparsity assumptions, our algorithms are guaranteed to find with high probability all entries of the sample covariance matrix which are larger, in absolute value, than some threshold μ. The resulting matrix is then nothing, but the sample covariance matrix, hard-thresholded at the value μ. The statistical properties of such thresholding were intensively studied, see, for example, Bickel & Levina (2008), El Karoui (2008a), Cai & Liu (2011) and references therein. The main motivation of these works was to derive a more accurate estimate of the population covariance matrix Σ, assuming it is sparse, thus overcoming the asymptotic inconsistency of S in the operator norm, in the joint limit $p, n \to \infty$ with $p/n \to c$; see, for example, El Karoui (2008b). Moreover, hard-thresholding with a threshold that slowly tends to zero as $p, n \to \infty$, was proved to be asymptotically minimax optimal under various sparsity models. Our work, in contrast, is concerned with the *computational effort* of computing this thresholded estimator, mainly for finite p, n and relatively large thresholds.

Focusing on the computational aspects, first of all note that the sample covariance matrix S can be represented as a product of two suitable matrices. Hence, using fast matrix multiplication methods, all its entries can be computed faster than $O(np^2)$. Currently, the fastest matrix multiplication algorithm for square matrices of size $N \times N$ has a time complexity of $O(N^{2.3727})$ (Williams, 2012). Hence, by expanding (with zero padding) the input sample matrix to a square matrix, all entries of S can be exactly evaluated using $O(\max\{n, p\}^{2.3727})$ operations.

In recent years, several works developed fast methods to *approximate* the product of two matrices, see, for example, Drineas *et al.* (2006), as well as Iwen & Spencer (2009) and Pagh (2013), which assume that the product is approximately sparse. In particular, in a setting similar to the one considered in this paper, the method of Pagh (2013), based on fast approximate matrix multiplication, can detect the large entries of a sparse covariance matrix in time complexity comparable to ours.

In addition, since the entries of S can be represented as inner products of all-pairs vectors, our problem is directly related to the *Maximum Inner Product Search* (MIPS) problem (Ram & Gray, 2012; Shrivastava & Li, 2014). In the MIPS problem, given a large dataset $\{\mathbf{x}_i\}$, the goal is to quickly find, for any query vector $\mathbf{y}$, its maximal inner product $\max_i \langle \mathbf{y}, \mathbf{x}_i \rangle$. Ram & Gray (2012) presented three algorithms to retrieve the maximal inner product, which can be generalized to find the k largest values.

While Ram & Gray (2012) did not provide a theoretical analysis of the runtime of their algorithms, empirically on several datasets they were significantly faster than direct computation of all inner products. Recently, Shrivastava & Li (2014) presented a simple reduction from the *approximate*-MIPS problem, of finding $\max_i \langle \mathbf{y}, \mathbf{x}_i \rangle$ up to a small distortion ϵ, to the well-studied k *nearest neighbor* problem (kNN). This allows one to solve the approximate-MIPS problem using any kNN procedure. The (exact or approximate) kNN search problem has been extensively studied in the last decades, with several fast algorithms; see Shakhnarovich *et al.* (2006), Jones *et al.* (2013), Arya *et al.* (1998) and references therein. For example, *kd*-tree (Bentley, 1975) is a popular exact algorithm for low-dimensional data, whereas *Local-sensitive hashing* (LSH) approximation methods (Datar *et al.*, 2004) are more suitable to high dimensions. Combining the reduction of Shrivastava & Li (2014) with the LSH-based algorithm of Har-Peled *et al.* (2012) yields an approximate solution of the MIPS problem with $O(np^{\gamma} \operatorname{poly} \log p)$ query time, where $\gamma \in (0, 1)$ controls the quality of the approximation. These methods can be exploited to approximate the sample covariance matrix, perhaps with no assumptions on the matrix, but with slower (or no) runtime guarantees. In Section 6, we empirically compare our sub-quadratic tree-based algorithm with some of the above methods.

Another line of work related to fast estimation of a sparse covariance matrix is the matrix sketching problem (Dasarathy *et al.*, 2013). Here, the goal is to recover an unknown matrix X, given only partial knowledge of it, in the form of a linear projection AXB with known matrices A and B. The matrix sketching problem is more challenging, since we are given only partial access to the input. In fact, recent solutions to this problem (Wimalajeewa *et al.*, 2013) have time complexity at least $O(np^2)$.

A more closely related problem is the *all-pairs similarity search* with respect to the cosine similarity or the Pearson correlation. Here, given a set of vectors $\{\mathbf{x}_i\}$, the task is to find the pair with highest cosine similarity, $\max_{i \neq j} \mathbf{x}_i{}^t \mathbf{x}_j / \|\mathbf{x}_i\| \|\mathbf{x}_j\|$. Two popular exact methods, suitable mainly for sparse inputs, are *All-Pairs* (Bayardo *et al.*, 2007) and *PPJoin+* (Xiao *et al.*, 2011). As in the nearest neighbors search problem, hashing can be adapted to obtain various approximation algorithms (Charikar, 2002). These algorithms can be used to rapidly compute a sparse correlation matrix. In a special case of the all-pairs similarity search, known as the *Light Bulb Problem* (Valiant, 1988), one is given p boolean vectors all of length n and taking values ± 1. The assumption is that all vectors are uniformly distributed on the n-dimensional boolean hypercube, apart from a pair of two vectors which have a Pearson correlation $\rho \gg \sqrt{\log p / n}$. The question is how fast can one detect this pair of correlated vectors. LSH-type methods, as well as bucketing coding (Dubiner, 2010), solve this problem with a sub-quadratic time complexity of $O(np^{g(\rho)})$ for a suitable function $g(\rho)$ of the correlation coefficient. More recently, Valiant (2015) developed a method with expected sub-quadratic time complexity of $O(np^{1.62})$ operations, where the exponent value 1.62 is independent of ρ, and directly related to the complexity of the fastest known matrix multiplication algorithm. It is easy to show that the methods proposed in our paper can solve the Light Bulb Problem using only $O(np \operatorname{poly} \log p)$ operations, but assuming a significantly stronger correlation of $\rho > O(\sqrt{p/n})$. Furthermore, our methods offer an interesting tradeoff between time complexity and correlation strength ρ. For example, our tree-based algorithm with $O(p^{\alpha})$ trees (instead of $O(\log p)$) can detect weaker correlations of strength $\rho > O(\sqrt{p^{1-\alpha}/n})$ with a time complexity of $O(np^{1+\alpha} \operatorname{poly} \log p)$ operations.

2. Notation and problem setup

For simplicity, in this paper we assume that the input data $(\mathbf{z}_1, \ldots, \mathbf{z}_n)$ is real-valued, though the proposed methods can be easily modified to the complex-valued case. For a vector $\mathbf{x} \in \mathbb{R}^n$, we denote

its ith entry by $\mathbf{x}_i$ (or $(\mathbf{x})_i$), its L_2-norm by $\|\mathbf{x}\| := \sqrt{\sum_{i=1}^n \mathbf{x}_i^2}$ and its L_0-norm by $\|\mathbf{x}\|_0 := |\{i : \mathbf{x}_i \neq 0\}|$. Similarly, for a matrix $A \in \mathbb{R}^{p\times p}$ we denote its kth row by A_k, its (i,j)th entry by A_{ij} (or $(A_i)_j$), its L_2-norm by $\|A\| := \sup_{\mathbf{x}\neq 0}(\|A\mathbf{x}\|/\|\mathbf{x}\|)$ and its Frobenius norm by $\|A\|_F := \sqrt{\sum_{ij} A_{ij}^2}$. For an integer $a \in \mathbb{N}$, let $[a] := \{1, 2, \ldots, a\}$. To simplify notations, the inner product between two complex vectors $\mathbf{x}, \mathbf{y} \in \mathbb{C}^n$ is defined with a normalization factor,

$$\langle \mathbf{x}, \mathbf{y} \rangle = \frac{1}{n-1} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i^{\dagger},$$

where † represents the complex conjugate, and will be relevant only when we discuss the Fourier transform which involves complex-valued numbers.

Given the input data $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$, let $\mathbf{x}_i = ((\mathbf{z}_1)_i - (\bar{\mathbf{z}}_i), \ldots, (\mathbf{z}_n)_i - (\bar{\mathbf{z}}_n)) \in \mathbb{R}^n$ be a mean centered vector of observations for the ith coordinate of Z. This leads to the simple representation of S, in terms of inner products

$$S_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle, \quad 1 \leqslant i, j \leqslant p. \tag{2.1}$$

For a matrix $A \in \mathbb{R}^{p\times p}$ and a threshold parameter μ, we define the set of large entries at level μ in the kth row to be

$$J_\mu(A_k) = \{j \,:\, |A_{kj}| \geqslant \mu\}$$

and the set of all its large entries at level μ by

$$J_\mu(A) = \bigcup_{k=1}^{p} \{k\} \times J_\mu(A_k).$$

As for the definition of matrix sparsity, we say that a matrix A is (r, μ)-*sparse* if, for every row k, $|J_\mu(A_k)| \leqslant r$. We say that A is (r, μ, R, q)-sparse if it is (r, μ)-sparse and, for every $k \in [p]$, the remaining small entries in the kth row are contained in an L_q-ball of radius R,

$$\left(\sum_{j \notin J_\mu(A_k)} |A_{kj}|^q \right)^{1/q} \leqslant R.$$

Here, we only consider the case where $q = 2$ and $R < \mu/2$.

2.0.0.1. *Problem Setup.* Let $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ be n input vectors whose covariance matrix S is (r, μ)-sparse, with $r \ll p$. In this paper, we consider the following task: Given $\{\mathbf{z}_i\}_{i=1}^n$ and the threshold value μ, find the set $J_\mu(S)$, which consists of all entries of S which are larger than μ in absolute value. A naive approach is to explicitly compute all p^2 entries of S and then threshold at level μ. This requires $O(np^2)$ operations and may be quite slow when $p \gg 1$. In contrast, if an oracle gave us the precise locations of all large entries, computing them directly would require only $O(npr)$ operations.

The key question studied in this paper is whether we can find the set $J_\mu(S)$, and compute the corresponding entries significantly faster than $O(np^2)$. In what follows, we present and analyze two sub-quadratic algorithms to discover $J_\mu(S)$, under the assumption that the matrix S is approximately sparse.

3. Sparse covariance estimation via sFFT

The first solution we present is based on a reduction of our problem to that of multiple independent instances of sparse Fourier transform calculations. Whereas standard FFT of a vector of length p requires $O(p \log p)$ operations, if the result is *a priori* known to be approximately sparse, it can be computed in sub-linear time. This problem, known as sFFT, has been intensively studied in the past few years; see Hassanieh *et al.* (2012), Akavia (2010), Gilbert *et al.* (2002), Iwen (2010). As described below, the computational gains for our problem then directly follow by an application of one of the available sFFT algorithms.

In what follows, we present this reduction, and focusing on the algorithm of Hassanieh *et al.* (2012), we analyze under which sparsity conditions on S it is guaranteed to succeed with high probability. To this end, we use the following definitions for the discrete Fourier transform (DFT) $\mathscr{F} : \mathbb{C}^p \to \mathbb{C}^p$ and its inverse $\mathscr{F}^{-1}$:

$$(\mathscr{F}[\mathbf{x}])_j = \sum_{l=1}^{p} \mathbf{x}_l \omega^{-jl} \quad (\mathscr{F}^{-1}[\mathbf{x}])_j = \frac{1}{p} \sum_{l=1}^{p} \mathbf{x}_l \omega^{jl},$$

where $\omega = \exp(2\pi \mathbf{i}/p)$, $\mathbf{i} = \sqrt{-1}$. Without any assumptions on the input $\mathbf{x}$, the fastest known method to compute its DFT is the FFT which requires $O(p \log p)$ operations. If the vector $\mathscr{F}[\mathbf{x}]$ is approximately r-sparse, with only r large entries, it is possible to estimate it in *sub-linear time*, by detecting and approximately evaluating only its large entries. In particular, Hassanieh *et al.* (2012) developed a randomized algorithm to compute the sparse Fourier transform with time complexity of $O(r \log p \log(p/r))$, which we refer to here as the sFFT algorithm. Formally, for any input vector $\mathbf{x} \in \mathbb{C}^p$ and parameters α, δ, r, the sFFT algorithm returns a sparse vector $\hat{\mathbf{x}}$ such that, with probability $\geqslant \frac{2}{3}$

$$\|\mathscr{F}[\mathbf{x}] - \hat{\mathbf{x}}\| \leqslant (1+\alpha) \min_{\|\mathbf{y}\|_0 \leqslant r} \|\mathscr{F}[\mathbf{x}] - \mathbf{y}\| + \delta \|\mathscr{F}[\mathbf{x}]\|. \tag{3.1}$$

The algorithm time complexity is $O((r/\alpha) \log(p/r) \log(p/\delta))$, though for our purposes we assume that α is fixed (e.g. $\alpha = 1$), and hence ignore the dependence on it. The output $\hat{\mathbf{x}}$ of the sFFT algorithm is represented as a pair $(J, \mathbf{y})$, where $J \subset \{1, \ldots, p\}$ is a set of indices and $\mathbf{y} \in \mathbb{C}^{|J|}$ are the values of $\hat{\mathbf{x}}$ at these indices, namely $\hat{\mathbf{x}}|_J = \mathbf{y}$ and $\hat{\mathbf{x}}|_{J^c} = 0$.

We now present the reduction from the sparse covariance estimation problem to sparse FFT. To this end, let us define a matrix $W = (\mathbf{w}_1, \ldots, \mathbf{w}_p) \in \mathbb{C}^{n \times p}$ whose jth column is $\mathbf{w}_j = (1/p) \sum_{l=1}^{p} \mathbf{x}_l \, \omega^{-jl}$. Note that, using standard FFT methods, W can be calculated in $O(np \log p)$ operations. The following lemma describes the relation between the matrix S and the matrix W.

LEMMA 3.1 For any $k \in [p]$, let $\mathbf{u}_k = (\langle \mathbf{x}_k, \mathbf{w}_1 \rangle, \ldots, \langle \mathbf{x}_k, \mathbf{w}_p \rangle) \in \mathbb{C}^p$. Then

$$\mathscr{F}[\mathbf{u}_k] = S_k. \tag{3.2}$$

Proof. Equation (3.2) follows directly by applying the inverse DFT on S_k,

$$\left(\mathscr{F}^{-1}[S_k]\right)_j = \frac{1}{p} \sum_{l=1}^{p} \langle \mathbf{x}_k, \mathbf{x}_l \rangle \omega^{jl} = \left\langle \mathbf{x}_k, \frac{1}{p} \sum_{l=1}^{p} \mathbf{x}_l \omega^{-jl} \right\rangle = \langle \mathbf{x}_k, \mathbf{w}_j \rangle = (\mathbf{u}_k)_j. \qquad \square$$

According to Lemma 3.1, each row S_k of S is the DFT of an appropriate vector $\mathbf{u}_k$. Since by assumption S_k is approximately sparse, we may thus find its set of large indices in sub-linear time by applying

the sFFT algorithm on the input $\mathbf{u}_k$. We then explicitly compute the corresponding entries in S_k directly from the original data $\mathbf{x}_1, \ldots, \mathbf{x}_p$.

Computing all p entries of all p vectors $\{\mathbf{u}_k\}$ requires a total of $O(np^2)$ operations. However, with this time complexity we could have computed all entries of the matrix S to begin with. The key point that makes this reduction applicable is that the sFFT algorithm is sub-linear in time and to compute its output, it reads at most $O(r\log(p/\delta)\log(p/r))$ coordinates of $\mathbf{u}_k$. In particular, it does not require *a priori* evaluation of all p entries of each vector $\mathbf{u}_k$. Hence, we may compute on-demand only the entries of $\mathbf{u}_k$ requested by the algorithm. Since computing a single entry of $\mathbf{u}_k$ can be done in $O(n)$ operations, the total number of operations required for a single sFFT run is $O(nr\log(p/\delta)\log(p/r))$.

To detect the large entries in all rows of S, all p outputs of the sFFT algorithm should simultaneously satisfy Equation (3.1) with high probability. Given that each sFFT run succeeds with probability $\geqslant \frac{2}{3}$, we show below that it suffices to invoke $m = O(\log p)$ independent queries of the sFFT algorithm on each input $\mathbf{u}_k$. The output for each row is the union of the large indices found by all sFFT runs.

In more detail, for each row k let $(J_1, \mathbf{y}_1), \ldots, (J_m, \mathbf{y}_m)$ be the outputs (indices and values) of sFFT on m independent runs with the same input $\mathbf{u}_k$. Our approximation for the kth row of S is then

$$\tilde{S}_{kj} = \begin{cases} S_{kj}, & j \in \bigcup_i J_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

The following lemma and corollary prove that $m = O(\log p)$ runs suffice to detect all large entries of S with a constant success probability.

LEMMA 3.2 Let $m = \lceil \log(3p)/\log(3) \rceil = O(\log(p))$. Then, for each $k \in [p]$ the following inequality holds with probability $\geqslant 1 - 1/3p$:

$$\left\| S_k - \tilde{S}_k \right\| \leqslant (1+\alpha) \min_{\|\mathbf{y}\|_0 \leqslant r} \|S_k - \mathbf{y}\| + \delta \|S_k\|.$$

COROLLARY 3.1 Let $m = \lceil \log(3p)/\log(3) \rceil$ as in the previous lemma. Then with probability $\geqslant \frac{2}{3}$, simultaneously, for all rows $k \in [p]$,

$$\|S_k - \tilde{S}_k\| \leqslant (1+\alpha) \min_{\mathbf{y}:\|\mathbf{y}\|_0 \leqslant r} \|S_k - \mathbf{y}\| + \delta \|S_k\|. \tag{3.4}$$

The proof of Corollary 3.1 follows from a standard union-bound argument, details omitted.

To conclude, we use multiple runs of the sFFT algorithm to find a set I of indices in S, whose entries S_{ij} may be potentially large, and which we then evaluate explicitly. The resulting approximation of S is compactly represented as $\{((i,j), S_{ij})\}_{(i,j)\in I}$. This procedure is summarized in Algorithm 1. The following Theorem, proved in the Appendix, provides a bound on its runtime and a guarantee for the accuracy of its output.

THEOREM 3.1 Assume that S is $(r, \mu, R, 2)$-sparse, where $\mu > 2R + \epsilon$ for known $R, \epsilon > 0$, and let $M = \max_i S_{ii}$. Then, Algorithm 1, which invokes the sFFT algorithm with parameters $\delta = \epsilon/\left(R + \sqrt{rM}\right)$ and $\alpha = 1$, has a runtime of $O\left(nrp \log^2 p \log\left(\left(R + \sqrt{rM}\right)p/\epsilon\right)\right)$ operations, and with probability $\geqslant \frac{2}{3}$, its output set I is guaranteed to include the set $J_\mu(S)$ of all large entries of S.

Algorithm 1 `sFFTCovEstimation`$(\mathbf{x}_1, \ldots, \mathbf{x}_p, r, R, \epsilon)$

Input:

$(\mathbf{x}_1, \ldots, \mathbf{x}_p)$: p vectors of dimension n.

r: bound on the number of large entries in each row.

R, ϵ: sparsity parameters.

Output: $\tilde{S}$: Compact representation of the large entries of S.

1: Compute $W = (\mathbf{w}_1, \ldots, \mathbf{w}_p) \in \mathbb{C}^{n \times p}$ using FFT, where $\mathbf{w}_j = \frac{1}{p} \sum_{l=1}^{p} \mathbf{x}_l \omega^{-jl}$
2: Set $I = \emptyset$
3: Compute all diagonal entries S_{ii} and the value $M = \max_i S_{ii}$
4: **for** $j = 1, \ldots, p$ **do**
5: Calculate $(J_1, \mathbf{y}_1), \ldots, (J_m, \mathbf{y}_m)$ by running $m = O(\log p)$ sFFT queries with input $\mathbf{u}_k = (\langle \mathbf{x}_k, \mathbf{w}_1 \rangle, \ldots, \langle \mathbf{x}_k, \mathbf{w}_p \rangle)$, $\delta = \frac{\epsilon}{R + \sqrt{rM}}$ and $\alpha = 1$
6: Add $\{k\} \times (\bigcup_{i=1}^{m} J_i)$ to I
7: **end for**
8: **for** $(i, j) \in I$ **do**
9: Calculate $S_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$
10: **end for**
11: **return** $\tilde{S} = \{((i, j), S_{ij})\}_{(i,j) \in I}$

4. Tree-based algorithm

We now present a second more direct method to efficiently detect and compute the large entries of a sparse covariance matrix. This method, which assumes that the threshold μ is *a priori* known, is based on a bottom-up construction of m random binary trees. To construct the lth tree, we first place at its p leaves the following n-dimensional vectors $\{\eta_{lj} \mathbf{x}_j\}_{j=1}^{p}$, where $\eta_{lj} \overset{i.i.d.}{\sim} N(0, 1)$. Then, at higher levels, the n-dimensional vector in each parent node is the sum of its offspring. After the construction of the trees, the main idea of the algorithm is to make recursive coarse-to-fine statistical group tests, where all indices of various subsets $\mathscr{A} \subseteq [p]$ are simultaneously tested whether they contain at least one large entry of S or not.

In more detail, given as input a row $k \in [p]$ and a set $\mathscr{A} \subseteq [p]$, we consider the following query or hypothesis testing problem:

$$\mathscr{H}_0 : J_\mu(S_k) \cap \mathscr{A} = \emptyset \quad \text{vs.} \quad \mathscr{H}_1 : J_\mu(S_k) \cap \mathscr{A} \neq \emptyset.$$

Assuming that S is $(r, \mu, R, 2)$-sparse, with $R < \mu/2$, one way to resolve this query is by computing the following quantity:

$$F(k, \mathscr{A}) = \sum_{j \in \mathscr{A}} \langle \mathbf{x}_k, \mathbf{x}_j \rangle^2. \tag{4.1}$$

Indeed, under $\mathscr{H}_0$, $F(k, \mathscr{A}) \leqslant R^2$, whereas under $\mathscr{H}_1$, $F(k, \mathscr{A}) \geqslant \mu^2$.

A direct calculation of (4.1) requires $O(n|\mathscr{A}|)$ operations, which may reach up to $O(np)$ when $|\mathscr{A}|$ is large. Instead, the algorithm *estimates* the value of $F(k, \mathscr{A})$ in Equation (4.1) using m i.i.d. samples $\{y_l\}_{l=1}^{m}$ of the random variable

$$Y(\mathscr{A}) = \left\langle \mathbf{x}_k, \sum_{j \in \mathscr{A}} \eta_j \mathbf{x}_j \right\rangle, \tag{4.2}$$

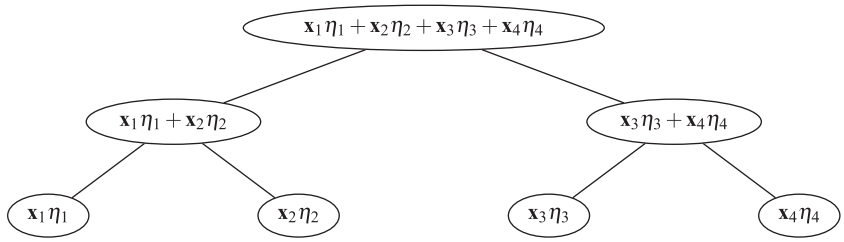

FIG. 1. Illustration of a random tree, for $p = 4$.

where $(\eta_1, \ldots, \eta_p) \sim N(0, I_p)$. Since by definition $\mathbb{E}Y(\mathscr{A}) = 0$ and $\mathrm{Var}[Y(\mathscr{A})] = F(k, \mathscr{A})$, Equation (4.1) can be estimated by the empirical variance of the m variables $\{y_l\}_{l=1}^m$. Again, given a specific realization of η_js, direct calculation of Equation (4.2) also requires $O(n|\mathscr{A}|)$ operations. Here, our construction of m binary trees comes into play, as it efficiently provides us samples of $\sum_{j\in\mathscr{A}} \mathbf{x}_j\eta_j$, for any subset of the form $\mathscr{A} = \{2^h i, \ldots, 2^h(i+1)\}$, where $h \in [\log_2 p]$ and $i \in [p/2^h]$. As described in Section 4.2 below, after the preprocessing stage of constructing the m trees, each query requires only $O(nm)$ operations. Moreover, we show that $m = O(\log p)$ trees suffice for our purpose, leading to $O(n \log p)$ query time.

To find large entries in the kth row, a divide and conquer method is applied: start with $\mathscr{A} = \{1, \ldots, p\}$, and check if $F(k, \mathscr{A})$ of Equation (4.1) is large by invoking an appropriate query. If so, divide $\mathscr{A}$ into two disjoint sets and continue recursively. With this approach, we reduce the number of required operations for each row from $O(np)$ to $O(nr \log^2 p)$, leading to an overall time complexity of $O(nrp \log^2 p)$.

4.1 *Preprocessing stage: constructing the trees*

To efficiently construct the m trees, first mp i.i.d. samples $\{\eta_{lj}\}$ are generated from a Gaussian distribution $N(0, 1)$. For simplicity, we assume that $p = 2^L$, for some integer L, leading to a full binary tree of height $L + 1$. Starting at the bottom, the value at the jth leaf in the lth tree is set to be n-dimensional vector $\eta_{lj}\mathbf{x}_j$. Then, in a bottom-up construction, the vector of each node is the sum of its two offspring (see Fig. 1). Since, each tree has $2p - 1$ nodes, and calculating the vector of each node requires $O(n)$ operations, the construction of m i.i.d. random trees requires $O(nmp)$ operations.

For future use, we introduce the following notation: for a given tree T, we denote by $T(h, i)$ the ith node at the hth level, where the root is considered to be at level zero. Furthermore, we denote by $I(h, i)$ the set of indices corresponding to the leaves of the sub-tree starting from $T(h, i)$. The vector stored at the node $T(h, i)$ is the following n-dimensional random variable:

$$\mathrm{Val}(T(h, i)) = \sum_{j\in I(h,i)} \eta_j \mathbf{x}_j$$

whose randomness comes only from the random variables η_j (we here consider the samples $\{\mathbf{x}_j\}$ as fixed). The entire tree construction procedure is described in Algorithm 2.

4.2 *Algorithm description*

As mentioned before, we assume that the threshold μ is an input parameter to the algorithm. Given a set $\mathscr{A} = I(h, i)$ and a vector $\mathbf{x}_k$, to estimate (4.1), the algorithm uses m i.i.d. samples of (4.2) obtained from

Algorithm 2 `ConstructTrees`$(\mathbf{x}_1, \ldots, \mathbf{x}_p, m)$

Input:
$(\mathbf{x}_1, \ldots, \mathbf{x}_p)$: p vectors of dimension n.
m: number of trees.

Output: m random binary trees $(T_1, \ldots, T_m)$.

Generate η_{lj}, for $j \in [p]$ and $l \in [m]$, where $\eta_{lj} \overset{i.i.d.}{\sim} N(0,1)$
for $l = 1, \ldots, m$ **do**
  for $j = 1, \ldots, p$ **do**
    Set $Val(T_l(L, j)) = \eta_{lj}\mathbf{x}_j$
  end for
  for $h = L-1, \ldots, 0$ **do**
    for $i = 1, \ldots, 2^h$ **do**
      Set $Val(T_l(h,i)) = Val(T_l(h+1, 2i)) + Val(T_l(h+1, 2i+1))$
    end for
  end for
end for

the relevant node $T(h,i)$ of the tree. For this, let $y_1, \ldots, y_m$ be the m i.i.d. samples of $Y = Y(I(h,i))$,

$$y_l = \langle \mathbf{x}_k, \mathrm{Val}(T_l(h,i)) \rangle = \sum_{j \in I(h,i)} \langle \mathbf{x}_k, \mathbf{x}_j \rangle \eta_{lj}.$$

Since $\mathbb{E}Y = 0$ and

$$\sigma^2 := \mathrm{Var}(Y) = \sum_{j \in I(h,i)} \langle \mathbf{x}_k, \mathbf{x}_j \rangle^2,$$

a natural estimator for (4.1) is the sample variance

$$\hat{\sigma}^2 := \frac{1}{m} \sum_{l=1}^{m} y_l^2.$$

Recall that, for a matrix S which is $(r, \mu, R, 2)$-sparse with $R < \mu/2$, the exact value of Equation (4.1) allows us to perfectly distinguish between the following two hypotheses:

$$\mathscr{H}_0 : I(h,i) \cap J_\mu(S_k) = \emptyset \quad \text{vs.} \quad \mathscr{H}_1 : I(h,i) \cap J_\mu(S_k) \neq \emptyset. \tag{4.3}$$

Given the estimate $\hat{\sigma}^2$, the algorithm considers the following test:

$$\hat{\sigma}^2 \underset{\mathscr{H}_0}{\overset{\mathscr{H}_1}{\gtrless}} \frac{3\mu^2}{4}. \tag{4.4}$$

If $\hat{\sigma}^2 \geqslant 3\mu^2/4$, the algorithm continues recursively in the corresponding sub-tree. Lemma 4.1 shows that $m = O(\log(1/\delta))$ trees suffice to correctly distinguish between the two hypotheses with probability at least $1 - \delta$.

In summary, the algorithm detects large entries in the kth row by processing the m random trees with the vector $\mathbf{x}_k$; starting from the root, it checks if one of its children contains a large entry using the

query presented in Equation (4.4), and continues recursively as required. This recursive procedure is described in Algorithm 3. Then, as described in Algorithm 4, the complete algorithm to detect all large entries of a sparse covariance matrix applies Algorithm 3 separately to each row. As in Algorithm 1, the output of Algorithm 4 is a compact representation of the large entries of the matrix, as a set of indices $I \subset [p] \times [p]$ and their corresponding entries $\{S_{ij}\}_{(i,j)\in I}$.

Algorithm 3 `Find`$(\mathbf{x}, h, i, m, \{T_l\}, \mu)$

Input:
$\mathbf{x}$: input vector of dimension n.
(h, i): tree-node index.
m: number of trees.
$\{T_l\}$: a collection of m trees.
μ: threshold parameter.

Output: The set of large entries in the current sub-tree.

if $h = L$ **then**
  return $\{i\}$
end if
Set $a = \frac{1}{m}\sum_{l=1}^{m}\langle \mathbf{x}, Val(T_l(h+1, 2i))\rangle^2$, $b = \frac{1}{m}\sum_{l=1}^{m}\langle \mathbf{x}, Val(T_l(h+1, 2i+1))\rangle^2$
Set $\mathcal{S}_a = \emptyset, \mathcal{S}_b = \emptyset$
if $a \geqslant \frac{3\mu^2}{4}$ **then**
  $\mathcal{S}_a =$ `Find`$(\mathbf{x}, h+1, 2i, m, \{T_l\}, \mu)$
end if
if $b \geqslant \frac{3\mu^2}{4}$ **then**
  $\mathcal{S}_b =$ `Find`$(\mathbf{x}, h+1, 2i+1, m, \{T_l\}, \mu)$
end if
return $\mathcal{S}_a \cup \mathcal{S}_b$

Algorithm 4 `SparseCovTree`$(\mathbf{x}_1, \ldots, \mathbf{x}_p, m, \mu)$

Input:
$(\mathbf{x}_1, \ldots, \mathbf{x}_p)$: p vectors of dimension n.
m: number of trees.
μ: threshold parameter.

Output: $\tilde{S}$: compact representation of large entries of S.

Construct m trees: $\{T_l\} =$ `ConstructTrees`$(\mathbf{x}_1, \ldots, \mathbf{x}_p, m)$
Set $I = \emptyset$
for $k = 1, \ldots, p$ **do**
  Add $\{k\} \times$ `Find`$(\mathbf{x}_k, 0, 1, m, \{T_l\}, \mu)$ to I
end for
for $(i, j) \in I$ **do**
  Calculate $S_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$
end for
return $\{((i, j), S_{ij})\}_{(i,j)\in I}$

4.3 *Theoretical analysis of the tree-based algorithm*

Two key questions related to Algorithm 4 are: (i) can it detect with high probability large entries of S; and (ii) what is its runtime. In this section, we study these questions under the assumption that the matrix S is approximately sparse. We start our analysis with the following lemma, proved in the Appendix, which shows that $m = O(\log(1/\delta))$ trees suffice for the test in Equation (4.4) to succeed with probability at least $1 - \delta$.

LEMMA 4.1 Assume that S is $(r, \mu, R, 2)$-sparse, where $R < \mu/2$. Then, for $m \geqslant 64 \log(1/\delta)$,

$$\Pr[\textbf{false alarm}] = \Pr\left[\hat{\sigma}^2 \geqslant \frac{3\mu^2}{4} \,\middle|\, \mathscr{H}_0\right] \leqslant \delta,$$

$$\Pr[\textbf{misdetection}] = \Pr\left[\hat{\sigma}^2 < \frac{3\mu^2}{4} \,\middle|\, \mathscr{H}_1\right] \leqslant \delta.$$

REMARK 4.1 While the constant of 64 is not sharp, the logarithmic dependence on δ is.

REMARK 4.2 The choice of η to be Gaussian is rather arbitrary. Standard concentration inequalities (Vershynin, 2010) imply that every sub-Gaussian distribution with zero mean and variance 1 will yield similar results, albeit with different constants.

Next, assuming that S is approximately sparse, Theorem 4.1 below shows that with high probability, Algorithm 4 indeed succeeds in finding all large entries of S. Most importantly, its runtime is bounded by $O(nrp \log^2 p)$ operations, both with high probability and in expectation.

THEOREM 4.1 Assume that S is $(r, \mu, R, 2)$-sparse, where $R < \mu/2$. Let $I(m)$ be the set of indices returned by Algorithm 4 with threshold μ and m trees. For a suitably chosen constant C that is independent of n, m, r and p, define

$$f(p, m) = \Pr\left[J_\mu(S) \subseteq I(m) \text{ and Runtime} \leqslant Cnpr \log^2 p\right].$$

Then, for $m(p) = \lceil 64 \log(2rpL^3) \rceil$, where $L = \log_2 p + 1$, the following conditions satisfy:

(i) For all $p \geqslant 8$, $f(p, m(p)) \geqslant \frac{2}{3}$.

(ii) $f(p, m(p)) \xrightarrow{p \to \infty} 1$.

(iii) $\mathbb{E}[\text{Runtime}] \leqslant C' nrp \log^2 p$ for some absolute constant C'.

REMARK 4.3 The sparsity of S is required only to bound the false alarm probability in Lemma 4.1. If S is not necessarily sparse, the algorithm will still locate with high probability all of its large entries. However, its runtime can increase significantly, up to $O(np^2 \log^2 p)$ operations in the worst case when $r = p$.

REMARK 4.4 Note that since our tree-based algorithm analyzes each row of S separately, in fact S need not be globally $(r, \mu, R, 2)$-sparse. Our algorithm is still applicable to a matrix S with k non-sparse rows and all other rows $(r, \mu, R, 2)$-sparse. On the non-sparse rows our algorithm will be quite slow, and would analyze all the leaves of the tree. However, on the sparse rows, it would detect the large entries in time $O(npr \log^2 p)$. If $k = O(\log p)$, the overall run time is, up to logarithmic factors in p, still $O(npr)$.

TABLE 1 *Comparison between the sFFT-based and tree-based algorithms*

	sFFT-based	SparseCovTree
Sparsity assumptions	$(r, \mu, R, 2)$-sparse with $\mu > 2R + \epsilon$	$(r, \mu, R, 2)$-sparse with $\mu > 2R$
Runtime bound	$O\big(nrp \log^2 p \log\big(\big(R + \sqrt{r}M\big)p/\epsilon\big)\big)$	$O(nrp \log^2 p)$
Probability to detect all large entries	$\geqslant \frac{2}{3}$	$\geqslant \frac{2}{3}$
Required input parameters	r, R, ϵ and $M = \max\{S_{ii}\}$	μ only
Dependencies on other algorithms	Based on the sFFT algorithm	Standalone

The runtime guarantees of Theorem 4 are probabilistic ones, where the algorithm runtime can reach up to $O(np^2 \log^2 p)$ operations, even when S is sparse. For an *a priori* known upper bound on the sparsity parameter r, one can slightly modify the algorithm to stop after $O(nrp \log^2 p)$ operations. This modification may decrease the probability to detect all large entries, but still maintain it to be at least $\frac{2}{3}$. This is true, since the event of finding all large entries contains the event of finding all large entries using a bounded number of operations, whereas the second event is not affected by the modification.

4.4 *Comparison between the sFFT-based and tree-based algorithms*

Table 1 compares the main properties of the sFFT-based and the tree-based algorithms. Interestingly, even though the two algorithms are very different, their success relies on precisely the same definition of matrix sparsity. Moreover, the difference in the input parameters is not significant since knowledge of R yields a lower bound for μ, and vice versa. Although the tree-based algorithm has a lower runtime complexity, and is easier to understand and implement, the sFFT-based algorithm is still of interest as it illustrates a connection between two different problems which seem unrelated at first glimpse; estimating the sparse Fourier transform of a given signal and detecting large entries of a sparse covariance matrix. Hence, advances in sFFT can translate to faster sparse covariance estimation.

5. Relation between sample size, sparsity of S and of Σ

The theoretical analysis in the previous two sections assumed that the sample covariance matrix S is approximately sparse. Typically, however, one may only assume that the population matrix Σ is sparse, whereas S is computed from a sample of n observations $\mathbf{z}_1, \ldots, \mathbf{z}_n$. In general, this may lead to a non-sparse matrix S, even when Σ is sparse. Nonetheless, we show below that if the underlying random variable, Z is sub-Gaussian with a sparse covariance matrix Σ, then given a sufficient number of samples $n = O(p \log p)$, with high probability the corresponding sample covariance matrix S is also approximately sparse, albeit with different parameters.

To this end, recall that a random variable Y is said to be sub-Gaussian if

$$\|Y\|_{\psi_2} := \sup_{q \geqslant 1} q^{-1/2} (\mathbb{E}|Y|^q)^{1/q} < \infty$$

and, similarly, a random variable Y is said to be sub-exponential if

$$\|Y\|_{\psi_1} := \sup_{q \geqslant 1} q^{-1} (\mathbb{E}|Y|^q)^{1/q} < \infty.$$

See, for example, Vershynin (2010). Lastly, a random vector $Z = (Z_1, \ldots, Z_p)$ is said to be sub-Gaussian if Z_i is sub-Gaussian for every $i \in [p]$.

The following theorem provides us sufficient conditions on Σ, the underlying random variable Z and the number of samples n to ensure, with high probability, the approximate sparsity of S.

THEOREM 5.1 Assume that Z is a sub-Gaussian random vector with covariance matrix Σ, which is $(r, \mu, R, 2)$-sparse where $2R < \mu$. Let $K = \max_i \|Z_i\|_{\psi_2}$ and $t = \min\{(\mu - 2R)/(2\sqrt{p - r} + 1), (\mu - R)/4, K^2\}$. Then, for $n > C(K^4/t^2)\log(54p^2)$, where C is an absolute constant, with probability $\geqslant \frac{2}{3}$, S is $(r, \mu - t, \frac{1}{2}(\mu - t), 2)$-sparse. Moreover, with high probability, every large entry of Σ (w.r.t. μ) is a large entry of S (w.r.t. $\mu - t$), and vice versa.

In addition to the probabilistic guarantees of Theorem 5.1 for fixed p and n, we can also deduce stronger asymptotical results, as $n, p \to \infty$.

THEOREM 5.2 Assume that Z is a sub-Gaussian random vector with covariance matrix Σ, which is $(r, \mu, R, 2)$-sparse where $2R < \mu$, and let t be as in Theorem 5.1. Then, as $n \to \infty$ with p fixed, or as $n, p \to \infty$ with $p \log p/n \to 0$, the probability that S is $(r, \mu - t, \frac{1}{2}(\mu - t), 2)$-sparse with $J_{\mu - t}(S) = J_\mu(\Sigma)$ converges to one.

Note that, for large $p \gg 1$, the parameter t in Theorem 5.1 is equal to $(\mu - 2R)/(2\sqrt{p - r + 1})$. Hence, the required number of samples for S to be approximately sparse is $n > O(\log(p)/t^2) = O(p \log p)$. With such a number of samples, we can replace the assumptions on S with corresponding assumptions on Σ, and obtain the following result analogous to Theorem 4.1.

COROLLARY 5.1 Assume that Σ is $(r, \mu, R, 2)$-sparse, where $R < \mu/2$. Then, as $n, p \to \infty$ with $p \log p/n \to 0$, with probability tending to 1, Algorithm 4 finds all large entries of S, which correspond to large entries of Σ, using at most $O(nrp \log^2 p)$ operations.

In various contemporary statistical applications, the number of samples n is comparable to the dimension p. In such a case, we may still detect the large entries of the population covariance matrix in sub-quadratic time. For example, we can divide the p variables into $K = p^{1-\alpha}$ distinct groups, each of size p^α, and separately find the largest entries in each of the K^2 sub-matrices of the full covariance matrix. In each pair, Corollary 5.1 applies, since if $p, n \to \infty$ with $p/n \to const$, then $p^\alpha \log p/n \to 0$. The overall runtime, up to logarithmic factors in p is now higher $O(np^{2-\alpha} r)$, but still sub-quadratic.

Finally, note that throughout this section we assumed that the underlying random variable Z is sub-Gaussian. It is an interesting question whether some of the above theorems continue to hold under weaker tail conditions.

6. Simulations

In this section, we illustrate the empirical performance of Algorithm 4, denoted by SparseCovTree, on input drawn from a Gaussian distribution with a sparse covariance matrix. Furthermore, we compare it with other algorithms that can be used to locate the large entries of S: (1) LSH-kNN (Datar *et al.*, 2004); (2) kd-tree (Bentley, 1975); (3) Dual-Ball-Ball (DBB) (Ram & Gray, 2012); (4) Dual-Ball-Cone (DBC) (Ram & Gray, 2012); and (5) direct calculation of all entries of S, at a cost of $O(np^2)$ operations. To the best of our knowledge, no public implementation of the sFFT algorithm of Hassanieh *et al.* (2012) is currently publicly available, thus we did not include it in the comparison. For the LSH-kNN and kd-tree algorithms, we used the mlpack library (Curtin *et al.*, 2013), whereas for DBB and DBC algorithms we

used the implementation kindly provided to us by Ram & Gray (2012). All codes are in C++ and were compiled with the O3 optimization flag.

We generated random population covariance matrices Σ as follows: first $r = \lfloor \log_2 p/3 \rfloor$ entries in each row (and their corresponding transposed entries) were selected uniformly to be ± 1, with all others set to zero. Then, the diagonal entries were set to be ± 1 as well. Lastly, to have a valid positive-definite covariance matrix, all diagonal entries were increased by the absolute value of the smallest eigenvalue of the resulting symmetric matrix plus one. Following this, we chose the threshold to be $\mu = 0.5$.

The space complexity of the SparseCovTree algorithm, which involves storing m trees, is $O(npm)$. This may exceed the available memory for large n, p and m. Thus, instead of simultaneously constructing and processing the entire m trees from the coarse level, we can construct and process the sub-trees of each node in the different fine levels separately, starting from some coarse level $h \in [\log_2 p]$. By doing so, the space complexity is reduced by a factor of 2^h. These modifications can only increase the probability to locate large entries, whereas the theoretical runtime bound of Theorem 4.1 increases to $O(nmp(\log_2 p - h)2^h)$ operations. Particularly, in our experiments we used $h = 5$, reducing the space complexity by a factor of 32. Moreover, the value of m as stated in Theorem 4.1 is required mostly for the theoretical analysis. Since the multiplicative factor appearing in Lemma 4.1 is not sharp, and the probability of failure converges to zero as the dimension increases, we can in practice use a smaller number of trees m and still detect most large entries of S, in sub-quadratic time. Here, we chose a fixed value of $m = 20$, leading to discovery of more than 99% of the large entries for all values of p considered.

As for the LSH-kNN tuning parameters, we chose the number of projections to be 10, hash width 4 and bucket size 3500. For the second hash size, we picked a large prime number 424577, whereas the number of tables was configured according to Slaney & Casey (2008) with $\delta = 1/\log p$. This configuration led to a more than 99% discovery rate of all large entries, in all tested dimensions.

Figure 2 shows the average runtime, in logarithmic scale, for various values of p from 1000 to 10,000. Surprisingly, all alternative solutions, apart from SparseCovTree, yield slower runtime performances than the direct calculation. We raise here several possible explanations for this. First, in all methods, except SparseCovTree and direct calculation, to locate negative large entries of S, we duplicate the input $\{\mathbf{x}_i\}$ with the opposite sign, thus potentially increasing the runtime by a factor of 2. Moreover, it was suggested (see Shrivastava & Li, 2014) that space-partitioning-based methods, such as kd-tree, DBB and DBC, may lead to slow runtime in high dimensions. For an empirical illustration of this issue, see Shrivastava & Li (2015). In our case, the dimension of the search problem is the number of samples n, which may indeed be very high. In contrast to these algorithms, the LSH method was originally designed to cope well in high dimensions. However, to locate most of $O(pr)$ large entries of S, we tuned the LSH-kNN algorithm to have a small misdetection probability, by setting $\delta = 1/\log p$. Such a small value of δ led to a large number of tables, thus increasing the runtime by a significant factor.

As for SparseCovTree, in low dimensions direct calculation is preferable. However, for larger values of p, SparseCovTree is clearly faster. Moreover, the slope on a log–log scale of the direct approach is roughly 2, whereas our method has a slope closer to 1.

In the above simulation, the underlying population matrix Σ was exactly sparse. Next, we empirically study the ability of the SparseCovTree algorithm to detect the large entries when the underlying population covariance matrix is not perfectly sparse, but only approximately so. To this end, we generated a population covariance matrix similar to the procedure described above, only that entries that previously were zero, are now $\pm\epsilon$ with equal probability. Figure 3 presents both the average runtime as well as the misdetection rate of our algorithm with $m = 10, 25, 50$ trees, as a function of ϵ, for a covariance matrix of size $p = 2000$ and $n = 20{,}000$ samples. Each row of Σ has an average of $r = 8$

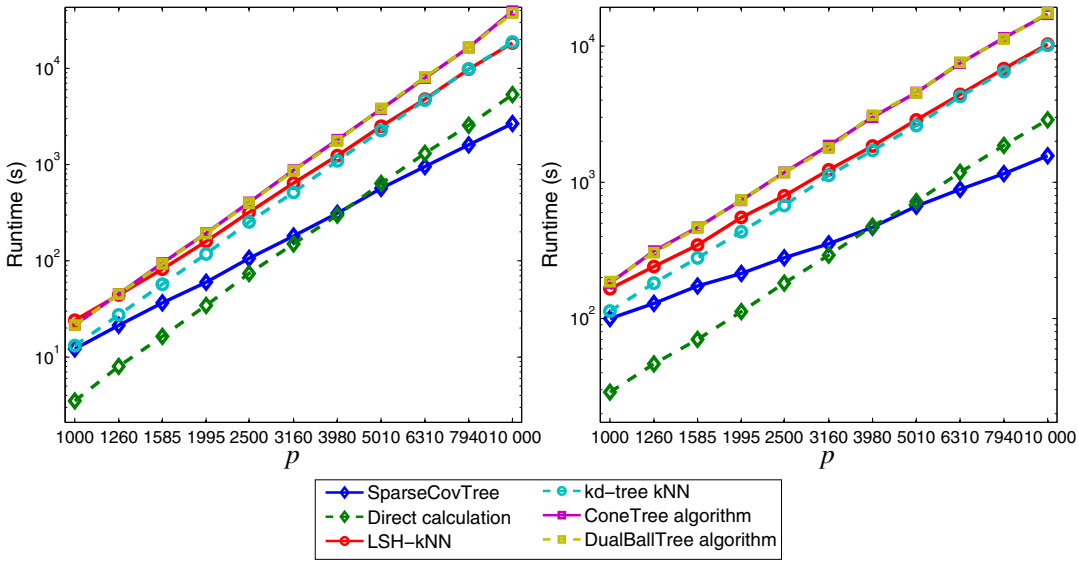

FIG. 2. Average runtime to detect large entries of a sparse covariance matrix as a function of the dimension for several algorithms, presented in a logarithmic scale in both axes. The input data were drawn from a Gaussian distribution with a random population covariance matrix and $r = \lfloor \log_2 p/3 \rfloor$. In the left panel, the number of samples increases with the dimension, $n = \lfloor p \log p \rfloor$, whereas in the right panel the number of samples is fixed, $n = 50{,}000$.

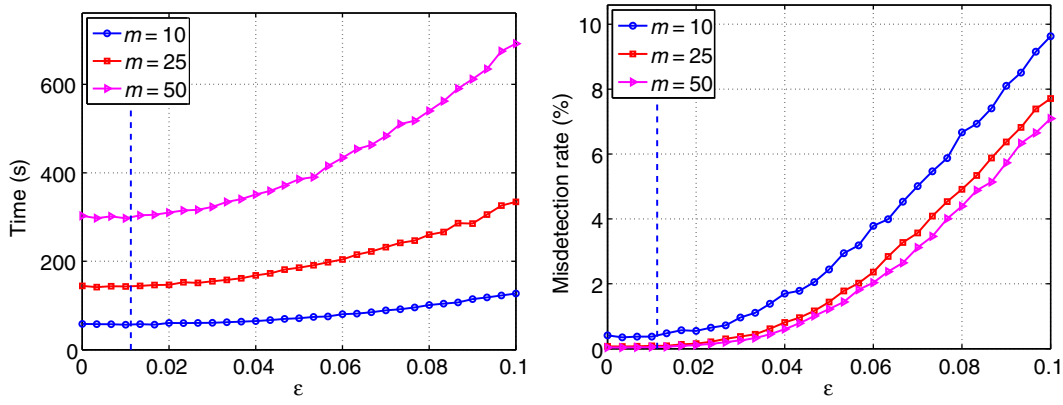

FIG. 3. Average runtime and misdetection rate (in percentage points) as a function of ϵ, for fixed $p = 2000$ and $n = 20{,}000$. The dashed vertical line is the critical value ϵ_{crit} where, for the population matrix, $R(\epsilon_{\text{crit}}) = \mu/2$.

large entries, all of size $\mu = 1$. Hence, $R = R(\epsilon) \approx (p - r)\epsilon^2$. The critical value ϵ_{crit} at which $R = \mu/2$ is thus $\epsilon_{\text{crit}} = 1/\sqrt{4(p - r)}$. As seen in the left panel, the runtime scales roughly linearly with the number of trees is nearly constant for $\epsilon < \epsilon_{\text{crit}}$ and slowly increases for larger values of ϵ. The right panel shows that, in accordance to our theory, the misdetection rate is also nearly constant as long as $\epsilon < \epsilon_{\text{crit}}$.

Finally, we demonstrate an application of the SparseCovTree algorithm to detect near duplicates in a large artificial dataset: Given a reference dataset with p elements, $\{\mathbf{x}_i\}_{i=1}^{p}$, with each $\mathbf{x}_i \in \mathbb{R}^n$, the goal is to quickly decide, for any query vector $\mathbf{y} \in \mathbb{R}^n$, whether there exists $\mathbf{x}_i$ such that $Sim(\mathbf{x}_i, \mathbf{y})$ is larger

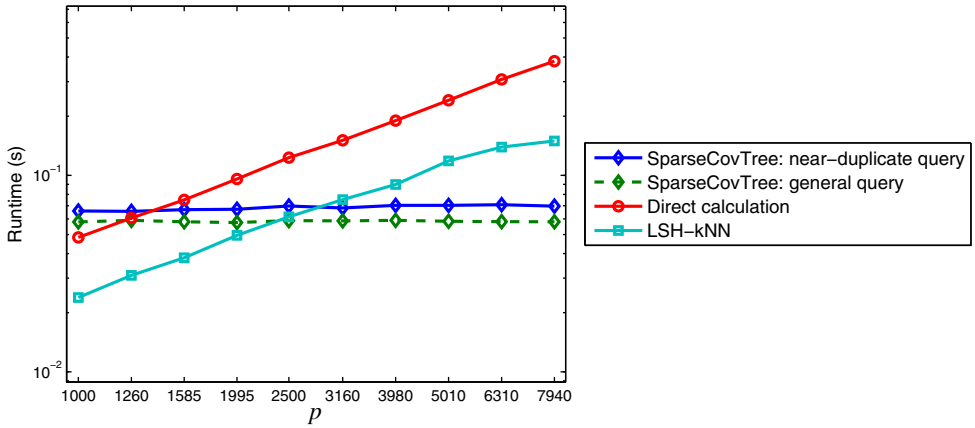

FIG. 4. Average runtime per single query to detect near-duplicates in a large artificial data set with p elements of dimension $n = 40{,}000$, presented in a logarithmic scale in both axes. For SparseCovTree, we present the average runtime both for a near-duplicate and for a general input query. For the other two methods, this partition did not affect the runtime significantly.

than a given threshold μ, where *Sim* is the *Cosine Similarity*. In contrast to the previous simulation, here the key quantity of interest is the average query time, and the computational effort of the preprocessing stage is typically ignored.

Specifically, we considered the following illustrative example: We first generated a reference set $\{\mathbf{x}_i\}_{i=1}^{p}$, whose p elements were all independently and uniformly distributed on the unit sphere S^{n-1}. Next, we evaluated the average runtime per query in two scenarios: (a) a general query $\mathbf{y}$, uniformly distributed on S^{n-1}, and hence with high probability, not a near-duplicate; (b) a near-duplicate query $\mathbf{u}$, generated as follows:

$$\mathbf{u} = \sqrt{1-\epsilon}\mathbf{x}_j + \sqrt{\epsilon}\frac{\mathbf{z} - \left(\mathbf{z}^T\mathbf{x}_j\right)\mathbf{x}_j}{\left\|\mathbf{z} - \left(\mathbf{z}^T\mathbf{x}_j\right)\mathbf{x}_j\right\|},$$

where $\mathbf{z}$ is uniformly distributed on S^{n-1} and the duplicate index j is chosen uniformly from the set of elements $[p]$.

In our simulations, the parameters $\epsilon = 0.25$, $n = 40{,}000$ and $m = 20$ trees were kept fixed, and we varied the size p of the reference set. For a fair comparison to LSH, we decreased its number of projections to 7, and set its misdetection probability per single projection to be $\delta = 0.3$. Empirically, both our method and LSH correctly detected a near-duplicate with a probability rate larger than 99%.

Figure 4 shows the average query runtime for SparseCovTree, LSH-kNN and the direct approach. In both LSH-kNN and direct approach, the average query runtime did not depend on whether the query was a near-duplicate or not, and hence we show only one of them. For SparseCovTree, in contrast, the average runtime for a general query is lower than for a near-duplicate query. The reason is that for a general query our algorithm detects that this is not a near-duplicate by processing only the nodes at the first few top levels of the tree. For this problem, the LSH method is clearly preferable over direct calculation for all tested values of p. When the number of elements p becomes sufficiently large, our method becomes faster. Most importantly, the runtime curves of SparseCovTree, both for the true and false near-duplicates, seem *almost constant* compared with the other runtime curves, which are approximately linear in p. This is consistent with the theoretical analysis, as SparseCovTree query time is $O(n \log^2 p)$ operations, which is sub-linear, whereas the direct calculation query time is $O(np)$ operations, which

is linear, thus significantly larger, and LSH-kNN query time is $O(np^{\rho}\,\mathrm{poly}\log p)$ operations for some constant $\rho > 0$, which is also sub-linear, but not logarithmic as the SparseCovTree query time.

7. Summary

In this paper, we considered the computational aspects of detecting the large entries of a sparse covariance matrix from given samples. We derived and theoretically analyzed two different algorithms for this task, under sparsity assumptions either on S itself, or on Σ and additional requirements on the underlying random variable Z and number of samples n. We next demonstrated the time efficiency of the algorithms, theoretically for both of them and empirically only for the tree-based one.

Our work raises several interesting questions for future research. If an oracle gave us the precise locations of all large entries, computing them directly would require $\Omega(npr)$ operations, a quantity lower by a poly $\log p$ factor from our results. This raises the following fundamental question: is it possible to reduce the poly-log factor, or is there a theoretical lower bound on the required number of operations? In addition, in this article we only considered an L_2-norm in the definition of matrix sparsity. It may be of interest to consider different norms, e.g. L_1, which could lead to different algorithmic results. Finally, both algorithms require partial knowledge on the sparsity parameters. It is thus of interest to efficiently estimate these parameters from the data, in case they are unknown.

Acknowledgements

We thank Robert Krauthgamer and Ohad Shamir for interesting discussions. We also thank Parikshit Ram and Alexander Gray for providing us with the code of their work and for various additional references. Finally, we also thank Rasmus Pagh and Anshumali Shrivastava for several references. This work was partially supported by a grant from the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).

References

Akavia, A. (2010) Deterministic sparse Fourier approximation via fooling arithmetic progressions. *23rd conference on learning theory (COLT)*, pp. 381–393.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R. & Wu, A. Y. (1998) An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, **45**, 891–923.

Bayardo, R. J., Ma, Y. & Srikant, R. (2007) Scaling up all pairs similarity search. *Proceedings of the 16th International Conference on World Wide Web*. New York: ACM, pp. 131–140 .

Bentley, J. L. (1975) Multidimensional binary search trees used for associative searching. *Commun. ACM*, **18**, 509–517.

Bickel, P. J. & Levina, E. (2008) Covariance regularization by thresholding. *Ann. Statist.*, 2577–2604.

Bien, J. & Tibshirani, R. J. (2011) Sparse estimation of a covariance matrix. *Biometrika*, **98**, 807–820.

Birnbaum, A. & Nadler, B. (2012) High-dimensional sparse covariance estimation: accurate thresholds for the maximal diagonal entry and for the largest correlation coefficient. *Technical report*. Weizmann Institute of Science.

Boucheron, S., Lugosi, G. & Massart, P. (2013) *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford: Oxford University Press.

Butte, A. J., Tamayo, P., Slonim, D., Golub, T. R. & Kohane, I. S. (2000) Discovering functional relationships between RNA expression and chemotherapeutic susceptibility using relevance networks. *Proc. Natl. Acad. Sci.*, **97**, 12182–12186.

Cai, T. & Liu, W. (2011) Adaptive thresholding for sparse covariance matrix estimation. *J. Amer. Statist. Assoc.*, **106**, 672–684.

Charikar, M. S. (2002) Similarity estimation techniques from rounding algorithms. *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*. New York: ACM, pp. 380–388.

Chaudhuri, S., Drton, M. & Richardson, T. S. (2007) Estimation of a covariance matrix with zeros. *Biometrika*, **94**, 199–216.

Curtin, R. R., Cline, J. R., Slagle, N. P., March, W. B., Ram, P., Mehta, N. A. & Gray., A. G. (2013) Mlpack: a scalable c++ machine learning library. *J. Mach. Learn. Res.*, **14**, 801–805.

Dasarathy, G., Shah, P., Bhaskar, B. N. & Nowak, R. (2013) Sketching sparse matrices. Preprint, 2013, arXiv:1303.6544.

Datar, M., Immorlica, N., Indyk, P. & Mirrokni, V. S. (2004) Locality-sensitive hashing scheme based on p-stable distributions. *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. New York: ACM, pp. 253–262.

Drineas, P., Kannan, R. & Mahoney, M. W. (2006) Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM J. Comput.*, **36**, 132–157.

Dubiner, M. (2010) Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Trans. Inform. Theory*, **56**, 4166–4179.

El Karoui, N. (2008a) Operator norm consistent estimation of large-dimensional sparse covariance matrices. *Ann. Statist.*, **36**, 2717–2756.

El Karoui, N. (2008b) Spectrum estimation for large-dimensional covariance matrices using random matrix theory. *Ann. Statist.*, 2757–2790.

Gilbert, A. C., Guha, S., Indyk, P., Muthukrishnan, S. & Strauss, M. (2002) Near-optimal sparse Fourier representations via sampling. *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*. New York: ACM, pp. 152–161.

Har-Peled, S., Indyk, P. & Motwani, R. (2012) Approximate nearest neighbor: towards removing the curse of dimensionality. *Theory Comput.*, **8**, 321–350.

Hassanieh, H., Indyk, P., Katabi, D. & Price, E. (2012) Nearly optimal sparse Fourier transform. *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*. New York: ACM, pp. 563–578.

Iwen, M. A. (2010) Combinatorial sub-linear time Fourier algorithms. *Found. Comput. Math.*, **10**, 303–338.

Iwen, M. A. & Spencer, C. V. (2009) A note on compressed sensing and the complexity of matrix multiplication. *Inform. Process. Lett.*,**109**, 468–471.

Johnstone, I. M. & Lu, A. Y. (2009) On consistency and sparsity for principal components analysis in high dimensions. *J. Amer. Statist. Assoc.*, **104**, 682–693.

Jones, P. W., Osipov, A. & Rokhlin, V. (2013) A randomized approximate nearest neighbors algorithm. *Appl. Comput. Harmon. Anal.*, **34**, 415–444.

Laurent, B. & Massart, P. (2000) Estimation of a quadratic functional by model selection. *Ann. Statist.*, **28**, 1302–1338.

Pagh, R. (2013) Compressed matrix multiplication. *ACM Trans. Comput. Theory (TOCT)*, **5**, 9.

Ram, P. & Gray, A. G. (2012) Maximum inner-product search using cone trees. *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York: ACM, pp. 931–939.

Rudelson, M. & Vershynin, R. (2013) Hanson–Wright inequality and sub-gaussian concentration. Preprint, 2013, arXiv:1306.2872.

Shakhnarovich, G., Indyk, P. & Darrell, T. (2006) *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. Cambridge, MA: MIT Press.

Shrivastava, A. & Li, P. (2014) Asymmetric lsh (ALSH) for sublinear time maximum inner product search (MIPS). *Advances in Neural Information Processing Systems*, (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence & K. Q. Weinberger eds). Curran Associates, Inc., pp. 2321–2329.

Shrivastava, A. & Li, P. (2015) Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). *31st Conference on Uncertainty in Artificial Intelligence (UAI)*.

Slaney, M. & Casey, M. (2008) Locality-sensitive hashing for finding nearest neighbors. *IEEE Signal Process. Mag.*, **25**, 128–131.

Valiant, G. (2015) Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *J. ACM*, **62**, 13.

Valiant, L. G. (1988) Functionality in neural nets. *Proceedings of the First Annual Workshop on Computational Learning Theory*. Los Altos, CA: Morgan Kaufmann Publishers Inc., pp. 28–39.

Vershynin, R. (2010) Introduction to the non-asymptotic analysis of random matrices. Preprint, 2010, arXiv:1011.3027.

Williams, V. V. (2012) Multiplying matrices faster than Coppersmith–Winograd. *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*. New York: ACM, pp. 887–898.

Wimalajeewa, T., Eldar, Y. C. & Varshney, P. K. (2013) Recovery of sparse matrices via matrix sketching. Preprint, 2013, arXiv:1311.2448.

Xiao, C., Wang, W., Lin, X., Yu, J. X. & Wang, G. (2011) Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, **36**, 15.

Appendix. Proofs

A.1 *Proof of Lemma 3.2*

Proof. For every independent run $i \in [m]$, by definition $(\mathbf{y}_i)_j = 0$ for all $j \notin J_i$. Hence,

$$\|S_k - \mathbf{y}_i\|^2 = \sum_{j \in J_i} (S_{kj} - (\mathbf{y}_i)_j)^2 + \sum_{j \notin J_i} (S_{kj})^2 \geqslant \sum_{j \notin J_i} (S_{kj})^2 = \|S_k - S_k^{J_i}\|^2, \tag{A.1}$$

where $(S_k^{J_i})_j = S_{kj}$ for $j \in J_i$ and otherwise $(S_k^{J_i})_j = 0$. Moreover, for every $i \in [m]$,

$$\left\|S_k\right\|^2 - \left\|S_k - S_k^{J_i}\right\|^2 = \sum_{j \in J_i} (S_{kj})^2 \leqslant \sum_{j \in \bigcup_i J_i} (S_{kj})^2 = \|S_k\|^2 - \left\|S_k - \tilde{S}_k\right\|^2.$$

Hence,

$$\left\|S_k - \tilde{S}_k\right\| \leqslant \left\|S_k - S_k^{J_i}\right\|. \tag{A.2}$$

By combining Equations (A.1) and (A.2), we obtain

$$\left\|S_k - \tilde{S}_k\right\| \leqslant \min_{i \in [m]} \left\|S_k - \mathbf{y}_i\right\|. \tag{A.3}$$

Since $\mathbf{y}_i$ is an output of the sFFT algorithm, according to Equation (3.1) it satisfies the following inequality with probability $\geqslant \frac{2}{3}$:

$$\left\|S_k - \mathbf{y}_i\right\| \leqslant (1 + \alpha) \min_{\|\mathbf{y}\|_0 \leqslant r} \|S_k - \mathbf{y}\| + \delta \|S_k\|. \tag{A.4}$$

Thus, by a union-bound argument, with probability $\geqslant 1 - (\frac{1}{3})^m = 1 - 1/3p$, there exists at least one index i for which Equation (A.4) holds. Combining this with Equation (A.3) concludes the proof. □

A.2 *Proof of Theorem 3.1*

Proof. First, we show that a sufficient condition for Algorithm 1 to detect all large entries of S is that Equation (3.4) holds for all rows. Then, from Corollary 3.1, the probabilistic guarantee will follow. Using Equation (3.4) with $\delta = \epsilon/(R + \sqrt{r}M)$ and $\alpha = 1$ yields

$$\left\|S_k - \tilde{S}_k\right\| \leqslant 2 \min_{\|\mathbf{y}\|_0 \leqslant r} \|S_k - \mathbf{y}\| + \frac{\epsilon}{R + \sqrt{r}M} \|S_k\|. \tag{A.5}$$

Since S is $(r, \mu, R, 2)$-sparse, it follows that $\|S_k\| \leqslant R + \sqrt{r}M$, which implies that

$$\left\|S_k - \tilde{S}_k\right\| \leqslant 2 \min_{\|\mathbf{y}\|_0 \leqslant r} \|S_k - \mathbf{y}\| + \epsilon.$$

Let $T_\mu(S_k)$ be the kth row of S thresholded at level μ,

$$(\mathrm{T}_\mu(S_k))_j = S_{kj} \mathbf{1}[|S_{kj}| \geqslant \mu].$$

Each row of S has at most r large entries, thus $\|T_\mu(S_k)\|_0 \leqslant r$ and, from Equation (A.5), we obtain

$$\left\|S_k - \tilde{S}_k\right\| \leqslant 2\|S_k - T_\mu(S_k)\| + \epsilon = 2 \left(\sum_{j \notin J_\mu(S_k)} (S_{kj})^2 \right)^{1/2} + \epsilon \leqslant 2R + \epsilon.$$

Now, assume to the contrary that there exists a large entry S_{kj}, for $j \in J_\mu(S_k)$, which the algorithm failed to detect. Then

$$\left\|S_k - \tilde{S}_k\right\|_2 \geqslant |S_{kj}| \geqslant \mu$$

meaning $\mu \leqslant 2R + \epsilon$, which contradicts the assumption.

As for the runtime of the algorithm, the first two steps, calculating the matrix W and the bound M, can be performed using only $O(np \log p)$ operations. Then, for every row k, the algorithm makes $O(\log p)$ queries of the sFFT algorithm, where each query requires $O(nr \log p \log((R + \sqrt{r}M)p/\epsilon))$ operations. As for the last step, evaluating the output $\tilde{S}$ requires $O(n|I|)$ operations. Since the size of I cannot exceed the number of sFFT queries multiplied by the number of operations for a single query, we conclude that the total number of operations is at most $O(nrp \log^2 p \log((R + \sqrt{r}M)p/\epsilon))$. □

A.3 *Proof of Lemma 4.1*

Proof. First, let us study the quantity σ^2 and the distribution of $\hat{\sigma}^2$, under both the null and under the alternative.

If $\mathscr{H}_0$ holds, since S is $(r, \mu, R, 2)$-sparse,

$$\sigma^2 = \sum_{j \in I(h,i)} \langle \mathbf{x}_k, \mathbf{x}_j \rangle^2 \leqslant \sum_{j \notin J_\mu(S_k)} \langle \mathbf{x}_k, \mathbf{x}_j \rangle^2 \leqslant R^2 < \mu^2/4.$$

In contrast, if $\mathscr{H}_1$ holds, then there is at least one entry $|S_{kj}| \geqslant \mu$, leading to

$$\sigma^2 = \sum_{j \in I(h,i)} \langle \mathbf{x}_k, \mathbf{x}_j \rangle^2 \geqslant \sum_{j \in J_\mu(S_k) \cap I(h,i)} \langle \mathbf{x}_k, \mathbf{x}_j \rangle^2 \geqslant \mu^2.$$

Therefore, the two hypotheses in Equation (4.3) may also be written as

$$\mathscr{H}_0 : \sigma^2 \leqslant R^2 \quad \text{vs.} \quad \mathscr{H}_1 : \sigma^2 \geqslant \mu^2.$$

Since, for every $l \in [m]$, $y_l \sim N(0, \sigma^2)$,

$$\hat{\sigma}^2 \sim \frac{\sigma^2}{m} \chi^2_m.$$

In Birnbaum & Nadler (2012) and Johnstone & Lu (2009), respectively, the following two concentration inequalities for a χ^2 random variable were derived for all $m \geqslant 2$ (see also Laurent & Massart, 2000):

$$\Pr\left[\frac{\chi^2_m}{m} > 1 + \epsilon\right] \leqslant \frac{1}{\sqrt{\pi m}(\epsilon + 2/m)} \exp\left(-\frac{m}{2}(\epsilon - \log(1+\epsilon))\right) \quad \forall \epsilon > 0, \tag{A.6}$$

$$\Pr\left[\frac{\chi^2_m}{m} < 1 - \epsilon\right] \leqslant \exp\left(-\frac{m\epsilon^2}{4}\right) \quad \forall 0 < \epsilon < 1. \tag{A.7}$$

We use these inequalities to bound the probabilities of false alarm and misdetection:

$$\Pr[\textbf{false alarm}] = \Pr\left[\hat{\sigma}^2 \geqslant \frac{3\mu^2}{4} \middle| \mathscr{H}_0\right] = \Pr\left[\frac{\chi^2_m}{m} \geqslant \frac{3\mu^2}{4\sigma^2} \middle| \sigma^2 \leqslant R^2\right] \leqslant \Pr\left[\frac{\chi^2_m}{m} \geqslant \frac{3\mu^2}{4R^2}\right].$$

Since $4R^2 \leqslant \mu^2$,

$$\Pr[\textbf{false alarm}] \leqslant \Pr\left[\frac{\chi^2_m}{m} \geqslant 3\right].$$

Using (A.6) with $\epsilon = 2$, we obtain, for $m \geqslant 3\log(1/\delta)$,

$$\Pr[\textbf{false alarm}] \leqslant \exp\left(-\frac{m}{2}(2 - \log 3)\right) \leqslant \delta.$$

Similarly, for the probability of misdetection,

$$\Pr[\textbf{misdetection}] = \Pr\left[\hat{\sigma}^2 < \frac{3\mu^2}{4} \middle| \mathscr{H}_1\right] = \Pr\left[\frac{\chi^2_m}{m} < \frac{3\mu^2}{4\sigma^2} \middle| \sigma^2 \geqslant \mu^2\right] \leqslant \Pr\left[\frac{\chi^2_m}{m} < \frac{3}{4}\right].$$

Using (A.7) with $\epsilon = \frac{1}{4}$, we obtain, for $m \geqslant 64\log(1/\delta)$,

$$\Pr[\textbf{misdetection}] \leqslant \Pr\left[\frac{\chi^2_m}{m} < \frac{3}{4}\right] \leqslant \exp\left(-\frac{m}{64}\right) \leqslant \delta.$$

□

A.4 *Proof of Theorem 4.1*

Proof. We first introduce the following notation. For each $k \in [p]$ and $h \in [L]$, divide the nodes into two disjoint sets

$$\mathscr{A}_k^h = \left\{(h,i) : i \in \left[2^h\right],\ I(h,i) \cap J_\mu(S_k) \neq \emptyset\right\},$$
$$\mathscr{B}_k^h = \left\{(h,i) : i \in \left[2^h\right],\ I(h,i) \cap J_\mu(S_k) = \emptyset\right\}.$$

We say that the algorithm visited or entered a node $T(h,i)$ while processing the trees with $\mathbf{x}_k$, if during its execution the call `Find(`$\mathbf{x}_k, h, i, \{T_l\}, \mu$`)` occurred. Note that as the algorithm processes the trees with input $\mathbf{x}_k$, in order to detect all large entries in the kth row, while at level h it *must* visit all nodes in $\mathscr{A}_k^h$. In contrast, to have a fast runtime, it should avoid processing nodes in $\mathscr{B}_k^h$. Moreover, recall that the tree construction stage deterministically requires $O(np\log p)$ operations. Thus, to prove the runtime bound, it suffices to show that, after the preprocessing stage, the algorithm uses at most $O(nrp\log^2 p)$ operations, with high probability for (i) and (ii), and in expectation for (iii).

Proof of (i) *and* (ii). Let V_k^h be the event that the algorithm visited all nodes in $\mathscr{A}_k^h$, but no nodes of $\mathscr{B}_k^h$, while processing the hth level of the trees with $\mathbf{x}_k$. Since S is sparse, $|\mathscr{A}_k^h| \leqslant r$, and therefore

$$\left|\bigcup_{k\in[p],h\in[L]} \mathscr{A}_k^h\right| \leqslant rpL.$$

Moreover, since the algorithm detects all large entries if and only if it visits all nodes of all sets $\mathscr{A}_k^h$, it suffices to bound the probability of V_k^h simultaneously occurring for all $k \in [p]$ and $h \in [L]$.

It has at most $2r$ new nodes to check in the $h+1$ level, unless $h = L$. According to Lemma 4.1 with $\delta = 1/2rpL^3$, the probability of the algorithm to enter a node in $\mathscr{B}_k^h$ or to skip a node in $\mathscr{A}_k^h$, is at most $1/2rpL^3$. Denote by $\bar{V}_k^h$ the complementary event of V_k^h. This implies

$$\Pr\left[\bar{V}_k^h\right] \leqslant 2r \cdot \frac{1}{2rpL^3} = \frac{1}{pL^3}. \tag{A.8}$$

Since

$$\Pr\left[V_k^h\right] = \Pr\left[V_k^h \mid V_k^1,\ldots,V_k^{h-1}\right]\Pr\left[V_k^1,\ldots,V_k^{h-1}\right] + \Pr\left[V_k^h \mid \exists i \in [h-1] : \bar{V}_k^i\right]\Pr\left[\exists i \in [h-1] : \bar{V}_k^i\right]$$

it follows that

$$\Pr\left[V_k^h \mid V_k^1,\ldots,V_k^{h-1}\right] = \frac{\Pr\left[V_k^h\right] - \Pr\left[V_k^h \mid \exists i \in [h-1] : \bar{V}_k\right]\Pr\left[\exists i \in [h-1] : \bar{V}_k^i\right]}{\Pr\left[V_k^1,\ldots,V_k^{h-1}\right]}.$$

Clearly, $\Pr[V_k^1,\ldots,V_k^{h-1}] \leqslant 1$, and thus

$$\Pr\left[V_k^h \mid V_k^1,\ldots,V_k^{h-1}\right] \geqslant \Pr\left[V_k^h\right] - \Pr\left[\exists i \in [h-1] : \bar{V}_k^i\right].$$

We use Equation (A.8) to obtain

$$\Pr\left[V_k^h \mid V_k^1,\ldots,V_k^{h-1}\right] \geqslant 1 - \frac{1}{pL^3} - (h-1)\frac{1}{pL^3} \geqslant 1 - \frac{1}{pL^2}.$$

Next, we derive, for a fixed k,

$$\Pr\left[\forall h : V_k^h\right] = \Pr\left[V_k^0\right] \cdot \prod_{h=1}^{L} \Pr\left[V_k^h \mid V_k^0, \ldots, V_k^{h-1}\right] \geqslant \left(1 - \frac{1}{pL^2}\right)^L$$

where clearly $\Pr[V_k^0] = 1$. Therefore,

$$\begin{aligned}\Pr\left[\forall k, h : V_k^h\right] &= 1 - \Pr\left[\exists k, h : \bar{V}_k^h\right] \geqslant 1 - \sum_{k=1}^{p} \Pr\left[\exists h : \bar{V}_k^h\right] \\ &\geqslant 1 - p\left(1 - \left(1 - \frac{1}{pL^2}\right)^L\right). \end{aligned} \tag{A.9}$$

To bound this expression, consider the function $g(x) = (1-x)^L$. Since $g''(\xi) \geqslant 0$ for all $\xi \in [0,1]$, it readily follows that, for all $x \in [0,1]$, $g(x) \geqslant (1 - xL)$. Hence, for $x = 1/pL^2$ we obtain

$$\left(1 - \frac{1}{pL^2}\right)^L \geqslant 1 - \frac{1}{pL}. \tag{A.10}$$

Combining Equation (A.9) with Equation (A.10) yields

$$\Pr\left[\forall k, h : V_k^h\right] \geqslant 1 - p\left(1 - \left(1 - \frac{1}{pL}\right)\right) = 1 - \frac{1}{\log_2 p} \xrightarrow{p\to\infty} 1$$

which proves (ii), and similarly, for $p \geqslant 8$,

$$\Pr\left[\forall k, h : V_k^h\right] \geqslant 2/3,$$

which proves (i).

Proof of (iii). Let $X_{h,i}^k$ be the indicator of the event that the algorithm entered the node $T(h,i)$, while processing the tree with $\mathbf{x}_k$. Recall that

$$\Pr\left[X_{h,i}^k = 1\right] \leqslant \Pr\left[\hat{\sigma}^2(I(h,i)) \geqslant \frac{3\mu^2}{4}\right].$$

For nodes in $\mathscr{A}_k^h$, this probability is clearly at most one, whereas for nodes in $\mathscr{B}_k^h$ it is at most δ, as $m \geqslant 64\log(1/\delta)$. By its definition, the algorithm makes at most cnm operations, for some absolute constant c, in each node it considers. Thus,

$$\begin{aligned}\mathbb{E}[\text{Runtime}] &= \mathbb{E}[\text{Number of visited nodes}] \cdot cmn \\ &= \sum_{k=1}^{p}\sum_{h=1}^{L}\sum_{i=1}^{2^h} \Pr\left[X_{h,i}^k = 1\right] \cdot cmn \\ &\leqslant \sum_{k=1}^{p}\sum_{h=1}^{L} \left(\left|\mathscr{A}_k^h\right| + \left|\mathscr{B}_k^h\right|\delta\right) \cdot cmn \\ &\leqslant p(rL + 2p\delta) \cdot cmn.\end{aligned}$$

For $m \geqslant 64 \log p$, $\delta = 1/p$, and we conclude

$$\mathbb{E}[\text{Runtime}] \leqslant C' nrp \log^2 p$$

for some absolute constant C'. □

A.5 *Proof of Theorem 5.1*

To prove Theorem 5.1, we first introduce the following auxiliary lemma. It provides exponential tail bounds on the deviations of a quadratic form from its mean value, when the underlying random vector has non-zero mean. This lemma thus generalizes previous concentration bounds for quadratic forms (Boucheron *et al.*, 2013; Rudelson & Vershynin, 2013), which all assumed that the vector has mean zero. Hence, it may be of independent interest.

LEMMA A.1 Let $Y = (Y_1, \ldots, Y_n)^t$ be a sub-Gaussian random vector, where all Y_is are independent and let $K = \max_i \|Y_i\|_{\psi_2}$. Let $A \in \mathbb{R}^{n \times n}$ be a real symmetric matrix. Then, for every $t > 0$,

$$\Pr[|Y^t A Y - \mathbb{E}[Y^t A Y]| > t] \leqslant 6 \exp\left[-c \min\left(\frac{t^2}{(n-1)K^4 \|A\|_F^2}, \frac{t}{K^2 \|A\|}\right)\right],$$

where c is a positive absolute constant.

Proof. First, observe that due to the independence of the Y_is,

$$Y^t A Y - \mathbb{E}\left[Y^t A Y\right] = \sum_i a_{ii}\left[Y_i^2 - \mathbb{E}Y_i^2\right] + \sum_{i \neq j} a_{ij}[Y_i Y_j - \mathbb{E}Y_i \mathbb{E}Y_j].$$

Since

$$(Y_i - \mathbb{E}Y_i)(Y_j - \mathbb{E}Y_j) = Y_i Y_j - Y_j \mathbb{E}Y_i - Y_i \mathbb{E}Y_j + \mathbb{E}Y_i \mathbb{E}Y_j$$

and A is symmetric, we obtain

$$\begin{aligned}\sum_{i \neq j} a_{ij}[Y_i Y_j - \mathbb{E}Y_i \mathbb{E}Y_j] &= \sum_{i \neq j} a_{ij}(Y_i - \mathbb{E}Y_i)(Y_j - \mathbb{E}Y_j) + 2\sum_{i \neq j} a_{ij}[Y_i \mathbb{E}Y_j - \mathbb{E}Y_i \mathbb{E}Y_j] \\ &= \sum_{i \neq j} a_{ij}(Y_i - \mathbb{E}Y_i)(Y_j - \mathbb{E}Y_j) + \sum_i b_i[Y_i - \mathbb{E}Y_i],\end{aligned}$$

where $b_i = 2\sum_{j \neq i} a_{ij} \mathbb{E}\mathbf{y}_j$. Therefore,

$$\left|Y^t A Y - \mathbb{E}\left[Y^t A Y\right]\right| \leqslant \left|\sum_i a_{ii}\left[Y_i^2 - \mathbb{E}Y_i^2\right]\right| + \left|\sum_{i \neq j} a_{ij}(Y_i - \mathbb{E}Y_i)(Y_j - \mathbb{E}Y_j)\right| + \left|\sum_i b_i[Y_i - \mathbb{E}Y_i]\right|$$

and the problem reduces to estimating the following probabilities:

$$\rho := \Pr\left[\left|Y^tAY - \mathbb{E}\left[Y^tAY\right]\right| > t\right] \leqslant \Pr\left[\left|\sum_i a_{ii}[Y_i^2 - \mathbb{E}Y_i^2]\right| > t/3\right] + \Pr\left[\left|\sum_{i\neq j} a_{ij}(Y_i - \mathbb{E}Y_i)(Y_j - \mathbb{E}Y_j)\right| > t/3\right] + \Pr\left[\left|\sum_i b_i[Y_i - \mathbb{E}Y_i]\right| > t/3\right] =: \rho_1 + \rho_2 + \rho_3. \tag{A.11}$$

As for the first term, note that $Y_i^2 - \mathbb{E}Y_i^2$s are independent centered sub-exponentials, with

$$\left\|Y_i^2 - \mathbb{E}Y_i^2\right\|_{\psi_1} \leqslant 4\|Y_i\|_{\psi_2}^2 \leqslant 4K^2.$$

Thus we can use Proposition 5.16 of Vershynin (2010) to obtain

$$\rho_1 \leqslant 2\exp\left[-c_1 \min\left(\frac{t^2}{K^4\sum_i a_{ii}^2}, \frac{t}{K^2 \max|a_{ii}|}\right)\right]$$

for some absolute constant c_1. Next, to bound the second sum, we follow the arguments of Rudelson & Vershynin (2013) in the proof of the Hanson–Wright inequality to obtain

$$\rho_2 \leqslant 2\exp\left[-c_2 \min\left(\frac{t^2}{K^4\|A\|_F^2}, \frac{t}{K^2\|A\|}\right)\right]$$

for some absolute constant c_2.

If $\mathbb{E}[Y] = 0$ or A is a diagonal matrix, then all coefficients $b_i = 0$. The third term in Equation (A.11) then vanished, and we recover a result similar to the original Hanson–Wright inequality. Assume then that $\mathbb{E}Y \neq 0$ and A is not diagonal, which leads to $\mathbf{b} \neq \mathbf{0}$. Since the random variables $Y_i - \mathbb{E}Y_i$s are centered independent sub-Gaussians with $\|Y_i - \mathbb{E}Y_i\|_{\psi_2} \leqslant \|Y_i\|_{\psi_2} + |\mathbb{E}Y_i| \leqslant 2K$, we then use Proposition 5.10 of Vershynin (2010) to obtain

$$\rho_3 \leqslant 2\exp\left[-c_3\frac{t^2}{K^2\|\mathbf{b}\|_2^2}\right]$$

for some absolute constant c_3. Since $\|\mathbf{b}\|_2^2 \leqslant (n-1)K^2\|A\|_F^2$, $\sum_i a_{ii}^2 \leqslant \|A\|_F^2$ and $\max|a_{ii}| \leqslant \|A\|$, the lemma follows. □

Proof of Theorem 5.1. Let A be the $n \times n$ matrix

$$A = \frac{1}{n-1}\left[I_n - \frac{1}{n}\mathbf{e}\mathbf{e}^\mathbf{t}\right],$$

where $\mathbf{e} = (1, \ldots, 1) \in \mathbb{R}^n$. Since

$$S_{ij} = \mathbf{z}_i^tA\mathbf{z}_j = \tfrac{1}{2}\left[\mathbf{z}_i^tA\mathbf{z}_i + \mathbf{z}_j^tA\mathbf{z}_j - (\mathbf{z}_i - \mathbf{z}_j)^tA(\mathbf{z}_i - \mathbf{z}_j)\right]$$

and $\Sigma_{ij} = \mathbb{E}[S_{ij}]$, it follows that

$$\Pr[|S_{ij} - \Sigma_{ij}| > t] \leqslant \Pr\left[\left|\mathbf{z}_i^t A \mathbf{z}_i - \mathbb{E}\left[\mathbf{z}_i^t A \mathbf{z}_i\right]\right| > \frac{2t}{3}\right] + \Pr\left[\left|\mathbf{z}_j^t A \mathbf{z}_j - \mathbb{E}\left[\mathbf{z}_j^t A \mathbf{z}_j\right]\right| > \frac{2t}{3}\right] + \Pr\left[\left|(\mathbf{z}_i - \mathbf{z}_j)^t A (\mathbf{z}_i - \mathbf{z}_j) - \mathbb{E}\left[(\mathbf{z}_i - \mathbf{z}_j)^t A (\mathbf{z}_i - \mathbf{z}_j)\right]\right| > \frac{2t}{3}\right].$$

Now, since $\mathbf{z}_i$ and $\mathbf{z}_i - \mathbf{z}_j$ are sub-Gaussian vectors with independent entries, we use Lemma A.1 to obtain

$$\Pr[|S_{ij} - \Sigma_{ij}| > t] \leqslant 18 \exp\left[-c \min\left(\frac{t^2}{(n-1)K^4 \|A\|_F^2}, \frac{t}{K^2 \|A\|}\right)\right]$$

for some absolute constant $c > 0$. Clearly, both $\|A\|_F$ and $\|A\|$ are bounded by $2/(n-1)$, and therefore we conclude that, for $t \leqslant K^2$,

$$\Pr[|S_{ij} - \Sigma_{ij}| > t] \leqslant 18 \exp\left(-\frac{Ct^2(n-1)}{K^4}\right) \tag{A.12}$$

for some absolute constant C. For $n > (CK^4/t^2)\log(54p^2)$ this probability is smaller than $1/3p^2$. Then, applying a simple union-bound argument yields

$$\Pr[\forall i,j : |S_{ij} - \Sigma_{ij}| \leqslant t] \geqslant \tfrac{2}{3}. \tag{A.13}$$

Next, assuming the event in Equation (A.13) holds, we show that, for sufficiently small t, sparsity of Σ implies sparsity of S. In particular, for $t \leqslant \min\{(\mu - 2R)/(2\sqrt{p-r}+1), (\mu - R)/4\}$, we show that S is $(r, \mu - t, \frac{1}{2}(\mu - t), 2)$-sparse and, moreover, that $J_\mu(\Sigma) = J_{\mu-t}(S)$.

Indeed, if $|\Sigma_{kj}| \geqslant \mu$, then, according to Equation (A.13),

$$|S_{kj}| \geqslant |\Sigma_{kj}| - |\Sigma_{kj} - S_{kj}| \geqslant \mu - t$$

meaning $J_\mu(\Sigma_k) \subseteq J_{\mu-t}(S_k)$. Similarly, if $|S_{kj}| \geqslant \mu - t$, then

$$|\Sigma_{kj}| \geqslant |S_{kj}| - |\Sigma_{kj} - S_{kj}| \geqslant \mu - 2t \geqslant \mu - \frac{\mu - R}{2} = \frac{\mu + R}{2} > R.$$

Since every element of Σ (in absolute value) is either larger than μ or smaller than R, the condition $|S_{kj}| > \mu - t$ thus implies that $|\Sigma_{kj}| \geqslant \mu$, and hence $J_\mu(\Sigma_k) \supseteq J_{\mu-t}(S_k)$. Combining these two results above yields that with high probability $J_\mu(\Sigma_k) = J_{\mu-t}(S_k)$, and entries of S are large if and only if the corresponding entries of Σ are large.

As for the gap,

$$\begin{aligned}\sum_{j \notin J_{\mu-t}(S_k)} (S_{kj})^2 &\leqslant \sum_{j \notin J_{\mu-t}(S_k)} (|\Sigma_{kj}| + t)^2 = \sum_{j \notin J_\mu(\Sigma_k)} (|\Sigma_{kj}| + t)^2 \\ &\leqslant \sum_{j \notin J_\mu(\Sigma_k)} (\Sigma_{kj})^2 + \sum_{j \notin J_\mu(\Sigma_k)} t^2 + 2 \sum_{j \notin J_\mu(\Sigma_k)} |\Sigma_{kj}| t \\ &\leqslant R^2 + (p-r)t^2 + 2t\sqrt{p-r}R = (R + \sqrt{p-r}t)^2.\end{aligned}$$

For $t \leqslant (\mu - 2R)/(2\sqrt{p-r} + 1)$, we obtain

$$\sum_{j \notin J_{\mu - t}(S_k)} (S_{kj})^2 \leqslant \left(\frac{1}{2}(\mu - t) \right)^2$$

which implies that S is $(r, \mu - t, \frac{1}{2}(\mu - t), 2)$-sparse. □

A.6 *Proof of Theorem 5.2*

Proof. For any n and p, let $\beta(n,p)$ be such that $n = (2K^4/ct^2)\log(\beta(n,p)p^2) + 1$. Note that, for n and p large enough, $\beta(n,p) > 0$. By applying a union-bound argument on Equation (A.12), we obtain, instead of Equation (A.13),

$$\Pr[\forall i,j : |S_{ij} - \Sigma_{ij}| \leqslant t] \geqslant 1 - \frac{1}{\beta(n,p)}.$$

Clearly, as $n \to \infty$ with $p \log p/n \to 0$, $\beta(n,p) \to \infty$, which implies that this probability converges to one. □