Efficient Nonnegative Tensor Factorization via Saturating Coordinate Descent

THIRUNAVUKARASU BALASUBRAMANIAM and RICHI NAYAK, Queensland University of Technology
CHAU YUEN, Singapore University of Technology and Design

With the advancements in computing technology and web-based applications, data are increasingly generated in multi-dimensional form. These data are usually sparse due to the presence of a large number of users and fewer user interactions. To deal with this, the Nonnegative Tensor Factorization (NTF) based methods have been widely used. However existing factorization algorithms are not suitable to process in all three conditions of size, density, and rank of the tensor. Consequently, their applicability becomes limited. In this article, we propose a novel fast and efficient NTF algorithm using the element selection approach. We calculate the element importance using Lipschitz continuity and propose a saturation point-based element selection method that chooses a set of elements column-wise for updating to solve the optimization problem. Empirical analysis reveals that the proposed algorithm is scalable in terms of tensor size, density, and rank in comparison to the relevant state-of-the-art algorithms.

CCS Concepts: • Computing methodologies → Factorization methods; • Information systems → Recommender systems; Spatial-temporal systems; • Theory of computation → Nonconvex optimization;

Additional Key Words and Phrases: Nonnegative tensor factorization, coordinate descent, element selection, saturating coordinate descent, pattern mining, recommender systems

ACM Reference format:
Thirunavukarasu Balasubramaniam, Richi Nayak, and Chau Yuen. 2020. Efficient Nonnegative Tensor Factorization via Saturating Coordinate Descent. ACM Trans. Knowl. Discov. Data 14, 4, Article 40 (May 2020), 28 pages.
https://doi.org/10.1145/3385654

1 INTRODUCTION

With the advent of tagging, sensor, and Internet of Things (IoT) technologies, online interaction data can be easily generated with tagging or spatio-temporal information. Tensor models become the natural choice to represent a multi-dimensional dataset, for example as \( \text{user} \times \text{image} \times \text{tag} \) or \( \text{user} \times \text{location} \times \text{time} \) \([7, 18, 25, 44]\). A Tensor Factorization (TF)-based method decomposes the tensor model into multiple factor matrices where each matrix learns the latent features inherent in the usage dataset \([25, 56]\). These factor matrices can be used to reconstruct (or approximate)
Table 1. Scalability in Terms of Size (Mode Length), Density (Percentage of Observed Entries in the Tensor) and Rank of the Tensor

| Algorithm | Mode length | Density | Rank |
|-----------|-------------|---------|------|
| APG [54]  | Low         | Low     | Low  |
| FMU [34]  | Medium      | Low     | Medium |
| FHALS [35] | Medium      | Low     | Medium |
| BCDP [51] | Medium      | Medium  | High  |
| CDTF [39] | High        | Medium  | Low   |
| GCD [5]   | High        | Medium  | Low   |
| SaCD      | High        | High    | High  |
| FSaCD     | High        | Very High | High  |

Algorithms are ranked as low, medium, high, and very high based on their capability in executing the process without running out of memory or out of time for synthetic datasets used in Sections 6.2 and 6.6. SaCD and FSaCD are the proposed algorithms.

The tensor to predict missing entries. These entries can be inferred as new items that have been generated based on correlations and dependencies between the data, and help in making recommendation generation and link prediction [2, 13, 14, 36]. A learned factor matrix reveals the latent features that are able to represent hidden patterns in the dataset in that particular dimension, and can help to understand people’s mobility patterns and usage [45].

A TF process is challenging for three main reasons. First, a tensor model tends to grow big as the data size increases. It especially becomes a problem for web applications due to a large Internet population and fewer user interactions. Second, due to fewer user interactions with all the items, the data generated are very sparse. Learning the associations or analyzing this sparse dataset is hard due to lack of correlation patterns. Third, the complexity of factorization process increases with an increase in the rank of the tensor. Here, the rank represents the number of hidden features in the dataset, in other words, it decides the size of factor matrices. Computing a large size sparse tensor model for exploring hidden latent relationships in the dataset is still a challenging problem. Table 1 shows the limited scalability of existing factorization algorithms. Nonnegative Tensor Factorization (NTF), which imposes the nonnegative constraint to the factor matrices learned during factorization, is the backbone of recommendation and pattern mining methods to understand interactions between features. The development of a fast and efficient NTF algorithm is needed for developing effective web recommendation and pattern mining methods.

We propose the Saturating Coordinate Descent (SaCD) factorization algorithm that reduces the complexity inherent in the factor matrix update by carefully selecting the important elements to be updated in each iteration. We propose a Lipschitz continuity-based element importance calculation to effectively select the elements for an update that will lead fast convergence. We also propose a fast-parallelized version of SaCD (named as FSaCD) to further speed up the factorization process that suffers from Intermediate Data Explosion (IDE). IDE is caused due to the materialization and storage of the intermediate data generated. The column-wise element selection and element update minimizes the complexity of IDE and parallelization becomes easier.

Extensive experiments with recommendation and pattern mining applications show improved scalability (i.e., size, density, and rank) performance of SaCD and FSaCD in comparison to all relevant baseline algorithms. The results also show that the efficiency is achieved at no cost of accuracy. In fact, SaCD achieves high accuracy when compared to other benchmarks.
The specific contribution of this article is four-fold.

1. We propose a novel Lipschitz continuity-based element importance calculation method to select certain elements for updating in the factorization process.

2. We propose a novel SaCD factorization algorithm that updates factor matrices by selecting elements using the proposed Lipschitz continuity-based element importance and leads fast convergence to a local optimum solution.

3. We apply SaCD in recommender systems and pattern mining effectively with high accuracy and scalability.

4. We design FSaCD by parallelizing SaCD on the single machine using multi-cores that further improves the performance on big and dense datasets.

To our best of knowledge, SaCD is the first algorithm that can efficiently process the large and sparse tensor model with large rank factor matrices, with and without parallelization.

2 RELATED WORK

TF based recommender systems and pattern mining methods: Tensor models have been widely applied in several web-based applications and have shown superiority in generating latent patterns [14, 19, 43]. The majority of model-based Collaborative Filtering methods, including Matrix Factorization (MF), fail to utilize a context along with users and items information because of their limited capability to deal with two-dimensional data only [44]. The capability of a tensor model to represent multidimensional data makes them a natural choice to represent the user interaction data [26]. The last decade has witnessed the rise of tensor-based recommendation methods [18, 36, 44, 46, 53, 55].

A three-dimensional tensor model represents the data as \((user \times item \times context)\) and a TF/NTF algorithm can be used to capture dimensional dependencies and general recommendations [22]. For example, a \((user \times item \times tag)\) tensor model has been used in social tagging systems for representing how users have used tags with items [9, 19]. A TF method with higher order singular value decomposition (HOSVD) has been used to recommend items based on tagging behavior or to recommend tags for items [47]. Link prediction in social networks has been solved using Alternating Least Square (ALS) based NTF by predicting the missing relations in the dataset, for example, identifying relations among users or recommending items to users by considering time as the additional context [14]. Researchers have used the \(p - core\) approach to minimize the size and sparsity of the tensor by only including the data with occurrence of \((user, item)\) at least \(p\) times, as the traditional TF/NTF methods are not scalable and their accuracy on the sparse dataset is low [7, 19, 36]. However, the \(p - core\) approach removes any users/items with less than \(p\) interactions, thus, leaving the recommendation process incomplete for some users.

Recent advancements in IoT allow recording additional spatio-temporal information easily in addition to the web interactions. This has led to focus research on Location-Based Social Networks (LBSNs) [53]. Researchers have successfully applied TF/NTF using ALS for automated mining of temporal patterns [5, 29]. For example, the Foursquare dataset with the user, venue, and time is represented as a tensor and the \(r\) number of patterns are identified by the factor matrix in the time mode where \(r\) is a tensor rank [56]. Similarly, the temporal trajectories of online gamers are derived using NTF [37]. These patterns allow learning the behavior of web users that helps to personalize and improve the interactions. Tensor rank is an important parameter in the factorization process that learns the hidden features of the dataset. These hidden features represent the data effectively in a lower dimension, where higher the number of hidden feature identified, higher is the chances of learning the true representation of dataset [44].
With the dependence of these methods on TF to learn the associations in the data, it is important that a factorization algorithm can efficiently deal the tensor models with the larger size, density, and rank.

**Fast TF Algorithms:** Factorization algorithms, namely, ALS [23], Multiplicative Update rule (MU) [12], Gradient Descent (GD) [54], and Coordinate Descent (CD) [17, 30, 50] have been commonly used to factorize the tensor models. These traditional algorithms have been extended to tensors based on MF. Researchers have explored the concepts of gradient calculation and tensor block to fasten the factorization process in these factorization algorithms. Fast Hierarchical ALS (FHALS) [35] and Fast MU (FMU) [34] are the optimized variants of ALS and MU by exploiting an efficient gradient calculation method for the faster execution. The traditional gradient calculation requires the tensor unfolding at each iteration which is relatively slow. FHALS and FMU use a simplified update rule with the gradient calculation that does not require the tensor unfolding. Accelerated Proximal Gradient (APG) [54] utilizes a low-rank approximation to speed up the factorization process. The Block Coordinate Descent (BCDP) [51] technique has been applied in NTF to minimize the complexity by processing the tensor in convex blocks. The element selection-based CD algorithm called Greedy Coordinate Decent (GCD) [17] has shown fast convergence in Nonnegative Matrix Factorization (NMF) process. But the extension of GCD to NTF involves high complexity because of the frequent gradient updates [5].

Few researchers have explored parallel and distributed computational methods to minimize the time complexity involved in the Matricized Tensor Times khatri-Rao Product (mttkrp) and matrices updates during TF [8, 11, 21, 32, 33, 39, 41]. CCD++ [52] is a CD algorithm for MF that updates the elements in a factor matrix column-wise. Coordinate Descent for Tensor Factorization (CDTF) [39] and Subset Alternating Least Square (SALS) [38] are two recent extensions of CCD++ for TF. While CDTF is a direct extension of CCD++ for TF, SALS is a special case with an additional constraint that controls the number of columns to update in a single iteration. The column-wise factor matrix update minimizes the complexity of mttkrp in these algorithms and introduces a huge advantage in the distributed and parallel environment. However, these implementations do not alter the computational complexity inherent in the underlying factorization algorithm. We have implemented CDTF for NTF and used it as a benchmark in experiments. SALS cannot be directly extended for NTF due to the additional constraint it imposes.

**Limitations:** While NTF has been applied widely with tensor-based methods to utilize multi-dimensional data, it suffers from the scalability issue due to the complex matrix and tensor products involved in the factorization process [39]. Existing factorization algorithms are limited to small datasets only (as listed in Table 1). While parallel and distributed computational methods improve the runtime and scalability by utilizing huge resources, performance on a normal machine with fewer memory requirements remains poor [32]. While optimizing the basic factorization process can improve the performance without depending on huge resources, not all the optimized factorization algorithms are easily extendable to a parallel and distributed environment to further improve the performance [24]. Most importantly, not all the factorization algorithms can effectively handle tensors of large size, sparsity, and rank [32]. We propose a factorization algorithm, SaCD, to efficiently process the large and sparse tensor model, with and without parallelization, by avoiding the frequent gradient calculations that are essential in existing algorithms [5, 17].

### 3 Basics: Preliminaries and Definitions

The notations used in this article are summarized in Table 2. The process of converting a tensor into a matrix is called as matricization or unfolding of tensor [49]. The mode – 1 matricization can be denoted as $X_1 \in \mathbb{R}^{(Q \times (PS))}$ for a third-order tensor $X \in \mathbb{R}^{(Q \times P \times S)}$ where $Q$, $P$, and $S$ denotes...
the mode length. Several matrix products are required for tensor-based processes [27]. We briefly introduce them.

### 3.1 Kronecker Product

For two matrices denoted as \( U \in \mathbb{R}^{(Q \times R)} \) and \( V \in \mathbb{R}^{(P \times R)} \), the Kronecker product is presented as \( U \otimes V \). The resultant matrix of size \( (QP \times R^2) \) is defined as follows:

\[
U \otimes V = \begin{bmatrix}
  u_{11}V & u_{12}V & u_{13}V & \ldots & u_{1r}V \\
  u_{21}V & u_{22}V & u_{23}V & \ldots & u_{2r}V \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  u_{q1}V & u_{q2}V & u_{q3}V & \ldots & u_{qr}V \\
\end{bmatrix},
\]

(1)

\[
= [u_1V \ u_2V \ u_3V \ \ldots \ u_rV],
\]

(2)

where \( u_r \) is \( r^{th} \) column of the factor matrix \( U \).

### 3.2 Khatri-Rao Product

The column-wise Kronecker product, called as Khatri-Rao product, is denoted as \( U \odot V \). The resultant matrix of size \( (QP \times R) \) is defined as:

\[
U \odot V = [u_1 \otimes v_1 \ \ldots \ u_r \otimes v_r],
\]

(3)

where \( u_r \) and \( v_r \) are columns of the matrices \( U \) and \( V \) respectively.
3.3 Hadamard Product

When the size of two matrices are same, the hadamard product can be calculated by the element-wise matrix product. It is denoted by $U \ast V$ and defined as:

$$
U \ast V = \\
\begin{bmatrix}
    u_{11} & u_{12} & u_{13} & \cdots & u_{1r}v_{1r} \\
    u_{21} & u_{22} & u_{23} & \cdots & u_{2r}v_{2r} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    u_{q1} & u_{q2} & u_{q3} & \cdots & u_{qr}v_{qr}
\end{bmatrix}.
$$

(4)

3.4 Tensor Factorization

TF, a dimensionality reduction technique, is an extension of MF for higher order. It factorizes a tensor into factor matrices that contain latent features. CANDECOMP/PARAFAC (CP) and Tucker are two well-known factorization techniques \[25\]. CP has shown to be less expensive in both memory and time as compared to Tucker \[25, 32\].

**Definition 1 (CP Factorization).** For a tensor $X \in \mathbb{R}^{(Q \times P \times S)}$ and rank $R$, CP factorization factorizes the tensor into a sum of component rank-one tensors \[10\] (as shown in Figure 1) as:

$$
X \cong [U, V, W] = \sum_{r=1}^{R} u_r \circ v_r \circ w_r,
$$

(5)

where $U \in \mathbb{R}^{(Q \times R)}$, $V \in \mathbb{R}^{(P \times R)}$ and $W \in \mathbb{R}^{(S \times R)}$ are factor matrices with $R$ hidden features, $R \in \mathbb{Z}_+$. The goal of CP TF is to approximate the input tensor and its objective function can be formulated as follows,

$$
\min_{U, V, W} f(U, V, W) = \|X - [U, V, W]\|^2.
$$

(6)

The learning process is modeled as the minimization of Euclidean loss function as shown in Equation (6). The optimization process (Equation (7)) aims to learn the factors to reduce the difference between the original tensor and approximated tensor, where the predicted value $\hat{x}_{qps}$ is calculated by the inner product of learned factors across all the features, as formulated in Equation (8) \[12\]:

$$
f(U, V, W) = \sum_{q=1}^{Q} \sum_{p=1}^{P} \sum_{s=1}^{S} \|x_{qps} - \hat{x}_{qps}\|^2.
$$

(7)

$$
\hat{x}_{qps} = \sum_{r=1}^{R} u_{qr} \cdot v_{pr} \cdot w_{sr}.
$$

(8)

3.5 Nonnegative Tensor Factorization

The basic idea of NTF is factorizing the $n$-dimensional tensor into $n$ factor matrices that satisfy the nonnegative constraint \[25\]. NTF can be achieved in traditional Tucker or CP factorization model by imposing constraints to maintain the nonnegative values.
The objective function for NTF based on CP model can be formulated as follows:

$$\min_{U, V, W \geq 0} f(U, V, W) = \|X - [U, V, W]\|^2. \quad (9)$$

The approximated tensor, after factorization, using the learned factor matrices will be a denser model in which a large portion of absent values are populated [31, 42]. For applications such as predictive modelling and recommender systems, these populated values can be inferred as potential prediction or recommendation [6, 18].

The goal of the optimization problem in Equation (9) is to find the accurate factor matrices $U$, $V$, and $W$. ALS [48] is the most common algorithm used to find the factor matrices [35, 40]. It gives equal importance to all the elements and alternatively updates the entire factor matrix by fixing all the other factor matrices. This unnecessary element updates can cause slow convergence and poor scalability [1, 17].

4 SATURATING COORDINATE DESCENT ALGORITHM FOR NTF

In this article, we propose the SaCD algorithm to solve Equation (9) in order to reduce the complexity inherent in the factor matrix update and to overcome the poor scalability faced by traditional algorithms like ALS [35], APG [54], GCD [17], CDTF [39], and BCDP [51]. Figure 2 provides the detail of the overall process.

4.1 Factor Matrix Update Rule

We first explain the learning process of factor matrix $U$ which is applicable to other factor matrices. The factor matrix $U$ is updated by solving Equation (10) and to solve the equation, we need to find the first order and second partial derivatives (Equation (11) and Equation (12), respectively) for $f$ in Equation (9) as follows:

$$\min_{U \geq 0} \|X_1 - U(W \odot V)^T\|^2, \quad (10)$$
\[ \frac{\partial f}{\partial U} = G = -X_1(W \odot V) + U(V^T V * W^T W), \] (11)

\[ \frac{\partial^2 f}{\partial U} = H = V^T V * W^T W, \] (12)

where \( \frac{\partial f}{\partial U} \) and \( \frac{\partial^2 f}{\partial U} \) are the first-order and second-order partial derivatives of the objective function \( f \) (Equation (9)) with respect to \( U \) and let us denote them as \( G \) (gradient) and \( H \) (second-order derivative), respectively.

The one variable gradient and second order derivative are computed as,

\[ g_{qr} = -(X_1(W \odot V))_{qr} + (U(V^T V * W^T W))_{qr}, \] (13)

\[ h_{rr} = (V^T V * W^T W)_{rr}, \] (14)

where \( g_{qr} \) and \( h_{rr} \) represents \( q, r \)th gradient \( G \) and second-order derivative \( H \), respectively.

Equation (9) becomes a quadratic equation in terms of the updated parameter when all other parameters are fixed. This leads to the following closed form CD update rule (i.e., one variable sub-problem) [35] for each parameter:

\[ \hat{u}_{qr} = \frac{g_{qr}}{h_{rr}}. \] (15)

The one variable sub-problem can be a simplified ALS update rule where each factor matrix is updated element-wise, thus it reduces the computational complexity. The nonnegative constraint can be added to Equation (15) as:

\[ u_{qr} \leftarrow \max(0, u_{qr} - \hat{u}_{qr}) - u_{qr}, \] (16)

where \( u_{qr} \) indicates the \( q, r \)th element of the factor matrix \( U \), and \( \hat{u}_{qr} \) indicates the computed element.

With the computed element value (\( \hat{u}_{qr} \)), the element \( u_{qr} \) can be updated as:

\[ u_{qr} \leftarrow u_{qr} + \hat{u}_{qr}. \] (17)

Since the calculation of \( X_1(W \odot V) \) and \( V^T V * W^T W \) for every element is expensive, it is better to calculate this value only once at each iteration, instead of calculating it for every element update, during the factorization process to find the best learning factor matrices.

In NMF, element selection has been proven to converge faster for updating the important elements repeatedly, instead of considering all elements [17]. The traditional measurement of element importance and update is computationally inefficient for NTF due to frequent gradient updates. Next, we show how an element importance can be efficiently calculated and only important elements can be updated that can avoid frequent gradient updates.

### 4.2 Proposed Lipschitz Element Importance

The existing partial differential equations based optimization algorithms like APG, FHALS, FMU, and GCD are prone to slow convergence due to inconsistent gradients, i.e., gradients shift on both directions from the global or local minima in the non-convex optimization curve [23]. Additional regularity conditions has been incorporated to speed-up the convergence in NMF [15]. In this article, we propose to smooth (Lipschitz smoothness) the continuous function \( f \) with a strong condition of Lipschitz continuity for fast convergence.

**Definition 2 (Lipschitz continuity).** Lipschitz continuity is a strong form of uniform continuity for a function if the function \( f \) is differentiable with Lipschitz constant \( L \).
The continuously differentiable function \( f \) is called Lipschitz continuous with Lipschitz constant \( L \), such that:

\[
\| f(u^k) - f(u^{k+1}) \| \leq L \| u^k - u^{k+1} \|, \forall u^k \in U^k, u^{k+1} \in U^{k+1},
\]

where \( U^k \) and \( U^{k+1} \) are two factor matrices at two consecutive iterations.

**Definition 3 (Lipschitz smoothness).** The Lipschitz continuous function, as defined in Equation (18), can be smoothed with the upper bound value for \( L \). It ensures the strong convexity of the function and achieves faster convergence. This property is called Lipschitz smoothness and the function \( f \) with this property is called \( L \)-smoothed. It can be defined as,

\[
f(u^k) \geq f(u^{k+1}) + f'(u^{k+1})(u^k - u^{k+1}) + \frac{L}{2} \| f(u^k) - f(u^{k+1}) \|.
\]

The upper bound value for \( L \) should be calculated to satisfy,

\[
f''(U, V, W) - LI \geq 0,
\]

where \( I \) is an Identity matrix.

Equations (11) and (12) can reveal that the objective function \( f \) is differentiable with respect to factor matrices as well as it is non-convex. We propose to analyse these properties using Lipschitz continuity and Lipschitz smoothness and measure the element importance in order to achieve faster convergence.

Let \( Z \) be the element importance matrix that holds the importance value for each element in a factor matrix. The importance of element \( u_{qr} \) can be calculated as the difference in the objective function as:

\[
z_{qr} = g_{qr} - \hat{g}_{qr},
\]

where \( g_{qr} \) represents the gradient of \( u_{qr} \) and \( \hat{g}_{qr} \) represents the new gradient value of the computed element \( \hat{u}_{qr} \).

With the precomputed \( g_{qr} \) as per Equation (13), we can compute \( \hat{g}_{qr} \) using one variable subproblem [17] as follows:

\[
\hat{g}_{qr} = g_{qr} + g_{qr} \hat{u}_{qr} + \frac{1}{2} \left( h_{rr} \hat{u}_{qr}^2 \right).
\]

Substituting Equations (13) and (22) in Equation (21), we get the element importance as:

\[
z_{qr} = -(g_{qr} \hat{u}_{qr}) - 0.5 \left( h_{rr} \hat{u}_{qr}^2 \right).
\]

In Equation (23), \( h_{rr} \) is the partial derivative of gradient \( g_{qr} \) as defined in Equation (14). An objective function holding Lipschitz continuity with differentiable gradients can be Lipschitz-smoothed (\( L \)-smoothed) (as proven in Lemma 4.1). This property will allow the function to converge faster by replacing the partial derivative of gradient (i.e. \( h_{rr} \) in (23)) with Lipschitz constant \( L \) [15].

We propose to calculate the difference in the \( L \)-smoothed objective function for each element’s update during the factorization using Lipschitz continuity [16] that can achieve faster convergence. The convergence of an optimization problem can be analyzed using curvature measure \( C_f \) that measures the deviation of the objective function \( f \) with the linear approximation. The Lipschitz smoothness of \( f \) has shown the best \( C_f \) with the improvement of convergence speed to \( \frac{1}{K} \) where \( K \) is the number of iterations [20]. We conjecture that use of Lipschitz continuity in the element importance calculation will speed up the convergence.

By applying the upper bound value of Lipschitz continuity in the one variable subproblem Equation (22), we have:

\[
\hat{g}_{qr} = g_{qr} + g_{qr} \hat{u}_{qr} + \frac{L}{2} \left( \hat{u}_{qr}^2 \right),
\]

\footnote{For a formal analysis of curvature measure and Lipschitz continuity, we direct the readers to [20].}
where $L$ is a positive scalar called Lipschitz constant.

With the redefined one variable sub-problem, we can define Lipschitz element importance by substituting Equations (13) and (24) in Equation (21) as:

$$z_{qr} = -(q_r \hat{u}_{qr}) - \frac{L}{2} (\hat{u}_{qr}^2).$$

(25)

**Lemma 4.1.** The gradient of the objective function Equation (9) satisfies Lipschitz continuity.

**Proof.** Supposedly we have two factor matrices $U^k$ and $U^{k+1}$ at two consecutive iterations.

$$\left\| \frac{\partial f}{\partial U^{k-1}} - \frac{\partial f}{\partial U^k} \right\| = \left\| X_1(W \odot V) + U^{k-1}(V^T * W^T W) - (X_1(W \odot V) + U^k(V^T * W^T W)) \right\|.$$  

(26)

Applying *Definition 2* and assuming the objective function is differentiable, we obtain:

$$\left\| \frac{\partial f}{\partial U^{k-1}} - \frac{\partial f}{\partial U^k} \right\| = L \left\| U^{k-1} - U^k \right\|.$$  

(27)

Equating Equations (26) and (27), we can identify the value for $L$ as $\left\| V^T * W^T W \right\|$. $L$ is the singular upper-bound value that defines the maximum curves an objective function allowed to have and making the function $f$ $L$-smoothed for faster convergence.

Solving Equations (26) and (27) for one variable subproblem with $u^k \in U^k$ and $u^{k+1} \in U^{k+1}$ proves Lemma 4.1. \hfill $\square$

### 4.3 SaCD

Once we have the element importance calculated for all the elements in the factor matrix $U$ using Equation (25) and stored them in $Z$, we now select a set of elements for every iteration using the proposed SaCD algorithm. The state-of-the-art GCD NMF algorithm uses greedy strategy in finding and updating single element multiple times in each row [17]. This requires a very expensive frequent gradient update. In the minimization optimization problem as formulated in Equation (9), the error is minimized for each iteration. It slowly reaches convergence, or reaches a saturation point beyond which updating will not minimize the objective function $f$. Hence, the contribution of each element in minimizing the objective function decreases with each iteration. Moreover, not all the elements will effectively minimize the objective function. It is sufficient and efficient to update every single element until it reaches the saturation point, instead of updating it for all the iterations. This avoids the expensive frequent gradient update as proven in Lemma 4.2.

As the element importance $z_{qr}$ is the difference in the objective function as per Equation (21), we can identify the saturation point $sp_{qr}$ of each element for every iteration $k$, by keeping track of the previous value of $z_{qr}^{k-1}$ as:

$$sp_{qr} = z_{qr}^{k} - z_{qr}^{k-1}.$$  

(28)

Additionally, for each iteration, we measure the total importance of a factor matrix as:

$$ti^k = \sum_{q=1}^{Q} \sum_{r=1}^{R} z_{qr}^k,$$  

(29)

$$ti^{k-1} = \sum_{q=1}^{Q} \sum_{r=1}^{R} z_{qr}^{k-1}.$$  

(30)
If the total importance of current iteration $t^k$ is more than the previous iteration $t^{(k-1)}$, we identify the saturation point $sp_{qr}$ as:

$$sp_{qr} = (z_{qr}^{k-1} - z_{qr}^k).$$  \(31\)

While the difference in the objective function gradually decreases, sometimes it increases. This happens when the optimization is reaching local or global minima. To avoid stuck in local minima, this redefinition of the saturation point is needed that further allows important elements to be updated until global minima is reached. We use this saturation point to decide if the element $u_{qr}$ is to be updated as per Equation (17). If $sp_{qr} < 0$, we avoid updating that element.

We have explained the proposed algorithm by describing the learning process of factor matrix $U$. Next we briefly show that the process is applicable to other factor matrices too.

**Lemma 4.2.** In the SaCD element selection, for each update of $u_{qr}$, it is not necessary to update gradient of all the columns of $q^{th}$ row and element importance $z_{qr}$.

**Proof.** Let $u_{qr} \in U$ be $q, r^{th}$ element of $U$ and $g_{qr} \in G$ represents $q, r^{th}$ gradient of $U$.

For every $r : R$, the set of important elements $e_r$ are selected based on the saturation point calculated using Equation (28) or Equation (31).

The subset $e_r$ is dependent on $r^{th}$ column of $G$.

For each update of $u_{qr} \in e_r$, $g_{qr} \in G$ alone needs to be updated as $e_r \perp g_y$ where $y \neq r$ and $g_{qr} \perp u_{nr}$ where $n \neq q$ and $\perp$ indicates that $e_r$ is independent of $g_y$.

Therefore, for each update of $u_{qr}$, it is enough to update $g_{qr}$ alone and it is not necessary to update $g_{qr}$ of all the columns of $q^{th}$ row and element importance $z_{qr}$. \(\square\)

**Updating solution for factor matrix $V$:** Taking the first and second order partial derivatives of the function $f$ with respect to $V$, we have a new solution for the gradient $G$ and second order derivative $H$ as:

$$\frac{\partial f}{\partial V} = G = -X_2(W \odot U) + V(U^T U \ast W^T W),$$  \(32\)

$$\frac{\partial^2 f}{\partial V} = H = U^T U \ast W^T W.$$  \(33\)

With the updated $G$ and $H$ as per Equations (32) and (33), the element importance is calculated according to Section 4.1 and the factor matrix update is performed in similar fashion.

**Updating solution for factor matrix $W$:** Taking the first- and second-order partial derivatives of the function $f$ with respect to $W$, we have a new solution for the gradient $G$ and second order derivative $H$ as:

$$\frac{\partial f}{\partial W} = G = -X_3(V \odot U) + W(U^T U \ast V^T V),$$  \(34\)

$$\frac{\partial^2 f}{\partial W} = H = U^T U \ast V^T V.$$  \(35\)

With the updated $G$ and $H$ as per Equations (34) and (35), the element importance is calculated according to Section 4.1 and the factor matrix update is performed in similar fashion.

Algorithm 1 details the process.

ACM Transactions on Knowledge Discovery from Data, Vol. 14, No. 4, Article 40. Publication date: May 2020.
Algorithm 1: SaCD Algorithm

**Input:** Tensor $X$; Randomly Initialized factor matrices $U \in \mathbb{R}^{Q \times R}$, $V \in \mathbb{R}^{P \times R}$, $W \in \mathbb{R}^{S \times R}$; Rank $R$; $Z = \emptyset$; Number of rows in any given factor matrix $\text{rows}$; Maximum number of iterations $\text{maxiters}$.

**Output:** Learned Factor matrices $U$, $V$, $W$

for $k = 1 : \text{maxiters}$ do
    compute $G$ and $H$ using Equations (11) and (12); $L = \|H\|$
    if $k == 1$ then
        for $r = 1 : R$ do
            for $q = 1 : \text{rows}$ do
                compute element importance $z_{qr}$ using Equation (25);
                store the initial element importance, $z^{k-1}_{qr} = z_{qr}$;
                if $z_{qr} > 0$ then
                    update the element $u_{qr}$ using Equation (17);
                end
            end
        end
    else
        for $r = 1 : R$ do
            for $q = 1 : \text{rows}$ do
                compute element importance $z_{qr}$ using Equation (25);
                identify the saturation point $sp_{qr}$ using Equation (28) or Equation (31);
                if $sp_{qr} > 0$ then
                    update the element $u_{qr}$ using Equation (17);
                end
                update the previous element importance, $z^{k-1}_{qr} = z_{qr}$;
            end
        end
    end
    repeat analogues lines 4 to 26 with $G$ and $H$ calculated using Equations (32) and (33) respectively to update elements of $V$ in lines 11 and 21;
    repeat analogues lines 4 to 26 with $G$ and $H$ calculated using Equations (34) and (35) respectively to update elements of $W$ in lines 11 and 21;
end

4.4 Fast Saturating Coordinate Descent (FSaCD) Using Parallelization

In this section we propose the FSaCD algorithm that leverages the column-wise element update to speed up the factorization process of SaCD. Instead of pre-computing the gradient using Equation (11), we propose to calculate the gradient column-wise during the column-wise element update. To further improve the performance of SaCD, we utilize the multiple cores of the single machine.

The pre-calculation of $G$ as per Equation (11) consists of an expensive $\text{mttkrp}$ operation (i.e., $X_1(V \odot W)$), a complex step in factorization that causes the IDE. IDE is caused due to the materialization and storage of the intermediate data ($V \odot W$). The calculation of $\text{mttkrp}$ is a rather infamous computational kernel with a lot of related work attempting to optimize the computations [3, 11, 41]. We utilize the concept of sparse tensor times vector product ($\text{sttvp}$) which redesigns
the NTF algorithm to calculate the column-wise $\text{mttkrp}$ [4]. $\text{sttvp}$ simplifies $\text{mttkrp}$ by multiplying sparse tensor to a vector instead of multiplying it to a matrix and minimizes the IDE. We call this Fast SaCD algorithm as FSaCD. We propose to calculate the element importance and $\text{mttkrp}$ column-wise and parallelize the calculation together.

Based on Algorithm 1, we calculate the element importance column-wise so that only the respective column of gradients is needed. It enables us to calculate the $\text{mttkrp}$ of only one column at a time as follows,

For simplicity, let us represent $\text{mttkrp}$ as,

$$M = X_1(W \odot V).$$  \hfill (36)

The $\text{mttkrp}$ operation is simplified using the sparseness of tensor and the column-wise $\text{mttkrp}$ $(M)$ can be calculated as,

$$m_r = \sum_{(q,p,s) \in \Omega^U_q} (x_{qps}(w_r, v_r)).$$  \hfill (37)

where $\Omega^U_q$ indicates a subset of $\Omega$ whose mode $U$’s index is $q$. $v_r$ and $w_r$ indicates the $r^{th}$ column of the factor matrices $V$ and $W$, respectively.

Now, the column-wise gradient to solve $U$ is calculated as:

$$g_r = m_r + (U(V^T V \ast W^T W))_r.$$  \hfill (38)

The column-wise element importance is calculated as:

$$z_r = -(g_r \ast \hat{u}_r) - \frac{L}{2} (\hat{u}_r \ast \hat{u}_r).$$  \hfill (39)

With the calculated gradient (Equation (38)) and element importance (Equation (39)), the column-wise update is performed as:

$$u_r \leftarrow u_r + \max \left(0, \frac{g_r}{h_{rr}} \right),$$  \hfill (40)

where $u_r$ indicates the $r^{th}$ column of the factor matrix $U$ and $h_{rr} = (V^T V \ast W^T W)_{rr}$.

The above process shows the learning of factor matrix $U$. Next we briefly show that the process is applicable to other factor matrices $V$ and $W$ too.

**Updating solution for factor matrix $V$.** The $\text{mttkrp}$ $(M = X_2(W \odot U))$ operation in Equation (32) can be calculated column-wise as:

$$m_r = \sum_{(q,p,s) \in \Omega^V_p} (x_{qps}(u_r, w_r)).$$  \hfill (41)

Now, the column-wise gradient to solve $V$ is computed as:

$$g_r = m_r + (V(U^T U \ast W^T W))_r.$$  \hfill (42)

The column-wise element importance is calculated as:

$$z_r = -(g_r \ast \hat{v}_r) - \frac{L}{2} (\hat{v}_r \ast \hat{v}_r).$$  \hfill (43)

With the calculated gradient (Equation (42)) and element importance (Equation (43)), the column-wise update is performed as:

$$v_r \leftarrow v_r + \max \left(0, \frac{g_r}{h_{rr}} \right),$$  \hfill (44)

where $v_r$ indicates the $r^{th}$ column of the factor matrix $V$ and $h_{rr} = (U^T U \ast W^T W)_{rr}$.
Updating solution for factor matrix $W$. The $\text{mttkrp} (M = X_3(V \odot U))$ operation in Equation (34) can be calculated column-wise as:

$$m_r = \sum_{(q,p,s) \in \Omega^y} (x_{qps} (v_r, u_r)).$$

(45)

Now, the column-wise gradient to solve $W$ is computed as:

$$g_r = m_r + (W(U^T U \ast V^T V))_r.$$

(46)

The column-wise element importance is calculated as:

$$z_r = -(g_r \ast \hat{w}_r) - \frac{L}{2} (\hat{w}_r \ast \hat{w}_r).$$

(47)

With the calculated gradient (Equation (46)) and element importance (Equation (47)), the column-wise update is performed as:

$$w_r \leftarrow w_r + \max \left(0, \frac{g_r}{h_{rr}} \right),$$

(48)

where $w_r$ indicates the $r^{th}$ column of the factor matrix $W$ and $h_{rr} = (U^T U \ast V^T V)_{rr}$.

With the column-wise gradients calculated as per Equations (38), (42), and (46), and the column-wise element importance calculated as per Equations (39), (43), and (47), the factor matrix update is performed in parallel as detailed in the Algorithm 2.

5 THEORETICAL ANALYSIS

We analyze SaCD in terms of convergence, time complexity, and memory requirement. We use the following symbols in the analysis: $R$ (rank), $K$ (maximum number of iterations), $M$ (number of factor matrices), and a three-mode tensor $X \in \mathbb{R}^{(Q \times Q \times Q)}$.

5.1 Convergence Analysis

In this section, we analyze the convergence of SaCD under the following assumptions.

Assumption 1. The objective function $f$ w.r.t each factor matrix $\nabla f(u^k)$ is continuous, differentiable and holds the Lipschitz continuity.

Assumption 2. Each element importance is calculated by Equation (25) for all iterations $K$ and the parameter $L^{(k-1)}$ obeys $l \leq L^{(k-1)} \leq L$.

Lemma 5.1. Based on the Assumptions 1 and 2, for given $K$ iterations, $\sum_{k=1}^{\infty} \|u^k - u^{k+1}\|^2 < \infty$.

Proof. For the element selection as per Equation (28), we have inequality and therefore:

$$f(u^k) - f(u^{k-1}) > L^k \|u^{k-1} - u^k\|^2 - L^{k-1} \|u^{k-2} - u^{k-1}\|^2.$$

(49)

If we sum the above inequality over $k$ from 1 to $K$, we have

$$f(u^1) - f(u^K) \geq \sum_{k=1}^{K} L^k \|u^{k-1} - u^k\|^2 - L^{k-1} \|u^{k-2} - u^{k-1}\|^2.$$

(50)

$$\geq \sum_{k=1}^{K} L^k \|u^{k-1} - u^k\|^2 \geq \sum_{k=1}^{K} l \|u^{k-1} - u^k\|^2.$$

As the function $f$ is lower bounded, for $k = \infty$, the proof satisfies. \qed
**ALGORITHM 2: FSaCD Algorithm**

**Input:** Tensor $X$; Randomly Initialized factor matrices $U \in \mathbb{R}^{Q \times R}$, $V \in \mathbb{R}^{P \times R}$, $W \in \mathbb{R}^{S \times R}$; Rank $R$; $Z = \emptyset$; Number of rows in any given factor matrix $\text{rows}$; Maximum number of iterations $\text{maxiters}$.

**Output:** Learned Factor matrices $U$, $V$, $W$

for $k = 1 : \text{maxiters}$ do

    compute $H$ using Equation (12); $L = \|H\|$

    if $k == 1$ then

        Parallel for $r = 1 : R$ do

            compute gradient $g_r$ using Equation (38);

            compute element importance $z_r$ using Equation (39);

            store the initial element importance, $z_r^{k-1} \leftarrow z_r$;

            if $z_r > 0$ then

                update $u_r$ using Equation (40);

            end

        end

    else

        Parallel for $r = 1 : R$ do

            compute gradient $g_r$ using Equation (38);

            compute element importance $z_r$ using Equation (39);

            for $q = 1 : \text{rows}$ do

                identify the saturation point $s_{qr}$ using Equation (28) or Equation (31);

                if $s_{qr} > 0$ then

                    update the element $u_{qr}$ using Equation (17);

                end

            end

            update the previous element importance, $z_{qr}^{k-1} = z_{qr}$;

        end

    end

repeat analogues lines 4 to 26 with $H$, $g_r$ and $z_r$ calculated using (33), (42) and (43) respectively to update the elements of $V$ in lines 11 and 21;

repeat analogues lines 4 to 26 with $H$, $g_r$ and $z_r$ calculated using (35), (46) and (47) respectively to update the elements of $W$ in lines 11 and 21;

end

The update rule in Equation (22) utilizes the Newton method [28] to apply the updates to the set of important elements that are identified by SaCD.

**Theorem 5.2.** The newton method update rule using SaCD converges faster to the optimal solution by reaching the saturation point.

**Proof.** The Newton method based update uses the update sequence,

$$u^{k+1} = \max \left( 0, u^k - \frac{f'(u^k)}{f''(u^k)} \right), k = 0, 1, \ldots$$

where $k$ indicates the $k^{th}$ iteration. Using SaCD, we select $e_{qr} \in E$ where $E$ is a set of elements to be updated in $U$. The gradients and second order derivatives for the sequence can be defined as:
\[ f'(u) = - \sum_{q,r \in E} g_{qr}^k, \]  
\[ f''(u) = \sum_{q,r} h_{qr}^k > \sum_{q,r} h_{qr}^k > 0. \]  
\[ \text{As per linear differential equations properties, for any positive } f''(u), f'(b) \leq f'(u) + (b - u)f''(u) \forall u, b \geq 0. \text{ As we know that } f''(u) > 0, \text{ and setting } b = u - (f''(u))/(f''(u)), \text{ we have} \]
\[ f'(u) - f'(u^{sat}) \leq 0. \]

With the initialized \( u \), suppose the update sequence that holds these properties will converge to a saturation point \( u^{sat} = \lim_{k \to \infty} u^k \) where \( f'(u^{sat}) \leq 0. \) For the larger value of \( k \) by continuity of given gradients and second-order derivatives,
\[ \frac{f''(u^k)}{f''(u^{sat})} < \frac{u^{sat}}{2f''(u^{sat})}, \]  
\[ u^{sat} - u^k < \frac{u^{sat}}{2f''(u^{sat})}. \]  

From (55) and (56), we have \( u^{sat} - u^{k+1} < 0 \) that literally contradicts \( u^{k+1} \leq u^{sat} \). Hence, it can be said that the proposed algorithm will converge faster to the optimal solution. \( \square \)

### 5.2 Time Complexity

**Lemma 5.3.** The time complexity of SaCD is \( O(|X| + (M + K)(Q^3R + QR^2 + QR + 1)) \)

**Proof.** Algorithm 1 includes five operations: initialization of factor matrices and respective importance matrix \( V \), unfolding of the tensor, gradient calculation, updating of factor matrices, and updating of importance matrix.

The random initialization of \( M \) number of factor matrices and respective importance matrix takes \( O(2MQR) \). Unfolding the tensor generally takes \( O(|X|) \). SaCD requires gradients \( G \) to be calculated before updating the elements that involves the calculation of two terms \(-X_1(W \odot V)\) and \( U(VV^T \ast WW^T)\) as shown in Equations (11) and (13) and requires \( O(Q^3R), O(QR^2) \), respectively. Let \( E \) be the total number of elements selected for update. Updating each factor matrix takes \( O(E) \) and calculating \( Z \) takes additional \( O(E) \), where \( E \leq QR \). For each iteration, the value of \( E \) reduces and at some point, it will reach 0. But if \( Z \) can be kept in memory, it can be updated while updating each entry for \( O(1) \). Thus, the time complexity of SaCD can be formulated as \( O(|X| + (M + K)(Q^3R + QR^2 + QR + 1)) \). \( \square \)

The time complexity of a third-order TF algorithms, including SaCD, remains to be cubical \( (Q^3R) \). However, the element selection in SaCD avoids frequent element updates and controls the complexity as the number of element updates \( E \leq QR \).

### 5.3 Memory Requirement

**Lemma 5.4.** The memory requirement of SaCD is \( O(|X| + MQ(3R + Q)) \).

**Proof.** For the factorization of an input tensor \( X \in \mathbb{R}^{Q \times Q \times Q} \), SaCD stores the following types of data in the memory at each iteration: unfolded tensor; factor matrices and respective importance matrices; and precomputed gradient and second order derivative. The unfolded tensor with respective to any mode requires \( O(|X|) \). The \( M \) number of factor matrices require \( O(MQR) \) while
Table 3. Real-world Dataset Details (M: Million)

| Dataset  | Tensor Size         | Density     |
|----------|---------------------|-------------|
| Delicious | 1797 × 24073 × 15752 | 0.0000003   |
| LastFM   | 1583 × 8383 × 3886  | 0.000002    |
| Movielens| 1129 × 3884 × 3693  | 0.000001    |
| Gowalla  | 1M × 2M × 24        | 0.000001    |

the respective importance matrices require the same amount of memory \(O(MQR)\). The precomputed gradients which is of the same size as the respective factor matrices will require another \(O(MQR)\). The Hadamard matrix which is a square matrix for each factor matrix will require an additional \(O(MQ^2)\) memory. Thus, the memory requirement of SaCD is \(O(|X| + MQ(3R + Q))\).

6 EMPIRICAL ANALYSIS

We would like to validate that SaCD can perform the factorization process efficiently as well as accurately. Experiments were conducted to answer the following questions:

Q1. How scalable is SaCD? What is its runtime performance?
Q2. How accurately can SaCD predict missing values and can be used in recommendation?
Q3. How accurately SaCD identify the unique patterns and can be used in pattern mining?
Q4. What is the impact of parallelization on SaCD?

6.1 Datasets

Several real-world and synthetic datasets were used to evaluate the performance of SaCD in comparison to the state-of-the-art algorithms. Table 3 details the four real-world datasets used. Delicious\(^2\) consists of 1,797 user’s tagging behavior on 24,073 URLs with 15,752 tags. LastFM\(^3\) consists of 1,583 users, 8,383 artists and 3,886 tags associated with the artists. Movielens\(^4\) consists of 1,129, 3,884 and 3,693 user, movies, and tags, respectively. Gowalla\(^5\), the LBSN Foursquare dataset, records the 1 Million users’ check-in activity at 2 Million locations.

6.2 Experimental Setup and Benchmarks

The source codes of SaCD and its parallelized version have been made available\(^6\). All single-core experiments were executed on Intel (R) Core\(^\text{TM}\) i7 – 6600U CPU @ 2.60GHz model with 16GB RAM. The multi-cores experiments in Section 6.4 were executed on Intel (R) Xeon(R) CPU E5 – 2680 v3 @ 2.50GHz model with 12GB RAM and 12 cores. For real-world datasets, we use five-fold cross validation with 80% of data used for training and 20% for testing.

We compare SaCD with the following benchmark algorithms.

(1) APG\(^{[54]}\) uses gradients to accelerate the convergence. The objective function is smoothed using the proximal gradients and the gradient calculation is simplified using the low-rank approximations. Instead of calculating the gradient using the original tensor \(X\), the low-rank approximations of the original tensor \(X\) is used to calculate the gradients.

---

\(^2\)https://del.icio.us/.
\(^3\)https://www.last.fm/.
\(^4\)https://movielens.org/.
\(^5\)http://www.yongliu.org/datasets/.
\(^6\)https://github.com/thirubs/SaCD.
(2) **FMU** [34] and **FHALS** [35] are optimized variations of MU and ALS, respectively. The tensor unfolding and the Kronecker product during the gradient calculations are simplified to minimize the computational cost.

(3) **BCDP** [51] decomposes the non-convex optimization function into multiple blocks of convex problem. And the convex blocks are solved cyclically to update the factor matrices. The convex blocks are smoothed for fast convergence.

(4) **CDTF** [39] is a latest CD algorithm for TF. For better scalability, the factor matrices are updated column-wise alternatively. For a fair comparison, we use the serial version of CDTF for NTF.

(5) **GCD** [5] is a element selection-based CD algorithm which selects important elements and update repeatedly for fast convergence. The elements are selected row-wise using the frequent gradient updates.

### 6.3 Evaluation Criteria

Tensor completion has its special properties, that discriminate it with factorization, such as the effect of the missing values on the rank/regularization selection and the optimization method. The **Root Means Square Error** (RMSE) is a commonly used metric to evaluate the tensor approximation performance. The recommendation quality is evaluated using precision, recall, and F1 score,

\[
RMSE = \sqrt{\frac{\sum (X_{test} - \hat{X}_{test})^2}{n}},
\]

where \(X_{test}\) is the test data, \(\hat{X}_{test}\) is approximated data and \(n\) is the number of elements in test data.

\[
\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}.
\]

\[
\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}.
\]

\[
\text{F1 score} = 2 \left( \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \right).
\]

We propose to use Pattern Distinctiveness (PD) to evaluate the quality of patterns learned using NTF as follows:

\[
PD = \text{Cosine}(w_i, w_r), \forall i, r \in [1, R], i < r,
\]

where **Cosine**\((w_i, w_r)\) indicates the cosine similarity of \(i^{th}\) and \(r^{th}\) column of a factor matrix \(W\).

**PD** measures the similarity of each pattern with other patterns. So higher the **PD** value, higher is the similarity between patterns. Since the objective is to identify unique patterns, lower the **PD** value, better the quality of learned patterns is demonstrated.

### 6.4 Scalability Analysis

We evaluate the scalability of SaCD and other algorithms, with regards to size (mode length), density, and rank of the tensor, using synthetic data of diverse characteristics. We randomly generated tensors of size ranging from \(64 \times 64 \times 64\) to \(16,384 \times 16,384 \times 16,384\), density ranging from 0.001 (dense) to 0.0000001 (sparse) and rank ranging from 10 to 125. Experiments using these synthetic data show the stability of SaCD and all the benchmarking algorithms in different data characteristics.

**Mode length.** We increase the mode length \(Q = P = S\) of each mode from \(2^6\) to \(2^{14}\) with setting the tensor density and rank to 0.00001 and 10, respectively. We set all the algorithms to run for
Fig. 3. Scalability Analysis using synthetic datasets. FMU and FHALS shows similar runtime performance and the lines are overlapped. o.m.: out of memory.

maximum iterations of 30. As shown in Figure 3(a), SaCD successfully handles the tensor of size $Q = P = S = 2^{14}$. Whereas APG [54] ran out of memory for the tensor size $Q = P = S > 2^7$, and FMU [34], FHALS [35], and BCDP [51] ran out of memory for the tensor size $Q = P = S > 2^{13}$. Overall SaCD can factorize $2^6$ to 7 times larger tensors when compared to existing algorithms. The runtime performance of SaCD is almost constant for the smaller size tensors; however, it increases linearly for large size tensors. This is due to the matricized tensor times Khatri–Rao product ($mttkrp$), $X_1(W \odot V)$ needed in Equation (11) for the gradient calculation. In general, SaCD yields a significant time saving in comparison to other algorithms, especially GCD and CDTF, due to the avoidance of frequent gradient updates.

Density. In this set of experiments, we fix the tensor mode length to $Q = P = S = 3,000$ and rank to 10 while decreasing the density from $1.0 \times 10^{-3}$ to $1.0 \times 10^{-7}$. As shown in Figure 3(b), SaCD is 13–70 times faster than existing algorithms for the very sparse dataset. The runtime performance of SaCD improves when the sparsity increases, due to a gradual reduction in the $mttkrp$ operation, as shown in Figure 3(d). On the other hand, the runtime performance of GCD and CDTF degrades with an increase in sparsity. In comparison to GCD (Figure 3(d)), there is a significant time saving due to avoidance of the frequent gradient update.

Rank. Here, we fix the tensor mode length $Q = P = S = 3,000$ and density to $0.00001$ while increasing the rank from 10 to 125. Figure 3(c) reveals SaCD outperforms all other algorithms
easily. As proven in lemma 4.2, SaCD avoids frequent gradients, hence the increase in rank does not adversely affect its performance.

6.5 Tensor Approximation Performance

In addition to scalable factorization process, it is essential that the approximated (i.e., reconstructed) tensor has good accuracy. The factor matrices learned using factorization is used to reconstruct the approximated tensor that will identify missing values. Figure 4 reports the RMSE performance of all the algorithms for the synthetic datasets used in previous section to evaluate the scalability. It is evident that SaCD does not compromise with accuracy for better runtime performance and produces the best result with less error in comparison to benchmarks.

6.6 Recommendation or Prediction Performance

The NTF problem can be considered as a solution to a recommendation or prediction problem where the estimated missing data are treated as prediction. The approximated tensor reconstructed using the factor matrices can be used to infer new values based on associations. In LastFM, Delicious, and Movielens datasets, the goal is to predict the missing entries of the tensor as accurately
Table 4. Pattern Distinctiveness (PD) and Runtime on the Gowalla Dataset (Lower Values are Better)

| Algorithm | PD     | Runtime in secs |
|-----------|--------|-----------------|
| FMU       | 0.69   | 633.95          |
| FHALS     | 0.41   | 623.66          |
| BCDP      | 0.63   | 1076.01         |
| CDTF      | 0.71   | 2537.64         |
| GCD       | 0.34   | 363.27          |
| SaCD      | 0.34   | 363.27          |

as possible. These entries are then inferred as “most likely items” that can be recommended to users.

It is evident from Figure 5 that SaCD does not compromise with accuracy for better runtime performance and produces the best result with less error in comparison to benchmarks. Especially in Movielens dataset, SaCD shows at least 2.5% accuracy improvement over other algorithms. Similar to synthetic datasets, APG and BCDP ran out of memory (o.m) to process the Delicious dataset, and both FMU and FHALS ran out of time (o.o.t) with an increase in the rank. While all other algorithms fail to process the Gowalla dataset for higher rank, only SaCD can successfully complete the process. The materialization of matrices in these algorithms requires large memory making them inefficient to deal with higher ranks. It can be noticed that the higher the rank, higher is the accuracy of SaCD. However, due to the increased memory requirements, APG, BCDP, FMU, and FHALS are not suitable for higher ranks. GCD and CDTF are able to process at higher ranks, however, they are significantly slower and run out of time.

Similar performance is obtained for the measures of precision, recall, and F1 score. It can be seen in Figure 6 that SaCD significantly outperforms other algorithms in the recommendation performance on all four real-world datasets.

6.7 Pattern Mining

In addition to missing value prediction, NTF can also be used to identify the patterns automatically. For LBSN datasets such as Gowalla, we have time as the 3rd mode. By setting the rank of a tensor as 4 in the factorization process, we identified four different patterns in the temporal factor matrix. Table 4 shows the PD values of all algorithms on the Gowalla dataset. SaCD outperforms all the baselines. GCD ran out of time due to the large mode length of the tensor. While FMU and FHALS can execute due to the low rank setting, they are not able to distinguish the patterns distinctly.

In Figure 7, we plot the values of the factor matrix in “temporal mode,” which have 24 hours as x-axis and y-axis represents the normalized value of the factor matrix in each column. Figure 7(e) shows the patterns obtained by SaCD. The red pattern shows a peak at 1 pm and 9 pm that probably indicates the lunch and dinner time. The pink pattern shows a very common 7 am to 10 pm activity. On the other hand, the blue pattern shows a unique pattern with a peak at 1 am, indicating night time activity. The green pattern shows activities between 1 pm and 6 am. With a proper domain knowledge, the kind of activities that happens in different time periods can be easily interpreted by using distinct patterns. It is interesting to note that unlike FMU and CDTF, SaCD avoids simultaneous elimination problem (a state where similar patterns are derived multiple times) [57]. In Figure 7(a), pink and blue patterns are highly similar, and in Figure 7(d), red pattern is same as the green pattern and pink pattern is same as the blue pattern. In comparison, patterns derived from SaCD are highly distinctive.
Fig. 5. Prediction and Runtime performance on real-world datasets. APG does not scale for these datasets while GCD runs out of time for Gowalla dataset. [o.m.: out of memory and o.o.t.: out of time]
Fig. 6. Precision, Recall, and F1 score on real-world datasets.
Fig. 7. Temporal patterns derived from the 3rd mode (time) of the tensor on the Gowalla dataset.

6.8 Parallelization

Figure 3(d) shows the time taken for $mttkrp$ and the factor matrix update without multi-core parallelization. SaCD allows to minimize the time taken for factor matrix update by avoiding the frequent gradient calculation; however, the complexity of $mttkrp$ remains the same as the traditional element selection-based CD algorithm.
Fig. 8. Scalability analysis of FSaCD and other algorithms using synthetic datasets.

If the calculation of $\text{mttkrp}$ is parallelized as in the proposed algorithm FSaCD and executed using the cores of a single machine, the time taken for $\text{mttkrp}$ can be reduced as shown in Figure 8(a). We evaluate FSaCD in terms of mode length, density, and rank of the tensor. We set the rank to 100 and density to 0.00001 to evaluate the performance in terms of mode length. As shown in Figure 8(b), FSaCD shows up to 8 times fast computing comparing to GCD, the next best algorithm. By setting the rank to 100 and mode length to $2^{13}$, we decreased the density to evaluate the performance in terms of density. As shown in Figure 8(c), FSaCD shows up to 234 times fast computing performance in comparison to GCD. It is interesting to note that SaCD and FSaCD show similar performance on sparse datasets showing that parallelization has no effect. To evaluate the performance in terms of rank, we set the density to 0.00001 and the mode length of the tensor to $2^{13}$ and increased the rank from 10 to 150. FSaCD shows 37.5 times improved performance in comparison to CDTF as shown in Figure 8(d). It ascertains that FSaCD can handle higher rank easily. It is also interesting to note that FSaCD shows a converged performance for ranks where the increase in the rank does not increase the runtime.

7 CONCLUSION

In this article, we propose an element selection-based coordinate descent algorithm SaCD that measures the important elements for optimization using Lipschitz continuity and provides the
saturated point for early stopping. The proposed Lipschitz continuity based element importance calculation introduces an additional regularity condition to the optimization process and helps to speed-up the convergence. SaCD can efficiently process NTF with higher mode length, rank, and density by reducing the frequent gradient updates. We also extend SaCD (called FSaCD) for the parallel environment to further improve the performance. We conducted theoretical and empirical studies to demonstrate the efficiency of SaCD. Theoretical analysis shows the time complexity and memory requirement as well as proves the fast convergence property of SaCD. Empirical analysis shows the superiority of SaCD and FSaCD in comparison to the state-of-the-art algorithms, in terms of tensor size (mode length), rank, and density without compromising the accuracy. Results show the applicability of SaCD in recommendation and pattern mining where efficiency is achieved at no cost of accuracy. SaCD and FSaCD require the element importance matrix stored in the memory which makes it challenging to extend in a distributed environment. In future work, we will explore SaCD and FSaCD to effectively handle the datasets in the distributed environment.

ACKNOWLEDGMENT

We like to express our gratitude to Dr. U Kang, Associate Professor, Seoul National University for his comments that had greatly improved the manuscript. His willingness to give his time so generously is very much appreciated.

REFERENCES

[1] Evrim Acar, Tamara G. Kolda, and Daniel M. Dunlavy. 2011. All-at-once optimization for coupled matrix and tensor factorizations. arXiv preprint arXiv:1105.3422 (2011).
[2] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge & Data Engineering 17, 6 (2005), 734–749.
[3] Brett W. Bader and Tamara G. Kolda. 2007. Efficient MATLAB computations with sparse and factored tensors. SIAM Journal on Scientific Computing 30, 1 (2007), 205–231.
[4] Tamara Kolda, Brett W. Bader, Evrim NMN Acar Ataman, Daniel Dunlavy, Robert Bassett, Casey J. Battaglino, Todd Plantenga, Eric Chi, and Samantha Hansen. 2017. MATLAB Tensor Toolbox Version 3.0-dev. Retrieved from https://www.tensortoolbox.org.
[5] Thirunavukarasu Balasubramaniam, Richi Nayak, and Chau Yuen. 2018. Nonnegative coupled matrix tensor factorization for smart city spatiotemporal pattern mining. In Proceedings of the 4th International Conference on Machine Learning, Optimization and Data Science (LOD’18). Springer.
[6] Thirunavukarasu Balasubramaniam, Richi Nayak, and Chau Yuen. 2018. People to people recommendation using coupled nonnegative Boolean matrix factorization. In Proceedings of the International Conference on Soft-Computing and Network Security (ICSNS’18). IEEE.
[7] Thirunavukarasu Balasubramaniam, Richi Nayak, and Chau Yuen. 2019. Sparsity constraint nonnegative tensor factorization for mobility pattern mining. In Proceedings of the Pacific Rim International Conference on Artificial Intelligence. Springer, 582–594.
[8] Alex Beutel, Partha Pratim Talukdar, Abhimanu Kumar, Christos Faloutsos, Evangelos E. Papalexakis, and Eric P. Xing. 2014. Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In Proceedings of the 2014 SIAM International Conference on Data Mining. SIAM, 109–117.
[9] Mohamed Reda Bouadjenek, Hakim Hacid, and Mokrane Bouzeghoub. 2016. Social networks and information retrieval, how are they converging? A survey, a taxonomy and an analysis of social information retrieval approaches and platforms. Information Systems 56 (2016), 1–18. https://www.sciencedirect.com/science/article/abs/pii/S030643791500160X?via%3Dihub.
[10] J. Douglas Carroll and Jih-Jie Chang. 1970. Analysis of individual differences in multidimensional scaling via an N-way generalization of “Eckart-Young” decomposition. Psychometrika 35, 3 (1970), 283–319.
[11] Joon Hee Choi and S. Vishwanathan. 2014. DFacTo: Distributed factorization of tensors. In Proceedings of the Advances in Neural Information Processing Systems. 1296–1304.
[12] Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun-ichi Amari. 2009. Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation. John Wiley & Sons.
[40] Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos. 2017. Tensor decomposition for signal processing and machine learning. IEEE Transactions on Signal Processing 65, 13 (2017), 3551–3582.

[41] Shaden Smith, Niranjan Ravindran, Nicholas D. Sidiropoulos, and George Karypis. 2015. SPLATT: Efficient and parallel sparse tensor-matrix multiplication. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS’15). IEEE, 61–70.

[42] Suvrit Sra, Dongmin Kim, and Bernhard Schölkopf. 2008. Non-monotonic poisson likelihood maximization. Max Planck Institute for Biological Cybernetics, Technical Report, 170.

[43] Lijun Sun and Kay W. Axhausen. 2016. Understanding urban mobility patterns with a probabilistic tensor factorization framework. Transportation Research Part B: Methodological 91, C (2016), 511–524.

[44] Panagiotis Symeonidis. 2016. Matrix and tensor factorization with recommender system applications. Graph-Based Social Media Analysis 39 (2016), 187.

[45] Panagiotis Symeonidis, Antonis Krinis, and Yannis Manolopoulos. 2013. Geosocialrec: Explaining recommendations in location-based social networks. In Proceedings of the East European Conference on Advances in Databases and Information Systems. Springer, 84–97.

[46] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. 2008. Tag recommendations based on tensor dimensionality reduction. In Proceedings of the 2008 ACM Conference on Recommender Systems. ACM, 43–50.

[47] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. 2009. A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis. IEEE Transactions on Knowledge and Data Engineering 22, 2 (2009), 179–192.

[48] Yoshio Takane, Forrest W. Young, and Jan De Leeuw. 1977. Nonmetric individual differences multidimensional scaling: An alternating least squares method with optimal scaling features. Psychometrika 42, 1 (1977), 7–67.

[49] Ledyard R. Tucker. 1966. Some mathematical notes on three-mode factor analysis. Psychometrika 31, 3 (1966), 279–311.

[50] Stephen J. Wright. 2015. Coordinate descent algorithms. Mathematical Programming 151, 1 (2015), 3–34.

[51] Yangyang Xu and Wotao Yin. 2013. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. SIAM Journal on Imaging Sciences 6, 3 (2013), 1758–1789.

[52] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. 2012. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In Proceedings of the 2012 IEEE 12th International Conference on Data Minin. IEEE, 765–774.

[53] Yonghong Yu and Xingguo Chen. 2015. A survey of point-of-interest recommendation in location-based social networks. In Proceedings of the Workshops at the 29th AAAI Conference on Artificial Intelligence. Vol. 130.

[54] Yu Zhang, Guoxu Zhou, Qibin Zhao, Andrzej Cichocki, and Xingyu Wang. 2016. Fast nonnegative tensor factorization based on accelerated proximal gradient and low-rank approximation. Neurocomputing 198, C (2016), 148–154.

[55] Nan Zheng, Qiudan Li, Shengcai Liao, and Leiming Zhang. 2010. Which photo groups should I choose? A comparative study of recommendation algorithms in Flickr. Journal of Information Science 36, 6 (2010), 733–750.

[56] Yu Zheng, Tong Liu, Yilun Wang, Yamin Zhu, Yanchi Liu, and Eric Chang. 2014. Diagnosing New York city’s noises with ubiquitous data. In Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing. ACM, 715–725.

[57] Hui Zou and Ming Yuan. 2008. The F∞-norm support vector machine. Statistica Sinica 18, 1 (2008), 379–398.

Received May 2019; revised November 2019; accepted February 2020