2018 IEEE 30th International Conference on Tools with Artificial Intelligence

Faster Matrix Completion Using Randomized SVD

Xu Feng
BNRist, Dept. Computer Science & Tech.
Tsinghua University
Beijing, China
fx17@mails.tsinghua.edu.cn

Wenjian Yu
BNRist, Dept. Computer Science & Tech.
Tsinghua University
Beijing, China
yu-wj@tsinghua.edu.cn

Yaohang Li
Dept. Computer Science
Old Dominion University
Norfolk, VA 23529, USA
yaohang@cs.odu.edu

***Abstract*—Matrix completion is a widely used technique for image inpainting and personalized recommender system, etc. In this work, we focus on accelerating the matrix completion using faster randomized singular value decomposition (rSVD). Firstly, two fast randomized algorithms (rSVD-PI and rSVD-BKI) are proposed for handling sparse matrix. They make use of an eigSVD procedure and several accelerating skills. Then, with the rSVD-BKI algorithm and a new subspace recycling technique, we accelerate the singular value thresholding (SVT) method in [1] to realize faster matrix completion. Experiments show that the proposed rSVD algorithms can be 6X faster than the basic rSVD algorithm [2] while keeping same accuracy. For image inpainting and movie-rating estimation problems (including up to 2×10^7 ratings), the proposed accelerated SVT algorithm consumes 15X and 8X less CPU time than the methods using `svds` and `lansvd` respectively, without loss of accuracy.**

***Index Terms*—matrix completion, randomized SVD, image inpainting, recommender system.**

I. Introduction

The problem of matrix completion, or estimating missing values in a matrix, occurs in many areas of engineering and applied science such as computer vision, pattern recognition and machine learning [1], [3], [4]. For example, in computer vision and image processing problems, recovering the missing or corrupted data can be regarded as matrix completion. A recommender system provides recommendations based on the user's preferences, which are often inferred with some ratings submitted by users. This is another scenario where the matrix completion can be applied.

The matrix which we wish to complete often has low rank or approximately low rank. Thus, many existing methods formulate the matrix completion as a rank minimization problem:

$$\min_{\mathbf{X}} rank(\mathbf{X}), \quad s.t.\ \mathbf{X}_{ij} = \mathbf{M}_{ij}, (i,j) \in \Phi, \tag{1}$$

where $\mathbf{M}$ is the incomplete data matrix and Φ is the set of locations corresponding to the observed entries. This problem is however NP-hard in general. A widely-used approach relies on the nuclear norm (i.e., the sum of singular values) as a convex relaxation of the rank operator. This results in a relaxed convex optimization, which can be solved with the singular value thresholding (SVT) algorithm [1]. The SVT algorithm has good performance on both synthetic data and real applications. However, it involves large computational expense while handling large data set, because the singular values exceeding a threshold and the corresponding singular vectors need to be computed in each iteration step. Truncated singular value decomposition (SVD), implemented with `svds` in Matlab or `lansvd` in PROPACK [5], is usually employed in the SVT algorithm [1]. Another method for matrix completion is the inexact augmented Lagrange multiplier (IALM) algorithm [6], which also involves singular value thresholding and was originally proposed for the robust principal component analysis (PCA) problem [7]. With artificially-generated low-rank matrices, experiments in [6] demonstrated that IALM algorithm could be several times faster than the SVT algorithm.

In recent years, randomized matrix computation has gained significant increase in popularity [2], [8]–[11]. Compared with classic algorithms, the randomized algorithm involves the same or fewer floating-point operations (*flops*), and is more efficient for truly large data sets. An idea of randomization is using random projection to identify the subspace capturing the dominant actions of a matrix. Then, a near-optimal low-rank decomposition of the matrix can be computed. A comprehensive presentation of the relevant techniques and theories are in [2]. This randomized technique has been extended to compute PCA of data sets that are too large to be stored in RAM [12], or to speed up the distributed PCA [13]. For general SVD computation, the approaches based on it have also been proposed [14], [15]. They outperform the classic deterministic techniques for calculating a few of largest singular values and corresponding singular vectors. Recently, a compressed SVD (cSVD) algorithm was proposed [11], which is based on a variant of the method in [2] but runs faster for image and video processing applications. It should be pointed out, these methods are not sufficient for accelerating the matrix completion. The SVT operation used in matrix completion requests accurate calculation of quite a large quantity of singular values. Thus, existing randomized SVD approaches cannot fulfill the accuracy requirement or cannot bring the runtime benefit. Besides, as sparse matrix is processed in matrix completion, special technique should be devised to make the randomized SVD approach really competitive.

In this work, we investigate the acceleration of matrix completion for large data using the randomized SVD techniques. We first review some existing acceleration skills for the basic randomized SVD (rSVD) algorithm, along with theoretic justification. Combining them we derive a fast randomized SVD

This work is supported by National Natural Science Foundation of China (No. 61872206).

2375-0197/18/$31.00 ©2018 IEEE
DOI 10.1109/ICTAI.2018.00098

algorithm (called rSVD-PI) and prove its correctness. Then, utilizing these techniques and the block Krylov-subspace iteration (BKI) scheme [16] we propose a rSVD-BKI algorithm for highly accurate SVD of sparse matrix. Finally, for matrix completion we choose the SVT algorithm (an empirical comparison in Section IV.A shows its superiority to the IALM algorithm), and accelerate it with the rSVD-BKI algorithm and a novel subspace recycling technique. This results in a fast SVT algorithm with same accuracy and reliability as the original SVT algorithm. To demonstrate the efficiency of the proposed fast SVT algorithm, several color image inpainting and movie-rating estimation problems are tested. The results show that the proposed method consumes 15X and 8X less CPU time than the methods using `svds` and `lansvd` respectively, while outputting same-quality results.

For reproducibility, the codes and test data in this work will be shared on GitHub (https://github.com/XuFengthucs/fSVT).

II. Preliminaries

We assume that all matrices in this work are real valued, although the generalization to complex-valued matrices is of no difficulty. In algorithm description, we follow the Matlab convention for specifying row/column indices of a matrix.

A. Singular Value Decomposition

Singular value decomposition (SVD) is the most widely used matrix decomposition [17], [18]. Let $\mathbf{A}$ denote an $m \times n$ matrix. Its SVD is

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}, \tag{2}$$

where orthogonal matrices $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \cdots]$ and $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \cdots]$ include the left and right singular vectors of $\mathbf{A}$, respectively. And, $\mathbf{\Sigma}$ is a diagonal matrix whose diagonal elements $(\sigma_1, \sigma_2, \cdots)$ are the singular values of $\mathbf{A}$ in descending order. Suppose $\mathbf{U}_k$ and $\mathbf{V}_k$ are the matrices with the first k columns of $\mathbf{U}$ and $\mathbf{V}$, respectively, and $\mathbf{\Sigma}_k$ is a diagonal matrix containing the first k singular values of $\mathbf{A}$. Then, we have the truncated SVD:

$$\mathbf{A} \approx \mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^{\mathrm{T}}. \tag{3}$$

It is well known that this truncated SVD, i.e. $\mathbf{A}_k$, is the best rank-k approximation of the matrix $\mathbf{A}$, in either spectral norm or Frobenius norm [17].

To compute truncated SVD, a common choice is Matlab's built-in `svds` [19]. It is based on a Krylov subspace iterative method, and is especially efficient for handling sparse matrix. For a dense matrix $\mathbf{A}$, `svds` costs $O(mnk)$ flops for computing rank-k truncated SVD. If $\mathbf{A}$ is sparse, the cost becomes $O(nnz(\mathbf{A})k)$ flops, where $nnz(\cdot)$ stands for the number of nonzeros of a matrix. Another choice is PROPACK [5], which is an efficient package in Matlab/Fortran for computing the dominant singular values/vectors of a large sparse matrix. The principal routine "`lansvd`" in PROPACK employs an intricate Lanczos method to compute the singular values/vectors directly, instead of computing the eigenvalues/eigenvectors of an augmented matrix as in Matlab's built-in `svds`. Therefore, `lansvd` is usually several times faster than `svds`.

B. Projection Based Randomized Algorithms

The randomized algorithms have shown their advantages for solving the linear least squares problem and low-rank matrix approximation [20]. An idea is using random projection to identify the subspace capturing the dominant actions of matrix $\mathbf{A}$. This can be realized by multiplying $\mathbf{A}$ with a random matrix on its right side or left side, and then obtaining the subspace's orthonormal basis matrix $\mathbf{Q}$. With $\mathbf{Q}$, a low-rank approximation of $\mathbf{A}$ can be computed which further results in the approximate truncated SVD. Because the dimension of the subspace is much smaller than that of $range(\mathbf{A})$, this method facilitates the computation of near-optimal decompositions of $\mathbf{A}$. A basic randomized SVD (rSVD) algorithm is described as Algorithm 1 [2].

Algorithm 1 basic rSVD

Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, rank parameter k, power parameter p
Output: $\mathbf{U} \in \mathbb{R}^{m\times k}$, $\mathbf{S} \in \mathbb{R}^{k\times k}$, $\mathbf{V} \in \mathbb{R}^{n\times k}$

1: $\mathbf{\Omega} = \mathrm{randn}(n, k+s)$
2: $\mathbf{Q} = \mathrm{orth}(\mathbf{A}\mathbf{\Omega})$
3: **for** $i = 1, 2, \cdots, p$ **do**
4: $\quad \mathbf{G} = \mathrm{orth}(\mathbf{A}^{\mathrm{T}}\mathbf{Q})$
5: $\quad \mathbf{Q} = \mathrm{orth}(\mathbf{A}\mathbf{G})$
6: **end for**
7: $\mathbf{B} = \mathbf{Q}^{\mathrm{T}}\mathbf{A}$
8: $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \mathrm{svd}(\mathbf{B})$
9: $\mathbf{U} = \mathbf{Q}\mathbf{U}$
10: $\mathbf{U} = \mathbf{U}(:, 1:k), \mathbf{S} = \mathbf{S}(1:k, 1:k), \mathbf{V} = \mathbf{V}(:, 1:k)$.

In Alg. 1, $\mathbf{\Omega}$ is a Gaussian i.i.d matrix. Other kinds of random matrix can replace $\mathbf{\Omega}$ to reduce the computational cost of $\mathbf{A}\mathbf{\Omega}$, but they also bring some sacrifice on accuracy. With the subspace's orthogonal basis $\mathbf{Q}$, we have the approximation $\mathbf{A} \approx \mathbf{Q}\mathbf{B} = \mathbf{Q}\mathbf{Q}^{\mathrm{T}}\mathbf{A}$. Then, performing the economic SVD on the $(k+s) \times n$ matrix $\mathbf{B}$ we obtain the approximate truncated SVD of $\mathbf{A}$. To improve the accuracy of the QB approximation, a technique called power iteration (PI) scheme can be applied [2], i.e. Steps 3∼6. It is based on the fact that matrix $(\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A}$ has exactly the same singular vectors as $\mathbf{A}$, but its singular value decays more quickly. Thus, performing the randomized QB procedure on $(\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A}$ can achieve better accuracy. The orthonormalization operation "orth()" is used to alleviate the round-off error in the floating-point computation. More theoretical analysis can be found in [2].

The s in Alg. 1 is an oversampling parameter, which enables $\mathbf{\Omega}$ with more than k columns used for better accuracy. s is a small integer, 5 or 10. "orth()" is achieved by a call to a packaged QR factorization (e.g., `qr(X, 0)` in Matlab).

The basic rSVD algorithm with the PI scheme has the following guarantee [2], [16]:

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^{\mathrm{T}}\mathbf{A}\| = \|\mathbf{A} - \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}\| \le (1+\epsilon)\|\mathbf{A} - \mathbf{A}_k\|, \tag{4}$$

with a high probability ($\mathbf{A}_k$ is the best rank-k approximation).

Another scheme called block Krylov-subspace iteration (BKI) can also be collaborated with the basic randomized QB procedure in Alg. 1. The resulted algorithm satisfies (4) as well, and has better accuracy with same number of iteration

(p in Alg. 1). In [16], it has been revealed that with the BKI scheme, the accuracy converges faster along with the iteration than using the PI scheme (Alg. 1). Specifically, the BKI scheme converges to the $(1+\varepsilon)$ low-rank approximation (4) in $\tilde{O}(1/\sqrt{\varepsilon})$ iterations, while the PI scheme requires $\tilde{O}(1/\varepsilon)$ iterations. This means that BKI based randomized SVD is more suitable for the scenario requiring higher accuracy.

Some accelerating skills have been proposed to speed up the basic rSVD algorithm [11], [14], [15], whose details will be addressed in the following section. However, they are developed individually and some of them just lack theoretic support. And, whether they are suitable for large sparse matrix is not well investigated.

C. Matrix Completion Algorithms

The matrix completion problem (1) is often relaxed to the problem minimizing the nuclear norm $\|\cdot\|_*$ of matrix:

$$\min_{\mathbf{X}} \|\mathbf{X}\|_*, \quad s.t.\ \mathcal{P}_\Phi(\mathbf{X}) = \mathcal{P}_\Phi(\mathbf{M}), \tag{5}$$

where $\mathcal{P}_\Phi(\cdot)$ is an orthogonal projector onto the span of matrices vanishing outside of set Φ. The solution of (4) can be approached by an iterative process:

$$\begin{cases} \mathbf{X}^i = shrink(\mathbf{Y}^{i-1}, \tau), \\ \mathbf{Y}^i = \mathbf{Y}^{i-1} + \delta\mathcal{P}_\Phi(\mathbf{M} - \mathbf{X}^i). \end{cases} \tag{6}$$

Here, $\tau > 0$, δ is a scalar step size, and $shrink(\mathbf{Y}, \tau)$ is a function which applies a soft-thresholding rule at level τ to the singular values of matrix $\mathbf{Y}$. As the sequence $\{\mathbf{X}^i\}$ converges, one derives the singular value thresholding (SVT) algorithm for matrix completion (i.e. Algorithm 2) [1].

Algorithm 2 SVT

Input: Sampled entries $\mathcal{P}_\Phi(\mathbf{M})$, tolerance parameter ϵ
Output: $\mathbf{X}_{\text{opt}}$

$\mathbf{Y}^0 = c\delta\mathcal{P}_\Phi(\mathbf{M})$, $r_0 = 0$
for $i = 1, 2, \cdots, i_{\max}$ **do**
  $k_i = r_{i-1} + 1$
  repeat
    $[\mathbf{U}^{i-1}, \mathbf{S}^{i-1}, \mathbf{V}^{i-1}] = \text{svds}(\mathbf{Y}^{i-1}, k_i)$
    $k_i = k_i + l$
  until $\mathbf{S}^{i-1}(k_i - l, k_i - l) \le \tau$
  $r_i = \max\{j : \mathbf{S}^{i-1}(j,j) > \tau\}$
  $\mathbf{X}^i = \sum_{j=1}^{r_i}(\mathbf{S}^{i-1}(j,j) - \tau)\mathbf{U}^{i-1}(:,j)(\mathbf{V}^{i-1}(:,j))^{\mathrm{T}}$
  if $\left\|\mathcal{P}_\Phi(\mathbf{X}^i) - \mathcal{P}_\Phi(\mathbf{M})\right\|_F / \|\mathcal{P}_\Phi(\mathbf{M})\|_F < \epsilon$ **then break**
  $\mathbf{Y}^i = \mathbf{Y}^{i-1} + \delta(\mathcal{P}_\Phi(\mathbf{M}) - \mathcal{P}_\Phi(\mathbf{X}^i))$
end for
$\mathbf{X}_{\text{opt}} = \mathbf{X}^i$

In Alg. 2, "svds($\mathbf{Y}, k$)" computes rank-k truncated SVD of $\mathbf{Y}$. There are some internal parameters which follow the empirical settings in [1]: $\tau = 5n$, where n is matrix column number, $l = 5$ and $c = \lceil \tau/(\delta \|\mathcal{P}_\Phi(\mathbf{M})\|_2) \rceil$. The value of δ affects the convergence rate, and one can slightly decrease it with the iteration.

Due to space limit, we omit the details of IALM algorithm [6]. In Section IV.A, with experiment we will show that the IALM is inferior to SVT algorithm for handling real data.

III. Faster Randomized SVD for Sparse Matrix

A. The Ideas for Acceleration

Because in each iteration of the SVT operation we need to compute truncated SVD of sparse matrix $\mathbf{Y}^{i-1}$, accelerating randomized SVD for sparse matrix is the focus. From Alg. 1, we see that Steps 2 and 7 occupy the majority of computing time if $\mathbf{A}$ is dense. However, for sparse matrix this is not true and optimizing other steps may bring substantial acceleration.

In existing work, some ideas were proposed to accelerate the basic rSVD algorithm. In [14], the idea of using eigendecomposition to compute the SVD in Step 8 of Alg. 1 was proposed. It was also pointed out that in the power iteration, orthonormalization after each matrix multiplication is not necessary. In [15], the power iteration was accelerated by replacing the QR factorization with LU factorization, and the Gaussian matrix is replaced with the random matrix with uniform distribution. In [11], the randomized SVD without power iteration was discussed for the dense matrix in image or video processing problem. It employs a variant of the basic rSVD algorithm, where the random matrix is multiplied to the left of $\mathbf{A}$. The algorithm is accelerated by using sparse random matrices and using eigendecomposition to obtain the orthonormal basis of the subspace.

Considering the situation for matrix completion, we decide only using the Gaussian matrix for $\mathbf{\Omega}$, because other choices are not suitable for sparse matrix (may cause $\mathbf{A\Omega}$ rank-deficient), and contribute little to the overall efficiency improvement. Other random matrix also degrades the accuracy of rSVD. The useful ideas for faster randomized SVD for sparse matrix are:

- use eigendecomposition for the economic SVD of $\mathbf{B}$;
- perform orthonormalization after every other matrix-matrix multiplication in the power iteration;
- perform LU factorization in the power iteration;
- replace the orthonormal $\mathbf{Q}$ with the left singular vector matrix $\mathbf{U}$.

We first formulate the eigendecomposition based SVD computation as an eigSVD algorithm (described as Alg. 3), where "eig()" computes eigendecomposition. Its correctness is given as Lemma 1.

Algorithm 3 eigSVD

Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m \ge n)$
Output: $\mathbf{U} \in \mathbb{R}^{m\times n}$, $\mathbf{S} \in \mathbb{R}^{n\times n}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$

$\mathbf{B} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$
$[\mathbf{V}, \mathbf{D}] = \text{eig}(\mathbf{B})$
$\mathbf{S} = \text{sqrt}(\mathbf{D})$
$\mathbf{U} = \mathbf{AVS}^{-1}$

Lemma 1. *The matrices $\mathbf{U}, \mathbf{S}, \mathbf{V}$ produced by Alg. 3 form the economic SVD of matrix $\mathbf{A}$.*

Proof. Suppose $\mathbf{A}$ has SVD as (2). Since $m \ge n$,

$$\mathbf{A} = \mathbf{U}(:, 1:n)\tilde{\mathbf{\Sigma}}\mathbf{V}^{\mathrm{T}}, \tag{7}$$

where $\tilde{\mathbf{\Sigma}}$, a square diagonal matrix, is the first n rows of $\mathbf{\Sigma}$. Eq. (7) is the economic SVD of $\mathbf{A}$. Then, Step 1 computes

$$\mathbf{B} = \mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{V}\tilde{\mathbf{\Sigma}}^2\mathbf{V}^{\mathrm{T}}. \tag{8}$$

The right-hand side is the eigendecomposition of $\mathbf{B}$. This means in Step 2, $\mathbf{D} = \tilde{\mathbf{\Sigma}}^2$ and $\mathbf{V}$ is the right singular vector matrix of $\mathbf{A}$. So, $\mathbf{S}$ in Step 3 equals $\tilde{\mathbf{\Sigma}}$, and lastly in Step 4 $\mathbf{U} = \mathbf{A}\mathbf{V}\mathbf{S}^{-1} = \mathbf{A}\mathbf{V}\tilde{\mathbf{\Sigma}}^{-1} = \mathbf{U}(:, 1:n)$. The last equality is derived from (7). This proves the lemma. □

Notice that eigSVD is especially efficient if $m \gg n$, when $\mathbf{B}$ becomes a small $n \times n$ matrix. Besides, the singular values in $\mathbf{S}$ are in ascending order. Numerical issues can arise if matrix $\mathbf{A}$ has not full column rank. Though more efficient than standard SVD, eigSVD is only applicable to special situations.

The idea that we can replace the orthonormal $\mathbf{Q}$ with the left singular matrix $\mathbf{U}$ can be explained with Lemma 2.

Lemma 2. *In the basic rSVD algorithm, orthonormal matrix $\mathbf{Q}$ includes a set of orthonormal basis of subspace $range(\mathbf{A\Omega})$ or $range((\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A\Omega})$. No matter how $\mathbf{Q}$ is produced, the results of basic rSVD algorithm do not change.*

Proof. The first statement is obviously correct by observing Alg. 1. The result of the basic rSVD algorithm is actually $\mathbf{QB} = \mathbf{Q}\mathbf{Q}^{\mathrm{T}}\mathbf{A}$, which further equals $\mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$. Notice that $\mathbf{Q}\mathbf{Q}^{\mathrm{T}}$ is an orthogonal projector onto the subspace $range(\mathbf{Q})$, if $\mathbf{Q}$ is an orthonormal matrix. The orthogonal projector is uniquely determined by the subspace [18], here equals to $range(\mathbf{A\Omega})$ or $range((\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A\Omega})$. So, no matter how $\mathbf{Q}$ is produced, $\mathbf{Q}\mathbf{Q}^{\mathrm{T}}$ does not change, and the basic rSVD algorithm's results do not change. □

Both QR factorization and SVD of a same matrix produce the orthonormal basis of its range space (column space), in $\mathbf{Q}$ and $\mathbf{U}$ respectively. So, with Lemma 2, we can replace $\mathbf{Q}$ with $\mathbf{U}$ from SVD in the basic rSVD algorithm.

Performing LU factorization is more efficient than QR factorization. It can be used while not affecting the correctness.

Lemma 3. *In the basic rSVD algorithm, the "orth()" operation in the power iteration, except the last one, can be replaced by LU factorization. This does not affect the algorithm's accuracy in exact arithmetic.*

Proof. Firstly, if the "orth()" is not performed, the power iteration produces $\mathbf{Q}$ including a set of basis of the subspace $range((\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A\Omega})$. As mentioned before, the "orth()" is just for alleviating the round-off error , and after using it $\mathbf{Q}$ still represents $range((\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A\Omega})$.

The pivoted LU factorization of a matrix $\mathbf{K}$ is:

$$\mathbf{P}\mathbf{K} = \mathbf{L}\mathbf{U}, \tag{9}$$

where $\mathbf{P}$ is a permutation matrix, and $\mathbf{L}$ and $\mathbf{U}$ are lower triangular and upper triangular matrices respectively. Obviously, $\mathbf{K} = (\mathbf{P}^{\mathrm{T}}\mathbf{L})\mathbf{U}$, where $\mathbf{P}^{\mathrm{T}}\mathbf{L}$ has the same column space as $\mathbf{K}$. So, replacing "orth()" with LU factorization (using $\mathbf{P}^{\mathrm{T}}\mathbf{L}$) also produces the basis of $range((\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A\Omega})$. Then, based on Lemma 2, we see this does not affect the algorithm's results in exact arithmetic. □

Notice that the LU factor $\mathbf{P}^{\mathrm{T}}\mathbf{L}$ has scaled matrix entries with linearly independent columns, since $\mathbf{L}$ is a lower-triangular matrix with unit diagonals and $\mathbf{P}$ just means row permutation. So, it also alleviates the round-off error. Finally, the orthonormalization or LU factorization in the power iteration can be performed after every other matrix multiplication. It harms the accuracy little, but remarkably reduces runtime.

B. Fast rSVD-PI Algorithm and rSVD-BKI Algorithm

Based on the above discussion, we find out that the eigSVD procedure can be applied to the basic rSVD to produce both the economic SVD of $\mathbf{B}$ and the orthonormal $\mathbf{Q}$. Because in practice $k + s \ll m$ or n and the matrices are not rank-deficient, using eigSVD induces no numerical issue. With these accelerating skills, we propose a fast rSVD-PI algorithm for sparse matrix (Alg. 4), where "lu(·)" denotes LU factorization function and its first output is "$\mathbf{P}^{\mathrm{T}}\mathbf{L}$".

Algorithm 4 rSVD-PI

Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, rank parameter k, power parameter p
Output: $\mathbf{U} \in \mathbb{R}^{m\times k}$, $\mathbf{S} \in \mathbb{R}^{k\times k}$, $\mathbf{V} \in \mathbb{R}^{k\times n}$

$\mathbf{\Omega} = \mathrm{randn}(n, k+s)$
$\mathbf{Q} = \mathbf{A\Omega}$
for $i = 0, 1, 2, 3, \cdots, p$ **do**
  if $i < p$ **then** $[\mathbf{Q}, \sim] = \mathrm{lu}(\mathbf{Q})$
  else $[\mathbf{Q}, \sim, \sim] = \mathrm{eigSVD}(\mathbf{Q})$ **break**
  $\mathbf{Q} = \mathbf{A}(\mathbf{A}^{\mathrm{T}}\mathbf{Q})$
end for
$\mathbf{B} = \mathbf{Q}^{\mathrm{T}}\mathbf{A}$
$[\mathbf{V}, \mathbf{S}, \mathbf{U}] = \mathrm{eigSVD}(\mathbf{B}^{\mathrm{T}})$
$ind = s+1 : k+s$
$\mathbf{U} = \mathbf{Q}\mathbf{U}(:, ind)$, $\mathbf{S} = \mathbf{S}(ind, ind)$, $\mathbf{V} = \mathbf{V}(:, ind)$.

Theorem 1. *Alg. 4 is mathematically equivalent to the basic rSVD algorithm (Alg. 1).*

Proof. One difference between Alg. 4 and Alg. 1 is in the power iteration (the "for" loop). Based on Lemma 1 we see that eigSVD accurately produces a set of orthonormal basis. And, based on Lemma 2 and 3, we see the power iteration in Alg. 4 is mathematically equivalent to that in Alg. 1. The other difference is the last three steps in Alg. 4. Its correctness is due to Lemma 1 and that the singular values produced by eigSVD is in the ascending order. □

For the scenario requiring higher accuracy, the BKI scheme [16] should be employed. Its main idea is to accumulate the subspaces generated in every iteration to form a larger subspace. Combining the accelerating skills we propose a fast BKI based rSVD algorithm (rSVD-BKI), i.e. Alg 5. Because the number of columns of $\mathbf{H}$ in Alg. 5 can be much larger than $\mathbf{Q}$'s in Alg. 4, we use "orth()" instead of eigSVD to produce $\mathbf{Q}$ finally. Similarly, we have the following theorem.

Theorem 2. *Alg. 5 is mathematically equivalent to the original BKI algorithm in [16].*

Algorithm 5 rSVD-BKI

Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, rank parameter k, power parameter p
Output: $\mathbf{U} \in \mathbb{R}^{m\times k}$, $\mathbf{S} \in \mathbb{R}^{k\times k}$, $\mathbf{V} \in \mathbb{R}^{k\times n}$

1: $\mathbf{\Omega} = \mathrm{randn}(n, k+s)$
2: $[\mathbf{H}_0, \sim] = \mathrm{lu}(\mathbf{A\Omega})$
3: **for** $i = 1, 2, 3, \cdots, p$ **do**
4: **if** $i < p$ **then** $[\mathbf{H}_i, \sim] = \mathrm{lu}(\mathbf{A}(\mathbf{A}^{\mathrm{T}}\mathbf{H}_{i-1}))$
5: **end for**
6: $\mathbf{H} = [\mathbf{H}_0, \mathbf{H}_1, ..., \mathbf{H}_p]$
7: $\mathbf{Q} = \mathrm{orth}(\mathbf{H})$
8: $\mathbf{B} = \mathbf{Q}^{\mathrm{T}}\mathbf{A}$
9: $[\mathbf{V}, \mathbf{S}, \mathbf{U}] = \mathrm{eigSVD}(\mathbf{B}^{\mathrm{T}})$
10: $ind = (k+s)(p+1) - k + 1 : (k+s)(p+1)$
11: $\mathbf{U} = \mathbf{QU}(:, ind)$, $\mathbf{S} = \mathbf{S}(ind, ind)$, $\mathbf{V} = \mathbf{V}(:, ind)$.

Both Alg. 4 and Alg. 5 can accelerate the randomized SVD for sparse matrix. They do not reduce the major term in computational complexity, but have smaller scaling constants and reduce other terms. They also inherit the theoretical error bound of the original algorithms [2], [16]. Their accuracy and efficiency will be validated with experiments in Section V.A. Based on the rSVD-BKI, we will derive a fast SVT algorithm for matrix completion problems in the following section.

IV. A Fast Matrix Completion Algorithm

A. The Choice of Algorithm

To evaluate the quality of matrix completion, we consider the mean absolute error (MAE),

$$\mathrm{MAE} = \frac{\sum_{ij\in\Phi} |\mathbf{M}_{ij} - \tilde{\mathbf{M}}_{ij}|}{|\Phi|}, \tag{10}$$

where $\mathbf{M}$ is the initial matrix, $\tilde{\mathbf{M}}$ is the recovered matrix, and $|\Phi|$ is the number of samples. MAE can be measured on the sampled matrix entries, or the whole matrix entries if the initial matrix is known. Before developing a faster matrix completion algorithm, we compare the SVT and IALM algorithms for recovering a $2,048 \times 2,048$ color image from 20% pixels in it (i.e. Case 2 in Section V.II). The MAE curves along the iteration steps produced with SVT and IALM algorithms are shown in Fig. 1. From it we see that SVT achieves much better accuracy than IALM, though the latter converges faster.

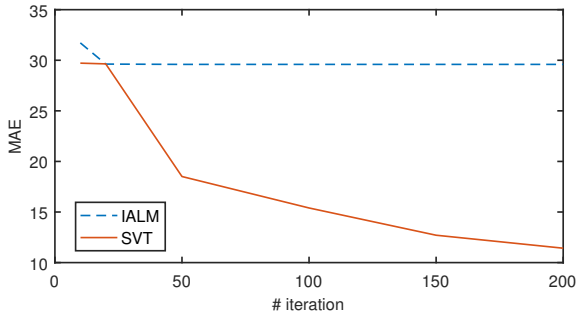


Fig. 1. The accuracy convergence of SVT and IALM algorithms.

A probable reason is that the IALM algorithm works well on some low-rank data, instead of the real data.

In this work, we focus on the acceleration of the SVT algorithm. A basic idea is replacing the truncated SVD in SVT algorithm with the fast rSVD algorithms in last section. However, with the iterations in SVT algorithm advancing, the rank parameter k_i becomes very large. Calculating so many singular values/vectors accurately is not easy. Firstly, the rSVD-BKI algorithm is preferable, which will be demonstrated with experiment in Section V.A. Secondly, with a large power p, its runtime advantage over `svds` or `lansvd` may lose, so that other accelerating technique is needed.

B. Subspace Recycling

The SVT algorithm uses an iterative procedure to build up the low rank approximation, where truncated SVD is repeatedly carried out on $\mathbf{Y}^i$. According to Step 11 in Alg. 2,

$$\left\|\mathbf{Y}^i - \mathbf{Y}^{i-1}\right\|_F = \delta \left\|\mathcal{P}_\Phi(\mathbf{M} - \mathbf{X}^i)\right\|_F \le \delta \left\|\mathbf{M} - \mathbf{X}^i\right\|_F \tag{11}$$

Because $\left\|\mathbf{M} - \mathbf{X}^i\right\|_F \to 0$ when iteration index i becomes large enough (see Theorem 4.2 in [1]), Eq. (11) means $\mathbf{Y}^i$ is very close to the $\mathbf{Y}^{i-1}$. So are the truncated SVD results of $\mathbf{Y}^i$ and $\mathbf{Y}^{i-1}$. The idea is to reuse the subspace of $\mathbf{Y}^{i-1}$ calculated in previous iteration step to speed up the SVD computation of $\mathbf{Y}^i$. This should be applied when i is large enough. Two recycling strategies are:

- Reuse the orthogonal basis $\mathbf{Q}$ in the rSVD-BKI for $\mathbf{Y}^{i-1}$, and then start from Step 8 in the rSVD-BKI algorithm for computing SVD of $\mathbf{Y}^i$.
- Reuse the left singular vectors $\mathbf{U}^{i-1}$ in last iteration to calculate SVD of $\mathbf{Y}^i$, with the following steps.

1: $\mathbf{B} = \mathbf{U}^{i-1\mathrm{T}}\mathbf{Y}^i$
2: $[\mathbf{V}^i, \mathbf{S}^i, \mathbf{U}^i] = \mathrm{eigSVD}(\mathbf{B}^{\mathrm{T}})$
3: $\mathbf{U}^i = \mathbf{U}^{i-1}\mathbf{U}^i$

The second strategy costs less time, because the size of $\mathbf{U}^{i-1}$ is $m \times k$ while the size of $\mathbf{Q}$ is $m \times (p+1)(k+s)$. However, it is less accurate than the first one. So, the second strategy is suitable for the situation where the error reduces rapidly in the iterative process of SVT algorithm, e.g. the image inpainting problem.

C. Fast SVT Algorithm

Based on the proposed techniques, we obtain a fast SVT algorithm described as Alg. 6. i_{reuse} represents the minimum iteration to execute subspace recycling, and q_{reuse} represents the maximum times of subspace recycling with one subspace. To guarantee the accuracy of randomized SVD, the power parameter p should increase with the iteration because the rank k_i of $\mathbf{Y}^i$ increases. Our strategy is increasing p by 1 once the relative error in Step 16 increases. This ensures a gradual decrease of error. And, if the error continuously decreases for 10 times, we reduce p by 1. This prevents overstating p. Other parameters follow the settings for Alg. 2 (see Section II.B).

Here we would like to explain the convergence of the proposed fast SVT algorithm. As proved in [16], the BKI based randomized SVD is able to attain any high accuracy if p is large enough. So is our rSVD-BKI algorithm. In the

Algorithm 6 fast SVT

Input: Sampled entries $\mathcal{P}_\Phi(\mathbf{M})$, tolerance ϵ
Output: $\mathbf{X}_{\text{opt}}$

$\mathbf{Y}^0 = c\delta\mathcal{P}_\Phi(\mathbf{M})$, $r_0 = 0$, $q = 0$, $p = 3$
for $i = 1, 2, \cdots, i_{\max}$ **do**
  $k_i = r_{i-1} + 1$, adjust the value of p
  repeat
    if $i < i_{\text{reuse}}$ **or** $q == q_{\text{reuse}}$ **then**
      $[\mathbf{U}^{i-1}, \mathbf{S}^{i-1}, \mathbf{V}^{i-1}] = \text{rSVD-BKI}(\mathbf{Y}^{i-1}, k_i, p)$
      $q = 0$
    else
      reuse $\mathbf{Q}$ or $\mathbf{U}$ in last execution of rSVD-BKI algorithm and compute $\mathbf{U}^{i-1}, \mathbf{S}^{i-1}, \mathbf{V}^{i-1}$
      $q = q + 1$
    end if
    $k_i = k_i + l$
  until $\mathbf{S}^{i-1}(k_i - l, k_i - l) \le \tau$
  $r_i = \max\{j : \mathbf{S}^{i-1}(j, j) > \tau\}$
  $\mathbf{X}^i = \sum_{j=1}^{r_i}(\mathbf{S}^{i-1}(j, j) - \tau)\mathbf{U}^{i-1}(:, j)(\mathbf{V}^{i-1}(:, j))^{\mathrm{T}}$
  if $\left\|\mathcal{P}_\Phi(\mathbf{X}^i) - \mathcal{P}_\Phi(\mathbf{M})\right\|_F / \|\mathcal{P}_\Phi(\mathbf{M})\|_F < \epsilon$ **then break**
  $\mathbf{Y}^i = \mathcal{P}_\Omega(\mathbf{Y}^{i-1}) + \delta(\mathcal{P}_\Phi(\mathbf{M}) - \mathcal{P}_\Phi(\mathbf{X}^i))$
end for
$\mathbf{X}_{\text{opt}} = \mathbf{X}^i$

fast SVT algorithm (Alg. 6), the k-truncated SVD is computed and k increases with the iterations. We initially set a p value which enables the rSVD-BKI algorithm attains same accuracy as `svds` for computing a few leading singular values/vectors. With the iteration advancing a mechanism gradually increasing p value is applied, such that rSVD-BKI can accurately compute more leading singular values/vectors. As a result, this accurate SVD computation guarantees that Alg. 6 behaves the same as the original SVT algorithm using `svds`. On the other hand, Theorem 4.2 in [1] proves the convergence of the original SVT algorithm. So, the convergence of our Alg. 6 is also guaranteed.

Notice that the subspace recycling technique is inspired by the theoretic analysis of (11). We have devised two recycling strategies and restrict their usage with parameters i_{reuse} and q_{reuse}. They, to some extent, ensure that the accuracy in the fast SVT algorithm will not degrade after incorporating the subspace recycling. This has been validated with extensive experiments, some of which are given in Section V.II and V.III.

V. Experimental Results

All experiments are carried out on a computer with Intel Xeon CPU @2.00 GHz and 128 GB RAM. The algorithms have been implemented in Matlab 2016a. `svds` in Matlab and `lansvd` in PROPACK [5] are used in Alg. 2, respectively. The resulted algorithms are compared with the proposed fast SVT algorithm (Alg. 6). The CPU time of different algorithms are compared, which is irrespective of the number of threads used in different SVT implementations.

The test cases for matrix completion are color images and movie-rating matrices from the MovieLens datasets [21]. Below, we first evaluate the proposed fast rSVD algorithms for sparse matrix and then validate the fast SVT algorithm.

A. Validation of Fast rSVD Algorithms

In this subsection, we first compare our rSVD-PI algorithm (Alg. 4) with the basic rSVD, cSVD (using `randn` as the random matrix) [11], pcafast [15], rSVDpack [14] algorithms. We consider a sparse matrix in size 45,115 $\times$ 45,115 obtained from the MovieLens dataset. The matrix has 97 nonzeros per row on average and is denoted by Matrix 1. Then, we randomly set some nonzero elements to zero to get two sparser matrices: Matrix 2 and 3 with 24 and 9 nonzeros per row on average, respectively. Setting rank $k = 100$, we performed the truncated SVD with different algorithms. The results are listed in Table I. Error there is the approximation error $\left\|\mathbf{A} - \tilde{\mathbf{A}}_k\right\|_F / \|\mathbf{A}\|_F$, where $\tilde{\mathbf{A}}_k$ denotes the computed rank-k approximation.

From the table we see that the proposed rSVD-PI algorithm has same accuracy as the basic rSVD algorithm, but is from 2.2X to 6.0X faster (Sp. in Table I denotes the speedup ratio to the basic rSVD). And, for a sparser matrix the speedup ratio increases. If the power iteration is not imposed ($p = 0$), cSVD and rSVDpack perform well, with at most 3.3X and 3.0X speedup respectively. When $p = 4$, the speedup ratios of these methods decrease. However, rSVDpack is better, due to the improvement of power iteration. pcafast also shows moderate speedup because it replaces QR with LU factorizaiton. These results verify the efficiency of our rSVD-PI algorithm for handling sparse matrix. It has up to **6.0X** speedup over the basic rSVD algorithm, and is several times faster than other state-of-the-art rSVD approaches.

Considering the scenario needing high accuracy, we compare rSVD-PI and rSVD-BKI algorithms with various matrices. Different values of power parameter p are tested and the results of `svds` are also given as the baseline. The results for Matrix 1 (setting $k = 100$) are listed in Table II. From it, we see that rSVD-BKI can reach the accuracy of `svds` in shorter runtime and a smaller $p = 4$. However, rSVD-PI cannot attain the accuracy of `svds` even when p is as large as 15. The experimental results show that rSVD-BKI achieves better accuracy than rSVD-PI in shorter CPU time, with much smaller p. This verifies that the rSVD-BKI algorithm (Alg. 5) is more efficient than rSVD-PI for high-precision computation.

As we have tested, to ensure the accuracy of SVD in the SVT iterations, the power p can increase to several tens while using rSVD-PI algorithm or similar randomized algorithms. This largely increase the runtime and makes rSVD-PI and those algorithms in Table I no competitive advantage over the standard SVD methods. So, we can only use rSVD-BKI in the following matrix completion experiments.

B. Image Inpainting

In this subsection, we test the matrix completion algorithms with a landscape color image. It includes $2,048 \times 2,048$ pixels, and we stack the three color channels of it to get a matrix in size of $6,144 \times 2,048$. Then, we randomly sample 10% and 20% pixels to construct Case 1 and Case 2 for

TABLE I
THE COMPUTATIONAL RESULTS OF DIFFERENT RANDOMIZED SVD ALGORITHMS ($k = 100$). THE UNIT OF CPU TIME IS SECOND

Setting		Matrix 1			Matrix 2			Matrix 3		
Algorithm	p	t_{cpu}	Error	Sp.	t_{cpu}	Error	Sp.	t_{cpu}	Error	Sp.
basic rSVD (Alg.1)	0	6.19	0.8166	*	5.10	0.9341	*	4.98	0.9506	*
cSVD [11]	0	2.76	0.8166	2.2	1.74	0.9352	2.9	1.51	0.9508	3.3
pcafast [15]	0	5.92	0.8188	1.0	5.04	0.9338	1.0	4.66	0.9506	1.1
rSVDpack [14]	0	2.59	0.8186	2.4	1.67	0.9355	3.1	1.67	0.9506	3.0
rSVD-PI (Alg.4)	0	**2.10**	0.8156	3.0	**1.10**	0.9342	4.8	**0.84**	0.9502	**6.0**
basic rSVD (Alg.1)	4	18.7	0.7305	*	13.2	0.8614	*	12.1	0.8804	*
cSVD [11]	4	14.9	0.7305	1.3	9.70	0.8614	1.4	8.51	0.8805	1.4
pcafast [15]	4	12.9	0.7305	1.5	8.36	0.8615	1.6	6.69	0.8805	1.8
rSVDpack [14]	4	11.7	0.7305	1.6	6.32	0.8617	2.1	5.40	0.8804	2.2
rSVD-PI (Alg.4)	4	**8.32**	0.7305	2.2	**3.18**	0.8615	4.2	**2.02**	0.8804	**6.0**

TABLE II
THE COMPARISON OF RSVD-PI AND RSVD-BKI ALGORITHMS

Algorithm	t_{cpu} (s)	Error	Sp.
`svds`	75.0	0.7289	*
rSVD-PI (Alg.4), $p = 2$	5.20	0.7345	14
rSVD-PI (Alg.4), $p = 15$	26.4	0.7290	2.8
rSVD-BKI (Alg.5), $p = 4$	22.0	0.7289	3.4

image inpainting, respectively. The error of image inpainting is measured with the MAE on all image pixels.

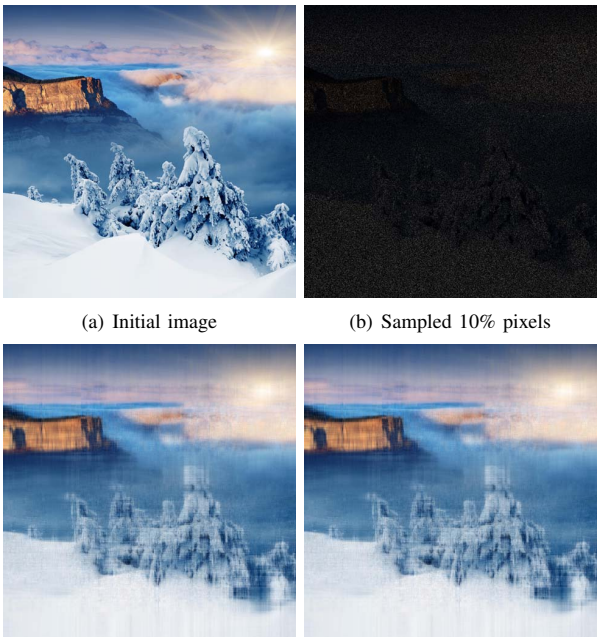
For the two cases, ϵ in the SVT algorithms is set 0.052 and 0.047 respectively. They correspond to the situation where the error of matrix completion does not decrease any more. The parameters for subspace recycling are $q_{reuse} = 10$, $i_{reuse} = 100$. And, we use the second recycling strategy reusing $\mathbf{U}$ matrix. Our fast SVT algorithm is compared with the SVT algorithm using `svds` and `lansvd`, see Table III.

TABLE III
THE RESULTS OF IMAGE INPAINTING (UNIT OF CPU TIME IS SECOND)

Test case	SVT (Alg.2)			fast SVT (Alg.6)			Sp1	Sp2
	$t_{\texttt{svds}}$	$t_{\texttt{lansvd}}$	MAE	$t_{w/o}$	$t_{w/}$	MAE		
Case 1	10,674	6,295	11.69	1,812	944	11.69	11.3	6.7
Case 2	19,358	10,008	8.854	3,254	**1,279**	8.854	**15.1**	**7.8**

In Table III, $t_{\texttt{svds}}$ and $t_{\texttt{lansvd}}$ denote the CPU time of the SVT algorithms (Alg. 2) using `svds` and `lansvd` for truncated SVD, respectively. $t_{w/o}$ and $t_{w/}$ denote the CPU time of our fast SVT Algorithm (Alg. 6) without and with subspace recycling, respectively. Sp1 and Sp2 are the ratios of $t_{\texttt{svds}}$ and $t_{\texttt{landsvd}}$ to the CPU time of our algorithm (with subspace recycling). We can see that the proposed algorithm is up to **15.1X** and **7.8X** faster than the SVT algorithms using `svds` and `lansvd`, respectively. Its memory cost is 512 MB, slightly larger than 420 MB used by the SVD algorithm with `lansvd`. All algorithms present the same accuracy (same MAE value), with same iteration numbers (400 for Case 1 and 700 for Case 2). The rank of the result matrix is 109 for Case 1 or 102 for Case 2. Comparing $t_{w/o}$ and $t_{w/}$ we see that the subspace recycling technique brings about 2X speedup, while not degrading the accuracy.

The recovered images from Case 1 are shown in Fig. 2, along with the original image. It reveals that our Alg. 6 produces same quality as the original SVT algorithm.



(a) Initial image (b) Sampled 10% pixels

(c) Recovered with Alg. 2 (d) Recovered with Alg. 6

Fig. 2. The initial image and recovered images from 10% pixels.

C. Rating Matrix Completion

The rating matrix includes users' ratings to movies, ranged from 0.5 to 5. For each dataset we keep a portion of ratings to be the training set. With them we recover the whole rating matrix and then use the remaining ratings to evaluate the accuracy of the matrix completion. In this experiment,

$q_{reuse} = 10$, $i_{reuse} = 50$, and the first subspace recycling strategy is used because it delivers better accuracy.

We first test a smaller dataset, including 10,000,054 ratings from 71,567 users judging 10,677 movies. We randomly sample 80% and 90% ratings as the training sets to obtain Case 3 and Case 4, respectively. The ϵ in the SVT algorithms is 0.16 and 0.19 for the both cases respectively, corresponding to the situation where the error of matrix completion does not decrease any more. The experimental results are in Table IV.

TABLE IV
THE RESULTS OF RATING MATRIX COMPLETION FOR A SMALLER DATASET

Test case	SVT (Alg.2)			fast SVT (Alg.6)		Sp1	Sp2
	$t_{\texttt{svds}}$	$t_{\texttt{lansvd}}$	MAE	$t_{w/}$	MAE		
Case 3	75,297	48,290	0.6498	15,133	0.6501	5.0	3.2
Case 4	19,813	12,509	0.6458	**3,771**	0.6460	**5.3**	**3.3**

According to Table IV, we see that the fast SVT algorithm has same accuracy as the original SVT algorithm. Here, MAE is measured on the remaining ratings. With the proposed techniques, the fast SVT algorithm is up to 5.3X and 3.3X faster than the methods using $t_{\texttt{svds}}$ and $t_{\texttt{lansvd}}$, respectively. From MAE we see that the error of rating estimation is on average 0.65, which is moderate.

Then, we test a larger dataset which includes 20,000,263 ratings from 138,493 users to 26,744 movies. It derives Case 5 and Case 6 by sampling 80% and 90% known ratings. The computational results are listed in Table V. They confirm the accuracy of the proposed algorithm again, and show its speedup up to 4.8X. It should be pointed out that the number of iterations to achieve the best quality in SVT algorithms are 362 for Case 5 and 293 for Case 6, which are larger than 208 for Case 3 and 153 for Case 4. But the ranks of the result matrix are 58 for Case 5 and 45 for Case 6 which are much smaller than 239 for Case 3 and 138 for Case 4. This explains why the CPU time for handling the larger dataset is less than that for handling the smaller dataset.

TABLE V
THE RESULTS OF RATING MATRIX COMPLETION FOR A LARGER DATASET

Test case	SVT (Alg.2)			fast SVT (Alg.6)		Sp1	Sp2
	$t_{\texttt{svds}}$	$t_{\texttt{lansvd}}$	MAE	$t_{w/}$	MAE		
Case 5	30,213	15,213	0.6676	6,582	0.6676	4.6	2.3
Case 6	19,951	9,785	0.6685	4,180	0.6685	4.8	2.3

VI. CONCLUSIONS

In this paper, we have presented two contributions. Firstly, a fast randomized SVD technique is proposed for sparse matrix. It results in two fast rSVD algorithms: rSVD-PI and rSVD-BKI. The former is faster than all existing approaches and up to 6X faster than the basic rSVD algorithm, while the latter is even better for problem requiring higher accuracy. Then, utilizing the rSVD-BKI, we propose a fast SVT algorithm for matrix completion. It also includes a new subspace recycling technique and is applied to the problems of image inpainting and rating matrix completion. The experiments with real data show that the proposed algorithm brings up to 15X speedup without loss of accuracy.

In the future, we will explore the application of this fast matrix completion algorithm to more AI problems.

REFERENCES

[1] J. F. Cai, E. J. Cands, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[2] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[3] H. Fang, Z. Zhang, Y. Shao, and C.-J. Hsieh, "Improved bounded matrix completion for large-scale recommender systems," in *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, Aug. 2017, pp. 1654–1660.

[4] D. Zhang, Y. Hu, J. Ye, X. Li, and X. He, "Matrix completion by truncated nuclear norm regularization," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 2192–2199.

[5] R. M. Larsen, "Propack-software for large and sparse svd calculations," *Available online. URL http://sun.stanford.edu/rmunk/PROPACK*, 2004.

[6] Z. Lin, M. Chen, and Y. Ma, "The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices," *arXiv preprint*, vol. arXiv:1009.5055, 2010.

[7] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM (JACM)*, vol. 58, no. 3, p. 11, 2011.

[8] M. W. Mahoney, "Randomized algorithms for matrices and data," *Foundations and Trends® in Machine Learning*, vol. 3, no. 2, pp. 123–224, 2011.

[9] D. P. Woodruff *et al.*, "Sketching as a tool for numerical linear algebra," *Foundations and Trends® in Theoretical Computer Science*, vol. 10, no. 1–2, pp. 1–157, 2014.

[10] P. G. Martinsson, "Randomized methods for matrix computations and analysis of high dimensional data," *arXiv preprint arXiv:1607.01649*, 2016.

[11] N. Benjamin Erichson, S. L. Brunton, and J. Nathan Kutz, "Compressed singular value decomposition for image and video processing," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 1880–1888.

[12] W. Yu, Y. Gu, J. Li, S. Liu, and Y. Li, "Single-pass PCA of large high-dimensional data," in *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, Aug. 2017, pp. 3350–3356.

[13] Y. Liang, M.-F. F. Balcan, V. Kanchanapally, and D. Woodruff, "Improved distributed principal component analysis," in *Advances in Neural Information Processing Systems*, 2014, pp. 3113–3121.

[14] S. Voronin and P.-G. Martinsson, "RSVDPACK: An implementation of randomized algorithms for computing the singular value, interpolative, and cur decompositions of matrices on multi-core and gpu architectures," *arXiv preprint arXiv:1502.05366*, 2015.

[15] H. Li, G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger, and M. Tygert, "Algorithm 971: An implementation of a randomized algorithm for principal component analysis." *ACM Transactions on Mathematical Software*, vol. 43, no. 3, pp. 1–14, 2017.

[16] C. Musco and C. Musco, "Randomized block Krylov methods for stronger and faster approximate singular value decomposition," in *Advances in Neural Information Processing Systems*, 2015, pp. 1396–1404.

[17] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[18] G. H. Golub and C. F. Van Loan, *Matrix Computations*. JHU Press, 2012.

[19] R. Lehoucq, D. Sorensen, and C. Yang, *ARPACK User's Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM Press, 1998.

[20] P. Drineas and M. W. Mahoney, "RandNLA: randomized numerical linear algebra," *Communications of the ACM*, vol. 59, no. 6, pp. 80–90, 2016.

[21] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 4, p. 19, 2016.