Fast Deterministic CUR Matrix Decomposition with Accuracy Assurance

Yasutoshi Ida [1 2] **Sekitoshi Kanai** [1] **Yasuhiro Fujiwara** [3] **Tomoharu Iwata** [3]
Koh Takeuchi [2 4] **Hisashi Kashima** [2 4]

Abstract

The deterministic CUR matrix decomposition is a low-rank approximation method to analyze a data matrix. It has attracted considerable attention due to its high interpretability, which results from the fact that the decomposed matrices consist of subsets of the original columns and rows of the data matrix. The subset is obtained by optimizing an objective function with sparsity-inducing norms via coordinate descent. However, the existing algorithms for optimization incur high computation costs. This is because coordinate descent iteratively updates all the parameters in the objective until convergence. This paper proposes a fast deterministic CUR matrix decomposition. Our algorithm safely skips unnecessary updates by efficiently evaluating the optimality conditions for the parameters to be zeros. In addition, we preferentially update the parameters that must be nonzeros. Theoretically, our approach guarantees the same result as the original approach. Experiments demonstrate that our algorithm speeds up the deterministic CUR while achieving the same accuracy.

1. Introduction

Matrix decomposition is a fundamental tool of machine learning, and it is used to decompose the data matrix into its low-rank approximations. Among the various matrix decomposition methods such as Singular Value Decomposition (SVD), CUR matrix decomposition (CUR) (Mahoney & Drineas, 2009) has been a popular method due to its high interpretability. This is because the decomposed matrices consist of the subsets of the original columns and rows of the data matrix. Namely, the decomposed matrices preserve the original elements in the data matrix. Thanks to its high interpretability, CUR has been successfully applied to a large number of domains, including gene expression data (Bien et al., 2010), network traffic data (Tong et al., 2008), bibliographic data (Sun et al., 2007), collaborative filtering (Mackey et al., 2011), hyperspectral medical image (Mahoney et al., 2006), and text data (Drineas et al., 2008).

In the literature, randomized algorithms (Mahoney & Drineas, 2009; Sun et al., 2007; Drineas et al., 2006; Tong et al., 2008) and deterministic algorithms (Bien et al., 2010; Mairal et al., 2011; Papailiopoulos et al., 2014) were proposed for CUR. Although randomized algorithms were proposed first, they would be disconcerting to the practitioners as they obtain a different result on every run (Bien et al., 2010; Mairal et al., 2011). Specifically, when the sizes of the decomposed matrices are small, they can obtain a poor approximate result because the variance of the approximate results is large (Sun et al., 2007; Bien et al., 2010). To overcome these drawbacks, researchers have focused on deterministic algorithms that utilize a sparse optimization approach (Bien et al., 2010; Mairal et al., 2011). In deterministic algorithms, matrix decomposition is seen as a convex optimization problem with sparsity-inducing norms on the basis of group Lasso (Yuan & Lin, 2006). In particular, they optimize an objective with respect to the parameter vectors corresponding to the columns and the rows of the data matrix. Since the unimportant parameter vectors are turned into zero vectors by the sparsity-inducing norms, they can deterministically obtain the important columns and rows, which are used to construct the decomposed matrices.

Although deterministic algorithms are attractive, they suffer from high computation costs. In order to optimize the objective, they usually use coordinate descent, which iteratively updates each parameter vector corresponding to each column and row of the data matrix (Bien et al., 2010). Unfortunately, the computation cost of updating a parameter vector is quadratic with respect to the number of columns or rows of the data matrix. In addition, they need to iteratively update all the parameters until convergence. For the aforementioned reasons, deterministic algorithms require longer processing times as the sizes of data matrices increase.

[1]NTT Software Innovation Center, Tokyo, Japan [2]Department of Intelligence Science and Technology, Kyoto University, Kyoto, Japan [3]NTT Communication Science Laboratories, Kyoto, Japan [4]RIKEN Center for Advanced Intelligence Project, Tokyo, Japan. Correspondence to: Yasutoshi Ida <yasutoshi.ida@ieee.org>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

This paper proposes a fast deterministic algorithm for CUR. Our approach utilizes two ideas to speed up the deterministic CUR. The first idea is to safely skip the updates of the parameters in coordinate descent. Since a high computation cost is needed for one update in coordinate descent, we can effectively reduce the total computation cost by skipping the updates. Specifically, we identify the rows and columns whose parameters *must be zeros* at linear time with respect to the number of columns or rows by approximately evaluating the optimality conditions for the parameters to be zeros. The second idea is to preferentially update the parameters that *must be nonzeros*. Because these nonzero parameters would correspond to the important columns and rows for the construction of the decomposed matrices, our algorithm is expected to effectively optimize the objective function. Similar to that in the first idea, it can identify the columns and rows whose parameters must be nonzeros at linear time. Another advantage of our algorithm is that it does not have additional hyperparameters, which incur additional computation costs for the tuning. Theoretically, our algorithm provably guarantees convergence to the same value of the objective function as that of the original algorithm. Experiments show that our method is up to $10\times$ faster than the original method, and up to $4\times$ faster than the state-of-the-art method while achieving the same accuracy.

2. Preliminary

In this section, we first explain the deterministic CUR by following (Bien et al., 2010; Mairal et al., 2011). Next, we describe coordinate descent, which optimizes the objective of deterministic CUR. Throughout the paper, given a matrix $\mathbf{A}$, $\mathbf{A}_{(i)}$ and $\mathbf{A}^{(i)}$ denote the i-th row vector and i-th column vector of $\mathbf{A}$, respectively. Similarly, given a set of indices $\mathcal{I}$, $\mathbf{A}_{\mathcal{I}}$ and $\mathbf{A}^{\mathcal{I}}$ denote the submatrices of $\mathbf{A}$ containing only $\mathcal{I}$ rows and columns, respectively. $\|\mathbf{A}\|_F$ represents the Frobenius norm of matrix $\mathbf{A}$.

2.1. Deterministic CUR Matrix Decomposition

CUR provides a low-rank approximation to a data matrix $\mathbf{X} \in \mathbb{R}^{n\times p}$. In particular, CUR decomposes the data matrix $\mathbf{X}$ into the form of a product of three matrices as $\mathbf{X} \approx \mathbf{CUR}$, where $\mathbf{C} \in \mathbb{R}^{n\times c}$, $\mathbf{U} \in \mathbb{R}^{c\times r}$, and $\mathbf{R} \in \mathbb{R}^{r\times p}$. Unlike other low-rank approximations such as Singular Value Decomposition (SVD), CUR extracts $\mathbf{C}$ and $\mathbf{R}$ as small numbers of the column and row vectors of $\mathbf{X}$, respectively. In other words, $\mathbf{C}$ and $\mathbf{R}$ are subsets of c columns and r rows of the original data matrix $\mathbf{X}$, respectively. This property helps practitioners to interpret the result more easily than that in the case of SVD (Tong et al., 2008).

To select $\mathbf{C}$ or $\mathbf{R}$, Bien et al. (2010) utilized a sparse optimization approach. In particular, the selection of $\mathbf{C}$ or $\mathbf{R}$ from $\mathbf{X}$ can be seen as a convex optimization problem with sparsity-inducing norms. For the selection of $\mathbf{C}$, the optimization problem is defined as follows:

$$\min_{\mathbf{W}\in\mathbb{R}^{p\times p}} \frac{1}{2}\|\mathbf{X}-\mathbf{XW}\|_F^2 + \lambda\sum_{i=1}^{p}\|\mathbf{W}_{(i)}\|_2, \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{p\times p}$ is the parameter matrix, and $\lambda \geq 0$ is a regularization constant. The term $\|\mathbf{W}_{(i)}\|_2$ induces $\mathbf{W}_{(i)}$ to be a zero vector; this sparsity-inducing norm is also used in group Lasso (Yuan & Lin, 2006). The regularization constant λ controls the degree of sparsity of the parameter matrix $\mathbf{W}$. If $\mathbf{W}_{(i)}$ is a zero vector, the corresponding column of the data matrix $\mathbf{X}^{(i)}$ can be considered as an unimportant column for problem (1). On the other hand, $\mathbf{X}^{(i)}$ is important when the corresponding $\mathbf{W}_{(i)}$ is a nonzero vector. Therefore, we can select columns $\mathbf{C}$ as $\mathbf{X}^{\mathcal{I}}$, where $\mathcal{I} \subseteq \{1, ..., p\}$ represents the indices corresponding to the nonzero row vectors of $\mathbf{W}$. We note that although the above problem handles the selection of $\mathbf{C}$, Mairal et al. (2011) naturally extended the problem to the simultaneous selection of both $\mathbf{R}$ and $\mathbf{C}$. Throughout the paper, we handle problem (1) focusing on simplicity; however, our approach can be easily applied to the problem of (Mairal et al., 2011) as described in Section 3.6.

Algorithm 1 Deterministic CUR

1: $\mathbb{A} = \{1, ..., p\}$, $\mathbf{W} \leftarrow \mathbf{0}$
2: **repeat**
3: **for** each $i \in \mathbb{A}$ **do**
4: update $\mathbf{W}_{(i)}$ by Equation (2);
5: **until** $\mathbf{W}$ converges

2.2. Coordinate Descent

Problem (1) is simply solved by using coordinate descent (Bien et al., 2010). The algorithm iteratively updates each parameter vector $\mathbf{W}_{(i)}$ corresponding to each row of the parameter matrix $\mathbf{W}$ until $\mathbf{W}$ converges. Suppose that $\mathbf{X}^{(i)}$ is normalized as $\|\mathbf{X}^{(i)}\|_2 = 1$. Then, the following equation is used to update $\mathbf{W}_{(i)}$:

$$\mathbf{W}_{(i)} = (1 - \lambda/\|\mathbf{z}_i\|_2)_+ \mathbf{z}_i, \tag{2}$$

where

$$(1-\lambda/\|\mathbf{z}_i\|_2)_+ = \begin{cases} 1-\lambda/\|\mathbf{z}_i\|_2, & \text{if } 1-\lambda/\|\mathbf{z}_i\|_2 > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

In these equations, $\mathbf{z}_i \in \mathbb{R}^{1\times p}$ is computed as follows:

$$\mathbf{z}_i = \mathbf{X}^{(i)\mathrm{T}}(\mathbf{X} - \textstyle\sum_{j\neq i}^{p} \mathbf{X}^{(j)}\mathbf{W}_{(j)}). \tag{4}$$

Algorithm 1 shows the pseudocode of coordinate descent. The inner loop (lines 3–4) performs Equation (2) to update each row of $\mathbf{W}$, and the outer loop (lines 2–5) repeats the update process until $\mathbf{W}$ converges. The computation cost

of Equation (4) is $\mathcal{O}(p^2 n)$ time. Therefore, Equation (2) also requires $\mathcal{O}(p^2 n)$ time. Equation (2) can be modified to have $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ time as described in (Huang et al., 2012). However, in any case, the computation cost is still large because p^2 or pn is prohibitive for a large data matrix.

3. Proposed Approach

This section presents our fast deterministic CUR. First, we provide an overview of our ideas in Section 3.1. Next, we provide their full descriptions in Sections 3.2, 3.3, and 3.4. We then describe our algorithm in Section 3.5. Finally, we introduce an extension of our algorithm for the simultaneous selection of both $\mathbf{R}$ and $\mathbf{C}$ in Section 3.6. The omitted proofs can be found in the supplementary file.

3.1. Ideas

To obtain the solution of CUR, coordinate descent requires a long processing time since the computation cost of Equation (2) is high, and the equation is performed for all the parameter vectors at every iteration until convergence.

To speed up coordinate descent, we safely skip the computations for the rows of $\mathbf{W}$ that must be *zero* vectors during the optimization. Our approach can effectively reduce the computation cost by skipping unnecessary computations of Equation (2). To identify the unnecessary rows of $\mathbf{W}$, we approximately evaluate the optimality conditions for the parameter vectors to be zero vectors. In particular, we compute the *upper* bounds of the optimality condition scores instead of the exact scores, which require $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ time. Because the computation of the upper bound requires only $\mathcal{O}(p)$ time, we can effectively reduce the processing time of the coordinate descent.

Another idea is to preferentially update the parameter vectors $\mathbf{W}_{(i)}$ that must be *nonzero* vectors. Since these nonzero parameter vectors correspond to the columns $\mathbf{C}$, as explained in Section 2.1, we can expect the algorithm to effectively optimize the objective by intensively updating the parameter vectors. We utilize the *lower* bounds of the optimality condition scores for the parameter vectors to be zero vectors to identify such parameter vectors. Because the additional computation cost is $\mathcal{O}(p)$ time, we can efficiently identify the parameter vectors that must be nonzero vectors.

3.2. Approximations of Optimality Condition Score

This section introduces the key approximations of the optimality condition scores for the parameter vectors to be zero vectors: the upper and lower bounds of the scores. As described in Section 3.1, the upper and lower bounds are used to identify the parameter vectors that must be zero vectors and nonzero vectors at the optimal solutions, respectively. First, we introduce the lemma of the optimality condition for the zero parameter vector and its score as follows:

Lemma 1 (Optimality Condition for Zero Parameter) *Let $K_i := \|\boldsymbol{z}_i\|_2$ be the optimality condition score for $\boldsymbol{W}_{(i)}$, where $\boldsymbol{z}_i$ is computed using Equation (4). Then, we have $\boldsymbol{W}_{(i)} = \boldsymbol{0}$ if and only if $K_i \leq \lambda$.*

We can check whether $\mathbf{W}_{(i)}$ is a zero vector by using Lemma 1. However, $\mathbf{z}_i$ in Lemma 1 incurs a high computation cost: it requires $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ time as described in Section 2.2. As a result, the computation cost of the optimality condition score K_i is also $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ time. To overcome this problem, we approximately compute the optimality condition score. In particular, we evaluate two types of approximated scores instead of the exact score: the upper and lower bounds of the optimality condition score. These bounds are defined as follows:

Definition 1 (Upper and Lower Bounds) *Let $\overline{K}_i$ and $\underline{K}_i$ be the upper and lower bounds of the optimality condition score K_i in Lemma 1, respectively. $\tilde{K}_i$ and $\tilde{\boldsymbol{W}}$ denote the optimality condition score and the parameter matrix before entering the inner loop (lines 3–4 in Algorithm 1) of the coordinate descent, respectively. Then, $\overline{K}_i$ and $\underline{K}_i$ are respectively defined as follows:*

$$\overline{K}_i = \tilde{K}_i + \|\Delta \boldsymbol{W}_{(i)}\|_2 + \|\boldsymbol{G}_{(i)}\|_2 \|\Delta \boldsymbol{W}\|_F, \tag{5}$$

and

$$\underline{K}_i = \tilde{K}_i - \|\Delta \boldsymbol{W}_{(i)}\|_2 - \|\boldsymbol{G}_{(i)}\|_2 \|\Delta \boldsymbol{W}\|_F, \tag{6}$$

where $\Delta \boldsymbol{W}_{(i)} = \boldsymbol{W}_{(i)} - \tilde{\boldsymbol{W}}_{(i)}$ and $\Delta \boldsymbol{W} = \boldsymbol{W} - \tilde{\boldsymbol{W}}$. $\boldsymbol{G}_{(i)} \in \mathbb{R}^{1 \times p}$ is the i-th row vector of $\boldsymbol{G} := \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \in \mathbb{R}^{p \times p}$.

Note that we can precompute $\tilde{K}_i$ and $\|\mathbf{G}_{(i)}\|_2$ before entering the inner loop and outer loop, respectively. Although the upper and lower bounds still require $\mathcal{O}(p^2)$ time, we introduce an efficient computation for these bounds in Section 3.3 and 3.4. The following lemma shows that $\overline{K}_i$ and $\underline{K}_i$ are the upper and lower bounds of K_i, respectively:

Lemma 2 (Upper and Lower Bounds) *We have $\overline{K}_i \geq K_i$ and $\underline{K}_i \leq K_i$ for the upper and lower bounds given in Definition 1.*

We use the upper and lower bounds as the approximations of the optimality condition score. The error bounds of the approximations are given as follows:

Lemma 3 (Error Bound) *Let ϵ be an error bound defined as $2\|\Delta \boldsymbol{W}_{(i)}\|_2 + 2\|\boldsymbol{G}_{(i)}\|_2 \|\Delta \boldsymbol{W}\|_F$. Then, we have $|\overline{K}_i - K_i| \leq \epsilon$ and $|\underline{K}_i - K_i| \leq \epsilon$ for the upper and lower bounds, respectively.*

3.3. Skipping Computations

This section introduces our first idea to skip the computations for the rows of $\mathbf{W}$ that must be *zero* vectors during the optimization. Because Equation (2) requires a high computation cost, we expect the coordinate descent to reduce the processing time by skipping the computations. To identify $\mathbf{W}_{(i)}$ that must be a zero vector, we utilize the upper bound of the optimality condition score $\overline{K}_i$ in Definition 1. Specifically, we utilize the following property of the upper bound:

Lemma 4 (Rows with Zero Vectors) *When $\overline{K}_i \leq \lambda$ holds, we have $\boldsymbol{W}_{(i)} = \boldsymbol{0}$ for the i-th row vector.*

According to Lemma 4, we can identify the rows that must be zero vectors by using the upper bounds $\overline{K}_i$. However, the computation cost of the upper bound is still high because Equation (5) requires $\mathcal{O}(p^2)$ time even if we precompute $\tilde{K}_i$ and $\|\mathbf{G}_{(i)}\|_2$ due to the computation of $\|\Delta\mathbf{W}\|_F$. Since the previous approaches require $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ times for Equation (2), the computation of the upper bound is not very efficient. Therefore, we introduce an efficient computation for the upper bound. In particular, we perform an online update for $\|\Delta\mathbf{W}\|_F$ in the upper bound when a parameter vector is updated as follows:

Lemma 5 (Online Update) *When $\boldsymbol{W}_{(j)}$ is updated to $\boldsymbol{W}'_{(j)}$, an upper bound $\overline{K}_{i\neq j}$ is computed as follows:*

$$\overline{K}_i = \tilde{K}_i + \|\Delta\boldsymbol{W}_{(i)}\|_2 + \delta\|\boldsymbol{G}_{(i)}\|_2, \tag{7}$$

where

$$\delta = \sqrt{\|\Delta\boldsymbol{W}\|_F^2 - \|\Delta\boldsymbol{W}_{(j)}\|_2^2 + \|\Delta\boldsymbol{W}'_{(j)}\|_2^2}. \tag{8}$$

The above online updating scheme[1] requires the following computation cost:

Lemma 6 (Computation Cost for Online Update) *Given precomputed $\tilde{K}_i$ and $\|\boldsymbol{G}_{(i)}\|_2$, the computation cost of Equation (7) is $\mathcal{O}(p)$ time when $\boldsymbol{W}_{(j)}$ is updated.*

Lemma 6 shows that our online updating scheme can compute the upper bound within $\mathcal{O}(p)$ time, which is lower than $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ times for Equation (2) in the previous approaches. Therefore, we can efficiently identify the rows that must be zero vectors, and skip the computation of Equation (2), which incurs a high computation cost.

Although the upper bound can be efficiently computed, the error bound of the upper bound ϵ in Lemma 3 is also important for reducing the processing time. This is because it is difficult to maintain the condition $\overline{K}_i \leq \lambda$ in Lemma 4 if ϵ is large. As a result, the number of skipped rows may be moderate. To increase the number of skipped rows, ϵ should be small. Fortunately, the error bounds of the upper and lower bounds have the following advantage:

Lemma 7 (Convergence of Error Bound) *If $\boldsymbol{W}$ reaches convergence by the coordinate descent, we have $\epsilon = 0$ for the error bound. Namely, the upper bound $\overline{K}_i$ and the lower bound $\underline{K}_i$ converge to the exact optimality condition score K_i when $\boldsymbol{W}$ converges.*

Lemma 7 indicates that the upper bound matches the optimality condition score when the parameter converges. Intuitively, the upper bound becomes increasingly tighter during the optimization as the bound depends on $\Delta\mathbf{W}_{(i)}$ and $\Delta\mathbf{W}$. Since ϵ becomes increasingly smaller during the optimization, the upper bounds can accurately identify the rows that must be zero vectors as the optimization progresses. As a result, we can effectively skip the computations of the rows by using the upper bound.

3.4. Selective Update

This section presents our second idea to preferentially update $\mathbf{W}_{(i)}$ that must be a *nonzero* vector. Since the nonzero parameter vectors correspond to the columns $\mathbf{C}$, we can expect the coordinate descent to effectively optimize the objective by intensively updating the nonzero parameter vectors. Specifically, we first construct the set including rows that must be nonzero vectors. Next, we perform coordinate descent on the set. We then update all the parameter vectors via the coordinate descent with the upper bounds until convergence.

To find $\mathbf{W}_{(i)}$ that must be a nonzero vector, we utilize the lower bound of the optimality condition score $\underline{K}_i$ in Definition 1. Similar to the upper bound, the lower bound has the following property:

Lemma 8 (Rows with Nonzero Vectors) *When $\underline{K}_i > \lambda$ holds, we have $\boldsymbol{W}_{(i)} \neq \boldsymbol{0}$ for the i-th row vector.*

Lemma 8 shows that we can identify a row that must have a nonzero vector by using the lower bound $\underline{K}_i$. We define a subset of the rows on the basis of Lemma 8 as follows:

Definition 2 (Row Set $\mathbb{M}$) *We define the set $\mathbb{M}$ by using the lower bound $\underline{K}_i$ as follows:*

$$\mathbb{M} = \{i \in \{1, ..., p\} | \underline{K}_i > \lambda\}. \tag{9}$$

The set $\mathbb{M}$ has the following property:

Lemma 9 (Row Set $\mathbb{M}$) *The set $\mathbb{M}$ contains the rows that must be nonzero vectors.*

[1] We note that $\|\Delta\mathbf{W}'_{(i)}\|_2$ is used instead of $\|\Delta\mathbf{W}_{(i)}\|_2$ in Equation (7) when $i = j$; however, this case would not be obtained in our algorithm because the coordinate descent updates the parameter vectors in a cyclic order.

As shown in Definition 2, the computation cost to construct the set $\mathbb{M}$ depends on the computation cost of the lower bound $\underline{K}_i$. Similar to that in the case of the upper bound, the lower bound requires $\mathcal{O}(p^2)$ time for the original computation of Equation (6) and $\mathcal{O}(p)$ time for the online updating scheme similar to Lemma 5. Since we must check the condition of Lemma 8 for each row to construct the set $\mathbb{M}$, we need $\mathcal{O}(p^2)$ time even if we use the online updating scheme. This is the motivation behind the lower bound computation using the upper bound. In particular, after the upper bound is updated by following Lemma 5, we compute the lower bound by utilizing the error bound of Lemma 3 as follows:

Lemma 10 (Computation using Upper Bound) *After the upper bound $\overline{K}_i$ is computed by using Equations (7) and (8), the lower bound $\underline{K}_i$ is computed as follows:*

$$\underline{K}_i = \overline{K}_i - 2\|\Delta \boldsymbol{W}_{(i)}\|_2 - 2\delta\|\boldsymbol{G}_{(i)}\|_2. \tag{10}$$

The computation cost of Equation (10) is as follows:

Lemma 11 (Computation Cost for Lower Bound) *After the upper bound is computed using Equations (7) and (8), Equation (10) requires $\mathcal{O}(1)$ time.*

Lemma 11 shows that the lower bound can be efficiently computed after the computation of the upper bound. This is because we can reuse the computed variables of the upper bounds for the computation of the lower bounds. Finally, we obtain the cost of constructing the set $\mathbb{M}$ as follows:

Lemma 12 (Computation Cost for Set $\mathbb{M}$) *We can construct the set $\mathbb{M}$ in Lemma 9 at $\mathcal{O}(p)$ time after the upper bounds are computed.*

The computation cost of $\mathcal{O}(p)$ time is significantly lower compared with the original computation based on Equation (6), which requires $\mathcal{O}(p^3)$ time to construct the set $\mathbb{M}$.

3.5. Algorithm

Algorithm 2 shows the pseudocode of our algorithm, namely Fast Deterministic CUR, which utilizes the above-mentioned definitions and lemmas. The algorithm utilizes two ideas as described in the previous sections: i) it safely skips the computations for the rows of $\mathbf{W}$ that must be zero vectors by using the upper bounds $\overline{K}_i$, and ii) it preferentially updates the parameter vectors corresponding to the row set $\mathbb{M}$, which must be nonzero vectors. Since the original deterministic CUR has the regularization constant λ as described in problem (1), we tune the regularization constant by using the sequential rule (Ghaoui et al., 2012) with a warm start (Friedman et al., 2007), which is a standard approach for the optimization with sparsity-inducing norms (Wang & Ye, 2014; Ndiaye et al., 2017; Ida et al., 2019). Specifically, it sequentially tunes the regularization constant λ with respect to the sequence $(\lambda_q)_{q=0}^{Q-1}$, where $\lambda_0 > \lambda_1 > ... > \lambda_{Q-1}$. The parameter matrices $\mathbf{W}$ are sequentially optimized for each regularization constant by using the coordinate descent, and the initial parameter matrix of the current λ_q is the result of the previous λ_{q-1}.

Algorithm 2 Fast Deterministic CUR

1: $\mathbb{A} = \{1, ..., p\}$, $\mathbf{W} \leftarrow \mathbf{0}$, $\tilde{\mathbf{W}} \leftarrow \mathbf{0}$, $\mathbf{G} \leftarrow \mathbf{X}^{\mathrm{T}}\mathbf{X}$
2: **for** each $i \in \mathbb{A}$ **do**
3: compute $\|\mathbf{G}_{(i)}\|_2$;
4: **for** $q = 0$ to $Q - 1$ **do**
5: $\mathbb{M} = \emptyset$;
6: **for** each $i \in \mathbb{A}$ **do**
7: compute the lower bound $\underline{K}_i$ by Equation (10);
8: **if** $\underline{K}_i > \lambda_q$ **then**
9: add i to $\mathbb{M}$;
10: **repeat**
11: **for** each $i \in \mathbb{M}$ **do**
12: update $\mathbf{W}_{(i)}$ by Equation (2);
13: **until** $\mathbf{W}$ converges
14: **repeat**
15: $\tilde{\mathbf{W}} \leftarrow \mathbf{W}$;
16: **for** each $i \in \mathbb{A}$ **do**
17: compute $\tilde{K}_i$;
18: **for** each $i \in \mathbb{A}$ **do**
19: compute the upper bound $\overline{K}_i$ by Equation (7);
20: **if** $\overline{K}_i \leq \lambda_q$ **then**
21: $\mathbf{W}_{(i)} \leftarrow \mathbf{0}$;
22: **else**
23: update $\mathbf{W}_{(i)}$ by Equation (2);
24: update δ by Equation (8);
25: **until** $\mathbf{W}$ converges

In Algorithm 2, we first precompute $\|\mathbf{G}_{(i)}\|_2$ for computing the upper bounds and lower bounds (lines 2–3). We next enter the loop of the sequential rule (lines 4–25). In the loop, we construct the set $\mathbb{M}$ by using the lower bounds of the optimality condition scores (lines 5–9). If the lower bound $\underline{K}_i$ is larger than λ_q, we add the index of the row to $\mathbb{M}$ (lines 8–9). It should be noted that when $\underline{K}_i$ is computed, we use the $\tilde{K}_i$, $\Delta\mathbf{W}_{(i)}$, and δ used in the last computation of the upper bound in the previous loop of the sequential rule. If $q = 0$, we compute the lower bounds by the original definition of Equation (6). Another strategy for computing the lower bounds in the case of $q = 0$ is to run lines 15–24 one time first and compute the lower bounds by using Equation (10). After constructing the set $\mathbb{M}$, we perform coordinate descent on the set (lines 10–13). We then enter the loop of the coordinate descent with the upper bounds of the optimality condition scores (lines 14–25). We set $\tilde{\mathbf{W}}$ in line 15 and compute $\tilde{K}_i$ (lines 16–17). The computation results are used to compute the upper bounds $\overline{K}_i$ (line 19). If the upper bound is less than or equal to λ_q, $\mathbf{W}_{(i)}$ turns to be $\mathbf{0}$ (lines 20–21). In other cases, $\mathbf{W}_{(i)}$ is updated as usual (lines 22–23). Subsequently, we update δ by following the online updating scheme of Lemma 5 (line 17).

The computation cost of Algorithm 2 is given as follows:

Theorem 1 (Computation Cost) *Let t_{u} be the total number of outer loops for the coordinate descent with the upper bounds. Suppose that S is the ratio of updates to the total number of inner loops, which is un-skipped by the upper bounds. If t_{m} is the total number of inner loops for the set $\mathbb{M}$, Algorithm 2 requires $\mathcal{O}(p\{n(t_{\mathrm{m}}+pt_{\mathrm{u}}S)+Q\})$ time.*

We expect the algorithm to reduce t_{u} by preferentially updating the parameter vectors on the basis of the set $\mathbb{M}$ in Lemma 9. In addition, S would be small when the algorithm skips a large number of updates by utilizing the upper bounds in Lemma 4. As a result, the total computation cost would be effectively reduced.

In terms of the optimization result, Algorithm 2 has the following property:

Theorem 2 (Optimization Result) *Suppose that Algorithm 2 has the same regularization constants as those of the original algorithm, and the coordinate descent updating the parameter vectors with a cyclic order converges. Then, Algorithm 2 converges to the same objective values as those of the original algorithm.*

The aforementioned theorem suggests that our algorithm achieves the same accuracy as that of the original algorithm. Therefore, we expect Algorithm 2 to speed up the deterministic CUR without degrading the accuracy.

3.6. Extension

We propose Algorithm 2 on the basis of problem (1) by following (Bien et al., 2010). Although it only selects the columns $\mathbf{C}$, Mairal et al. (2011) naturally extended the problem to achieve simultaneous selection of both rows $\mathbf{R}$ and columns $\mathbf{C}$ by using the row-wise and column-wise regularization terms. Although these regularization terms overlap, we can handle the terms by using the overlapping norm (Jacob et al., 2009). Specifically, we define the following optimization problem for the simultaneous selection by combining the extension of (Mairal et al., 2011) and the overlapping norm (Jacob et al., 2009):

$$\min_{\mathbf{W}\in\mathbb{R}^{p\times n}} \frac{1}{2}\|\mathbf{X}-\mathbf{X}\mathbf{W}\mathbf{X}\|_F^2+\lambda_r\sum_{i=1}^{p}\|\mathbf{V}_{(i)}\|_2+\lambda_c\sum_{j=1}^{n}\|\mathbf{H}^{(j)}\|_2, \quad (11)$$

where $\mathbf{V}\in\mathbb{R}^{p\times n}$ and $\mathbf{H}\in\mathbb{R}^{p\times n}$ are latent variables: the parameter matrix is decomposed into a sum of the latent variables as $\mathbf{W}=\mathbf{V}+\mathbf{H}$; $\sum_{i=1}^{p}\|\mathbf{V}_{(i)}\|_2$ and $\sum_{j=1}^{n}\|\mathbf{H}^{(j)}\|_2$ are the overlapping norms, which correspond to row-wise and column-wise regularization terms, respectively; and $\lambda_r,\lambda_c\geq 0$ are regularization constants for the norms.

To solve problem (11), we perform row-wise and column-wise coordinate descents. Therefore, we have two types of condition scores for the parameters to be zeros as follows:

Lemma 13 (Optimality Condition for Zero Parameter) *Let $R_i := \|\boldsymbol{X}^{(i)\mathrm{T}}\{\boldsymbol{X}-(\boldsymbol{X}\boldsymbol{W}-\boldsymbol{X}^{(i)}\boldsymbol{V}_{(i)})\boldsymbol{X}\}\boldsymbol{X}^{\mathrm{T}}\|_2$ and $C_j := \|\boldsymbol{X}^{\mathrm{T}}\{\boldsymbol{X}-\boldsymbol{X}(\boldsymbol{W}\boldsymbol{X}-\boldsymbol{H}^{(j)}\boldsymbol{X}_{(j)})\}\boldsymbol{X}_{(j)}^{\mathrm{T}}\|_2$ be the optimality condition scores for the parameters to be zeros for $\boldsymbol{V}_{(i)}$ and $\boldsymbol{H}^{(j)}$, respectively. Then, we obtain $\boldsymbol{V}_{(i)}=\boldsymbol{0}$ if and only if $R_i\leq\lambda_r$, and $\boldsymbol{H}^{(j)}=\boldsymbol{0}$ if and only if $C_j\leq\lambda_c$.*

The columns $\mathbf{C}$ and the rows $\mathbf{R}$ are selected on the basis of the indices corresponding to the nonzero rows and columns in $\mathbf{V}$ and $\mathbf{H}$, respectively. The detail can be found in the supplementary file. Then, we can define the upper and lower bounds for the condition scores as follows:

Definition 3 (Upper and Lower Bounds) *Let $\overline{R}_i$ and $\underline{R}_i$ be the upper and lower bounds of the optimality condition score R_i, respectively. $\tilde{R}_i$ denotes the optimality condition score before entering the inner loop of the coordinate descent. Then, $\overline{R}_i$ and $\underline{R}_i$ are respectively defined as $\overline{R}_i=\tilde{R}_i+\rho_i$ and $\underline{R}_i=\tilde{R}_i-\rho_i$, where $\rho_i:=\boldsymbol{G}_{(i)}^{(i)}\|\Delta\boldsymbol{V}_{(i)}\|_2\|\boldsymbol{F}\|_F+\|\boldsymbol{G}_{(i)}\|_2\|\Delta\boldsymbol{W}\|_F\|\boldsymbol{F}\|_F$, and $\boldsymbol{F}:=\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\in\mathbb{R}^{n\times n}$. Similarly, the upper bound $\overline{C}_j$ and the lower bound $\underline{C}_j$ of the optimality condition score C_j are respectively defined as $\overline{C}_j=\tilde{C}_j+\sigma_j$ and $\underline{C}_j=\tilde{C}_j-\sigma_j$, where $\sigma_j:=\|\boldsymbol{G}\|_F\|\Delta\boldsymbol{H}^{(j)}\|_2\boldsymbol{F}_{(j)}^{(j)}+\|\boldsymbol{G}\|_F\|\Delta\boldsymbol{W}\|_F\|\boldsymbol{F}^{(j)}\|_2$.*

The bounds in Definition 3 can realize our two ideas as described in the paper: i) safely skipping the computations for rows and columns by using the upper bounds, and ii) prioritizing the update order by using the lower bounds.

4. Related Work

The deterministic CUR is a convex optimization problem with sparsity-inducing norms as shown in problem (1). As these norms correspond to the group-level regularizations used in group Lasso (Yuan & Lin, 2006), we can use several techniques for group Lasso to speed up CUR.

To efficiently solve group Lasso, screening rules (Tibshirani et al., 2012; Wang et al., 2013; Bonnefoy et al., 2015; Ndiaye et al., 2017) are popular methods, which eliminate several parameters before achieving optimization. Since they reduce the size of the problem, we can expect the optimization algorithm to reduce the processing time. Dual Polytope Projections (DPP) is a screening rule that utilizes the geometric property of the dual solution (Wang et al., 2013). Unfortunately, DPP may eliminate parameters incorrectly because it theoretically requires the exact solution of the previous λ in the sequential rule, which is not practically available in the iterative optimization algorithm. Bonnefoy et al. (2015) proposed a dynamic method of screening: it eliminates the parameters, not only before the optimization but also during the optimization. As a result, it can be expected to eliminate a large number of parameters. However,

Table 1. Wall clock times on each dataset. We omit some computation results, which could not finish within a month.

Dataset	Wall clock time (s)				Reduction ratio (%)
	origin	SSR	SSR+WS	ours	(ours/origin)
colon-cancer	1.384×10^6	–	–	$\mathbf{4.793 \times 10^5}$	**34.64**
Bioresponse	1.461×10^6	–	–	$\mathbf{9.908 \times 10^5}$	**67.80**
QSAR-TID-52	9.628×10^3	1.301×10^4	8.875×10^3	$\mathbf{6.323 \times 10^3}$	**65.67**
Madelon	2.555×10^3	1.194×10^3	1.008×10^3	$\mathbf{1.006 \times 10^3}$	**39.36**
Slashdot	1.827×10^3	2.122×10^3	1.636×10^3	$\mathbf{8.592 \times 10^2}$	**47.04**

if the number of eliminated parameters is small, the processing time can increase due to the overhead of the screening process (Ida et al., 2019).

Sequential strong rule (SSR) (Tibshirani et al., 2012) is a heuristic strategy for screening, which approximately eliminates the parameters. Although it can eliminate parameters incorrectly, this can be avoided by checking the Karush–Kuhn–Tucker (KKT) condition after the optimization. In spite of the fact that SSR is a relatively old method, it still achieves the state-of-the-art results compared with the recent screening methods (Ndiaye et al., 2017).

Although our idea of selective update using lower bounds in Section 3.4 can be seen as a screening, we utilize another idea of skipping computations using upper bounds in Section 3.3. As a general problem, screenings could not speed up the optimization when the number of eliminated features is small (Johnson & Guestrin, 2016; 2017; Ida et al., 2019). On the other hand, our method can effectively skip updates by using upper bounds during the optimization even if lower bounds cannot eliminate so many features because our bounds become increasingly tighter during the optimization as shown in Lemma 7. In addition, our screening using lower bounds is efficient because it only requires $\mathcal{O}(p)$ time as shown in Lemma 12.

Ida et al. (2019) proposed a fast block coordinate descent algorithm that focuses on sparse group Lasso. It checks whether the parameter vector of each group is a zero vector by using bound approximations similar to our method. However, because their bounds specialize in models whose parameters take the form of a vector, it does not apply to CUR whose parameters are in a matrix form. In theory, their lower bounds do not converge to the exact value while our bounds guarantee the convergences as shown in Lemma 7.

All the aforementioned methods including our method can further improve the efficiency by utilizing a warm start strategy (Friedman et al., 2007; Ndiaye et al., 2017). The warm start uses the solution of the previous λ as the initial parameters of the current λ in the sequential rule. As a result, it empirically speeds up the optimization algorithm.

5. Experiment

We evaluated the processing times and values of the objectives. We compared our method with the original deterministic CUR (*origin*) (Bien et al., 2010), the sequential strong rule (*SSR*) (Tibshirani et al., 2012), and the sequential strong rule with warm start (*SSR+WS*) (Ndiaye et al., 2017). We tuned λ for all the approaches based on the sequential rule by following the methods in (Tibshirani et al., 2012; Wang & Ye, 2014; Ndiaye et al., 2017; Ida et al., 2019). The search space was a non-increasing sequence of Q parameters $(\lambda_q)_{q=0}^{Q-1}$ defined as $\lambda_q = \lambda_{max} 10^{-\gamma q/Q-1}$. λ_{max} is the smallest λ for which all the parameters are zeros at the optimal solutions and it was computed by following (Tibshirani et al., 2012). We used $\gamma = 4$ and $Q = 100$ (Wang & Ye, 2014; Ndiaye et al., 2017; Ida et al., 2019). We stopped the algorithm for each λ_q when the relative tolerance of the parameter matrix dropped below 10^{-5} for all the approaches (Johnson & Guestrin, 2016; 2017; Ida et al., 2019). We stopped the sequential rule when all the parameters were nonzeros since our purpose is to select the subset of the columns. We conducted the experiments on five datasets from the LIBSVM[2] (Chang & Lin, 2011) and OpenML[3] (Vanschoren et al., 2013) websites, namely colon-cancer, Bioresponse, QSAR-TID-52, Madelon, and Slashdot, and the sizes of the data matrices were 62×2000, 3751×1776, 877×1024, 2000×500, and 3782×1079, respectively. Each experiment was conducted with one CPU core and 264 GB of main memory on a 2.20 GHz Intel Xeon server running Linux.

5.1. Processing Time

We evaluated the processing times of the sequential rules for each method. Table 1 shows the wall clock times for the five datasets. Note that the processing times include precomputation times for a fair comparison. Our method was faster than the previous methods for all the datasets. It reduced the processing time by up to 34.64% compared to

[2] https://www.csie.ntu.edu.tw/~cjlin/libsvm/

[3] https://www.openml.org/

Table 2. Numbers of updates for Eq. (2). Our method effectively reduces the number of bottleneck computations.

Dataset	# of updates for Eq. (2)	
	origin	ours
colon-cancer	8.344×10^7	$\mathbf{4.181 \times 10^7}$
Bioresponse	9.048×10^7	$\mathbf{7.655 \times 10^7}$
QSAR-TID-52	1.991×10^6	$\mathbf{1.131 \times 10^6}$
Madelon	1.847×10^6	$\mathbf{2.162 \times 10^5}$
Slashdot	6.571×10^5	$\mathbf{1.898 \times 10^5}$

the original method. Although SSR+WS was competitive in comparison to our method on the Madelon dataset, our method was faster than SSR+WS on the other datasets. This is because the speed-up of SSR+WS is moderate due to the overheads of checking the KKT conditions when the number of eliminated parameters is small. Unfortunately, SSR and SSR+WS could not finish the computations within a month on the colon-cancer and Bioresponse datasets due to the over heads. On the other hand, our method quickly extracts the parameter vectors that must be nonzero vectors at $\mathcal{O}(p)$ time. In addition, it efficiently skips the parameter vector that must be a zero vector at $\mathcal{O}(p)$. Thanks to the small overheads, our method reduces the processing times even on datasets that are not suitable for SSR and SSR+WS.

Table 2 shows the numbers of updates of Equation (2) for the original method and our method on each dataset. The results suggest that our approximations of the upper and lower bounds effectively reduced the number of updates by skipping unnecessary updates and preferentially updating the parameters. Since Equation (2) is the main bottleneck as described in Section 2.2, our method could effectively reduce the processing time as shown in Table 1.

Figure 1 shows the mean and standard deviation for approximation errors of the upper and lower bounds during the optimization (lines 14–25 in Algorithm 2). The result was obtained with $\lambda_q = 1.2$ on the Slashdot dataset; the ratio of zero parameters was 80% in this setting. Similar results were obtained for the other settings. This result suggests that the error bounds become increasingly smaller during the optimization; it supports our theoretical result of Lemma 7. Namely, the upper and lower bounds become tight during the optimization. Thanks to the small approximation errors, our method effectively identifies the parameter vectors that must be nonzero and zero vectors by using the lower and upper bounds, respectively.

We also investigated the relationship between the processing time and the size of the data matrix. We used the gene expression data from (Ramaswamy et al., 2002), which is a data matrix of 190×16063. We randomly sampled 100, 500, 1000, 5000, and 10000 columns from the data matrix, and evaluated the processing times to select 1% of all the columns from these five data matrices on two CPU cores. As shown in Figure 2, our method was faster than the existing methods for all sizes. Specifically, for 1000 columns, our method was $10\times$ and $4\times$ faster than the original method and SSR+WS, respectively. We omit the results of the comparison methods for 10000 columns because they could not finish the computations within a month. The results show that our method achieves higher efficiency than the existing methods.

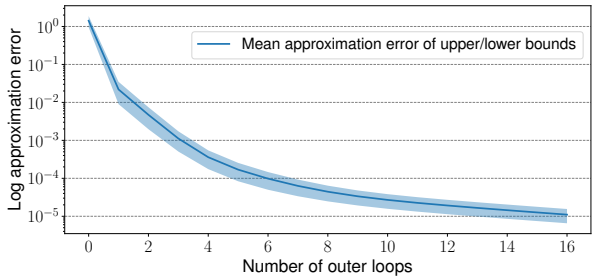

Figure 1. Mean and standard deviation for approximation errors of the upper and lower bounds during optimization. Our bounds become tight during the optimization.

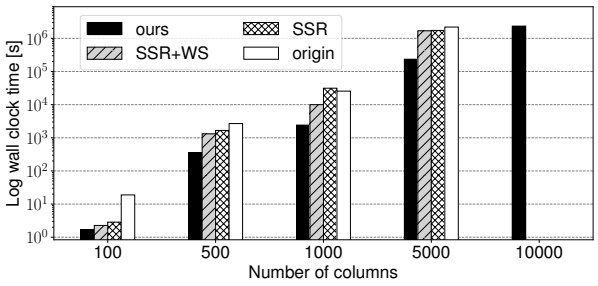

Figure 2. Log wall clock time vs. number of columns with the gene expression data. Our method is up to $10\times$ and $4\times$ faster than the original method and SSR+WS, respectively. We omit some computation results, which could not finish within a month.

5.2. Accuracy

We evaluated the value of the objective function to confirm the effectiveness of our method. Table 3 shows the results: they are final values of the objective functions in the sequential rules. The values of the objectives of our method are the same as those of the original method. This is because our method is guaranteed to yield the same value of the objective function as that of the original method, as described in Theorem 2. We note that SSR and SSR+WS also achieved the same objectives as those of the original

Table 3. Values of objectives. Our method converges to the same objective as that of the original method.

Dataset	Objective	
	origin	ours
colon-cancer	$\mathbf{1.336}$	$\mathbf{1.336}$
Bioresponse	$\mathbf{9.683 \times 10^1}$	$\mathbf{9.683 \times 10^1}$
QSAR-TID-52	$\mathbf{5.727 \times 10^2}$	$\mathbf{5.727 \times 10^2}$
Madelon	$\mathbf{9.390 \times 10^2}$	$\mathbf{9.390 \times 10^2}$
Slashdot	$\mathbf{1.837 \times 10^3}$	$\mathbf{1.837 \times 10^3}$

method for QSAR-TID-52, Madelon, and Slashdot datasets because they are safe screening methods. However, they could not finish the computations within a month on the other datasets as shown in Table 1. Table 3 shows that our method can maintain the accuracy while speeding up CUR.

6. Conclusion

We proposed a fast deterministic CUR matrix decomposition. The main bottleneck of the original method is the coordinate descent, which requires a large number of parameter updates. Our method utilizes two ideas to tackle this problem: i) it safely skips the updates by identifying the parameters that must be *zeros*, and ii) it preferentially updates the parameters that must be *nonzeros*. The key is to approximately evaluate the optimality conditions for the zero parameters by using the upper and lower bounds of the optimality condition scores. In addition, it provably guarantees the same results as those of the original method. The experimental results showed that our method is up to $10\times$ faster than the original method, and up to $4\times$ faster than the state-of-the-art method without requiring any additional hyperparameters or incurring any loss of accuracy.

Although we handle only matrix decomposition in this paper, our method can be extended to tensor decomposition. Furthermore, our ideas can be generalized for various types of structured data such as graph, tree, and heterogeneous data. These future works will enable a wide range of applications to be implemented more efficiently.

References

Bien, J., Xu, Y., and Mahoney, M. W. CUR from a Sparse Optimization Viewpoint. In *NeurIPS*, pp. 217–225, 2010.

Bonnefoy, A., Emiya, V., Ralaivola, L., and Gribonval, R. Dynamic Screening: Accelerating First-Order Algorithms for the Lasso and Group-Lasso. *IEEE Trans. Signal Processing*, 63(19):5121–5132, 2015.

Chang, C. and Lin, C. LIBSVM: A Library for Support Vector Machines. *ACM TIST*, 2(3):27:1–27:27, 2011.

Drineas, P., Kannan, R., and Mahoney, M. W. Fast Monte Carlo Algorithms for Matrices III: Computing a Compressed Approximate Matrix Decomposition. *SIAM J. Comput.*, 36(1):184–206, 2006.

Drineas, P., Mahoney, M. W., and Muthukrishnan, S. Relative-Error CUR Matrix Decompositions. *SIAM J. Matrix Analysis Applications*, 30(2):844–881, 2008.

Friedman, J., Hastie, T., Höfling, H., and Tibshirani, R. Pathwise Coordinate Optimization. *Ann. Appl. Stat.*, 1(2): 302–332, 12 2007.

Ghaoui, L. E., Viallon, V., and Rabbani, T. Safe Feature Elimination for the Lasso and Sparse Supervised Learning Problems. *Pacific Journal of Optimization*, 8(4):667–698, 2012.

Huang, J., Breheny, P., and Ma, S. A Selective Review of Group Selection in High-Dimensional Models. *Statistical Science*, 27(4):481–499, 2012.

Ida, Y., Fujiwara, Y., and Kashima, H. Fast Sparse Group Lasso. In *NeurIPS*, pp. 1700–1708, 2019.

Jacob, L., Obozinski, G., and Vert, J. Group Lasso with Overlap and Graph Lasso. In *ICML*, pp. 433–440, 2009.

Johnson, T. B. and Guestrin, C. Unified Methods for Exploiting Piecewise Linear Structure in Convex Optimization. In *NeurIPS*, pp. 4754–4762, 2016.

Johnson, T. B. and Guestrin, C. StingyCD: Safely Avoiding Wasteful Updates in Coordinate Descent. In *ICML*, pp. 1752–1760, 2017.

Mackey, L. W., Talwalkar, A., and Jordan, M. I. Divide-and-Conquer Matrix Factorization. In *NeurIPS*, pp. 1134–1142, 2011.

Mahoney, M. W. and Drineas, P. CUR Matrix Decompositions for Improved Data Analysis. *Proc. Natl. Acad. Sci. U.S.A.*, 106(3):697–702, 2009.

Mahoney, M. W., Maggioni, M., and Drineas, P. Tensor-CUR Decompositions for Tensor-based Data. In *KDD*, pp. 327–336, 2006.

Mairal, J., Jenatton, R., Obozinski, G., and Bach, F. R. Convex and Network Flow Optimization for Structured Sparsity. *JMLR*, 12:2681–2720, 2011.

Ndiaye, E., Fercoq, O., Gramfort, A., and Salmon, J. Gap Safe Screening Rules for Sparsity Enforcing Penalties. *Journal of Machine Learning Research*, 18(1):4671–4703, 2017.

Papailiopoulos, D. S., Kyrillidis, A., and Boutsidis, C. Provable Deterministic Leverage Score Sampling. In *KDD*, pp. 997–1006, 2014.

Ramaswamy, S., Tamayo, P., Rifkin, R., Mukherjee, S., Yeang, C.-H., Angelo, M., Ladd, C., Reich, M., Latulippe, E., Mesirov, J., Poggio, T., Gerald, W., Loda, M., and Lander, E. Multiclass Cancer Diagnosis using Tumor Gene Expression Signatures. *Proceedings of the National Academy of Sciences*, 98, 01 2002.

Sun, J., Xie, Y., Zhang, H., and Faloutsos, C. Less is More: Compact Matrix Decomposition for Large Sparse Graphs. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, pp. 366–377, 2007.

Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J., and Tibshirani, R. J. Strong Rules for Discarding Predictors in Lasso-type Problems. *Journal of the Royal Statistical Society Series B*, 74(2):245–266, 2012.

Tong, H., Papadimitriou, S., Sun, J., Yu, P. S., and Faloutsos, C. Colibri: Fast Mining of Large Static and Dynamic Graphs. In *KDD*, pp. 686–694, 2008.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

Wang, J. and Ye, J. Two-Layer Feature Reduction for Sparse-Group Lasso via Decomposition of Convex Sets. In *NeurIPS*, pp. 2132–2140, 2014.

Wang, J., Zhou, J., Wonka, P., and Ye, J. Lasso Screening Rules via Dual Polytope Projection. In *NeurIPS*, pp. 1070–1078, 2013.

Yuan, M. and Lin, Y. Model Selection and Estimation in Regression with Grouped Variables. *Journal of the royal statistical society, series B*, 68:49–67, 2006.