Utilizing Common Substructures to Speedup Tensor Factorization for Mining Dynamic Graphs

Wei Liu, Jeffrey Chan, James Bailey, Christopher Leckie,
and Ramamohanarao Kotagiri
Dept of Computing and Information Systems, The University of Melbourne, Australia
{wei.liu, jeffrey.chan, baileyj, caleckie, kotagiri}@unimelb.edu.au

ABSTRACT

In large and complex graphs of social, chemical/biological, or other relations, frequent substructures are commonly shared by different graphs or by graphs evolving through different time periods. Tensors are natural representations of these complex time-evolving graph data. A factorization of a tensor provides a high-quality low-rank compact basis for each dimension of the tensor, which facilitates the interpretation of frequent substructures of the original graphs. However, the high computational cost of tensor factorization makes it infeasible for conventional tensor factorization methods to handle large graphs that evolve frequently with time.

To address this problem, in this paper we propose a novel iterative tensor factorization (`ITF`) method whose time complexity is linear in the cardinalities of all dimensions of a tensor. This low time complexity means that when using tensors to represent dynamic graphs, the computational cost of `ITF` is linear in the size (number of edges/vertices) of graphs and is also linear in the number of time periods over which the graph evolves. More importantly, an error estimation of `ITF` suggests that its factorization correctness is comparable to that of the standard factorization method. We empirically evaluate our method on publication networks and chemical compound graphs, and demonstrate that `ITF` is an order of magnitude faster than the conventional method and at the same time preserves factorization quality. To the best of our knowledge, this research is the first work that uses important frequent substructures to speed up tensor factorizations for mining dynamic graphs.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining

General Terms

Algorithms

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

Keywords

Tensor factorization; dynamic graphs; scalability.

1. INTRODUCTION

Widely used statistical data mining methods that detect dynamic patterns are usually based on matrix operations, such as SVD [7] and PCA [14], which can only deal with two "dimensions" of the original data (by the "row dimension" and the "column dimension" of matrices). When data sets are multi-dimensional such as dynamic time-evolving networks, tensor-based methods can more effectively and naturally represent all dimensions of the data. However, similar to performing SVD on matrices, the use of tensors raises the problem of tensor factorizations, from which it is possible to identify the principle low-rank factors of a tensor (i.e., the linear combinations of original dimensions that contribute the most to the overall variance of the tensor). These low-rank factors are important since the new feature space formed by these factors preserves the main variance of the original tensor.

To successfully perform tensor factorization on dynamic graphs, a major challenge that needs to be addressed is how to handle scalability. Scalability is an issue not only because tensors are composed of multiple dimensions, but also because the cardinality of each dimension in itself can be very large. For example, if we use a 3-mode tensor, $authors \times journalNames \times publicationTime$, to represent a dynamic publication network, each of the three dimensions can have thousands or more unique values. For another example, when one uses a tensor to represent a collection of chemical compounds (e.g., $atoms \times bonds \times compounds$) where each compound is a graph of atoms [11], the cardinality of the third dimension of this tensor is usually in the tens of thousands. Moreover, with new bioassays constantly being tested over time, the number of compounds that need to be incorporated into the tensor will tend continue to grow larger. This phenomenon makes the computational cost of the standard tensor factorization extremely expensive and infeasible for analyzing time-varying graphs/networks[1].

Given the scalability challenge, a good tensor factorization method should be able to *(1)* factorize large tensors efficiently using much less time than standard methods, while at the same time *(2)* obtain low-rank factors that preserve the main variance of the tensors. To achieve this goal, in this

[1]Due to the generic nature of the iterative tensor factorization technique we propose, we use the two terms "network" and "graph" interchangeably.

paper we propose to reduce the time complexity of tensor factorization by updating the factor matrices in an iterative manner. The rationale behind this design is based on the observation that, in many practical domains, the most important substructures of graphs are commonly shared by all graphs, or by graphs evolving in different time periods. In other words, it is usually not the important substructures, but the actual combinations of these substructures, which are changing with time or changing across different graphs. If one has discovered the most important substructures of a set of graphs at an earlier time, then in many domains the graphs at a later time can generally be well explained by those earlier substructures using an efficient updating procedure. From the perspective of tensor factorization on graphs, if we have found low-rank factors of a subset of a tensor, then after we iteratively update these factors at a very low cost, the rest of the tensor data can also be well represented by the updated factors. The notion of an important substructure is essentially a column vector from a low-rank factor matrix, which comprises a linear combination of the original features (e.g., edges). Note that we do not assume the structural changes of the graphs are smooth – even if the graphs change frequently, the later graphs can still be explained by the earlier factor matrices as long as the later graphs are comprised of important substructures (i.e., rather than being comprised of noisy or random substructures).

In the preceding chemical compound example, although there are a huge number of compounds that can be encoded into a tensor, most of these compounds share many common substructures, such as aromatic rings, hydroxyls and amines. The major difference among chemical compounds, in terms of preserving the main variance of the data, is in what formulations of those substructures comprise each compound. There might be some special substructures that only exist in a few specific compounds, but we note that such substructures would contribute very little in representing the main variance of a data set, and thus would have little influence in the decomposed factor matrices. Similarly for publication networks, the publication patterns of an author at different times would vary in the concrete combinations of journals or conferences in which the author has published. For example, we expect authors in the number theory domain to continue to publish in number theory journals, rather than change their focus to marketing or finance journals. To this end, the combination of number theory journals are the common important factors that can be discovered early on from a subset of a tensor.

Based on these observations, in this paper we propose a novel factorization strategy that first factorizes a subset of a tensor, and then iteratively refines its factor matrices by efficient and effective updating operations on the remaining data in the tensor. Moreover, we generalize the iterative factorization process from one mode to all modes of a tensor, which further reduces its overall time complexity. In summary, the major contributions of this paper are as follows:

1. We propose an iterative tensor factorization method (`ITF`), which significantly reduces the time complexity of standard tensor factorization.

2. We derive an analytical error estimation of the iterative approximation made by `ITF`, and demonstrate that `ITF` can theoretically preserve as much variance as a standard method, when important substructures are frequently shared among different graphs.

3. We apply the general `ITF` algorithm to practical problems of discovering emerging publication trends and classifying chemical compounds, both of which demonstrate that `ITF` is capable of significantly reducing computation time while solving the two practical problems with high accuracy in comparison to the standard factorization method.

The rest of the paper is structured as follows. We review related literature in Section 2. Section 3 defines the tensor factorization problem, and introduces our `ITF` algorithm with an error estimation. We explain how tensor decomposition can be applied to dynamic graph mining in Section 4. Details of the dataset we use and the empirical evaluations are reported in Section 5. We conclude in Section 6 with directions for future work.

2. LITERATURE REVIEW

In this section, we review existing methods that are closely related to this research. A detailed survey on tensor factorization methods can be found in [9].

Many factorization methods have been proposed for tensor analysis based broadly on two approaches: Tucker decomposition [19] and canonical polyadic decomposition (aka PARAFAC/CANDECOMP decomposition) [3], both of which can be considered as higher-order generalizations of matrix SVD and PCA. Kroonenberg *et al.* [12] proposed to use the method of alternating least squares (ALS) in solving Tucker decomposition for three-way arrays, which was then extended and popularized by Kapteyn *et al.* [8] for n-way arrays. Sun *et al.* [16,17] have proposed methods for incremental tensor analysis, which are aimed at solving the decomposition problem when there is a stream of many tensors: in [16] they used sliding windows to track the changes of covariance matrices, based on which their algorithms made decisions on whether to update the projection matrices; and in [17] they introduced the notion of a "forgetting factor" which is added onto older covariance matrices. This forgetting factor controlled the amount of information that could be considered in future updates. However, these methods did not avoid the extremely expensive computational costs of diagonalization/eigen-decomposition on any mode of a tensor, which severely limits the efficiency of their methods. Moreover, they did not provide any analysis of the error estimation of their methods. Kolda *et al.* [10] designed algorithms to improve memory efficiency for sparse tensors, but when tensors are not highly sparse their method requires much more CPU time than normal decompositions. Recently, Schifanella *et al.* [15] proposed to use metadata, which is a priori background knowledge in addition to the original data, to design a domain hierarchy by which they reduce the time of tensor decomposition. However, this method cannot address more general scenarios where such metadata is not available for the original data.

The major difference between our proposed iterative method and the above existing methods is that the method proposed in this paper avoids the full decomposition / diagonalization on the covariance matrix in all modes of a tensor, which is the main cause of the high computational complexities of tensor factorization. Furthermore, we provide an error estimation of our method, which illustrates its suitability for

Table 1: Notation used in this paper

Notation	Definition and Description
a	lowercase normal font represents scalar values
$\mathbf{a}$	boldface lowercase represents vectors
$\mathbf{U}$	boldface uppercase represents matrices
$\mathbf{U}_{i,j}$	the scalar at the $\{i,j\}$ position of $\mathbf{U}$
$\mathcal{X}$, $\mathcal{Y}$, $\mathcal{S}$	calligraphic font represents tensors
$\mathcal{X}_{i,j,k,...}$	the scalar at the $\{i,j,k,...\}$ position of $\mathcal{X}$
$\mathbf{U}^i$	the projection factor matrix on the ith dimension of $\mathcal{X}$
$\mathbb{U}$	the set of projection matrices $\mathbf{U}^i$ on all dimensions i; we also use $\mathbf{U}^d\|_{d=i}^j$ to define the range of i in $\mathbb{U}$
n_i	the cardinality of the ith dimension of the original tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_M}$
r_i	the desired rank of the ith dimension of the original tensor $\mathcal{X}$, which is also the cardinality of the ith dimension of the core tensor $\mathcal{Y}$
$\times_d$	the tensor mode product of a tensor and a matrix on the tensor's dth dimension

mining graph data. A detailed analytical presentation of the proposed method is provided in the next section.

3. TENSOR DECOMPOSITION

The dimensionality of a tensor is also called the tensor's *modes*. For example, matrices are tensors of 2 modes. Table 1 lists the notation we use in the rest of the paper. We denote scalars by letters in lowercase normal font, e.g., a; vectors (tensors of 1 mode) by boldface lowercase letters, e.g., $\mathbf{a}$; matrices (tensors of 2 modes) by boldface uppercase letters, e.g., $\mathbf{U}$; and tensors with 3 or more modes by letters of calligraphic font, e.g., $\mathcal{X}$. A tensor can be represented in general by a multi-dimensional array in $\mathbb{R}^{n_1 \times n_2 \times ... \times n_M}$, where n_i $(1 \le i \le M)$ is the cardinality of the ith mode, and M is the total number of modes. Now we give the following essential definitions to introduce the overall tensor factorization problem.

Definition 1. (**Tensor mode product**): The product of the dth mode of a tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_M}$ and a matrix $\mathbf{U} \in \mathbb{R}^{n_d \times r}$, denoted by $\mathcal{X} \times_d \mathbf{U}$, is a new tensor of size $\mathbb{R}^{n_1 \times ... \times n_{d-1} \times r \times n_{d+1} \times ... \times n_M}$ defined element-wise as:

$$(\mathcal{X} \times_d \boldsymbol{U})_{i_1,...,i_{d-1},r,i_{d+1},...,i_M} = \sum_{i_d=1}^{n_d} \mathcal{X}_{i_1,i_2,...,i_M} \boldsymbol{U}_{i_d,r}.$$

Definition 2. (**Tensor mode unfolding**): The unfolding (i.e., matricization) on the dth mode of a tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_M}$, denoted by unfold$(\mathcal{X}, d)$, leads to a matrix $\mathbf{X} \in \mathbb{R}^{n_d \times \prod_{i \ne d} n_i}$ by keeping the index d fixed and varying the other indices of $\mathcal{X}$.

Tucker Factorization

The overall problem we aim to solve in this paper is to efficiently factorize a tensor in the manner of Tucker factorization [19], which approximates a large tensor by using a small core tensor by changes of basis:

Definition 3. (**Tucker tensor factorization**): Given a M-mode tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_M}$ and desired low ranks on each mode $r_1, r_2, ..., r_M$, factorize $\mathcal{X}$ into a core tensor $\mathcal{Y} \in \mathbb{R}^{r_1 \times r_2 \times ... \times r_M}$ and M projection matrices $\mathbf{U}^d \in \mathbb{R}^{n_d \times r_d} (1 \le d \le M)$, such that the factorization error $\|\mathcal{X} - \mathcal{Y} \times_1 \mathbf{U}^1 \times_2 \mathbf{U}^2 .... \times_M \mathbf{U}^M\|$ is minimized.

In other words, the Tucker tensor factorization is equivalent to solving the following minimization problem:

$$\min f(\mathcal{Y}, \mathbf{U}^d) = \min \|\mathcal{X} - \mathcal{Y} \times_1 \mathbf{U}^1 .... \times_M \mathbf{U}^M\| \quad (1)$$

where $1 \le d \le M$. The core tensor $\mathcal{Y}$ is an approximation of the original tensor $\mathcal{X}$ by changing its basis through projecting all original data into $\mathbf{U}^d$ on each dimension. The function $f(\mathcal{Y}, \mathbf{U}^d)$ can be minimized to zero if $r_d = n_d$ for all modes d. How to solve Equation 1 scalably and effectively is the main challenge we address in this paper. We start from introducing the standard method of solving Equation 1 in the next subsection.

Algorithm 1 HOSVD$(\mathcal{X}, r_1, r_2, ..., r_M)$: the standard method for higher-order singular value decomposition.

Input: the original tensor $\mathcal{X}$, and the desired ranks on each mode r_1, r_2, ..., r_M.
Output: a core tensor $\mathcal{Y}$, and projection factor matrices $\mathbf{U}^d (1 \le d \le M)$.

1: **for** d = 1,2,...,M **do**
2: $\quad \mathbf{X}_d \leftarrow$ unfold($\mathcal{X}$,d) by Definition 2;
3: $\quad$ Compute the covariance matrix of the dth mode unfolded matrix $\mathbf{C}_d = \mathbf{X}_d \mathbf{X}_d^T$;
4: $\quad \mathbf{U}^d \leftarrow$ the first r_d eigenvectors of $\mathbf{C}_d$;
5: **end for**
6: $\mathcal{Y} \leftarrow \mathcal{X} \times_1 (\mathbf{U}^1)^T \times_2 (\mathbf{U}^2)^T ... \times_M (\mathbf{U}^M)^T$;
7: **return** $\mathcal{Y}$ and $\mathbf{U}^d (1 \le d \le M)$.

Algorithm 2 Tucker$(\epsilon, \mathcal{X}, r_1, r_2, ..., r_M)$: the standard alternating least square method for Tucker tensor decomposition.

Input: convergence threshold ϵ, original tensor $\mathcal{X}$, and the desired ranks $r_1, r_2, ..., r_M$.
Output: core tensor $\mathcal{Y}$, and the projection matrices $\mathbf{U}^d (1 \le d \le M)$.

1: Initialize $\mathbf{U}^d (1 \le d \le M)$ by HOSVD$(\mathcal{X}, r_1, r_2, ..., r_M)$;
2: **repeat**
3: $\quad$ **for** d = 1, 2, ..., M **do**
4: $\quad\quad \mathcal{X}' \leftarrow \mathcal{X} \times_i (\mathbf{U}^i|_{i=1\&i\ne d}^M)^T$;
5: $\quad\quad \mathbf{X}'_d \leftarrow$ unfold($\mathcal{X}'$,d);
6: $\quad\quad$ Covariance matrix of current mode $\mathbf{C}'_d \leftarrow \mathbf{X}'_d \mathbf{X}'^T_d$;
7: $\quad\quad \mathbf{U}^d \leftarrow$ the first r_d eigenvectors of $\mathbf{C}'_d$;
8: $\quad$ **end for**
9: $\quad \mathcal{Y} \leftarrow \mathcal{X} \times_d (\mathbf{U}^d|_{d=1}^M)^T$.
10: **until** the increase of $||\mathcal{Y}||$ is less than ϵ.
11: **return** $\mathcal{Y}$ and $\mathbf{U}^d (1 \le d \le M)$.

3.1 Standard Method for Tensor Decomposition

Higher-order singular value decomposition (HOSVD) [4] forms the basis of many tensor decomposition methods developed recently [5, 18]. Details of the HOSVD method are shown in Algorithm 1.

In line 2 of Algorithm 1, the matrix $\mathbf{X}_d$ obtained from unfolding on the dth mode of $\mathcal{X}$ is of size $n_d \times \prod_{i \ne d} n_i$, and hence the formulation of the covariance matrices has com-

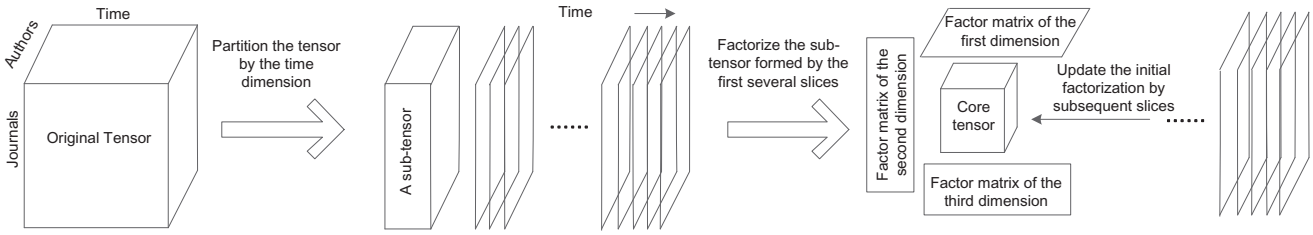


Figure 1: Graphical representation of the main process of the one-mode ITF algorithm. For illustrative purposes, we use the notion of two dimensional "slices" to represent the partitions of the original tensor. However we note that the ITF algorithm is designed to deal with partitions of any number of dimensions.

plexity $O(n_d^2 \prod_{i \neq d} n_i)$. The covariance matrix $\mathbf{C}$ obtained in line 3 is of size $n_d \times n_d$, which means the time complexity for computing the eigenvectors of $\mathbf{C}$ in line 4 is $O(n_d^3)$ [2]. Therefore the overall time complexity for Algorithm 1 is $O(\sum_{d=1}^{M} n_d^3 + \sum_{d=1}^{M} n_d^2 \prod_{i \neq d} n_i)$, which is in cubic time for each mode.

In the standard Tucker tensor factorization, it is common to use alternating least squares (ALS) (shown in Algorithm 2) to repeat the singular vector projection process until the variance of the core tensor $\mathcal{Y}$ stops increasing. Because the size of the projected matrix $\mathbf{X'}_d$ formed in line 5 of Algorithm 2 is of size $n_d \times \prod_{i \neq d} r_i$, the complexity of the covariance matrix formulation in line 6 is $O(n_d^2 \prod_{i \neq d} r_i)$. The body of the *for loop* of the Tucker method is very similar to HOSVD, which makes it easy to show that the overall time complexity of the Tucker method is $O(C(\sum_{d=1}^{M} n_d^3 + \sum_{d=1}^{M} n_d^2 \prod_{i \neq d} r_i))$, where the first term is for the eigen-decomposition and the second is for computing the covariance matrices (the factor C is the number of iterations needed for convergence, which can be non-trivial). The complexity of the standard Tucker method is also in cubic time to each mode of a tensor, which is very expensive.

3.2 ITF Method for One-mode Iterative Factorization

In this subsection, we explain how we solve the factorization problem iteratively in one mode. The solution to the problem in Equation 1 has the following property:

$$\mathcal{X} \approx \mathcal{Y} \times_1 \mathbf{U}^1 .... \times_M \mathbf{U}^M \tag{2}$$

For brevity, we use $\mathbb{U}$ to denote the product of projection factor matrices $\mathbf{U}^d|_{d=1}^M$, and use $\mathcal{Y}\mathbb{U}$ to denote the product of the core tensor $\mathcal{Y}$ with all projection matrices. Then Equation 2 can be rewritten as $\mathcal{X} \approx \mathcal{Y}\mathbb{U}$. From the orthogonality of $\mathbb{U}$ we know that $\mathcal{Y} = \mathcal{Y}\mathbb{U}\mathbb{U}^T \approx \mathcal{X}\mathbb{U}^T$, which means $\mathcal{Y}$ is $\mathcal{X}$ projected on the subspace formed by $\mathbb{U}$. Similarly we have

$$\mathcal{X} \approx \mathcal{Y}\mathbb{U} \approx (\mathcal{X}\mathbb{U}^T)\mathbb{U} \approx \mathcal{X}\mathbb{U}^T\mathbb{U}. \tag{3}$$

Now we derive our one-mode ITF method which refines the core tensor and projection matrices iteratively in one mode. As illustrated in Figure 1, we first factorize a small sub-tensor $\mathcal{S}$ from the original tensor $\mathcal{X}$, and then update the obtained factorization by gradually taking into account the remaining data of $\mathcal{X}$. Counting from the zero index on the dth mode of $\mathcal{X}$, we extract a sub-tensor whose dth mode is the same size as the desired core tensor and whose other modes are the same sizes as those of $\mathcal{X}$, so we have a sub-tensor $\mathcal{S} \in \mathbb{R}^{n_1 \times n_2 \times ... n_{d-1} \times r_d \times n_{d+1} ... \times n_M}$.

Without loss of generality, we assume it is the *1st* mode of $\mathcal{X}$ where a sub-tensor $\mathcal{S}$ is extracted, so $\mathcal{S} \in \mathbb{R}^{r_1 \times n_2 ... \times n_M}$. We denote the formulation of $\mathcal{S}$ by $\mathcal{S} \equiv \mathcal{X}_{1:r_1}$ where we use the subscripts to represent the size of $\mathcal{S}$ on the *1st* mode and omit the fixed sizes n_d $(2 \leq d \leq M)$ on other modes. By partitioning the first mode of $\mathcal{X}$ and extracting the sub-tensor, the original tensor can be represented as:

$$\mathcal{X} = \{\mathcal{X}_{1:r_1},\ \mathcal{X}_{r_1+1},\ \mathcal{X}_{r_1+2},\ ...,\ \mathcal{X}_{n_1}\},$$

where the size of $\mathcal{X}_{r_1+i}$ $(1 \leq i \leq n_1 - r_1)$ is $1 \times n_2 \times n_3 ... \times n_M$, which we call a *slice* of $\mathcal{X}$.

Suppose we have obtained the factorization of the sub-tensor: $\mathcal{S} \equiv \mathcal{X}_{1:r_1} \approx \tilde{\mathcal{Y}} \times_1 \tilde{\mathbf{U}}^1 \times_2 \tilde{\mathbf{U}}^2 ... \times_M \tilde{\mathbf{U}}^M$, where $\tilde{\mathcal{Y}}$ is of the same size as the desired core tensor, $\tilde{\mathbf{U}}^1$ is of size $r_1 \times r_1$, and $\tilde{\mathbf{U}}^i|_{i=2}^M$ is of size $n_i \times r_i$. Then from Equation 3 we know $\mathcal{X}_{1:r_1} \approx \mathcal{X}_{1:r_1}\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}$, where $\tilde{\mathbb{U}}$ denotes $\tilde{\mathbf{U}}^i|_{i=1}^M$. Since the next *slice* after $\mathcal{X}_{1:r_1}$ (e.g., $\mathcal{X}_{r_1+1}$) was not included in the decomposition of $\mathcal{S}$, it is possible that this next *slice* cannot be recovered from the projection in the subspace formed by $\tilde{\mathbb{U}}$. So we introduce the notion of "*departure*", denoted by Δ, with which the projection recovery in Equation 3 can hold true for $\mathcal{X}_{r_1+1}$:

$$\mathcal{X}_{r_1+1} = \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T\tilde{\mathbb{U}} + \Delta. \tag{4}$$

We now explain the rationale of using the departure Δ from a graph mining perspective. By using a tensor to represent dynamic graphs, the columns of $\tilde{\mathbb{U}}$ (i.e., the singular vectors) are necessarily the major substructures (linear combinations of original features) of previous graphs. The information contained in Δ denotes new formulations of substructures from the current graph that cannot be represented by previously discovered substructures in $\tilde{\mathbb{U}}$. Intuitively, when important representative substructures are shared among current and previous graphs, the value of Δ would always be small.

After factorizing the sub-tensor, we form a new tensor $\{\tilde{\mathcal{Y}}, \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T + \Delta\tilde{\mathbb{U}}^T\}$, where the second term of this new tensor comprises the component of $\mathcal{X}_{r_1+1}$ that can be represented in the subspace formed by $\tilde{\mathbb{U}}$, and also the "*departure*" of $\mathcal{X}_{r_1+1}$ (i.e., Δ) that cannot be recovered from the subspace projection. We factorize this new tensor into

$$\{\tilde{\mathcal{Y}}, \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T + \Delta\tilde{\mathbb{U}}^T\} = \mathcal{Y}' \times_1 \mathbf{U}^{1'} \times_2 \mathbf{U}^{2'} ... \times_M \mathbf{U}^{M'} \tag{5}$$

where $\mathbf{U}^{1'}$ is of size $(r_1 + 1) \times r_1$ and $\mathbf{U}^{i'}|_{i=2}^M$ are all of size $r_i \times r_i$. Then the new updated projection matrices for $\{\mathcal{X}_{1:r_1}, \mathcal{X}_{r_1+1}\}$ are $\mathbb{U}'\tilde{\mathbb{U}}$, where $\mathbb{U}'$ denotes the set of projection matrices $\mathbf{U}^{i'}|_{i=1}^M$.

Algorithm 3 `ITF_OneMode`$(\mathcal{X}, r_1, r_2, ..., r_M, l)$: iteratively decomposing one mode of a tensor by using the *1st* mode as an illustration.

Input: original tensor $\mathcal{X}$, the desired ranks r_1, r_2, ..., r_M, and a mode index l on which the tensor is to iteratively decomposed (we use l=1 as an example to explain this algorithm in the paper's texts).
Output: core tensor $\mathcal{Y}$, and the set of all projection matrices $\mathbb{U}$.

1: Form a sub-tensor $\mathcal{X}_{1:r_l}$ from the first r_1 slices of $\mathcal{X}$;
2: $\{\tilde{\mathcal{Y}}, \tilde{\mathbb{U}}\} \leftarrow$ `HOSVD`$(\mathcal{X}_{1:r_l}, r_1, r_2, ..., r_M)$;
3: $\mathcal{Y}_{old} \leftarrow \tilde{\mathcal{Y}}$;
4: $\mathbb{U}_{old} \leftarrow \tilde{\mathbb{U}}$;
5: **for** i=1, 2, ..., $n_1 - r_1$ **do**
6: Obtain the "departure" $\Delta \leftarrow \mathcal{X}_{r_1+i}\mathbb{U}^T\mathbb{U} - \mathcal{X}_{r_1+i}$;
7: Form a tensor $\mathcal{X}_{new} \leftarrow \{\mathcal{X}_{1:r_l}, \mathcal{X}_{r_1+i}\mathbb{U}_{old} + \Delta\mathbb{U}_{old}\}$
8: $\{\mathcal{Y}_{new}, \mathbb{U}_{new}\} \leftarrow$ `HOSVD`$(\mathcal{X}_{new}, r_1, r_2, ..., r_M)$;
9: $\mathbb{U} = \mathbb{U}_{new}\mathbb{U}_{old}$;
10: **end for**
11: $\mathcal{Y} \leftarrow \mathcal{X}\mathbb{U}^T$;
12: **return** $\mathcal{Y}$ and $\mathbb{U}$.

We then take $\{\mathcal{X}_{1:r_1}, \mathcal{X}_{r_1+1}\}$ as a new sub-tensor that replaces $\mathcal{S}$, and treat $\mathcal{X}_{r_1+2}$ as the new *next slice* from the original $\mathcal{X}$. This iterative process is repeated until we reach the final *slice* of $\mathcal{X}$ (i.e., $\mathcal{X}_{n_1}$). We call such an iterative updating procedure the one-mode `ITF` method.

The complete procedure of `ITF_OneMode` is presented in Algorithm 3, where lines 2 and 8 reuse HOSVD as a subroutine. Line 1 has the time complexity of $O(r_1^3 + \sum_{d=2}^{M} n_d^3 + r_1^2 \prod_{i\neq 1} n_i + \sum_{d=2}^{M} n_d^2 \prod_{i\neq d} n_i)$, line 8 has the complexity of $O((r_1+1)^3 + \sum_{d=2}^{M} r_d^3 + (r_1+1)^2 \prod_{i\neq 1} r_i + \sum_{d=2}^{M} r_d^2 \prod_{i\neq d} r_i)$, so the overall time complexity of `ITF_OneMode` is $O(r_1^3 + \sum_{d=2}^{M} n_d^3 + r_1^2 \prod_{i\neq 1} n_i + \sum_{d=2}^{M} n_d^2 \prod_{i\neq d} n_i + (n_1 - r_1)\big[(r_1+1)^3 + \sum_{d=2}^{M} r_d^3 + (r_1+1)^2 \prod_{i\neq 1} r_i + \sum_{d=2}^{M} r_d^2 \prod_{i\neq d} r_i\big])$, where among the five terms the first four terms are for decomposing the first sub-tensor, and the last term is for updating all successive $n_1 - r_1$ slices of the original tensor. Note that n_1 only appears as a linear coefficient in the complexity expression, and that is why it is possible to design a graph mining technique based on `ITF` that is linear in the size of the graphs. Moreover, as we explain in the next subsection, this computational complexity can be further significantly reduced by applying `ITF` onto all modes of a tensor.

Compared to the complexity of standard decomposition methods, the reduced complexity of `ITF_OneMode` arises from computations on the magnitude of r_1 rather than n_1 in both the covariance matrices formulation and their eigen-decompositions. We note that the standard Tucker method becomes a special case of `ITF_OneMode` when $r_1 = n_1$.

3.3 ITF Method for All-mode Factorization

By applying the above iterative updating process to all modes of a tensor, we obtain the final `ITF` method shown in Algorithm 4. Starting from the first mode (line 2 to 4), this algorithm constructs and factorizes a sub-tensor extracted from all modes of the original tensor (line 21) before proceeding to the iterative refinement of factor matrices. We use a recursion to formulate sub-tensors, so that after one mode completes its updating procedure, the algorithm recursively returns the decomposition of that mode to other modes, until it reaches the last mode. This makes Algorithm

Algorithm 4 `ITF`$(\mathcal{X}, r_1, r_2, ..., r_M, l)$: a recursive algorithm for iteratively decomposing all modes of a tensor.

Input: original tensor $\mathcal{X}$, the desired ranks r_1, r_2, ..., r_M, and a mode index l (for recursion purposes).
Output: core tensor $\mathcal{Y}$, and the set of all projection matrices $\mathbb{U}$.

1: // l is uninitialized in the beginning of the algorithm.
2: **if** mode index l is uninitialized **then**
3: Initialize $l \leftarrow 1$;
4: **end if**
5: **if** l has not reached the last mode of $\mathcal{X}$ **then**
6: // A recursion for applying `ITF` to all modes.
7: $l \leftarrow l + 1$;
8: $\{\tilde{\mathcal{Y}}, \tilde{\mathbb{U}}\} \leftarrow$ `ITF`$(\mathcal{X}, r_1, r_2, ..., r_M, l)$;
9: $\mathcal{Y}_{old} \leftarrow \tilde{\mathcal{Y}}$;
10: $\mathbb{U}_{old} \leftarrow \tilde{\mathbb{U}}$;
11: // Iteratively update factor matrices on the lth mode
12: **for** i=1, 2, ..., $n_l - r_l$ **do**
13: Obtain the "departure" $\Delta \leftarrow \mathcal{X}_{r_l+i} - \mathcal{X}_{r_l+i}\mathbb{U}^T\mathbb{U}$;
14: Form a tensor $\mathcal{X}_{new} \leftarrow \{\mathcal{X}_{1:r_l}, \mathcal{X}_{r_l+i}\mathbb{U}_{old} + \Delta\mathbb{U}_{old}\}$
15: $\{\mathcal{Y}_{new}, \mathbb{U}_{new}\} \leftarrow$ `HOSVD`$(\mathcal{X}_{new}, r_1, r_2, ..., r_M)$;
16: $\mathbb{U} = \mathbb{U}_{new}\mathbb{U}_{old}$;
17: **end for**
18: $\mathcal{Y} \leftarrow \mathcal{X}\mathbb{U}^T$;
19: **else**
20: // Factorize a sub-tensor extracted from all modes
21: $\{\tilde{\mathcal{Y}}, \tilde{\mathbb{U}}\} \leftarrow$ `HOSVD`$(\mathcal{X}_{1:r_1,1:r_2,1:r_3,...,1:r_l}, r_1, r_2, ..., r_M)$;
22: $\mathcal{Y}_{old} \leftarrow \tilde{\mathcal{Y}}$;
23: $\mathbb{U}_{old} \leftarrow \tilde{\mathbb{U}}$;
24: // Iteratively update factor matrices on the last mode
25: **for** i=1, 2, ..., $n_l - r_l$ **do**
26: Obtain the "departure" $\Delta \leftarrow \mathcal{X}_{r_l+i} - \mathcal{X}_{r_l+i}\mathbb{U}^T\mathbb{U}$;
27: $\mathcal{X}_{new} \leftarrow \{\mathcal{X}_{1:r_1,1:r_2,1:r_3,...,1:r_l}, \mathcal{X}_{r_l+i}\mathbb{U}_{old} + \Delta\mathbb{U}_{old}\}$
28: $\{\mathcal{Y}_{new}, \mathbb{U}_{new}\} \leftarrow$ `HOSVD`$(\mathcal{X}_{new}, r_1, r_2, ..., r_M)$;
29: $\mathbb{U} = \mathbb{U}_{new}\mathbb{U}_{old}$;
30: **end for**
31: $\mathcal{Y} \leftarrow \mathcal{X}\mathbb{U}^T$;
32: **end if**
33: **return** $\mathcal{Y}$ and $\mathbb{U}$.

4 different to Algorithm 3 in that any tensor being factorized in Algorithm 4 is of size $r_1 \times r_2 \times r_3 \times ... \times r_M$, in contrast to that of $r_1 \times n_2 \times n_3 \times ... \times n_M$ in Algorithm 3.

For Algorithm 4, the complexity of line 21 (for decomposing the first sub-tensor) is $O(\sum_{d=1}^{M} r_d^3 + \sum_{d=1}^{M} r_d^2 \prod_{i\neq d} r_i)$, and those of line 15 and line 28 on mode i (for decomposing a new sub-tensor containing a successive slice) are $O((r_i+1)^3 + \sum_{d\neq i} r_d^3 + (r_i+1)^2 \prod_{d\neq i} r_d + \sum_{d\neq i} r_d^2 \prod_{i\neq d} r_i)$. On each mode i, there exist $n_i - r_i$ slices to be updated, so the overall complexity of Algorithm 4 is $O(\sum_{d=1}^{M} r_d^3 + \sum_{d=1}^{M} r_d^2 \prod_{i\neq d} r_i + \sum_{i=1}^{M} (n_i - r_i)\big[(r_i+1)^3 + \sum_{d\neq i} r_d^3 + (r_i+1)^2 \prod_{d\neq i} r_d + \sum_{d\neq i} r_d^2 \prod_{i\neq d} r_i\big])$. This complexity expression explains that using *one* slice to update the decomposition is cheaper than using *multiple* slices at a time. It also shows that the complexity of `ITF` is *linear in all modes* of a tensor, with all $n_i (1 \leq d \leq M)$ being linear coefficients only. We can observe that the reduction in time complexity of `ITF` compared to that of the standard Tucker method results from the avoidance of computations on the magnitude of n_i, but on r_i, for all modes.

Similar to the standard method, we apply the ALS method to `ITF` by replacing line 1 and line 7 of Algorithm 2 with our

iterative decompositions. We omit the details of ALS for ITF due to page limits.

3.4 Error Estimation of ITF

Our ITF method uses the notion of departure ("Δ") in the tensor approximations. By deriving the following theorem, we show that in ITF the update of slice $\mathcal{X}_{r_1+1}$ on the factorization of sub-tensor $\mathcal{X}_{1:r_1}$ is an effective approximation of a standard direct decomposition on $\{\mathcal{X}_{1:r_1}, \mathcal{X}_{r_1+1}\}$.

THEOREM 1. *Given the factorization of all previously observed slices $\mathcal{X}_{1:r_1+1} = \tilde{\mathcal{Y}}\tilde{\mathbb{U}}$ and a new data slice $\mathcal{X}_{r_1+1}$, denote the information loss of projecting $\mathcal{X}_{r_1+1}$ onto $\tilde{\mathbb{U}}$ by $\Delta = \mathcal{X}_{r_1+1} - \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}$, then the approximation error of using ITF compared to using the standard method is no higher than $||\Delta - \Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}||_F$, where $||\cdot||_F$ represents the Frobenius norm.*

PROOF. From Equation 5 we have:

$$\{\mathcal{Y}, \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T + \Delta\tilde{\mathbb{U}}^T\} = \mathcal{Y}'\mathbb{U}'.$$

By multiplying both sides of the above equation by $\tilde{\mathbb{U}}$, we obtain:

$$\begin{aligned} & \{\mathcal{Y},\ \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T + \Delta\tilde{\mathbb{U}}^T\}\tilde{\mathbb{U}} &=& (\mathcal{Y}'\mathbb{U}')\tilde{\mathbb{U}} \\ \Rightarrow\quad & \{\mathcal{Y}\tilde{\mathbb{U}},\ \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T\tilde{\mathbb{U}} + \Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}\} &=& (\mathcal{Y}'\mathbb{U}')\tilde{\mathbb{U}} \\ \Rightarrow\quad & \{\mathcal{X}_{1:r_1},\ \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T\tilde{\mathbb{U}} + \Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}\} &=& (\mathcal{Y}'\mathbb{U}')\tilde{\mathbb{U}} \\ \Rightarrow\quad & \{\mathcal{X}_{1:r_1},\ \mathcal{X}_{r_1+1} - \Delta + \Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}\} &=& (\mathcal{Y}'\mathbb{U}')\tilde{\mathbb{U}} \end{aligned} \quad (6)$$

The tensor $\{\mathcal{X}_{1:r_1},\ \mathcal{X}_{r_1+1} - \Delta + \Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}\}$ is in the size of $(r_1+1)\times n_2\times n_3\times ...\times n_M$ and the slice $\{\mathcal{X}_{r_1+1} - \Delta + \Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}\}$ is in $n_2 \times n_3 \times ... \times n_M$.

By contrast, a direct decomposition on the new tensor $\{\mathcal{X}_{1:r_1}, \mathcal{X}_{r_1+1}\}$ is

$$\{\mathcal{X}_{1:r_1}, \mathcal{X}_{r_1+1}\} = \hat{\mathcal{Y}}\hat{\mathbb{U}}. \quad (7)$$

Then by incorporating the definition of the Frobenius norm[2], the approximation error of using ITF compared to using the standard method is

$$\begin{aligned} & || \hat{\mathcal{Y}}\hat{\mathbb{U}} - (\mathcal{Y}'\mathbb{U}')\tilde{\mathbb{U}} ||_F \\ &= || \{\mathcal{X}_{1:r_1}, \mathcal{X}_{r_1+1}\} - \{\mathcal{Y},\ \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T + \Delta\tilde{\mathbb{U}}^T\}\tilde{\mathbb{U}} ||_F \\ &= || \{\mathcal{X}_{1:r_1}, \mathcal{X}_{r_1+1}\} - \{\mathcal{Y}\tilde{\mathbb{U}},\ \mathcal{X}_{r_1+1}\tilde{\mathbb{U}}^T\tilde{\mathbb{U}} + \Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}\} ||_F \\ &= || \{\mathcal{X}_{1:r_1}, \mathcal{X}_{r_1+1}\} - \{\mathcal{X}_{1:r_1},\ \mathcal{X}_{r_1+1} + (\Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}} - \Delta)\} ||_F \\ &= \sqrt{\sum_{i=1}^{\min\{r_1+1, n_2, n_3, ..., n_M\}} \sigma_i} \\ &\le \sqrt{\sum_{i=1}^{\min\{n_2, n_3, ..., n_M\}} \sigma'_i} \\ &= ||\Delta - \Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}}||_F \end{aligned}$$

where σ and σ' denote the singular values of tensor $\{\mathcal{X}_{1:r_1}, \mathcal{X}_{r_1+1}\} - \{\mathcal{X}_{1:r_1},\ \mathcal{X}_{r_1+1} + (\Delta\tilde{\mathbb{U}}^T\tilde{\mathbb{U}} - \Delta)\}$ and $\{\Delta - \Delta\tilde{\mathbb{U}}^T\}$ respectively. □

[2]Given a tensor $\mathcal{A}$ and its smallest cardinality $min^{\mathcal{A}}$, the Frobenius norm can be derived as $||\mathcal{A}||_F = \sqrt{tr(\mathcal{A}\mathcal{A}^*)} = \sqrt{\sum_{i=1}^{min^{\mathcal{A}}} \sigma_i^{\mathcal{A}}}$, where $tr()$ is the trace function, $\sigma_i^{\mathcal{A}}$ are $\mathcal{A}$'s singular values, and $\mathcal{A}^*$ is the conjugate transpose of $\mathcal{A}$.

This theorem demonstrates that, compared to the standard method, the approximation error of ITF in the worst case is upper bounded by a linear expression of the departure Δ and the projection matrices $\tilde{\mathbb{U}}$, where the columns of $\tilde{\mathbb{U}}$ indicate previously discovered major frequent substructures. This result can be used to estimate the effects of the iterative updating operations on graph mining. As discussed in Section 3.2, when important frequent substructures are commonly shared across different graphs, the value of the departure Δ would typically be small, which suggests low factorization errors of ITF. This expectation is further confirmed by the empirical results in Section 5.

4. APPLYING ITF TO GRAPH MINING

In the previous section we introduced different ways of decomposing tensors. Now we explain how we use these tensor decomposition methods to *(1)* weight emerging edges (word pairs) in publication networks, and *(2)* classify graphs.

4.1 Weighting Emerging Edges

By using the example of publication networks, we define an edge in a graph as a pair of key words extracted from paper titles. We treat the mode of word pairs in a tensor as a feature of the network, and use tensor factorization to perform feature selection (i.e., dimensionality reduction) on this mode. Given the original data tensor $\mathcal{X}$, both the standard Tucker factorization (Algorithm 2) and our ITF method (Algorithm 4) produce two outputs, the core tensor $\mathcal{Y}$, and the projection matrices $\mathbf{U}^d$ $(1 \le d \le M)$. Moreover, the matrices in all dimensions of $\mathcal{Y}$ are arranged in decreasing order of the magnitudes of their variances (i.e., the ordered singular values for the case of matrix decomposition), which is an indication of the significance of the corresponding column vector in $\mathbf{U}^d$. In this regard, to analyze the original tensor $\mathcal{X}$, we only need to keep a small number of column vectors in each projection matrix $\mathbf{U}^d$, the variances of whose corresponding matrices in $\mathcal{Y}$ can preserve the major variances of $\mathcal{X}$. Thus, suppose word pairs are in the first mode of the tensors, in order to reduce the number of features in the word pair mode (assume it is the first mode of $\mathcal{X}$), we only have to look at the r_1 columns contained in $\mathbf{U}^1$. The r_1 columns of $\mathbf{U}^1$ are linear combinations of the original features, so if we take the Frobenius norm of each of the n_1 rows of $\mathbf{U}^1$, these norms can be used as indications of the significance (weights) of the original n_1 features. To this end, all original n_1 word pairs have the obtained weights associated with them. These weights suggest the significance of word pairs in each time period, which we use to weight emerging research topics.

4.2 Classifying Graphs

We apply ITF to classification problems by utilizing the new data points that lie in the new low-dimension feature space formed by factor matrices. Given a set of graphs, each of which is associated with a class, the task is to predict the class of a new graph. For many type of graphs, one can generalize their adjacency matrices and extend them to a tensor of 5 modes: *vertices* × *vertices* × *verticesLabels* × *edgeLabels(or weights)* × *observations(graph instances)*. One can think of the 5-mode tensors as standard *features* × *observations* data sheets where the *features* are represented by 4-mode tensors. Suppose it is the first vertex dimension on which we want to obtain low-dimension new features, then

the new feature spaces are defined by the columns of factor matrix $\mathbf{U}^1$. Given the original data points $D_{original}$, we obtain new data points D_{new} which lie in the new spaces by projecting the original data onto the first factor matrix: $D_{new} = D_{original}\mathbf{U}_1$. When D_{new} is of multi-mode, we unfold D_{new} (c.f. Definition 2) on its instance dimension so that it can be learned by standard classifiers. These new data points D_{new} produced by `ITF` are what we use in the classification process in comparison with data points produced by the standard Tucker method.

5. EXPERIMENTS AND ANALYSIS

The objectives of our experiments are to *(1)* evaluate the computation time costs and preservation of variances of the `ITF` algorithms for different settings of the desired ranks, and to *(2)* test the practical usefulness of the variance preserved by `ITF` in terms of weighting emerging edges and classifying graphs. We implement[3] `ITF` by using the Matlab Tensor Toolbox [1]. We use the DBLP bibliography [13] to obtain the publication networks, and use chemical compound data to perform graph classification. The convergence threshold ϵ of ALS (c.f. Algorithm 2) is set to 10^{-4} for both the standard Tucker and the `ITF` method. Experiments are conducted on a PC with a 3.0GHz CPU and 4GB memory.

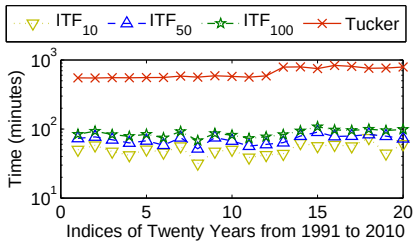
5.1 Weighting Publication Networks

From the DBLP repository, we select journals and conferences whose full book/proceeding titles contain the phrases "data mining" or "knowledge discovery" over a 20 year period from 1991 to 2010. Four properties (i.e., dimensions) are extracted: *(i)* pairs of words that appeared in paper titles; *(ii)* (co-)authors; *(iii)* booktitles (names of journals and conferences/workshops proceedings); *(iv)* time of publication. To avoid using trivial features, we only include word pairs that appeared at least 5 times and authors who have at least 5 publication records in DBLP through the 20 years. After removing stop words, we obtain 11917 word pairs (with 1897 unique words), 6399 authors and 171 booktitles, which form 20 tensors (one for each year) in the space of $\mathbb{R}^{11917\times6399\times171}$. For tracking the changes of patterns over time, we calculate the difference of all consecutive two years of tensors and use the resulting "difference tensors" as the input of the decomposition algorithms.

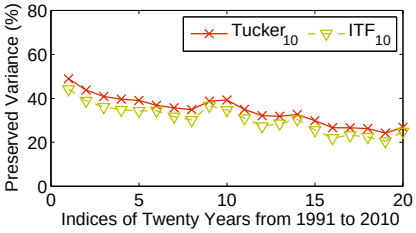
In the settings of the following experiments, we evaluate `ITF` on the first dimension (i.e., word pairs) of the tensors, and set the desired number of singular vectors of other dimensions (r_2 and r_3) consistently to 50. We note that the focus of this paper is not to determine the best number of singular vectors in approximating the original tensors, but to investigate the improvements of `ITF` over the standard Tucker method on a consistent number of singular vectors. Hence we do not evaluate the effects of different r_2 and r_3 values, but study the performance of different factorization methods using constant values of r_2 and r_3.

5.1.1 Factorizing Tensors

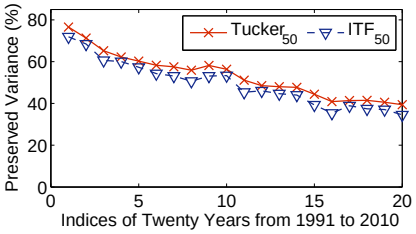
Comparisons between the Tucker method and `ITF` method in factorizing the 20 years of tensors are shown in Figure 2. Figure 2(a) demonstrates that compared to the standard Tucker method, `ITF` uses significantly less time to factorize the same tensor on the same number of desired singular vectors. This advantage of `ITF` is more apparent when r_1 is small – for example when $r_1 = 10$, `ITF` is able to finish the decomposition process in less than 50 minutes for all the 20 years. By contrast, the standard method whose time complexity is independent of the value of r_1 takes a much longer time to finish (e.g., about 10 times longer than `ITF`). More-



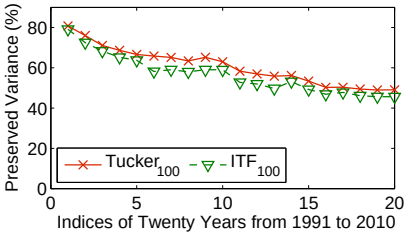
(a) Computation time in log-scale.



(b) Preserved variance when $r_1 = 10$.



(c) Preserved variance when $r_1 = 50$.



(d) Preserved variance when $r_1 = 100$.

Figure 2: Comparisons between Tucker and `ITF` on computation time and preserved variance when factorizing tensors of DBLP data. We can observe that `ITF` is able to preserve almost the same amount of variance compared to the standard method while using significantly less computation time. The notations $\text{Tucker}_{\{10,50,100\}}$ and $\texttt{ITF}_{\{10,50,100\}}$ in the sub-figures' legends represent the standard method and the `ITF` method for $r_1 = \{10, 50, 100\}$. Note that the time complexity of standard method is a function of n_1 (besides other dimensions) and is independent of r_1.

[3]We provide our implementation of `ITF` at `http://people.eng.unimelb.edu.au/liuw/ITF.html`.

Table 2: Comparisons of ITF and Tucker on paper weight assignments. The weight of a paper is the sum of the weights of word pairs that can be matched with the paper's title, where word pairs are the edges of emerging subgraphs weighted by the ITF or the Tucker method. In this table, columns 2 to 5 show examples of paper weight formulations, while in the last column the p-values are obtained from paired t-tests of two groups of weights on 50 papers in each year: one group contains weights assigned by ITF, and the other group contains weights assigned by the Tucker method.

Year	Paper title examples	Matched edges (word pairs of paper titles) from subgraphs weighted by ITF	ITF weights	Tucker weights	t-tests from top 50 papers
1996	A density-based algorithm for discovering clusters in large spatial databases with noise.	algorithm&density (0.5571), cluster&database (0.6496), large&spatial (0.9252), algorithm&database (0.1757), algorithm&noise (0.6544), density&spatial (0.7119), algorithm&spatial (0.1785), algorithm&cluster (0.2078)	3.9664	3.9015	0.7365
1997	New algorithms for fast discovery of association rules.	algorithm&discovery (0.974), algorithm&association (0.97), fast&rule (0.7516), algorithm&rule (0.9681), fast&discovery (0.1134)	3.7771	3.6983	0.8744
1998	An efficient approach to clustering in large multimedia databases with noise.	clustering&noise (0.8097), clustering&database (0.2281), efficient&multimedia (0.1027), clustering&large (0.1495), efficient&large (0.7172), clustering&time (0.1464), efficient&time (0.1713), clustering&database (0.9997), large&time (0.1282), approach&large (0.0937), large&multimedia (0.9548)	4.5013	4.3550	0.6893
1999	Fast and effective text mining using linear-time document clustering.	fast&linear (0.1022), effective&time (0.6865), effective&text (0.7618), mining&text (0.0545), document&time (0.7505), fast&mining (0.4674), clustering&document (0.7231), fast&text (0.7005)	4.2465	4.4001	0.8541
2000	FreeSpan: frequent pattern-projected sequential pattern mining.	mining&pattern (0.9605), mining&project (1.0), mining&sequential (0.7944), frequent&sequential (0.6906), frequent&pattern (0.746)	4.1915	4.0825	0.7976
2001	Co-clustering documents and words using bipartite spectral graph partitioning.	graph&partition (0.6302), clustering&word (0.8961), clustering&graph (0.6363), bipartite&graph (0.978)	3.1407	3.2672	0.9368
2002	Privacy preserving association rule mining in vertically partitioned data.	data&rule (0.6316), association&rule (0.9663), data&mining (0.4072), privacy&rule (0.6591), privacy&vertically (0.5433), preserving&vertically (0.6749)	3.8824	4.1041	0.5831
2003	Mining distance-based outliers in near linear time with randomization and a simple pruning rule.	mining&pruning (0.834), distance&outlier (0.8845), mining&outlier (0.775), base&rule (1.0), mining&rule (0.4337)	3.9273	3.7618	0.6498
2004	Systematic data selection to mine concept-drifting data streams.	data&selection (0.4054), data&mine (0.5996), data&stream (0.9991), concept&drift (0.8912), concept&stream (0.1294), selection&system (0.9889)	4.0256	4.1218	0.8519
2005	Discovering evolutionary theme patterns from text: an exploration of temporal text mining.	discovering&temporal (0.352), discovering&mining (0.5347), mining&text (0.7071), discovering&pattern (0.7155), pattern&text (0.5704), pattern&temporal (0.746), evolutionary&mining (0.7072)	4.3329	4.5085	0.7176

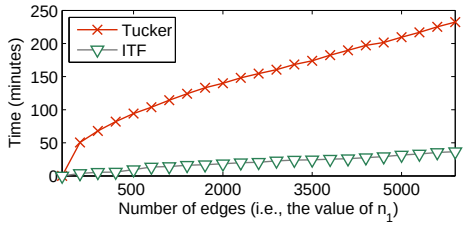


Figure 3: Computation time of the standard Tucker method and the ITF method on a controlled number of graph edges when r_1 is fixed at 100.

over, the variances preserved by ITF are highly comparable to the standard decomposition method. The values shown in Figures 2(b), 2(c) and 2(d) illustrate that the core tensors obtained from ITF can represent almost as much variance as those obtained from the standard method.

We are also interested in the computation time of the standard and ITF methods when the size of the tensor grows. This scenario is closely related to the dynamic graph mining problem in an online setting, since the number of edges (e.g., word pairs) in practical networks and graphs are usually very likely to increase with time. We control the number of word pairs in the original graphs (which is also the cardinality of the first dimension of the original tensor), and vary it from 500 to more than 5000. The computation time needed in the decompositions by the standard Tucker method and the ITF method is shown in Figure 3. The trends in the figure demonstrate that the computational cost of ITF is not only linear to the value of n_1 but also much lower than that of the standard decomposition method. Moreover, it also can be observed that the larger the size of the graph (i.e., the value of n_1), the larger the difference between the computation time needed for ITF and for the standard Tucker method.

5.1.2 *Weighting Emerging Topics*

In Section 4 we explained how the outputs of tensor factorization can be applied in weighting the original word pairs, and in this experiment we use that theory to evaluate the correctness of ITF against the Tucker method. After we decomposed twenty years of tensors, from each year we obtained a list of weights for the original word pairs. To discard trivial word pairs, we only consider those whose weights are higher than the threshold 0.0001 (note that the column vectors in the projection matrices are *unit vectors*, which means the weight of a word pair is typically between 0 and 1). After deleting those word pairs that do not pass this threshold, we obtain subgraphs of the original graphs. To find how popular a research topic is, we weight paper titles by using the edges of these subgraphs. We calculate the sum

of the weights of the word pairs that a paper title contains as the weight of that paper title. Thus, paper titles that have higher weights are the ones whose words occupy more variance in the original tensor.

To test whether the weights of paper titles assigned by ITF are similar to those from the standard Tucker method, in each year we select 50 papers that are the highest weighted by ITF, and compare them with the weights produced by the Tucker method. Since the weights suggest the variance in their corresponding tensors, we conduct these comparisons to check whether the weights found by ITF agree with those found by the Tucker method on the same set of word pairs. We apply paired t-tests on the two groups of weights, one from ITF and the other from the Tucker method, under the null hypothesis that the weights are not significantly different between the two groups. The last column[4] of Table 2 shows the p-values returned from the t-tests, which are mostly very large. This does not reject the null hypothesis, suggesting that ITF and Tucker assign insignificantly different weights to the same set of word pairs. This result validates the correctness of the factorization produced by ITF, in that the main variance of key word pairs preserved by ITF is as good as that preserved by the computationally expensive Tucker method.

5.2 Classifying Chemical Compounds

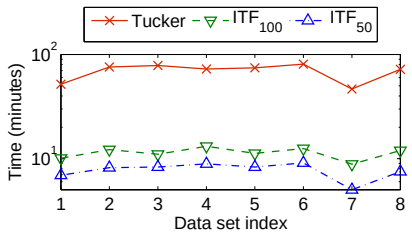
In this experiment, we compare the IFT and the Tucker method on graph classification problems, where we classify the points that are projected into the row-rank feature spaces defined by the factor matrices. We use chemical compound data sets from anti-cancer bioassays[5], where each compound is treated as a graph of atoms. The task is to predict whether a compound is positive or negative in anti-cancer activities. Although there are limited types of atoms and bonds in the data, the total number of compounds is very large (Table 3).

Table 3: Statistics of chemical compound data sets. "pos%" represents the proportion of compounds that are positive.

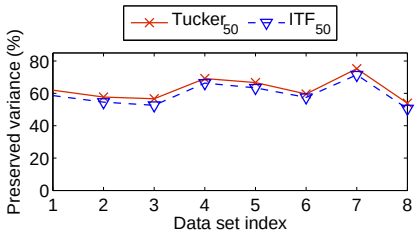
Index	#graphs	pos%	Description
1	27784	8.2	Breast Cancer
2	40700	5.9	Colon Cancer
3	40152	7.8	Leukemia
4	40460	5.1	Lung Cancer
5	40209	4.1	Melanoma
6	40691	5.1	Ovarian Cancer
7	27585	5.7	Prostate Cancer
8	40164	4.9	Renal Cancer

5.2.1 Factorizing Tensors

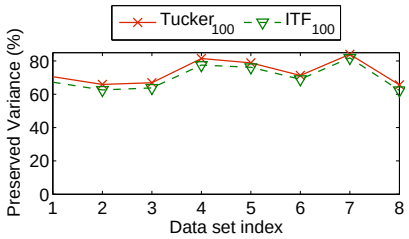
Similar to the graph formulations discussed in Section 4.2, for each chemical compound data set we construct a tensor of 5 modes, where the types of atoms in a compound are converted to labels of vertices, and the lengths of bonds between atoms are treated as weights of edges. Figure 4 shows comparisons of the ITF and the Tucker method in their factorization time and preserved variance of each data set. It is easy to observe that ITF factorizes each data set by using much less time than the Tucker method, while preserving similar amounts of variance. Such results indicate that ITF is able to capture the most important substructures that are shared across different compounds.



(a) Log-scale computation time.



(b) Preserved variance when $r_1 = 50$.



(c) Preserved variance when $r_1 = 100$.

Figure 4: Comparisons between the Tucker and the ITF methods on factoring tensors of chemical compound data sets.

5.2.2 Classification on Compounds

In addition to the evaluation of preserved variances, we test the correctness of the factor matrices produced by ITF by examining the new data points that lie in the new spaces defined by the factor matrices. A linear logistic regression is used to learn and classify these data points by 5-fold cross validation with 10 repeated runs. The results of classification on different factorization methods is reported in Table 4. Note that due to the class imbalance, we use the area under the ROC curve (AUC) as the evaluation metric, which is more appropriate than using overall accuracy as the metric.

While there are various ways to compare classifiers across multiple data sets, we adopt the strategy proposed in [6] which evaluates classifiers by performing Friedman tests. We compare the ITF and the Tucker methods on the same ranks ($r_1 = 50$ and $r_1 = 100$), where p–values that are lower than 0.05 reject the hypothesis with 95% confidence that the classifiers in the comparison are not statistically different. As we can see from the high p-values in the bottom of table Table 4, the data points projected onto subspaces formed by ITF are *insignificantly* different to those by the

[4]Due to space limit, in this table we only present comparisons in nine years time from 1996 to 2005.

[5]http://pubchem.ncbi.nlm.nih.gov

Table 4: Comparisons of ITF and Tucker on classifying the data points located in their corresponding subspaces, using linear logistic regression. "*F. test*" represents the Friedman significance test.

Data	AUC from logistic regression			
	Tucker_{50}	ITF_{50}	Tucker_{100}	ITF_{100}
Breast	.583±.09	.593±.08	.740±.05	.722±.05
Colon	.670±.06	.678±.04	.754±.06	.713±.02
Leukemia	.602±.04	.607±.06	.699±.07	.683±.04
Lung	.608±.05	.645±.06	.652±.03	.633±.05
Melanoma	.687±.06	.703±.05	.693±.06	.710±.04
Ovarian	.699±.04	.711±.04	.740±.04	.731±.02
Prostate	.641±.04	.635±.03	.686±.04	.689±.05
Renal	.568±.02	.589±.04	.661±.06	.653±.08
F. test	0.2573		0.5236	

Tucker method from the perspective of distinguishing class labels. This suggests that the factor matrices generated by ITF capture as much intrinsic variance as those by the Tucker method in both class labels, and that the new features (e.g., columns of a factor matrix) from ITF are competent representatives of the row-rank basis of the compound data sets. Moreover, the setting of cross validation also indicates that ITF is insensitive to the orderings of graphs.

6. CONCLUSIONS AND FUTURE WORK

The main focus of this paper is on scalably factorizing tensors that represent dynamic graphs. A good tensor factorization method should be able to decompose tensors in a computationally cheap manner and also be able to preserve the main variance of the data. We have shown that the proposed ITF method is able to significantly reduce computation time compared to the standard factorization method, while the variance preserved by ITF is comparably effective. The superiority of ITF arises from the fact that important substructures are usually shared among different graphs, or graphs across different time periods. We have applied ITF to the weighting of emerging research topics in dynamic publication networks/graphs, and have shown that the time complexity of ITF is linear in the size of the networks or graphs, which makes ITF feasible to be applied in an online setting. We have also applied ITF to graph classification problems using chemical compound data sets, showing that ITF is not only a significant speed up of the Tucker method but also captures the intrinsic variance of the original data with respect to different class labels. Moreover, the results from cross validations on graph classification experiments demonstrate that the iterative algorithm ITF is insensitive to the orderings of graphs.

The advantage of ITF is not limited to Tucker decomposition. Other tensor factorization methods such as canonical polyadic decomposition and non-negative tensor factorization can also benefit from the iterative procedure of ITF. These are the future research directions of ITF we are planning to investigate.

Acknowledgements

This research was supported under Australian Research Council's Discovery Projects funding scheme (DP110102621).

7. REFERENCES

[1] B. Bader and T. Kolda. Efficient Matlab computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, 2007.

[2] J. Bunch and J. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, pages 231–236, 1974.

[3] J. Carroll and J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[4] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[5] L. De Lathauwer and D. Nion. Decompositions of a higher-order tensor in block terms. *SIAM J. Matrix Analysis and Applications*, 30(3):1067–1083, 2008.

[6] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[7] G. Golub and C. Van Loan. *Matrix computations*, volume 3. Johns Hopkins Univ Pr, 1996.

[8] A. Kapteyn, H. Neudecker, and T. Wansbeek. An approach to n-mode components analysis. *Psychometrika*, 51(2):269–275, 1986.

[9] T. Kolda and B. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455, 2009.

[10] T. Kolda and J. Sun. Scalable tensor decompositions for multi-aspect data mining. In *Proc of ICDM*, pages 363–372, 2008.

[11] X. Kong and P. Yu. Semi-supervised feature selection for graph classification. In *Proc of KDD*, pages 793–802, 2010.

[12] P. Kroonenberg and J. De Leeuw. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 45(1):69–97, 1980.

[13] M. Ley. The DBLP computer science bibliography: evolution, research issues, perspectives. In *String Processing and Information Retrieval*, pages 481–486. Springer, 2002.

[14] K. Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. *ACM SIGMOD Record*, 27(2):166–176, 1998.

[15] C. Schifanella, K. Candan, and M. Sapino. Fast metadata-driven multiresolution tensor decomposition. In *Proc of CIKM*, pages 1275–1284, 2011.

[16] J. Sun, S. Papadimitriou, and P. Yu. Window-based tensor analysis on high-dimensional and multi-aspect streams. In *Proc of ICDM*, pages 1076–1080, 2006.

[17] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proc of KDD*, pages 374–383, 2006.

[18] J. Sun, D. Tao, S. Papadimitriou, P. Yu, and C. Faloutsos. Incremental tensor analysis: theory and applications. *ACM Transactions on Knowledge Discovery from Data*, 2(3):11, 2008.

[19] L. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.