Faster Algorithm for Structured John Ellipsoid Computation

Zhao Song† Xin Yang‡ Yuanyuan Yang§ Tianyi Zhou¶

Abstract

Computing John Ellipsoid is a fundamental problem in machine learning and convex optimization, where the goal is to compute the ellipsoid with maximal volume that lies in a given convex centrally symmetric polytope defined by a matrix $A \in \mathbb{R}^{n \times d}$. In this work, we show two faster algorithms for approximating the John Ellipsoid.

- For sparse matrix $A$, we can achieve nearly input sparsity time $\text{nnz}(A) + d^\omega$, where $\omega$ is exponent of matrix multiplication. Currently, $\omega \approx 2.373$.
- For the matrix $A$ which has small treewidth $\tau$, we can achieve $n\tau^2$ time.

Therefore, we significantly improves the state-of-the-art results on approximating the John Ellipsoid for centrally symmetric polytope [Cohen, Cousins, Lee, and Yang COLT 2019] which takes $nd^2$ time.

*The authors would like to thank Guanghao Ye and Lichen Zhang for their generous and priceless help.
†zsong@adobe.com. Adobe Research.
‡yangxin.yx@bytedance.com. Bytedance Research.
§yyangh@cs.washington.edu. University of Washington.
¶t8zhou@ucsd.edu. University of California San Diego.
1 Introduction

The famous theorem of Fritz John [Joh48] states that any convex polytope can be well approximated by the maximal volume inscribed ellipsoid of the polytope. John Ellipsoid is defined as the maximal volume inscribed ellipsoid (MVIE), and it has a lot of important applications including high-dimensional sampling [Vem05], linear programming [LS14] and differential privacy [NTZ13]. Furthermore, it is known [Tod16] that computing MVIE is equivalent to the D-optimal design problem, which is a fundamental problem in statistics.

The John Ellipsoid can be computed in polynomial time, as it can be written as a constrained convex optimization problem and hence can be solved by various convex optimization algorithms such as first-order methods [Kha96, KY05] and second-order interior point methods [NN94, SF04]. Recently, [CCLY19] found a simple fixed-point iteration to compute the John Ellipsoid. They studied the case of centrally symmetric polytopes given by a set of inequalities, which can be formulated as follows: Given a rank $d$ matrix $A \in \mathbb{R}^{n \times d}$, the polytope $P$ is defined as $P := \{ x \in \mathbb{R}^d : -1_n \leq Ax \leq 1_n \}$, where $1_n$ is an all-1 vector with dimension $n$. In this setting, [CP15] showed that computing MVIE can be reduced to the problem of computing $\ell_\infty$ Lewis weights of the matrix $A$. The $\ell_\infty$ Lewis weights of matrix $A$ is a vector $w \in \mathbb{R}^n$, which can be considered as a weighted version of leverage scores of $A$. The property of $w$ naturally satisfies a fixed-point iteration [CCLY19].

The most time-consuming part of the algorithm in [CCLY19] is to compute quadratic forms $a^T B^{-1} a$, where $B$ is a weighted version of $A^T A$ and $a$ is a row vector in $A$. This is a standard linear algebra task and in [CCLY19], it is done via computing Cholesky decomposition and solving linear systems. However, these operations require $O(nd^2)$ time, which could be prohibitively large.

Motivated by this, we propose a sketched-based algorithm with improved running time. Sketching is a popular technique in randomized linear algebra and has been widely applied in a lot of linear algebra tasks [CW13, NN13, BWZ16, RSW16, SWZ17, XZZ18, SWZ19, LSZ19, JSWZ21, SY21, DLY21, BPSW21, HSWZ22, SXYZ22, GS22]. Furthermore, we can even speed up our algorithm when the matrix $A$ has certain special structures, i.e., with small treewidth.

| References | #Iters. | Cost per iter. |
|------------|---------|----------------|
| [CCLY19]   | $\epsilon^{-1} \log(n/d)$ | $nd^2$ |
| Theorem 1.1| $\epsilon^{-1} \log(n/d)$ | $\epsilon^{-1} \text{nnz}(A) \log(n/\delta) + \epsilon^{-2} d^\omega \log(n/\delta)$ |
| Theorem 1.4| $\epsilon^{-1} \log(n/d)$ | $n\tau^2$ |

Table 1: Let $A \in \mathbb{R}^{n \times d}$. Let $\omega$ denote the exponent of matrix multiplication. Currently $\omega \approx 2.373$ [Wil12, LG14, AW21]. Let $\epsilon \in (0, 1)$ denote the accuracy parameter. Let $\delta \in (0, 1)$ denote the failure probability. We use $\text{nnz}(A)$ to denote the number of non-zeros in the matrix $A$. We use $\tau$ to denote the treewidth of $A$. Note that $\tau \leq d$. Our algorithm (Theorem 1.1, Algorithm 1) is better than the result in [CCLY19] when $\epsilon \in (1/d, 1)$ and $n \geq d^\omega$. Since $\tau \leq d$, then for any matrix $A$ that has small $\tau$, our algorithm (Theorem 1.4, Algorithm 2) is always better than [CCLY19].

1.1 General Result

The main result of this paper is the following theorem:

**Theorem 1.1** (Our result I, general $A$). Given a matrix $A \in \mathbb{R}^{n \times d}$, we define a centrally symmetric polytope $P$ as follows:

$$\{ x \in \mathbb{R}^d : -1_n \leq Ax \leq 1_n \}.$$
Figure 1: Time complexity comparison between CCLY19 (denotes [CCLY19]) and ours, assuming \( n = d^a \), \( \epsilon = \Theta(1) \), and ignoring the log factors. The x-axis is corresponding to \( a \) and y-axis is corresponding to \( b \). The \( n^b \) is the total running time.

Then, given \( \epsilon \in (0, 1) \), there exists an algorithm (Algorithm 1) that outputs an ellipsoid \( Q \) satisfies:

\[
\frac{1}{\sqrt{1+\epsilon}} \cdot Q \subseteq P \subseteq \sqrt{d} \cdot Q.
\]

in \( O((\epsilon^{-1} \log(n/\delta) \cdot \text{nnz}(A) + \epsilon^{-2} \log(n/\delta) \cdot d^a) \cdot \epsilon^{-1} \log(n/d)) \) time, where \( \omega \approx 2.373 \) denote the current matrix multiplication exponent [Wil12, LG14, AW21].

The geometric interpretation of the output ellipsoid is shown in Figure 2.

1.2 Treewidth Result

In this section, we further generalize our result to the treewidth setting. Before stating our main result, we introduce several definitions for treewidth. To begin with, we introduce the definition of generalized dual graph of a given matrix.

**Definition 1.2.** The generalized dual graph of the matrix \( A \in \mathbb{R}^{n \times d} \) with block structure \( n = \sum_{i=1}^{m} n_i \) is the graph \( G_A = (V, E) \) with \( V = \{1, \ldots, d\} \).

We say an edge \((i, j) \in E\) if and only if \( A_{r,i} \neq 0 \) and \( A_{r,j} \neq 0 \) for some \( r \), where we use \( A_{r,i} \) to mean the submatrix of \( A \) in row \( i \) and column block \( r \).

Next, we define the tree decomposition and treewidth (see [BGHK95, Dav06, LMS13, DLY21]),

**Definition 1.3.** A tree decomposition is a mapping of graphs into trees. For graph \( G \), the tree decomposition is defined as pair \((M, T)\), where \( T \) is a tree, and \( M : V(T) \to 2^{V(G)} \) is a family of subsets of \( V(G) \) called bags labelling the vertices of \( T \), satisfies that:

- The vertices maintained by all bags is the same as those of graph \( G \): \( \cup_{t \in V(T)} M(t) = V(G) \).
• For every vertex \( v \in V(G) \), the nodes \( t \in V(T) \) satisfying \( v \in M(t) \) is a connected subgraph of \( T \), and

• For every edge \( e = (u, v) \in V(G) \), there exist a node \( t \in V(T) \) so that \( u, v \in M(t) \).

where \( V(\cdot) \) denote the vertex set of a graph.

The width of a tree decomposition \((M, T)\) is \( \max \{|M(t)| - 1 : t \in T| \). The treewidth of \( G \) is the minimum width over all tree decompositions of \( G \).

\[ \frac{1}{\sqrt{1+\epsilon}} Q \subseteq P \subseteq \sqrt{d} \cdot Q. \]

Figure 2: The geometric interpretation of our output ellipsoid. Let \( P \) be a given input polytope. We can find an ellipsoid \( Q \) so that \( \frac{1}{\sqrt{1+\epsilon}} Q \subseteq P \subseteq \sqrt{d} \cdot Q \).

Usually, the treewidth of a tree is NP-hard to compute \([FLS^{+}18, ACP87]\). Nevertheless, it’s possible to find a width-\( O(\text{tw}(G) \log^3 n) \) tree decomposition within \( O(m \text{ poly log } n) \), where \( m \) denotes the number of edges, \( n \) denotes the number of vertices and \( \text{tw}(G) \) denote the treewidth of graph \( G \) \([BGS21]\).

Now, we are ready to state our secondary main result,

**Theorem 1.4 (Our result II, small treewidth \( A \)).** Given \( A \in \mathbb{R}^{n \times d} \) that \( A \) has treewidth \( \tau \). Let \( P \) denote a centrally symmetric polytope defined as \( \{x \in \mathbb{R}^d : -1_n \leq Ax \leq 1_n\} \). For \( \epsilon \in (0, 1) \), there is an algorithm (Algorithm 2) that runs in time

\[ O(\epsilon^{-1} \cdot (n \tau^2) \cdot \log(n/d)) \]

and outputs an ellipsoid \( Q \) such that

\[ \frac{1}{\sqrt{1+\epsilon}} \cdot Q \subseteq P \subseteq \sqrt{d} \cdot Q. \]

**Roadmap.** The rest of the paper is organized as follows. In Section 2, we present the technique overview for this paper. In Section 3, we provide some preliminaries for treewidth and John Ellipsoid. In Section 4, we give the formal definition for the John Ellipsoid. In Section 5, we present our main algorithm (Algorithm 1) for approximating John Ellipsoid inside symmetric polytopes and show the running time for the algorithm. In Section 6, we prove the correctness of our implementation. In Section 7, we further improve our running time for small treewidth setting. In Section 8, we give a sparsification tool for the matrix that has pattern \( A^TWA \), where \( W \) is some non-negative diagonal matrix.
Figure 3: (a) A graph $G(V,E)$ (b) The tree decomposition for graph $G$. We can see that the union of the vertices in all bags are nodes $a, \ldots, i$, which is the same as $V(G)$. For every edge $u,v \in V(G)$, we can find at least one bag contains both $u$ and $v$. For example, for edge $(c,b)$ in graph $G$, bag 1 contains both $c$ and $b$. Furthermore, the bags containing any one node in $(a)$ is a subgraph of the tree $(b)$. For example, the bags contains node $c$ are bags 1, 2, 3, which is a subgraph of the tree. Similarly, we can see that the bags containing node $f$ is bags 3, 5, which is also a subgraph of the tree. For edge $(c,f)$, bag 2 and 3 both contain vertices $c$ and $f$. For edge $(i,g)$, bag 5 contains vertices $i$ and $g$.

2 Technique Overview

To begin with, we define $\sigma: \mathbb{R}^n \rightarrow \mathbb{R}^n$ as

$$\sigma_i(v) = a_i^T \left( \sum_{j=1}^{n} v_j a_j a_j^T \right)^{-1} a_i$$

$$= a_i^T (A^T \text{diag}(v)A)^{-1} a_i.$$

Fixed Point Iteration  Let $w^*$ be the optimal solution to program Eq. (4). By its optimal condition (Lemma 4.2), $w^*$ satisfies $w^*_i \cdot (1 - \sigma_i(w^*)) = 0$ for all $i \in [n]$, or equivalently

$$w^*_i = w^*_i \cdot \sigma_i(w^*)$$

Intuitively, by this observation, our algorithm updates the vector $w$ by the fixed point iteration, i.e., $w_{k+1,i} = w_{k,i} \cdot \sigma_i(w_k)$. Then, we have

$$w_{k+1,i}$$

$$= a_i^T \sqrt{w_{k,i}} (A^T \text{diag}(w_{k,i})A)^{-1} \sqrt{w_{k,i}} a_i$$

$$= a_i^T \sqrt{w_{k,i}} (A^T \text{diag}(w_{k,i})A)^{-1/2} \cdot (A^T \text{diag}(w_{k,i})A)^{-1/2} \sqrt{w_{k,i}} a_i$$

$$= \| (A^T \text{diag}(w_{k,i})A)^{-1/2} \sqrt{w_{k,i}} a_i \|_2^2$$

The most time-consuming computation for the above fixed point iteration is to compute $\sigma_i(w)$, and we use dimension deduction techniques to speed up this computation. There are two major ways to do dimensionality reduction, one is non-oblivious way such as leverage score sampling [SS11, BSS12, SXZ22, Zha22], the other one is oblivious way such as sketching method [CW13, NN13]. Prior work [CCLY19] only uses sketching method. In this work, we use both leverage score sampling and sketching method.
Leverage Score Sampling Observe that for $k \in [T-1]$, we can write $w_{k,i} = b_{k,i}^\top ((B_k)^\top B_k)^{-1} b_{k,i}$, where $B_k = \sqrt{\text{diag}(w_k)} \cdot A$ and $b_{k,i}$ is the $i$-th row vector of $B_k \in \mathbb{R}^{n \times d}$.

Hence, $w_{k,i}$ is the leverage score of the $i$-th row of the matrix $B_k$. From the well-known folklore properties of leverage scores (see [SS11, CCLY19] for example), we have:

**Lemma 2.1** (folklore). For $k \in [T]$ and $i \in [n]$, we have $0 \leq w_{k,i} \leq 1$. Moreover, $\sum_{i=1}^n w_{k,i} = d$.

Additionally, applying sampling matrix is a quite standard way in the field of numerical linear algebra (see [CW13, BWZ16, RSW16, SWZ17, SWZ19, CLS19, BLSS20, DSW22]). Here, we apply a sampling matrix based on a constant approximation of the leverage score, we can approximate the matrix $B^\top DB$ up to a multiplicative $(1 + \epsilon_0)$.

Sketching In order to further speed up the algorithm, we apply sketching techniques at line 20 in Algorithm 1, where we use random Gaussian matrix of dimension $s \times d$ to speed up the calculation while maintaining enough accuracy.

Following all the tools above, we are able to prove the following conclusion.

**Lemma 2.2** ($\phi_i$, informal version of Lemma 6.4). Let $u \in \mathbb{R}^n$ ($u$ is the non-normalized version of $v$) denote the vector computed during the Algorithm 1, fix the number of iterations executed in the algorithm as $T = O(\epsilon^{-1} \log(n/d))$ and $s = 1000/\epsilon$, define $\phi_i(u) = \log \sigma_i(u)$, then for $i \in [n]$:

$$\phi_i(u) \leq \frac{1}{T} \log \left(\frac{n}{d}\right) + \epsilon/250 + \epsilon_0$$

holds with probability $1 - \delta - \delta_0$.

Small Treewidth Setting The calculation of $\sigma_i(w)$ can also be sped up with small treewidth setting. When the constraint matrix $A$ is an incidence matrix for a graph, it is natural to parameterize the graph in terms of its treewidth $\tau$. [DLY21] extends this graph notion into linear program, and one of their contributions is to show that if $A$ has treewidth $\tau$, then one can compute a permutation matrix $P \in \mathbb{R}^n$ such that the Cholesky factorization $PA^\top WAP^\top = LL^\top$ is sparse during the iterative algorithm, i.e., $L \in \mathbb{R}^{n \times n}$ has column sparsity $\tau$. For each iteration, the time needed for Cholesky decomposition is $O(n\tau^2)$ and computing $\sigma(w)$ also take $O(n\tau^2)$. Then, we provide an implementation that takes $O((n\tau^2) \cdot T)$ to find the $(1 + \epsilon)$-approximation of John Ellipsoid.

3 Preliminaries

For notations, we use $N(\mu, \sigma^2)$ to denote the normal distribution with mean $\mu$ and variance $\sigma^2$. Given two vectors $x$ and $y \in \mathbb{R}^d$, we use $\langle x, y \rangle$ to denote the inner product between $x$ and $y$, i.e., $\langle x, y \rangle = \sum_{i=1}^d x_i y_i$. We use $1_n$ to denote an all-1 vector with dimension $n$. For any matrix $A \in \mathbb{R}^{d \times d}$, we say $A \succeq 0$ (positive semi-definite) if for all $x \in \mathbb{R}^d$ we have $x^\top Ax \geq 0$. For a function $f$, we use $\tilde{O}(f)$ to denote $f \cdot \text{poly}(\log f)$. For a matrix $A$, we use $A^\top$ to denote the transpose of matrix $A$. For a matrix $A$, we use $\text{nnz}(A)$ to denote the number of non-zero entries in $A$. For a square and full rank matrix $A$, we use $A^{-1}$ to denote the inverse of matrix $A$.

For a positive integer, we use $[n]$ to denote the set $\{1, 2, \cdots, n\}$. For a vector $x$, we use $\|x\|_2$ to denote the entry-wise $\ell_2$ norm of $x$, i.e., $\|x\|_2 := (\sum_{i=1}^n x_i^2)^{1/2}$. We say a vector is $\tau$-sparse if it has at most $\tau$ non-zero entries.

For a random variable $X$, we use $\mathbb{E}[X]$ to denote its expectation. We use $\text{Pr}[\cdot]$ to denote the probability.
3.1 Treewidth Preliminaries

We begin by introducing the definition of treewidth for a given matrix.

**Definition 3.1** (Treewidth $\tau$). Given a matrix $A \in \mathbb{R}^{n \times d}$, we construct its graph $G = (V, E)$ as follows: The vertex set are columns $[d]$; An edge $(i, j) \in E$ if and only if there exists $k \in [n]$ such that $A_{k,i} \neq 0, A_{k,j} \neq 0$. Then, the treewidth of the matrix $A$ is the treewidth of the constructed graph. In particular, every column of $A$ is $\tau$-sparse.

Next, we present the definition for Cholesky factorization.

**Definition 3.2** (Cholesky Factorization). Given a positive-definite matrix $P$, there exists a unique Cholesky factorization $P = LL^\top \in \mathbb{R}^{d \times d}$, where $L \in \mathbb{R}^{d \times d}$ is a lower-triangular matrix with real and positive diagonal entries.

Then, we introduce some results based on the Cholesky factorization of a given matrix with treewidth $\tau$:

**Lemma 3.3** ([BGHK95, Dav06, DLY21]). For any positive diagonal matrix $H \in \mathbb{R}^{n \times n}$, for any matrix $A^\top \in \mathbb{R}^{d \times n}$ with treewidth $\tau$, we can compute the Cholesky factorization $A^\top HA = LL^\top \in \mathbb{R}^{d \times d}$ in $O(n^2 \tau^2)$ time, where $L \in \mathbb{R}^{d \times d}$ is a lower-triangular matrix with real and positive entries. $L$ satisfies the property that every column is $\tau$-sparse.

3.2 Basic Algebra

Next, we provide a basic algebra claim that used in our paper.

**Claim 3.4.** Given vector $w$, we can show $A^\top \text{diag}(w)A = \sum_{i=1}^{n} w_i a_i a_i^\top$, where $a_i$ is the $i$-th column of matrix $A$.

**Proof.** We have,

$$ A^\top \text{diag}(w) = [w_1 a_1, w_2 a_2, \cdots, w_n a_n] $$

Then the $x,y$ element for $A^\top \text{diag}(w)A$ is $\sum_{i=1}^{n} w_i a_y a_i x$. Hence, $A^\top \text{diag}(w)A = \sum_{i=1}^{n} w_i a_i a_i^\top$. □

3.3 Good Rounding

Lemma 3.5 gives a geometric interpretation of the approximation factor in Definition 4.3. Note that for the exact John Ellipsoid $Q^*$ of the same polytope, $Q^* \subseteq P \subseteq \sqrt{d} \cdot Q^*$.

**Lemma 3.5** ((1 + $\epsilon$)-approximation is good rounding, Lemma 2.3 in arXiv\(^1\) version of [CCLY19]). Let $P$ be defined as Definition 4.1. Let $w \in \mathbb{R}^n$ be a (1 + $\epsilon$)-approximation of Eq. (4). Using that $w$, we define

$$ Q := \{x \in \mathbb{R}^d : x^\top (\sum_{i=1}^{n} w_i a_i a_i^\top) x \leq 1\}. $$

Then the following property holds:

$$ \frac{1}{\sqrt{1+\epsilon}} \cdot Q \subseteq P \subseteq \sqrt{d} \cdot Q. $$

Moreover, $\text{vol}(\frac{1}{\sqrt{1+\epsilon}} Q) \geq \exp(-\epsilon d/2) \cdot \text{vol}(Q^*)$.

\(^1\)https://arxiv.org/pdf/1905.11580.pdf
3.4 Convexity
Here, we show the convexity of $\phi_i$.

Lemma 3.6 (Convexity, Lemma 3.4 in [CCLY19]). For $i = 1, \cdots, n$, let $\phi_i : \mathbb{R}^n \to \mathbb{R}$ be the function defined as

$$\phi_i(v) = \log \sigma_i(v) = \log (a_i^\top (\sum_{j=1}^{n} v_j a_j a_j^\top)^{-1} a_i).$$

Then $\phi_i$ is convex.

3.5 Leverage Score
We state a useful tool for leverage score from previous work [DSW22]. In [DSW22], they proved a stronger version that computes the leverage score for the matrix in the form of $A(I - V^\top V)$. We only compute the leverage score for matrix $A$ here.

Lemma 3.7 (Leverage score, Lemma 4.3 in [DSW22]). Given a matrix $A \in \mathbb{R}^{n \times d}$, we can compute a vector $\tilde{\sigma} \in \mathbb{R}^m$ in $\tilde{O}(\epsilon_\sigma^{-2} (\text{nnz}(A) + d^2))$ time, so that, $\tilde{\sigma}$ is an approximation of the leverage score of matrix $A$, i.e.,

$$\tilde{\sigma} \in (1 \pm \epsilon_\sigma) \cdot \sigma(A),$$

with probability at least $1 - \delta_\sigma$. The $\tilde{O}$ hides the $\log(d/\delta_\sigma)$ factor.

4 Problem Formulation
In this section, we give the formal definition for the John Ellipsoid of a symmetric polytope. We first give a characterization of any symmetric polytope.

Definition 4.1 (Symmetric convex polytope). We define a symmetric convex polytope as

$$P := \{ x \in \mathbb{R}^d : |\langle a_i, x \rangle| \leq 1, \forall i \in [n] \}. $$

We define matrix $A \in \mathbb{R}^{n \times d}$ associate with the above polytope $P \subset \mathbb{R}^d$ as a collection of column vectors, i.e., $A = (a_1, a_2, \cdots, a_n)^\top$, and we assume $A$ is full rank. Note that since $P$ is symmetric, the John Ellipsoid of it must be centered at the origin. Since any origin-centered ellipsoid $E$ is of the form $x^\top G^{-2} x \leq 1$, for a positive definite matrix $G$, we can search over the optimal ellipsoid by searching over the possible matrix $G$ as in [CCLY19]:

Maximize $\log((\det(G))^2)$,

subject to: $G \succeq 0$ \quad (3)

$$\|Ga_i\|_2 \leq 1, \forall i \in [n]$$

In [CCLY19] it is shown that the optimal $G$ must satisfy $G^{-2} = A^\top \text{diag}(w) A$, for the matrix $A$ and vector $w \in \mathbb{R}_{\geq 0}^n$. Thus, optimizing over $w$, we have the following optimization program:
Minimize \( \sum_{i=1}^{n} w_i - \log \det(\sum_{i=1}^{n} w_i a_i a_i^\top) - d \), \hspace{1cm} (4)

subject to: \( w_i \geq 0, \forall i \in [n] \).

Additionally, the optimality condition for this \( w \) has been studied in [Tod16]:

**Lemma 4.2** (Optimality criteria, Proposition 2.5 in [Tod16]). A weight \( w \in \mathbb{R}^n \) is optimal for program (Eq. (4)) if and only if

\[
\sum_{i=1}^{n} w_i = d, \\
a_j^\top Q^{-1} a_j = 1, \text{ if } w_j \neq 0 \\
a_j^\top Q^{-1} a_j < 1, \text{ if } w_j = 0.
\]

where \( Q := \sum_{i=1}^{n} w_i a_i a_i^\top \in \mathbb{R}^{d \times d} \).

Besides finding the exact John Ellipsoid, we can also find an \((1 + \epsilon)\)-approximate John Ellipsoid:

**Definition 4.3** ((1 + \(\epsilon\))-approximate John Ellipsoid). For \( \epsilon > 0 \), we say \( w \in \mathbb{R}_{\geq 0}^{n} \) is a \((1 + \epsilon)\)-approximation of program (Eq. (4)) if \( w \) satisfies

\[
\sum_{i=1}^{n} w_i = d, \\
a_j^\top Q^{-1} a_j \leq 1 + \epsilon, \forall j \in [n].
\]

where \( Q := \sum_{i=1}^{n} w_i a_i a_i^\top \in \mathbb{R}^{d \times d} \).

5 Input Sparsity Algorithm, Running time

In this section, we present the running time needed for our algorithm (Algorithm 1). We first introduce some facts that are useful to our proof.

**Fact 5.1.** For any real numbers \( a \geq 1 \) and \( b \geq 2 \), we have

\[
\log(ab) \leq 2a \cdot \log b
\]

**Proof.** We have

\[
\log(ab) \leq \log a + \log b \\
\leq a + \log b \\
\leq a \log b + \log b \\
\leq a \log b + a \log b \\
\leq 2a \log b.
\]

where the third step follows from \( \log b \geq 1 \), the forth step follows from \( a \geq 1 \).

Thus, we complete the proof.
Algorithm 1 Faster Algorithm for approximating John Ellipsoid inside symmetric polytopes

1: procedure FastApproxGeneral($A \in \mathbb{R}^{n \times d}$) \hfill \triangleright \text{Theorem 1.1}
2: \begin{itemize}
3:    \item $s \leftarrow \Theta(\epsilon^{-1})$
4:    \item $T \leftarrow \epsilon^{-1} \log(n/d)$
5:    \item $\epsilon_0 \leftarrow \Theta(\epsilon)$
6:    \item $N \leftarrow \Theta(\epsilon_0^2 d \log(nd/\delta))$
7:    \item for $i = 1 \rightarrow n$ do
8:        \begin{itemize}
9:            \item Initialize $w_{1,i} = \frac{d}{n}$
10:        \end{itemize}
11:    \end{itemize}
12: for $k = 1, \ldots, T - 1$ do
13:    \begin{itemize}
14:        \item $W_k = \text{diag}(w_k)$.
15:        \item $B_k = \sqrt{W_k}A$
16:        \item Let $S_k \in \mathbb{R}^{s \times d}$ be a random matrix where each entry is chosen i.i.d from Gaussian distribution $\mathcal{N}(0,1)$
17:        \begin{itemize}
18:            \item Ideally we want to compute $w_{k+1,i} = \| (B_k^T B_k)^{-1/2} (\sqrt{w_k}a_i) \|^2_2$ by Eq. (2).
19:        \end{itemize}
20:    \end{itemize}
21: for $i = 1 \rightarrow n$ do
22:    \begin{itemize}
23:        \item $\bar{w}_{k+1,i} \leftarrow \frac{1}{s} \| \tilde{Q}_k \sqrt{\bar{w}_{k,i} a_i} \|^2_2$
24:    \end{itemize}
25: end for
26: for $i = 1 \rightarrow n$ do
27:    \begin{itemize}
28:        \item $u_i = \frac{1}{T} \sum_{k=1}^{T} w_{k,i}$ \hfill \triangleright \text{Lemma 6.4}
29:    \end{itemize}
30: end for
31: for $i = 1 \rightarrow n$ do
32:    \begin{itemize}
33:        \item $v_i = \sum_{j=1}^{d} u_j$ \hfill \triangleright \text{Lemma 6.5}
34:    \end{itemize}
35: end for
36: $V = \text{diag}(v)$.
37: return $V$ and $A^T VA$ \hfill \triangleright \text{Approximate John Ellipsoid inside the polytope}
38: end procedure

Fact 5.2. For any $a \geq 1$ and $b \geq 2$, we have

$$a + \log(ab) \leq 3a \log b$$

Proof. Using Fact 5.1, we have

$$\log(ab) \leq 2a \log b$$
Then we have
\[ a + \log(ab) \leq a + 2a \log b \leq 3a \log b \]
where the last step follows from \( a \leq a \log b \).

**Fact 5.3.** For any \( n, d \) such that \( 2 \leq d \leq n \leq \text{poly}(d) \). For any \( \delta \in (0, 0.1) \), we have
\[ \log(d \log(n/d)/\delta) = O(\log(d/\delta)) \]

**Proof.** Let \( c > 1 \) denote some constant value such that \( n \leq d^c \).
Then we can write
\[
d \log(n/d) \leq d \log(d^{c-1}) = (c - 1)d \log d \\
\leq cd \log d \\
\leq cd^2
\]
where the first step follows from \( n \leq d^c \), and the last step follows from \( \log d \leq d \).
Thus
\[
\log(d \log(n/d)/\delta) \leq \log(cd^2/\delta) \\
\leq 2c \log(d^2/\delta) \\
\leq 2c \log(d^2/\delta^2) \\
= 4c \log(d/\delta) \\
= O(\log(d/\delta)).
\]
where the second step follows from Fact 5.1, the third step follows from \( \delta \in (0, 1) \). \( \square \)

Next, we show the running time of Theorem 1.1.

**Lemma 5.4** (Performance of Algorithm 1). Given a symmetric convex polytope, for all \( \epsilon \in (0, 1) \), Algorithm 1 can find a \((1 + \epsilon)^2\)-approximation of John Ellipsoid inside this polytope with \( \epsilon_0 = \Theta(\epsilon) \) and \( T = O(\epsilon^{-1} \log(n/d)) \) in time
\[ O((\epsilon^{-1} \log(d/\delta) \cdot \text{nnz}(A) + \epsilon^{-2} \log(n/\delta) \cdot d^\omega)T), \]
where \( \omega \approx 2.373 \) denote the current matrix multiplication exponent [Wil12, AW21].

**Proof.** At first, initializing the vector \( w \in \mathbb{R}^n \) takes \( O(n) \) time. In the main loop, the per iteration running time can be decomposed as follows:

- Calculating matrix \( B_k \in \mathbb{R}^{n \times d} \) takes \( O(\text{nnz}(A)) \) time. Due to the structure of matrix \( W_k \), we only need to multiply the non-zero entries of \( i \)-th row by \( w_{k,i} \) to get matrix \( B_k \). The total non-zero entries here is \( \text{nnz}(A) \).
- Initializing matrix \( S_k \in \mathbb{R}^{s \times d} \), where \( s = \Theta(\epsilon^{-1}) \), takes \( O(\epsilon^{-1} n) \) time.
- Generating diagonal matrix \( D_k \in \mathbb{R}^{n \times n} \) takes \( O(\epsilon_{s-2}^2(\text{nnz}(A) + d^\omega)) \) time by using Lemma 3.7.
- Computing matrix \( \tilde{H}_k = (B_k^T D_k B_k)^{-1/2} \) contains three steps.
We first compute \( B_k^T D_k B_k \in \mathbb{R}^{d \times d} \), where \( D_k \) is a diagonal matrix with \( N \) non-zero entries and \( N = \Theta(\epsilon_0^{-2} d \log(nd/\delta)) \). It takes \( O(\epsilon_0^{-2} d^2 \log(nd/\delta)) \) time by using fast matrix multiplication. As \( n = \text{poly}(d) \), we can simplify it as \( O(\epsilon_0^{-2} d^2 \log(n/\delta)) \).

Second, we compute the inverse of the result in the first step, which takes \( O(d^2) \) time.

Third, we take the square root of the result in second step. To take square root of a matrix \( T \in \mathbb{R}^{d \times d} \), we can first decompose \( T \) as \( U \Sigma V^\top \) using SVD, which takes \( O(d^2) \). Then we take the square root of the diagonal matrix \( \Sigma \), which takes \( O(d) \). Then, we multiply them back together to get \( T^{1/2} \), which takes \( O(d^2) \). Hence, the time needed for the final step is \( O(d^2) + O(d) + O(d^2) = O(d^2) \)

As \( O(d^2) \) is less than \( O(\epsilon_0^{-2} d^2 \log(n/\delta)) \), the total running time for computing \( H_k \) is \( O(\epsilon_0^{-2} d^2 \log(n/\delta)) \).

- Computing matrix \( \tilde{Q}_k \) takes \( O(\epsilon^{-1} d^2) \) time.
- Updating vector \( w_{k+1} \) takes \( O(\epsilon^{-1} \text{nnz}(A)) \) time. We need \( O(\epsilon^{-1} \text{nnz}(a_i)) \) time for each iteration to compute \( \frac{1}{\sqrt{n}} \| \tilde{Q}_k \sqrt{w_{k+1}} \|_2^2 \). Hence to update vector \( w_{k+1} \), we need \( \sum_{i=1}^{n} O(\epsilon^{-1} \text{nnz}(a_i)) = O(\epsilon^{-1} \text{nnz}(A)) \).

Hence, with \( \epsilon_\sigma = \Theta(1) \) and \( \delta_\sigma = \frac{\delta}{\log(n/\delta)} \), the overall per iteration running time for the main loop is

\[
O(\text{nnz}(A)) + O(\epsilon^{-1} n) + \tilde{O}(\epsilon_\sigma^{-2}(\text{nnz}(A) + d\omega)) + O(\epsilon_0^{-2} d^2 \log(n/\delta)) + O(\epsilon^{-1} d^2) + O(\epsilon^{-1} \text{nnz}(A))
\]

\[
= \tilde{O}(\epsilon_\sigma^{-2}(\text{nnz}(A) + d\omega)) + O(\epsilon_0^{-2} d^2 \log(n/\delta)) + O(\epsilon^{-1} d^2) + O(\epsilon^{-1} \text{nnz}(A))
\]

\[
= O(\epsilon_\sigma^{-2} \text{nnz}(A) + d\omega) \log(d/\delta_\sigma) + \epsilon_0^{-2} d^2 \log(n/\delta) + \epsilon^{-1} d^2 + \epsilon^{-1} \text{nnz}(A))
\]

\[
= O((\epsilon_\sigma^{-2} \log(d/\delta_\sigma) + \epsilon^{-1}) \text{nnz}(A) + (\epsilon_\sigma^{-2} \log(d/\delta_\sigma) + \epsilon_0^{-2} \log(n/\delta)) d\omega + \epsilon^{-1} d^2)
\]

\[
= O((\log(d/\delta_\sigma) + \epsilon^{-1}) \text{nnz}(A) + (\log(d/\delta_\sigma) + \epsilon_0^{-2} \log(n/\delta)) d\omega + \epsilon^{-1} d^2)
\]

where the first step comes from \( \text{nnz}(A) > n \) and \( \text{nnz}(A) > d \), the second step follows from the definition of \( \tilde{O} \), the third step follows from reorganization, the fourth step follows from \( \epsilon_\sigma = \Theta(1) \).

Note that without loss of generality, we can assume \( 2 \leq d \leq n \leq \text{poly}(d) \). For convenient of the simplifying complexity related to logs, we can assume \( n \geq 2d \) and \( \delta \in (0,0.1) \) and \( \epsilon \in (0,0.1) \).

We can try to further simplify \( \log(d/\delta_\sigma) + \epsilon^{-1} \), using the definition of \( \delta_\sigma = \frac{\delta}{\log(n/\delta)} \), then we can have

\[
\log(d/\delta_\sigma) + \epsilon^{-1} = \log\left(\frac{d \log(n/\delta)}{\delta_\sigma}\right) + \epsilon^{-1}
\]

\[
= O(\epsilon^{-1} \log\left(\frac{d \log(n/\delta)}{\delta}\right))
\]

\[
= O(\epsilon^{-1} \log(d/\delta))
\]

where the first step follows from definition of \( \delta_\sigma \), the second step follows from Fact 5.2 and the last step follows from Fact 5.3.

Hence yields the total running time for the main loop as

\[
O((\epsilon^{-1} \log(d/\delta) \cdot \text{nnz}(A) + (\log(d/(\delta_\sigma)) + \epsilon_0^{-2} \log(n/\delta)) \cdot d\omega + \epsilon^{-1} \cdot d^2) T).
\]

Then, computing the average of vector \( w \) from time 1 to \( T \), and computing the vector \( v_i \) takes \( O(nT) \) time. Finally, note that we don’t have to output \( A^\top VA \). Instead, we can just output \( A \) and vector \( v \), which takes \( O(n) \) time.
Therefore, by calculation, the running time of Algorithm 1 is:

\[
O((\epsilon^{-1} \log(d/\delta) \cdot \text{nnz}(A) + (\log(d/(\delta \epsilon)) + \epsilon_0^{-2} \log(n/\delta)) \cdot d^\omega + \epsilon^{-1} \cdot d^2)T)
\]

\[
= O((\epsilon^{-1} \log(d/\delta) \cdot \text{nnz}(A) + (\log(d/(\delta \epsilon)) + \epsilon^{-2} \log(n/\delta)) \cdot d^\omega)T)
\]

where the first step comes from \(\epsilon_0 = \Theta(\epsilon)\) and \(\omega \geq 2\), and the last step follows from \(n > d\) and \(\epsilon \in (0, 1)\). Note that \(\omega\) denotes the exponent of matrix multiplication [Wil12, LG14, AW21].

6 Input Sparsity Algorithm, Correctness Part

In Section 6.1, we provide a novel telescoping lemma. In Section 6.2, we provide a high probability bound on \(\lambda_i(w)\). In Section 6.3, we show the upper bound of \(\phi_i\). In Section 6.4, we show the correctness of our implementation.

6.1 Telescoping Lemma

The following lemma is a novel telescoping lemma (compared to Lemma C.4 in arXiv version of [CCLY19]). The previous work telescoping lemma only handles sketching. Our Lemma handles both sketching and sampling.

**Lemma 6.1 (Telescoping, Algorithm 1).** Fix \(T\) as the number of main loops executed in Algorithm 1. Let \(u\) (note that the output of algorithm is \(v\), and \(u\) is the non-normalization version of \(v\)) be vector generated during the Algorithm 1. Then for \(i \in [n]\), with probability \(1 - \delta_0\),

\[
\phi_i(u) \leq \frac{1}{T} \log \frac{n}{d} + \frac{1}{T} \sum_{k=1}^{T} \log \frac{\tilde{w}_{k,i}}{w_{k,i}} + \epsilon_0
\]

**Proof.** We define

\[
u := (u_1, u_2, \cdots, u_n) \in \mathbb{R}^n.
\]

For \(k = 1, \cdots, T - 1\), we define

\[
w_k := (w_{k,1}, \cdots, w_{k,n}) \in \mathbb{R}^n
\]

and

\[
\tilde{w}_{k+1} := (w_{k,1} \cdot \sigma_1(w_{k}), \cdots, w_{k,n} \cdot \sigma_n(w_{k})).
\]

By the convexity of \(\phi_i\) (Lemma 3.6)

\[
\phi_i(u) = \phi_i\left(\frac{1}{T} \sum_{k=1}^{T} w_k\right) \leq \frac{1}{T} \sum_{k=1}^{T} \phi_i(w_k)
\]
\[
\begin{align*}
&= \frac{1}{T} \sum_{k=1}^{T} \log \sigma_i(w_k) \\
&= \frac{1}{T} \sum_{k=1}^{T} \log \frac{\hat{w}_{k+1,i}}{w_{k,i}} \\
&= \frac{1}{T} \sum_{k=1}^{T} \log \frac{\hat{w}_{k+1,i} \cdot \hat{w}_{k,i} \cdot \hat{w}_{k,i}}{w_{k,i} \cdot \hat{w}_{k,i} \cdot \hat{w}_{k,i}} \\
&= \frac{1}{T} \left( \sum_{k=1}^{T} \log \frac{\hat{w}_{k,i}}{w_{k,i}} + \sum_{k=1}^{T} \log \frac{\hat{w}_{k,i}}{w_{k,i}} + \sum_{k=1}^{T} \log \frac{\hat{w}_{k,i}}{w_{k,i}} \right) \\
&= \frac{1}{T} \log \frac{n \hat{w}_{T+1,i}}{d} + \frac{1}{T} \sum_{k=1}^{T} \log \frac{\hat{w}_{k,i}}{w_{k,i}} + \frac{1}{T} \sum_{k=1}^{T} \log \frac{\hat{w}_{k,i}}{w_{k,i}} \\
&\leq \frac{1}{T} \log \frac{n}{d} + \log(1 + \epsilon_0) + \frac{1}{T} \sum_{k=1}^{T} \log \frac{\hat{w}_{k,i}}{w_{k,i}} \\
&\leq \frac{1}{T} \log \frac{n}{d} + \epsilon_0 + \frac{1}{T} \sum_{k=1}^{T} \log \frac{\hat{w}_{k,i}}{w_{k,i}} \\
\end{align*}
\]

where the first step uses the definition of \( \phi_i(u) \), the second step uses the convexity of \( \phi_i \), the third step uses the definition of \( \phi_i \), the fourth step uses the definition of \( \sigma_i \), the fifth step comes from reorganization, the sixth step comes from reorganization, the seventh step comes from reorganization, the eighth step uses our initialization on \( w_1 \), the ninth step comes from Lemma 2.1, the tenth step uses Corollary 8.5, and the final step comes from the fact \( \log(1 + \epsilon_0) \leq \epsilon_0 \).

Note that, the tenth step only holds with probability \( 1 - \delta_0 \), which gives us the high probability argument in the lemma statement.

\[\square\]

### 6.2 High Probability Bound of \( \lambda_i \)

**Lemma 6.2** (Implicitly in Lemma C.5 and Lemma C.6 in arXiv version of [CCLY19]). *If \( s \) is even, define \( \lambda_i(w_k) = \log \frac{\hat{w}_{k,i}}{w_{k,i}} \) then we have*

\[
\mathbb{E}[\lambda_i(w_k)] = \frac{2}{s} \\
\mathbb{E}[(\exp(\lambda_i(w_k)))^\alpha] \leq \left( \frac{n}{d} \right)^{\frac{n}{s}} \left( 1 + \frac{2\alpha}{sT - 2\alpha} \right)^T.
\]

where the randomness is taken over the sketching matrices \( \{S^{(k)}\}_{k=1}^{T-1} \).

We provide a high probability bound of \( \lambda_i \) as follows.
Lemma 6.3 (High probability Argument on $\lambda_i(w)$). Let $\lambda_i(w) = \log \frac{\hat{w}_{k,i}}{w_{k,i}}$. Then we have

$$\Pr[\exp(\lambda_i(w)) \geq 1 + \epsilon] \leq \left( \frac{n}{d} \right)^{\alpha} \frac{4\alpha}{(1 + \epsilon)^{\alpha}}.$$  

Moreover, with our choice of $s, T$, with large enough $n$ and $d$, we have:

$$\Pr[\exp(\lambda_i(w)) \geq 1 + \epsilon] \leq \frac{\delta}{n}$$

**Proof.** In the proof, we pick $\alpha = \frac{2}{\epsilon} \log \frac{n}{\delta}$. By the choice of $\alpha$, we have that:

$$\alpha \geq \frac{\log(n/\delta)}{\log \left( \frac{1 + \epsilon}{1 + \epsilon / 4} \right)}$$  \hspace{1cm} (5)

$$sT \geq 4\alpha$$  \hspace{1cm} (6)

Then, for $i \in [n]$, by Markov Inequality on the $\alpha$ moment of $\exp(\lambda_i(w))$, we have that:

$$\Pr[\exp(\lambda_i(w)) \geq 1 + \epsilon] = \Pr[\exp(\lambda_i(w))^\alpha \geq (1 + \epsilon)^\alpha]$$

$$\leq \mathbb{E}[\exp(\lambda_i(w))^\alpha] \leq \left( \frac{n}{d} \right)^{\alpha} \cdot \frac{1 + 2\alpha}{sT - 2\alpha}\frac{(1 + \epsilon)^\alpha}{(1 + \epsilon/4)^\alpha}$$

$$\leq \frac{n}{d} \cdot \frac{1 + 2\alpha}{sT/2} \frac{(1 + \epsilon)^\alpha}{(1 + \epsilon/4)^\alpha}$$

where the first step comes from calculation, the second step comes from Markov Inequality, the third step comes from applying Lemma 6.2, the fourth step comes from the choice of $\alpha$ that $sT \geq 4\alpha$, and the final step comes from $1 + x \leq e^x$.

Moreover, for large enough $n$ and $d$, we have that:

$$\left( \frac{n}{d} \right)^{\alpha} \leq \left( \frac{n}{d} \right)^{\epsilon/10 \log(n/\delta)} \leq 1 + \epsilon/10$$  \hspace{1cm} (7)

Also, we have:

$$e^\frac{4}{\epsilon} = e^\frac{\epsilon}{10} \leq 1 + \epsilon/10$$  \hspace{1cm} (8)

Hence,

$$\Pr[\exp(\lambda_i(w)) \geq 1 + \epsilon] \leq \left( \frac{1 + \epsilon/10}{1 + \epsilon} \right)^\alpha$$

$$\leq \left( \frac{1 + \epsilon/4}{1 + \epsilon} \right)^\alpha$$

$$\leq \frac{\delta}{n}$$

where the first step comes from applying Eq (7) and Eq. (8), the second step comes from calculation, and the last step comes from Eq. (5).  \hfill \square
6.3 Upper Bound of $\phi_i$

Then, we show the upper bound of $\phi_i$.

**Lemma 6.4 (\(\phi_i\), formal version of Lemma 2.2).** Let $u$ be the vector generated during the Algorithm 1, fix the number of iterations executed in the algorithm as $T$ and $s = 1000/\epsilon$, with $1 - \delta - \delta_0$, we have

$$\phi_i(u) \leq \frac{1}{T} \log \left( \frac{n}{d} \right) + \frac{\epsilon}{250} + \epsilon_0$$

\forall i \in [n].

**Proof.** To begin with, by Lemma 6.1, we have that, with probability $1 - \delta_0$,

$$\phi_i(u) \leq \frac{1}{T} \log \frac{n}{d} + \frac{1}{T} \sum_{k=1}^{T} \log \frac{\tilde{w}_{k,i}}{w_{k,i}} + \epsilon_0$$

$$= \frac{1}{T} \log \frac{n}{d} + \frac{1}{T} \sum_{k=1}^{T} \lambda_i(w_k) + \epsilon_0$$

We have with probability $1 - \delta - \delta_0$, for all $i \in [n]$:

$$\phi_i(u) \leq \frac{1}{T} \log \frac{n}{d} + \frac{\epsilon}{1000} + \epsilon_0$$

$$\leq \frac{1}{T} \log \frac{n}{d} + \frac{\epsilon}{250} + \epsilon_0$$

where the first step follows from Lemma 6.3. \qed

6.4 Main Result

In terms of Definition 4.3, to show Algorithm 1 provides a reasonable approximation of the John Ellipsoid, it is necessary to prove that for the output $v \in \mathbb{R}^n$ of Algorithm 1, $\sigma_i(v) \leq 1 + O(\epsilon)$, $\forall i \in [n]$ Our main result is shown below.

**Theorem 6.5 (Correctness).** Let $\epsilon_0 = \frac{\epsilon}{1000}$. Let $v \in \mathbb{R}^n$ be the output of Algorithm 1. For all $\epsilon \in (0, 1)$, when $T = O(\epsilon^{-1} \log(n/d))$, we have

$$\Pr \left[ \sigma_i(v) \leq (1 + \epsilon)^2, \forall i \in [n] \right] \geq 1 - \delta - \delta_0$$

Moreover,

$$\sum_{i=1}^{n} v_i = d.$$

Therefore, Algorithm 1 provides $(1 + \epsilon)^2$-approximation to program Eq. (4)

**Proof.** We set

$$T = 1000 \epsilon^{-1} \log(n/d) \quad \text{and} \quad \epsilon_0 = \epsilon/1000,$$
By Lemma 6.4, we know the succeed probability is $1 - \delta - \delta_0$. Then, we have for $i \in [n]$,

$$\log \sigma_i(u) = \phi_i(u)$$

$$\leq \frac{1}{T} \log(n/d) + \epsilon/250 + \epsilon_0$$

$$\leq \frac{\epsilon}{50}$$

$$\leq \log(1 + \epsilon)$$

where the first step uses the definition of $\sigma_i$, the second step uses Lemma 2.2, the third step comes from calculation, and the last step comes from the fact that when $0 < \epsilon < 1$, $\frac{\epsilon}{50} \leq \log(1 + \epsilon)$.

In conclusion, $\sigma_i(u) \leq 1 + \epsilon$.

Because, we choose $v_i = \sum_{j=1}^{d} u_j u_i$, then

$$\sum_{i=1}^{n} v_i = d.$$

Next, we have

$$\sigma_i(v) = a_i^\top (A^\top VA)^{-1} a_i$$

$$= a_i^\top \left( \sum_{i=1}^{n} u_i \right) A^\top UA)^{-1} a_i$$

$$= \sum_{i=1}^{n} u_i \sigma_i(u)$$

$$\leq (1 + \epsilon) \cdot \sigma_i(u)$$

$$\leq (1 + \epsilon) \cdot (1 + \epsilon)$$

where the first step uses the definition of $\sigma_i(v)$, the second step uses the definition of $V$, the third step uses the definition of $\sigma_i(u)$, the fourth step comes from $u_i$ is at most $(1 + \epsilon)$ true leverage score, and the summation of true leverage scores is $d$ (by Lemma 2.1), the last step comes from $\sigma_i(u) \leq (1 + \epsilon)$.

Thus, we complete the proof.

---

**Theorem 6.6 (Correctness part of Theorem 1.1).** Given a matrix $A \in \mathbb{R}^{n \times d}$, we define a centrally symmetric polytope $P$ as follows:

$$\{ x \in \mathbb{R}^d : -1_n \leq Ax \leq 1_n \}.$$ 

Then, given $\epsilon \in (0, 1)$, Algorithm 1 that outputs an ellipsoid $Q$ satisfies:

$$\frac{1}{\sqrt{1 + \epsilon}} \cdot Q \subseteq P \subseteq \sqrt{d} \cdot Q.$$ 

**Proof.** By combining Theorem 6.5 and Lemma 3.5, we can complete the proof.

---

### 7 Faster Algorithm for Small Treewidth Setting

In this section, we provide an algorithm (Algorithm 2) that approximate the John Ellipsoid in $O(\epsilon^{-1} \cdot (n \tau^2) \cdot \log(n/d))$ time with small treewidth setting. In Section 7.1, we prove the correctness of our implementation. In Section 7.2, we show the running time of it.
Algorithm 2 Faster Algorithm for approximating John Ellipsoid inside symmetric polytopes (under tree width setting)

1: procedure FastApproxTreeWidth($A \in \mathbb{R}^{n \times d}$) \Comment{Theorem 1.4}
2: \Comment{A symmetric polytope given by $-1_n \leq Ax \leq 1_n$ where $A \in \mathbb{R}^{n \times d}$}
3: \Comment{$s \leftarrow \Theta(\epsilon^{-1})$}
4: $T \leftarrow \epsilon^{-1} \log(n/d)$
5: for $i = 1 \rightarrow n$ do
6: Initialize $w_{1,i} = \frac{d}{n}$
7: end for
8: for $k = 1, \cdots, T - 1$ do
9: $W_k = \text{diag}(w_k)$.
10: $B_k = \sqrt{W_k}A$
11: $L_k \leftarrow \text{Cholesky decomposition matrix for } B_k^\top B_k \text{ i.e., } L_k L_k^\top = B_k^\top B_k$ \Comment{$O(n\tau^2)$ time}
12: for $i = 1 \rightarrow n$ do
13: $w_{k+1,i} \leftarrow b_{k,i}^\top(L_k L_k^\top)^{-1}b_{k,i}$ \Comment{$O(\tau^2)$ time}
14: end for
15: end for
16: for $i = 1 \rightarrow n$ do
17: $u_i = \frac{1}{T} \sum_{k=1}^{T} w_{k,i}$
18: end for
19: $U = \text{diag}(u)$. \Comment{$U$ is a diagonal matrix with the entries of $u$}
20: return $U$ and $A^\top UA$ \Comment{Approximate John Ellipsoid inside the polytope}
21: end procedure

7.1 Correctness

Note that for Algorithm 2, we compute the exact leverage score of each row, the randomness of sketching matrix $S$ and diagonal sampling $D$ doesn’t play a role in our analysis. It immediately follows that the following corollary holds:

**Corollary 7.1** (Telescoping, Algorithm 2). Fix $T$ as the number of main loops executed in Algorithm 2. Let $u$ (note that the output of algorithm is $v$, and $u$ is the non-normalization version of $v$) be vector generated during the Algorithm 2. Then for $i \in [n]$,

$$\phi_i(u) \leq \frac{1}{T} \log \frac{n}{d}$$

Next, we prove the correctness of our implementation with small treewidth setting.

**Theorem 7.2** (Correctness of Algorithm 2). Let $u$ be the output of Algorithm 2. For all $\epsilon \in (0, 1)$, when $T = O(\epsilon^{-1} \log(n/d))$, we have:

$$\sigma_i(u) \leq (1 + \epsilon)$$

$$\sum_{i=1}^{n} u_i = d$$

**Proof.** We set

$$T = 1000 \epsilon^{-1} \log(n/d)$$
We also have for $i \in [n]$,
\[
\log \sigma_i(u) = \phi_i(u)
\leq \frac{1}{T} \log(n/d)
\leq \frac{\epsilon}{50}
\leq \log(1 + \epsilon)
\]
where the first step uses the definition of $\sigma_i(u)$, the second step follows from Corollary 7.1, the third step follows from calculation, and the last step follows from the fact that for small $\epsilon$, $\epsilon/50 \leq \log(1 + \epsilon)$. In conclusion, $\sigma_i(u) \leq 1 + \epsilon$.

Additionally, since for $k \in [T]$, each row of $w_{k,i}$ is a leverage score of some matrix, according to Lemma 2.1, we have:
\[
\sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{1}{T} \sum_{k=1}^{T} w_{k,i}
= \frac{1}{T} \sum_{k=1}^{T} \sum_{i=1}^{n} w_{k,i}
= \frac{1}{T} \sum_{k=1}^{T} d
= \frac{1}{T} Td
= d
\]
where the first line uses the definition of $u$, the second step follows from reorganization, the third step follows from Lemma 2.1, the fourth and the final step comes from calculation.

Thus, we complete the proof. 

\section*{7.2 Running Time}

The rest of this section is to prove the running time of Algorithm 2. We first show the time needed to compute the leverage score with small treewidth setting.

\textbf{Lemma 7.3.} Given the Cholesky factorization $LL^\top$. Let $a_i^\top$ denote the $i$-th row of $A$, for each $i \in [n]$. Let $B = \sqrt{H} A \in \mathbb{R}^{n \times d}$ where $H$ is a nonnegative diagonal matrix. Let $\sigma_i = b_i^\top (B^\top B)^{-1} b_i$. We can compute $\sigma \in \mathbb{R}^n$ in $O(n \tau^2)$ time.

\textbf{Proof.} Let $LL^\top = B^\top B$ be Cholesky facotization decomposition. Then, we have
\[
b_i^\top (B^\top B)^{-1} b_i = b_i^\top L^{-\top} L^{-1} b_i
= (L^{-1} b_i)^\top (L^{-1} b_i).
\]
Using the property of elimination tree, we have each row of $B$ has sparsity $\tau$ and they lie on a path of elimination tree $T$. In this light, we are able to output $L^{-1} b_i$ in $O(\tau^2)$ time, and then compute a solution of sparsity $O(\tau)$ (Lemma 5.4 in [DLY21]).

Therefore, we can compute the score for a single column in $O(\tau^2)$. In total, it takes $O(n \tau^2)$. 

\section*{18}
Figure 4: (a) A $10 \times 10$ positive definite matrix $P = AA^\top$, where the blue dot represent the non-zero elements in $P$. (b) The Cholesky factor $L$ of $AA^\top$. (c) The corresponding elimination tree for matrix $P$, where each node represent one column in the Cholesky factor. We can see that, as the row index of the first subdiagonal nonzero entry of the 6-th column is 8, the parent of node 6 is 8. Furthermore, the non-zero pattern of this column is $\{6, 8, 10\}$, which is a subset of vertices on the path from node 6 to the root in the elimination tree.

Next, we show our main result.

**Theorem 7.4 (Performance of Algorithm 2).** For all $\epsilon \in (0, 1)$, we can find a $(1+\epsilon)$-approximation of John Ellipsoid defined by matrix $A$ with treewidth $\tau$ inside a symmetric convex polytope in time $O((n\tau^2) \cdot T)$ where $T = \epsilon^{-1} \log(n/d)$.

**Proof.** At first, initializing the vector $w$ takes $O(n)$ time. In the main loop, the per iteration running time can be decomposed as follows:

- Using Lemma 3.3, calculating the Cholesky decomposition for $B_k^\top B_k$ takes $O(n\tau^2)$ time.
- Using Lemma 7.3, computing $w_{k+1}$ takes $O(n\tau^2)$ time.

Hence, the overall per iteration running time for the main loop is $O(n\tau^2)$ time, hence yields the total running time for the main loop as $O((n\tau^2)T)$.

Then, computing the average of vector $w$ from time 1 to $T$, and computing the vector $v_i$ takes $O(nT)$ time. Finally, note that we don’t have to output $A^\top VA$. Instead, we can just output $A$ and vector $v$, which takes $O(n)$ time.

Therefore, by calculation, the running time of Algorithm 2 is: $O((n\tau^2)T)$. Thus, we complete the proof. \qed

8 Sampling

In this section, we provide the sparsification tool used in Line 15 of Algorithm 1. Especially, we show how to approximate the matrix that has pattern $A^\top WA$, where $W$ is some non-negative diagonal matrix, by using sample matrix $D$.

**Lemma 8.1 (Matrix Chernoff Bound [Tro11]).** Let $X_1, \ldots, X_s$ be independent copies of a symmetric random matrix $X \in \mathbb{R}^{d \times d}$ with $E[X] = 0$, $\|X\| \leq \gamma$ almost surely and $\|E[X^\top X]\| \leq \sigma^2$. Let $C = \frac{1}{s} \sum_{i \in [s]} X_i$. For any $\epsilon \in (0, 1)$,

$$\Pr[\|C\| \geq \epsilon] \leq 2d \cdot \exp\left(-\frac{s\epsilon^2}{\sigma^2 + \gamma\epsilon/\sqrt{3}}\right).$$
To better monitor the whole process, it is useful to write \( H(w) \) as \( A^\top WA \), where \( A \in \mathbb{R}^{n \times d} \) is the constraint matrix and \( W \) is a diagonal matrix with \( W = \text{diag}(w) \). The sparsification process is then sample the rows from the matrix \( \sqrt{WA} \).

We define the leverage score as follows:

**Definition 8.2.** Let \( B \in \mathbb{R}^{n \times d} \) be a full rank matrix. We define the leverage score of the \( i \)-th row of \( B \) as

\[
\sigma_i(B) := b_i^\top (B^\top B)^{-1} b_i,
\]

where \( b_i \) is the \( i \)-th row of \( B \).

Next we define our sampling process as follows:

**Definition 8.3 (Sampling process).** For any \( w \in K \), let \( H(w) = A^\top WA \). Let \( p_i \geq \beta \cdot \sigma_i(\sqrt{WA})/d \), suppose we sample with replacement independently for \( s \) rows of matrix \( \sqrt{WA} \), with probability \( p_i \) of sampling row \( i \) for some \( \beta \geq 1 \). Let \( (i(j)) \) denote the index of the row sampled in the \( j \)-th trial. Define the generated sampling matrix as

\[
\tilde{H}(w) := \frac{1}{s} \sum_{j=1}^{s} \frac{1}{p_{i(j)}} w_{i(j)} a_{i(j)} a_{i(j)}^\top.
\]

For our sampling process defined as Definition 8.3, we can have the following guarantees:

**Lemma 8.4 (Sample using Matrix Chernoff).** Let \( \epsilon_0, \delta_0 \in (0, 1) \) be precision and failure probability parameters, respectively. Suppose \( \tilde{H}(w) \) is generated as in Definition 8.3, then with probability at least \( 1 - \delta_0 \), we have

\[
(1 - \epsilon_0) \cdot H(w) \preceq \tilde{H}(w) \preceq (1 + \epsilon_0) \cdot H(w).
\]

Moreover, the number of rows sampled is

\[
s = \Theta(\beta \cdot \epsilon_0^{-2} d \log(d/\delta_0)).
\]

**Proof.** Similar to the proof of Lemma 5.2 in [DSW22], except we use \( \sqrt{WA} \) instead of \( A \). \qed

**Corollary 8.5.** Let \( \epsilon_0 \) denote the parameter defined as Algorithm 1. Then we have with probability \( 1 - \delta_0 \)

\[
(1 - \epsilon_0) \cdot \tilde{w}_i \leq \tilde{w}_i \leq (1 + \epsilon_0) \tilde{w}_i,
\]

for all \( i \in [n] \).

**Proof.** Since if \( (1 - \epsilon_0)A \preceq B \preceq (1 + \epsilon_0)A \), then for all \( x \), we know \( (1 - \epsilon_0) \cdot x^\top Ax \leq x^\top Bx \leq (1 + \epsilon_0) \cdot x^\top Ax \). Thus, using lemma (Lemma 8.4) implies the weights guarantees. \qed

**9 Conclusion**

Our paper studies the problem computing John Ellipsoid of a symmetric polytope. For the aforementioned problem, we use leverage score-based sampling with sketching techniques to speed up the state-of-the-art [CCLY19] \( O(nd^2) \) time and achieve per iteration running time \( \tilde{O}(\epsilon^{-1} \text{nnz}(A) + \epsilon^{-2} d^2) \). Additionally, under the extra condition that the matrix defining polytope has low treewidth \( \tau \), we give an algorithmic solution for approximate John Ellipsoid in time \( O(n\tau^2) \) cost per iteration.
References

[ACP87] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[BGHK95] Hans L Bodlaender, John R Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[BGS21] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1000–1008. IEEE, 2021.

[BLSS20] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 775–788, 2020.

[BPSW21] Jan van den Brand, Binghui Peng, Zhao Song, and Omri Weinstein. Training (over-parametrized) neural networks in near-linear time. In *ITCS*, 2021.

[BSS12] Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

[BWZ16] Christos Boutsidis, David P Woodruff, and Peilin Zhong. Optimal principal component analysis in distributed and streaming models. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 236–249, 2016.

[CCLY19] Michael B Cohen, Ben Cousins, Yin Tat Lee, and Xin Yang. A near-optimal algorithm for approximating the john ellipsoid. In *Conference on Learning Theory*, pages 849–873. PMLR, 2019.

[CLS19] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *STOC*, 2019.

[CP15] Michael B Cohen and Richard Peng. Lp row sampling by lewis weights. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 183–192, 2015.

[CW13] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference (STOC)*, pages 81–90, 2013.

[Dav06] Timothy A Davis. *Direct methods for sparse linear systems*. SIAM, 2006.

[DLY21] Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: a multiscale representation of robust central path. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1784–1797, 2021.
[DSW22] Yichuan Deng, Zhao Song, and Omri Weinstein. Discrepancy minimization in input-sparsity time. *arXiv preprint arXiv:2210.12468*, 2022.

[FLS+18] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, Michał Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms (TALG)*, 14(3):1–45, 2018.

[GS22] Yuzhou Gu and Zhao Song. A faster small treewidth sdp solver. *arXiv preprint arXiv:2211.06033*, 2022.

[HSWZ22] Hang Hu, Zhao Song, Omri Weinstein, and Danyang Zhuo. Training overparametrized neural networks in sublinear time. In *arXiv preprint arXiv: 2208.04508*, 2022.

[Joh48] Fritz John. Extremum problems with inequalities as subsidiary conditions. In *Studies and Essays Presented to R. Courant on his 60th Birthday, January 8, 1948*, pages 187–204. Interscience Publishers, Inc., New York, N. Y., 1948.

[JSWZ21] Shuhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. A faster algorithm for solving general lps. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 823–832, New York, NY, USA, 2021. Association for Computing Machinery.

[Kha96] Leonid G Khachiyan. Rounding of polytopes in the real number model of computation. *Mathematics of Operations Research*, 21(2):307–320, 1996.

[KY05] Piyush Kumar and E Alper Yildirim. Minimum-volume enclosing ellipsoids and core sets. *Journal of Optimization Theory and Applications*, 126(1):1–21, 2005.

[LG14] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303, 2014.

[LMS13] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 3(105), 2013.

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $O(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow. In *Foundations of Computer Science (FOCS)*, 2014 IEEE 55th Annual Symposium on, pages 424–433. IEEE, 2014.

[LSZ19] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *Conference on Learning Theory*, pages 2140–2157. PMLR, 2019.

[NN94] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*, volume 13. Siam, 1994.

[NN13] Jelani Nelson and Huy L Nguyê$n$. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2013.

[NTZ13] Aleksandar Nikolov, Kunal Talwar, and Li Zhang. The geometry of differential privacy: the sparse and approximate cases. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 351–360. ACM, 2013.
[RSW16] Ilya Razenshteyn, Zhao Song, and David P. Woodruff. Weighted low rank approximations with provable guarantees. In Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing, STOC ’16, page 250–263, 2016.

[SF04] Peng Sun and Robert M Freund. Computation of minimum-volume covering ellipsoids. Operations Research, 52(5):690–706, 2004.

[SS11] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. SIAM Journal on Computing, 40(6):1913–1926, 2011.

[SWZ17] Zhao Song, David P Woodruff, and Peilin Zhong. Low rank approximation with entrywise ℓ1-norm error. In Proceedings of the 49th Annual Symposium on the Theory of Computing (STOC), 2017.

[SWZ19] Zhao Song, David P Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In SODA. arXiv preprint arXiv:1704.08246, 2019.

[SXYZ22] Zhao Song, Zhaozhuo Xu, Yuanyuan Yang, and Lichen Zhang. Accelerating frank-wolfe algorithm using low-dimensional and adaptive data structures. arXiv preprint arXiv:2207.09002, 2022.

[SXZ22] Zhao Song, Zhaozhuo Xu, and Lichen Zhang. Speeding up sparsification with inner product search data structures. 2022.

[SY21] Zhao Song and Zheng Yu. Oblivious sketching-based central path method for linear programming. In International Conference on Machine Learning, pages 9835–9847. PMLR, 2021.

[Tod16] Michael J Todd. Minimum-volume ellipsoids: Theory and algorithms. SIAM, 2016.

[Tro11] Joel A Tropp. Improved analysis of the subsampled randomized hadamard transform. Advances in Adaptive Data Analysis, 3:115–126, 2011.

[Vem05] Santosh Vempala. Geometric random walks: a survey. In Combinatorial and computational geometry, volume 52 of Math. Sci. Res. Inst. Publ., pages 577–616. Cambridge Univ. Press, Cambridge, 2005.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC), pages 887–898. ACM, 2012.

[XZZ18] Chang Xiao, Peilin Zhong, and Changxi Zheng. Bourgan: generative networks with metric embeddings. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS), pages 2275–2286, 2018.

[Zha22] Lichen Zhang. Speeding up optimizations via data structures: Faster search, sample and maintenance. Master’s thesis, Carnegie Mellon University, 2022.