Stream-based Community Discovery via Relational Hypergraph Factorization on Evolving Networks

Christian Bockermann and **Felix Jungermann**
Technical University of Dortmund,
Artificial Intelligence Group
Baroper Strasse 301, Dortmund, Germany
{bockermann,jungermann}@ls8.cs.tu-dortmund.de

Abstract

The discovery of communities or interrelations in social networks has become an important area of research. The increasing amount of information available in these networks and its decreasing life-time poses tight constraints on the information processing – storage of the data is often prohibited due to its sheer volume.

In this paper we adapt a flexible approach for community discovery offering the integration of new information into the model. The continuous integration is combined with a time-based weighting of the data allowing for disposing obsolete information from the model building process.

We demonstrate the usefulness of our approach by applying it on the popular *Twitter* network. The proposed solution can be directly fed with streaming data from *Twitter*, providing an up-to-date community model.

1 Introduction

Social networks like *Twitter* or *Facebook* have recently gained a lot of interest in data analysis. A social network basically consists of various types of entities – such as *users*, *keywords* or *resources* – which are in some way related to one another. A central question is often the discovery of groups of individuals within such networks - the finding of *communities*. Thus, we are seeking for a clustering of the set of entities into subsets where the individuals within each subset are most similar to each other and are most dissimilar to the entities of all other subsets. The similarity of entities is provided by their relations to one another.

The relations between different entities are implied by the communication taking place within the network. Users exchange messages, which contain references to other users, are tagged with keywords or link to external resources by means of URLs. Figure 1 shows a message from the *Twitter* platform, which implies relations between the user `yarapavan`, the URL `http://j.mp/fpga-mr` and the tag `#ML`.

A natural perception of a social network is that of a connected graph, which models each entity as a node and contains (weighted) edges between related entities. Such a graph can be easily described by its adjacency matrix: with d being the number of entities in our social network, we will end up with a (sparse) matrix $\boldsymbol{A}$ of size d^2, where $\boldsymbol{A}_{i,j} = w$ if entity i is related to j with weight w and 0 otherwise. However this representation is not well-suited for n-ary relations.

Figure 1: Example tweets of the *Twitter* platform

A well-established representation of multi-dimensional relations is given by tensors [1; 2; 5; 17; 12; 6; 19]. A tensor is a multi-way array and can be seen as a generalization of a matrix. Tensors have been successfully used in multi-dimensional analysis and recently gained attention in social network mining [1; 2; 5]. In the case of social networks, tensors can be used to describe n-ary relations by using one tensor for each type of relations. Ternary relations of type (*user*,*tag*,*url*) can then be described by a mode-3 tensor $\boldsymbol{\mathcal{X}}$ with

$$\boldsymbol{\mathcal{X}}_{i,j,k} = \begin{cases} w & \text{if user } i\text{, tag } j \text{ and url } k \text{ are related} \\ 0 & \text{otherwise.} \end{cases}$$

More complex n-ary relations will be reflected in tensors of mode-n.

Tensor based Community Discovery

Community discovery in such tensor representations is mapped to a decomposition of the tensors into a product of matrices $\boldsymbol{U}^{(i)} \in \mathbb{R}^{m_i \times k}$ which approximates the tensor

$$\boldsymbol{\mathcal{X}} \approx [z] \prod_i \times_{d_i} \boldsymbol{U}^{(i)}.$$

Each of the matrices $\boldsymbol{U}^{(i)}$ in turn reflects a mapping of entities to clusters $\{1, \ldots, k\}$. The $[z]$ factor is a super-diagonal tensor which serves as a "glue element" – see Section 3 for details. A variety of different decomposition techniques such as Tucker3 or PARAFAC (CP) has been previously proposed [3; 14; 7]. Approximation is commonly measured by some divergence function. In [5] the authors proposed a clustering framework based on tensor decompositions which has been generalized for Bregman divergences. In [4] Bader et al. used CP tensor decomposition to detect and track informative discussions from the Enron email dataset by working on the ternary relation (*term*,*author*,*time*). These approaches have been applied to decompose single tensors. In [16] the authors introduced METAFAC, which is a factorization of a *set of tensors* with shared factors ($\boldsymbol{U}^{(q)}$ matrices). This allows for the discovery of one global clustering based on multiple tensor descriptions of the data. The time complexity for these tensor

decompositions is generally given by the number of non-zero elements of the tensors (provided that a sparse representation is used).

Stream-based Community Discovery
The majority of the tensor decomposition methods so far is based on a static data set. To incorporate streaming data, the stream is broken down into blocks and the decompositions are re-computed for each of the new blocks [16]. A common way to handle time is to introduce a trade-off factor of the old data and the data contained in the new blocks.

In [18] Sun et al. presented dynamic tensor analysis. They handle n-ary relations by tensor decomposition using stream-based approximations of correlation matrices. They also presented a stream-based approach which is not really comparable to ours. They are processing a tensor containing data by unfolding the tensors to every single mode and after that they are handling every column of the resulting matrices in a stream to update their model. In reality, we cannot assume such an original tensor to be given. In contrast to [18], we consider multiple relations which have to be updated at each iteration instead of just one.

Contributions

The critical bottle-neck within the tensor decomposition methods often is their runtime. As of [16], the runtime for a decomposition of a set of tensors can be bound by $O(N)$, where N is the number of entries in all tensors. However, this number can be rather large – we extracted about 590k entries (relations) from 200k messages of the *Twitter* platform.

In this work, we present an adaption of the METAFAC framework proposed in [16]. Our contributions are as follows:

1. We integrate a sampling strategy into the METAFAC framework. Effectively we limit the maximum size of the tensors – and therefore N – and use a least-recently-used approach to replace old entities if the limit of an entity type exceeds.
2. We introduce a time-based weighting for relations contained within the tensors. These weights will decrease over time, reflecting the decreasing importance of links within the social networks.
3. We present an adaption of the METAFAC factorization which allows for a *continuous integration* of new relations into the factorization model. Instead of running the optimization in a per-block mode, we provide a way to simultaneously optimize the model while new data arrives.
4. Finally, we provide an evaluation of our proposed adaptions on real-world data.

The rest of this paper is structured as follows: Section 2 formalizes the problem and presents the METAFAC approach on which this work is based. Following that, we give an overview of tensor decomposition in Section 3 and provide the basics for the multilinear algebra terminology required. In Section 4 we introduce our stream-based adaption of the METAFAC algorithm. We evaluated our streaming approach on real world data (Section 5) and present our findings in Section 6.

2 Multi-Relational Graphs

As denoted above, a social network generally consists of a set of related entities. In general, we are given sets $V_1, \ldots, V_k$ of entities of different types, such as *users*, *keywords* or *urls*. Let V_i be the i-th type of entities, e.g. V_1 corresponds to *users*, V_2 refers to *keywords* and so on. A *relation* then is a tuple of entities, e.g. a *user-keyword* relation (u_1, k_1) is an element of $V_1 \times V_2$. We also refer to $R := V_1 \times V_2$ as the *relation type* R of the relation (u_1, k_1).

The entities are given as strings, and we define a mapping φ_i for each entity type V_i, which maps entities to integers

$$\varphi_i : V_i \to \{0, \ldots, |V_i| - 1\}.$$

The mapping φ_i can be some arbitrary bijective function. For some $w \in V_i$ we refer to $\varphi_i(w)$ as the index of w. We denote the string of an entity given by its index j by $\varphi^{-1}(j)$. This allows us to identify each entity by its index and enables us to describe a set of relations between entities by a tensor.

A tensor $\boldsymbol{\mathcal{X}}$ is a generalization of a matrix and can be seen as a higher-order matrix. A mode-k tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_k}$ is a schema with k dimensions where

$$\boldsymbol{\mathcal{X}}_{i_1,\ldots,i_k} \in \mathbb{R}, i_j \in I_j$$

denotes the entry at position $(i_1, \ldots, i_k)$. For $k = 2$ this directly corresponds to a simple matrix whereas $k = 3$ is a cube.

With the mappings φ_i of entities and the tensor schema, a set of relations $X \subseteq V_{i_1} \times \ldots \times V_{i_{l(i)}}$ can be defined as a mode-k tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{|V_{i_1}| \cdots |V_{i_{l(i)}}|}$ with

$$\boldsymbol{\mathcal{X}}_{\nu_1,\ldots,\nu_{l(i)}} = \begin{cases} 1 & \text{if } (\varphi_1^{-1}(\nu_1), \ldots, \varphi_{l(i)}^{-1}(\nu_{l(i)})) \in X \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\varphi_i^{-1}(\nu_i)$ denotes the mapping φ_i that corresponds to the i-th relation type, and $V_{i_1} \times \ldots \times V_{i_{l(i)}}$ are the indexes of the entity types used in the relation i.

2.1 MetaGraph

Following the above approach for $k = 2$, we would be considering only binary relations, which correspond to edges in the graph representation of the social network. Thus the adjacency matrix for such a graph would be resembled within a collection of mode-2 tensors.

MetaGraph introduced by [16] is a relational hypergraph representing multi-dimensional data in a network of entities. A MetaGraph is defined as a graph $G = (V, E)$, where each vertex corresponds to a set of entities of the same type and each edge is defined as a *hyper-edge* connecting two or more vertices. By the use of hyper-edges, the MetaGraph captures multi-dimensional relations of the social network and therefore provides a framework to model n-ary relations.

Given the notion of relation types defined above, each relation type $R_i = V_{i_1} \times \ldots \times V_{i_{l(i)}}$ corresponds to a hyper-edge in the MetaGraph G. Each relation type $R_i = (v_{i_1}, \ldots, v_{i_l})$ observed within the social network is reflected in a hyper-edge of the MetaGraph. Given a fixed set of relation types $R_1, \ldots, R_n$, we can model the occurrence of relations of type R_i by defining a Tensor $\boldsymbol{\mathcal{X}}^{(i)}$ for each R_i as described in (1).

This approach results in a description of the social network by means of different relational aspects $R_1, \ldots, R_n$. Each type R_i of relations for which a tensor is defined, reflects a subset of all the relations of the network. Capturing the complete set of relations among all entities would obviously result in $|\mathcal{P}(V)| = 2^{|V|}$ different tensors.

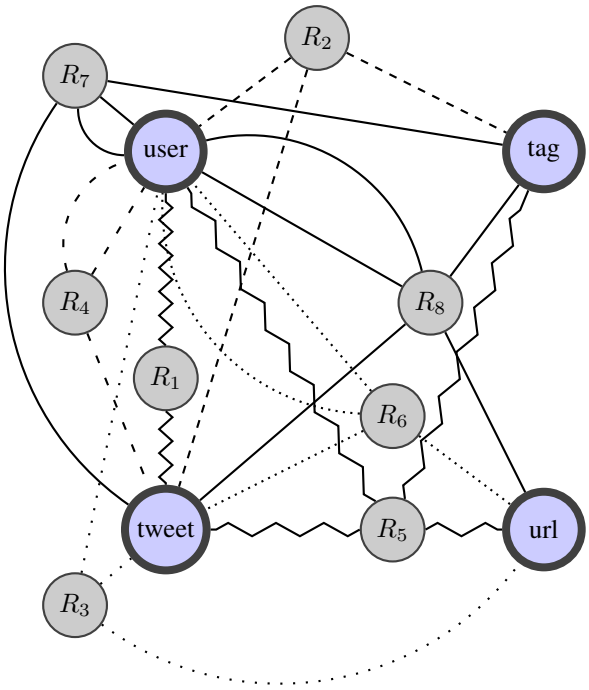

Figure 2: MetaGraph for *Twitter*

Figure 2 visualizes the metagraph we used for modeling possible relations in the microblogging framework *Twitter*. As an example, the relation R_8 referring to (*user*,*tweet*,*tag*,*url*) is represented by a hyper-edge connecting four vertices.

2.2 Community Discovery Problem

With the use of tensors we have an approximated description of our social networks by means of a set of relation types $R_1, \ldots, R_n$. Thus we can describe our network graph G by means of the data tensors which are defined according to the observed relations $R_1, \ldots, R_n$ in G, i.e.

$$G \mapsto \left\{ \boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(n)} \right\}.$$

Based on this description we seek for a further partitioning of the tensor representation into clusters of entities.

The solution proposed in [16] is a factorization of the tensors $\boldsymbol{\mathcal{X}}^{(i)}$ into products of matrices $\boldsymbol{U}^{(q)}$ which share a global factor $[z]$ and some of the $\boldsymbol{U}^{(q)}$ matrices. Let $\boldsymbol{\mathcal{X}}^{(i)}$ be the tensor describing $V_{i_1} \times \ldots \times V_{i_{l(i)}}$, then we can factorize this as

$$\boldsymbol{\mathcal{X}}^{(i)} \approx [z] \prod_{j=1}^{l(i)} \times_j \boldsymbol{U}^{(i_j)}. \tag{2}$$

Within this factorization, the $[z]$ factor is a super-diagonal tensor containing non-zero values only at positions $(i, i, \ldots, i)$. The $\boldsymbol{U}^{(q)}$ are $\mathbb{R}^{|V_q| \times k}$ matrices, where $|V_q|$ is the number of entities of the q-th entity type and k is the number of communities we are looking for. For tensors which relate to relation types with overlapping entity types (e.g. (*user*,*keyword*,*tag*) and (*user*,*keyword*,*url*)) the corresponding factorizations share the related $\boldsymbol{U}^{(q)}$ matrices (e.g. $\boldsymbol{U}^{user}$ and $\boldsymbol{U}^{keyword}$). The $\times_j$ is the mode-j product of a tensor with a matrix.

With an appropriate normalization as used in [16], the $\boldsymbol{U}^{(q)}$ matrices only contain values of $[0, 1]$ which can be interpreted as probability values. Based on this, the value of $\boldsymbol{U}^{(q)}_{l,m}$ can be seen as the probability of entity $\varphi_q^{-1}(l)$ belonging to cluster $m \in \{1, \ldots, k\}$ and we can simply map an entity to its cluster $C(l)$ by

$$C(l) = \arg\max_m \boldsymbol{U}^{(q)}_{l,m}. \tag{3}$$

Thus, the community discovery is mapped onto the simultaneous factorization of a set of tensors. The objective is to find a factorized representation, which resembles the original data tensors $\{\boldsymbol{\mathcal{X}}^{(i)}\}$ as closely as possible. Given some distance measure $D : \mathbb{R}^{I_1 \times \ldots \times I_l} \times \mathbb{R}^{I_1 \times \ldots \times I_l} \to \mathbb{R}$ this leads to the following optimization problem:

$$\arg\min_{[z], \{\boldsymbol{U}^{(q)}\}} \sum_{i=1}^{n} D(\boldsymbol{\mathcal{X}}^{(i)}, [z] \prod_{j=1}^{l(i)} \times_j \boldsymbol{U}^{(i_j)}) \tag{4}$$

2.3 Batch Processing of Evolving Tensors

To capture the evolving behavior of the data tensors Lin et.al. proposed a batch processing approach. The stream is processed as disjoint sliding windows. Let t denote the current window and denote by $\mathbf{z}_{t-1}, \{\boldsymbol{U}^{(q)}_{t-1}\}$ the model obtained so far. Then the time-based optimization problem of [16] yielding the new model $\mathbf{z}_t, \{\boldsymbol{U}^{(q)}_t\}$ is given as

$$\arg\min_{\boldsymbol{z}_t, \{\boldsymbol{U}^{(q)}_t\}} (1-\alpha) \sum_{i=1}^{n} D(\boldsymbol{\mathcal{X}}^{(i)} [\boldsymbol{z}_t] \prod_{j=1}^{l(i)} \times_j \boldsymbol{U}^{(i)}_t) + \alpha L_{prior} \tag{5}$$

$$L_{prior} = D([\boldsymbol{z}_{t-1}][\boldsymbol{z}_t]) + \sum_{i=1}^{n} D(\boldsymbol{U}^{(i)}_{t-1} \boldsymbol{U}^{(i)}_t). \tag{6}$$

The L_{prior} reflects the divergence between the new and the old model, whereas the α specifies a trade-off between the models.

3 Tensors & Tensor Factorizations

The previous sections presented tensors as a mathematical structure to model multi-dimensional relations and motivated their use to describe multi-relational data such as community networks. In this section we will introduce tensors and their factorizations in more detail.

3.1 Tensor Decomposition

Tensors can be decomposed into a sum of component rank-one tensors [11]. A popular method for factorizing a tensor into a product of matrices is the PARAFAC decomposition (*CP*-decomposition) by [9]. Using CP, a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ can be decomposed to

$$\boldsymbol{\mathcal{X}} = \left[\sum_{r=1}^{R} \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r \right] + \boldsymbol{\mathcal{E}} \tag{7}$$

where R is a positive integer and $\boldsymbol{a}_r \in \mathbb{R}^I$, $\boldsymbol{b}_r \in \mathbb{R}^J$, $\boldsymbol{c}_r \in \mathbb{R}^K$. See Figure 3 for a schema of this decomposition. This decomposition is an approximation of $\boldsymbol{\mathcal{X}}$ by an error of $\boldsymbol{\mathcal{E}}$.

Neglecting this error tensor $\boldsymbol{\mathcal{E}}$ we are left with the approximation of $\boldsymbol{\mathcal{X}}$ by

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r. \tag{8}$$

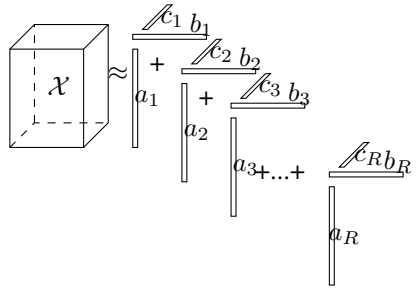

Figure 3: CP tensor decomposition

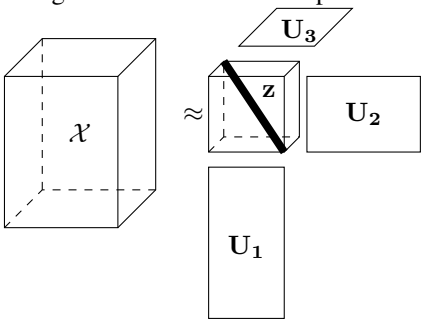

Figure 4: *CP* tensor decomposition incorporating weights

By breaking equation (8) further down into its elementwise form, we get

$$x_{ijk} \approx \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr} \tag{9}$$

Since we later refer to the elements of this sum as probability values, we need to normalize the rank-one tensors $\boldsymbol{a}_r$, $\boldsymbol{b}_r$ and $\boldsymbol{c}_r$ for $r = 1, ..., R$ to length one. With this normalization, we are left with the following form

$$x_{ijk} \approx \sum_{r=1}^{R} z_r a_{ir} b_{jr} c_{kr} \tag{10}$$

where $\boldsymbol{z} \in \mathbb{R}^R$ is a weight-vector. This decomposition sometimes is called higher-order singular value decomposition (HOSVD) ([10]). The rank-one tensors $\boldsymbol{a}_r$, $\boldsymbol{b}_r$ and $\boldsymbol{c}_r$ for $r = 1, ..., R$ represent the singular values and can be used to derive a clustering of the data.

$\boldsymbol{a}_r$, $\boldsymbol{b}_r$ and $\boldsymbol{c}_r$ for $r = 1, ..., R$ can be combined in matrices $\boldsymbol{U}^1 \in \mathbb{R}^{I \times R}$, $\boldsymbol{U}^2 \in \mathbb{R}^{J \times R}$ and $\boldsymbol{U}^3 \in \mathbb{R}^{K \times R}$. We construct a super-diagonal tensor $[\boldsymbol{z}] \in \mathbb{R}^{R \times ... \times R}$ of $\boldsymbol{z}$ containing just zeros apart from positions $z_{k,...,k}$ where it contains value z_k. This allows us to write eq. (10) as *n-mode product* (see [11] for further information about tensor calculation):

$$\boldsymbol{\mathcal{X}} \approx ((([\boldsymbol{z}] \times_1 \boldsymbol{U}^1) \times_2 \boldsymbol{U}^2) \times_3 \boldsymbol{U}^3) = [\boldsymbol{z}] \prod_{i=1}^{3} \times_i \boldsymbol{U}^i \tag{11}$$

Figure 4 visualizes the *n-mode product* for better understanding.

3.2 METAFAC - Metagraph Factorization

As mentioned before, the Metagraph is a description of a multi-relational graph G by means of a set of tensors $\{\boldsymbol{\mathcal{X}}^{(i)}\}$. The objective of the METAFAC algorithm is to derive tensor decompositions of the $\boldsymbol{\mathcal{X}}^{(i)}$ with shared factors $[z], U^{(q)}$ which closely resemble the $\boldsymbol{\mathcal{X}}^{(i)}$. To measure the approximation, [16] proposed the Kullback Leibler divergence D_{KL} [13], thus implying the following optimization problem:

$$\underset{[z],\{U^{(q)}\}}{\arg\min} \sum_{i=1}^{n} D_{KL}(\boldsymbol{\mathcal{X}}^{(i)}, [\boldsymbol{z}] \prod_{i_1,...,i_{l(i)}} \times_j \boldsymbol{U}^{(i_j)}) \tag{12}$$

To solve for (12) the authors derived an approximation scheme by defining

$$\boldsymbol{\mu}^{(i)} = \text{vec}(\boldsymbol{\mathcal{X}}^{(i)} \oslash ([\boldsymbol{z}] \prod_{j=1}^{l(i)} \times_j \boldsymbol{U}^{(i_j)})) \tag{13}$$

$$\boldsymbol{\mathcal{S}}^{(i)} = \text{fold}(\boldsymbol{\mu}^{(i)} * (\boldsymbol{z} * \boldsymbol{U}^{M_i} * \cdots * \boldsymbol{U}^{1_i})^T) \tag{14}$$

where $\oslash$ is the elementwise division of tensors, and $*$ is the Khatri-Rao product of matrices. These values are then be used to update $\boldsymbol{z}$ and the $\{\boldsymbol{U}^{(q)}\}$ iteratively using

$$\boldsymbol{z} = \frac{1}{n} \sum_{i=1}^{n} \text{acc}(\boldsymbol{\mathcal{S}}^{(i)}, M_i + 1) \tag{15}$$

$$\boldsymbol{U}^q = \sum_{l:e_l \sim v_q} \text{acc}(\boldsymbol{\mathcal{S}}^{(i)}, q, M_e + 1) \tag{16}$$

where acc is the accumulation-function of tensors and $M_i + 1$ is the last mode of tensor $\boldsymbol{\mathcal{S}}^{(i)}$. This update is carried out iteratively until the the sum in 4 converges. The optimization is shown in Algorithm 1.

Algorithm 1 MF algorithm

Input: Meta-Graph $G = (V, E)$, data tensors $\{\boldsymbol{\mathcal{X}}^{(e)}\}$
Output: z and $\{\boldsymbol{U}^q\}$

procedure METAFAC(G, $\{\boldsymbol{\mathcal{X}}^{(e)}\}$)
  Initialize z, $\{\boldsymbol{U}^q\}$
  repeat
    for all $e \in E$ **do**
      compute $\{\boldsymbol{\mathcal{S}}^{(e)}\}$ by eq. (13), (14)
      compute z by eq. (15)
    end for
    for all $q \in V$ **do**
      update $\{\boldsymbol{U}^q\}$ by eq. (16)
    end for
  until *convergence*
end procedure

The batched version of the METAFAC approximation can be derived by using the KL-divergence in equations (5),(6). An appropriate approximation scheme has been proposed by the following update function:

$$\boldsymbol{z} = (1 - \alpha) \sum_{i=1}^{n} \text{acc}(\boldsymbol{\mathcal{S}}^{(i)}, M_i + 1) + \alpha \boldsymbol{z}_{t-1} \tag{17}$$

$$\boldsymbol{U}^{(q)} = (1 - \alpha) \sum_{l:e_l \sim v_q} \text{acc}(\boldsymbol{\mathcal{S}}^{(j)}, q, M_i + 1) + \alpha \boldsymbol{U}^{(q)}_{t-1} \tag{18}$$

4 Stream-based Community Discovery with Tensors

In this section we present our adaptions of the METAFAC framework by introducing a sampling-based tensor representation of graphs and using time-stamped relations to induce a decrease of impact of relations to reflect the decreasing importance of *Twitter* messages.

Given a social network we are provided with a sequence M of messages

$$M := \langle m_0, m_1, \ldots \rangle$$

where each message m_i implies a set of relations $\mathcal{R}(m_i)$. Let $\tau(m_i) \geq 0$ be the arrival time of m_i. This results in an overall sequence of relations

$$S := \langle \mathcal{R}(m_0), \mathcal{R}(m_1), \ldots \rangle$$

which are continuously added to the evolving social network graph G. Hence we are faced with a sequence

$$\langle G_{t_0}, G_{t_1}, \ldots \rangle$$

of graphs where each G_{t_i} contains the relations of all messages up to time t_i.

Let t, t' be points in time with $t < t'$. In the following we will by $G_{[t,t']}$ denote the graph implied by only the messages of time-span $[t, t']$, hence $G_t = G_{[0,t]}$. Accordingly the graphs are represented by the corresponding tensor as

$$G_{[t,t']} \mapsto \left\{ \boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(n)} \right\}_{[t,t']}.$$

4.1 The MFSTREAM Algorithm

The METAFAC approach uses a sliding window of some fixed window size w_s to manage streams. Given a sequence of time-points t_j for $j \in \mathbb{N}$ with $t_j = t_{j-1} + w_s$, it factorizes $\{\boldsymbol{\mathcal{X}}^{(i)}\}_{[t_{j-1}, t_j]}$ based on a trade-off factor α as denoted in equation (5).

Our MFSTREAM algorithm interleaves the optimization of METAFAC by adding new relations during optimization and uses a time-based weighting function to take into account the relations' decreasing importance. Additionally, the optimization is carried out over only a partial set of relations as older relations tend to become obsolete for adjusting the model. We will present the time-based weighting and the sampling strategy in the following and present the complete algorithm in 4.4.

4.2 Time-based Relation Weighting

So far we considered the property of two or more entities to be related as binary property, i.e. if entities i, j and k are related, then

$$\boldsymbol{\mathcal{X}}_{i,j,k} = w,$$

with $w \in \{0, 1\}$. With the extraction of relations from time-stamped messages – as provided within the *Twitter* platform – we are interested in incorporating the age of these relations to reflect the decreasing up-to-dateness of the information.

Hence we associate each relation $r \in R_i$ with a timestamp $\tau(r)$ of the time at which this relation has been created (i.e. the time of the message from which it has been extracted). With S being a set of relations extracted from messages this leaves us with the tensor representation of relation type $R_i = V_{i_1} \times \ldots \times V_{i_{l(i)}}$ as

$$\boldsymbol{\mathcal{X}}^{(i)}_{i_1,\ldots,i_{l(i)}} = \begin{cases} \tau(r) & \text{if } r = (\varphi_1^{-1}(\nu_1), \ldots, \varphi_{l(i)}^{-1}(\nu_{l(i)})) \in S \\ 0 & \text{otherwise} \end{cases} \quad (19)$$

In addition to that, we introduce a *global clock*, denoted by $\tau_{\max}$, which represents the largest (i.e. the most recent) timestamp of all relations observed so far:

$$\tau_{\max} := \max \{ \tau(r) \mid r \in X \}.$$

Storing the timestamp $\tau(r)$ for each entry r in the tensors allows us to define a weighting function for the relations based on the global clock value. A simple example for a parametrized weighting function is given as

$$\omega_{\alpha,\beta}(r) := \frac{\alpha}{\alpha + \frac{1}{\beta}(\tau_{\max} - \tau(r))}. \quad (20)$$

4.3 Sampling

The runtime of each iteration of the approximation scheme is basically manifested by the maximum number N of non-zero entries in the tensors. To reduce the overall optimization time, we restrict the size of the tensors, i.e. number of entities of each type, by introducing constants $C_q \in \mathbb{N}$ and providing new entity mappings φ_q by

$$\varphi_q : V_q \to \overline{V}_q \text{ with } \overline{V}_q = \{1, \ldots, C_q\}.$$

This has two implications: Clearly, these φ_q will not be bijective anymore if $|V_q| > C_q$. Moreover, the size of the $\boldsymbol{U}^{(q)}$ matrices will also be limited to $C_q \times k$.

We deal with these imposed restrictions by defining dynamic entity mappings φ_q, which maps a new entity e (i.e. an entity that has not been mapped before) to the next free integer of $\{1, \ldots, C_q\}$. If no such element exists, we choose $f \in \{1, \ldots, C_q\}$ as the element that has longest been inactive, i.e. not been mapped to by φ_q. The relations affected by f are then removed from all tensors and the current cluster model, i.e. $\boldsymbol{U}^{(q)}_{f,i} = \frac{1}{k} \; \forall \; i = 1, \ldots, k$.

Effectively this introduces a "current window" $\{\boldsymbol{\mathcal{X}}^{(i)}\}$ of relations, that affect the adaption of the clustering in the next iteration. In contrast to the original METAFAC approach this also frees us from having to know the number of entities and a mapping of the entities beforehand.

4.4 Continuous Integration

With the prerequisites of section 4.2 and 4.3 we now present our stream adaption MFSTREAM as Algorithm 2. MFSTREAM is a purely dynamic approach of METAFAC which adds new relations to the data tensors $\{\boldsymbol{\mathcal{X}}^{(q)}\}$ and fits the model $[z], \{\boldsymbol{U}^{(q)}\}$ after a specified number of T messages. This differentiates our approach from METAFAC as the optimization is performed by running a single iteration of the optimization loop – with respect to the time-based weighting – after adding the relations of T messages to the tensors. The time complexity per iteration of the MFSTREAM algorithm is the same as for the METAFAC algorithms (see Section 3.2). Due to the fixed tensor dimensions, the maximum number of non-zero elements N is constant, which implies $O(1)$ runtime.

5 Evaluation

For the evaluation of our approach we extracted relations of the *Twitter* website. *Twitter* is a blogging platform giving users the opportunity to inform other users by very small snippets of text containing a maximum of 140 characters. In spite of such limitations *users* are not only posting messages – called *tweets* – but also enriching their *tweets* by *tags*, *urls* or *mentions*, which allows users to address other *users*. This brings up the entity types {*user*, *tweet*, *tag*, *url* }.

To discover clusters on the above mentioned entities present on the *Twitter* platform, we constructed a meta-graph for *Twitter* as shown in Figure 2. The entity types

Algorithm 2 The MFSTREAM algorithm.

Input: MetaGraph $G = (V, E)$, Stream $M = \langle m_i \rangle$, capacities C_q, number of clusters k, constant $T \in \mathbb{N}$
procedure MFSTREAM
  Initialize z, $\{\boldsymbol{U}^{(q)}\}$, $c := 0$
  while $M \neq \emptyset$ **do**
    $m := m_c$, $c := c + 1$ ▷ *Pick the next message from the stream*
    for all $(r_{j_1}, \ldots, r_{j_{l(j)}}) \in \mathcal{R}(m)$ **do**
      for all $p = 1, \ldots, l(j)$ **do**
        if $\varphi_p(r_{j_p}) = nil$ **then** ▷ *Replacement needed?*
          if $|\varphi_p| = C_q + 1$ **then**
            $f^* := \arg\min_{f \in \varphi_p} \tau(f)$
            $\boldsymbol{U}^{(p)}_{f^*,s} := \frac{1}{k} \; \forall \; s = 1, \ldots, k$
          else
            $f^* := \min_{f \in \{1,\ldots,C_p\}} \varphi_p^{-1}(f) = nil$ ▷ *Pick next unmapped f^**
          end if
          $\varphi_p(r_{j_p}) := f^*$, $\tau(f^*) := \tau(m)$
        end if
      end for
    end for
    $\nu_i := \varphi_p(r_{j_i})$ for $i = 1, \ldots, l(j)$
    $\boldsymbol{\mathcal{X}}^{(p)}_{\nu_1,\ldots,\nu_{l(j)}} := \tau(m)$ ▷ *Update corresponding tensor*
    if $c \equiv 0 \mod T$ **then** ▷ *Single opt.-iteration every T steps*
      for all $i \in \{1, \ldots, n\}$ **do**
        compute $\{\boldsymbol{\mathcal{S}}^{(i)}\}$ by eq. (14) and (13)
        update z by eq. (15)
      end for
      for all $j \in \{1, \ldots, q\}$ **do**
        update $\{\boldsymbol{U}^{(j)}\}$ by eq. (16)
      end for
    end if
  end while
end procedure

- R_1: a *user* writing a *tweet*.
- R_2: a *user* writing a *tweet* containing a special *tag*.
- R_3: a *user* writing a *tweet* containing a special *url*.
- R_4: a *user* mentioning another *user* in a written *tweet*.
- R_5: a *user* writes a *tweet* containing a *tag* and an *url*.
- R_6: a *user* writing a *tweet* containing an *url* and mentioning another *user*.
- R_7: a *user* writes a *tweet* containing a *tag* and a mentioned *user*.
- R_8: a *user* mentioning another *user* in a *tweet* containing a *tag* and an *url*.

themselves imply as much as $\mathcal{P}(V) = 2^4$ possible relation types, some of which will not arise or are redundant. E.g. since each *tweet* is written by a user, there is no relation (*tweet*,*tag*) which does not also refer to a *user*. Hence, our MetaGraph is based on the relation types $\{R_1, \ldots, R_8\}$ given as:

We extracted 1000 seed users and their direct *friends* and *followers*. *Followers* are following a *user* which means that messages of the *user* are directly visible for the *followers* at their *twitter* website. *Friends* are all the users a particular *user* is following. We used an English stopword filter to extract users which are writing in English language and processed all *friends* and *followers* of the seed users, revealing about 478.000 *users*. For these, we extracted all the messages written between the 19th and 23rd of February 2010. Out of these 2.274.000 *tweets* we used the *tweets* written at the 19th of February for our experiments, leaving about 389.000 *tweets* from 41.000 *users*.

5.1 Evaluating the Model

For a comparison of the clusterings produced by MFSTREAM and the METAFAC approaches we employ the "within cluster" point scatter [8]. This is given as

$$W(C) := \sum_{k=1}^{K} N_k \sum_{C(i)=k} \parallel x_i - \overline{x}_k \parallel^2 \qquad (21)$$

where K is the number of clusters, x_i is a member of a cluster $C(i)$ and $\overline{x}_k$ is the centroid of a cluster k. It can be seen as a sum of dissimilarities between elements in the particular clusters.

We created clusterings on a stream of 200k messages with MFSTREAM and restricted the tensor dimensions to $C_{user} = C_{tweet} = 5000$ and $C_{tag} = C_{url} = 1000$. We employed several weighting functions such as $\omega_{1,1}, \omega_{10,1000}$ and $\omega_{100,1000}$ as well as a binary weighting which equals the unweighted model (i.e. $w \in \{0, 1\}$).

To be able to compare the clusterings of MFSTREAM and METAFAC, we processed messages until the first entity type V_i reached its limit and stored the resulting clustering on disk. Then we reset the φ mappings and started anew, revealing a new clustering every time an entity type i reached C_i, revealing a total of 93 clusterings. We applied METAFAC on the messages that have been used to create these 93 clusterings and computed their similarities using $W(C)$. Figure 5 shows that MFSTREAM delivers results comparable to METAFAC for different weighting functions. Figure 7 shows that using timestamped values instead of binary values for calculation of the MFSTREAM delivers better results. The decrease of T, which implies a larger number of optimization steps, intuitively increases the quality

Weighting	$W(C)$ (mean)	std. deviation
METAFAC	$5.685 \cdot 10^7$	$1.32 \cdot 10^7$
binary	$7.511 \cdot 10^7$	$2.00 \cdot 10^7$
$\omega_{1,1}$	$6.142 \cdot 10^7$	$1.57 \cdot 10^7$
$\omega_{\mathbf{1,1000}}$	$\mathbf{6.002 \cdot 10^7}$	$\mathbf{1.38 \cdot 10^7}$
$\omega_{10,1000}$	$6.272 \cdot 10^7$	$1.43 \cdot 10^7$
$\omega_{100,1000}$	$6.724 \cdot 10^7$	$1.55 \cdot 10^7$

Figure 5: Mean $W(C)$ of different weights ($T = 20$), comparing MFSTREAM and METAFAC

T	$W(C)$ (mean)	std. deviation
5	$\mathbf{6.043 \cdot 10^7}$	$\mathbf{1.35 \cdot 10^7}$
10	$6.133 \cdot 10^7$	$1.36 \cdot 10^7$
50	$6.058 \cdot 10^7$	$1.40 \cdot 10^7$
100	$6.465 \cdot 10^7$	$1.49 \cdot 10^7$
250	$10.882 \cdot 10^7$	$3.13 \cdot 10^7$
500	$43.419 \cdot 10^7$	$11.47 \cdot 10^7$

Figure 6: Mean of $W(C)$ with different update steppings T (weight used: $\omega_{1,1000}$)

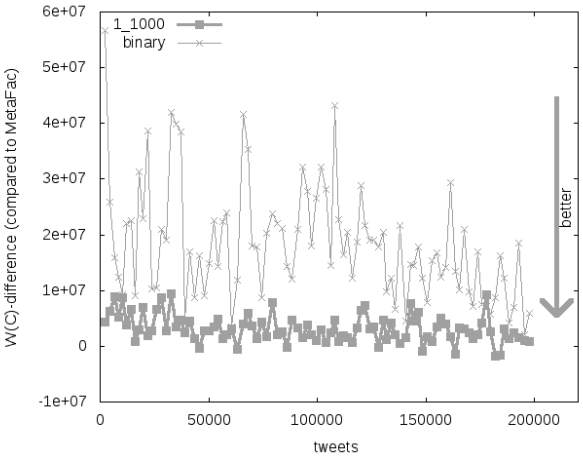

Figure 7: $W(C)$ for MFSTREAM compared to the METAFAC clusterings.

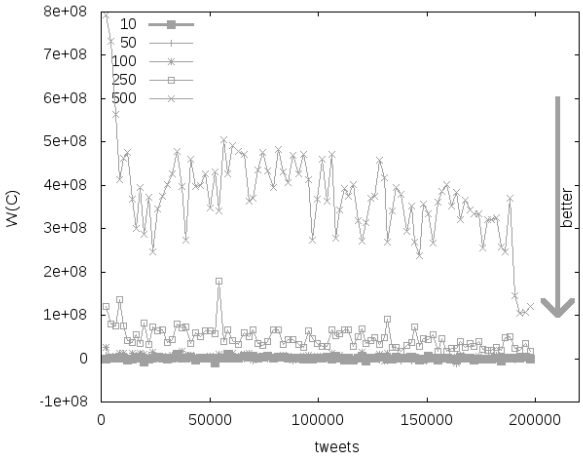

Figure 8: Relative $W(C)$ of MFSTREAM using different update step sizes T

of MFSTREAM as is attested by Figures 6 and 8.

In addition, we made experiments to show the effect of the update frequency T. Figure 9 shows the relative runtime of MFSTREAM where $T = 1$ corresponds to the baseline at 1.0. Raising T results in shorter runtime, since the model is updated less frequently, which is the major time factor. The upper curve shows the runtime for updating after 5 relations ($T = 5$), the middle one shows $T = 10$, and the latter refers to $T = 50$.

Varying sizes of entity types by the C_q results in clusterings of different numbers of entities, which cannot be directly compared by $W(C)$. Hence, we normalized $W(C)$ by the variance $\mathcal{V}$ of each clustering. Larger models of course incorporate more information, which results in more stable clusterings as can be seen in Figure 10.

6 Conclusion and Future Work

In this work we presented MFSTREAM, a flexible algorithm for clustering multi-relational data from evolving networks, derived from the METAFAC framework by [16]. The main improvement of our approach is the reduction of the approximation scheme on to a small relevant window of relations. The proposed time-based weighting of relations contributes to this reduction by removing obsolete information that is not relevant to the model adaption anymore.

MFSTREAM is able to handle relations containing new, unseen entities by offering a replacement strategy for the set of entities considered at optimization time. This makes it especially suitable to continuously integrate new data from a stream. We evaluated MFSTREAM on real-world data crawled from the *Twitter* platform and showed its comparability to METAFAC.

The use of backend storage for off-loading obsolete data that can be re-imported into the optimization window at a later stage might be an interesting advancement. Also, concurrent criteria *runtime* and *quality* offer a starting point for multi-objective optimization. Additionally, recent works [15] motivate further improvements to handle a dynamic number k of clusters within MFSTREAM.

References

[1] E. Acar, S. A. Çamtepe, M. S. Krishnamoorthy, and B. Yener. Modeling and multiway analysis of chatroom tensors. In *ISI*, pages 256–268, 2005.

[2] E. Acar, S. A. Çamtepe, and B. Yener. Collective sampling and analysis of high order tensors for chatroom communications. In *ISI*, pages 213–224, 2006.

[3] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[4] B. Bader, M. W. Berry, and M. Browne. *Survey of Text Mining II*, chapter Discussion tracking in Enron email using PARAFAC, pages 147–163. Springer, 2007.

[5] A. Banerjee, S. Basu, and S. Merugu. Multi-way clustering on relation graphs. In *SDM*. SIAM, 2007.

[6] D. Cai, X. He, and J. Han. Tensor space model for document analysis. In E. N. Efthimiadis, S. T. Dumais, D. Hawking, and K. Järvelin, editors, *SIGIR*, pages 625–626. ACM, 2006.

[7] R. Harshman. Foundations of the parafac procedure: Models and conditions for an "explanatory" multi-

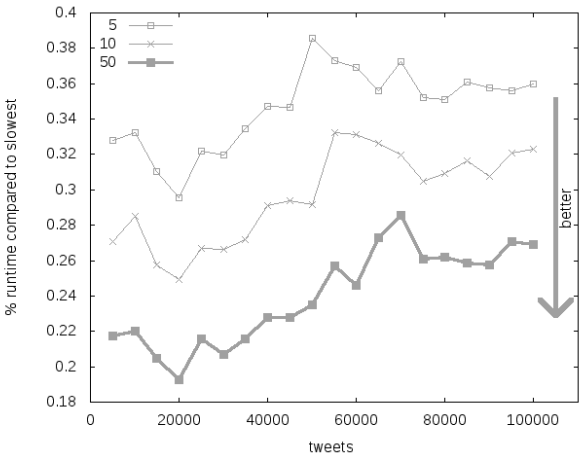

Figure 9: Relative runtime of MFSTREAM using different numbers of relations for update.

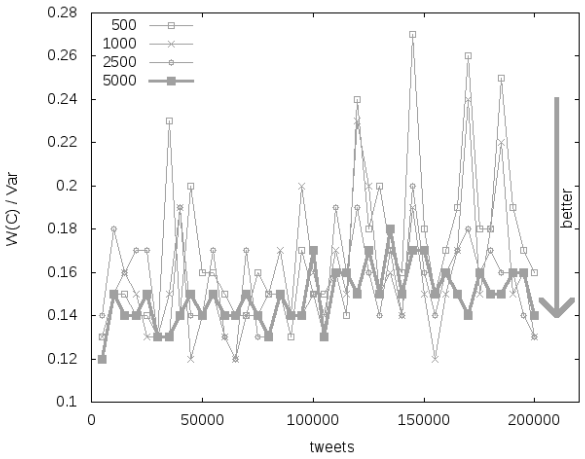

Figure 10: $W(C)/\mathcal{V}$ of MFSTREAM using different sizes of models.

modal factor analysis. *UCLA Working Papers in Phonetics*, 16, 1970.

[8] T. Hastie, R. Tibshirani, and F. J. The elements of statistical learning-data mining, inference and prediction. *Springer, Berlin Heidelberg New York*, 2001.

[9] H. Kiers. Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3):105–122, 2000.

[10] E. Kilmer and C. D. Moravitz Martin. Decomposing a tensor. *SIAM News*, 37(9), 2004.

[11] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September 2009.

[12] T. G. Kolda, B. W. Bader, and J. P. Kenny. Higher-order web link analysis using multilinear algebra. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 242–249, 2005.

[13] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[14] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278, 2000.

[15] Y.-R. Lin, J. Sun, N. Cao, and S. Liu. Contextour: Contextual contour visual analysis on dynamic multi-relational clustering. In *Proceedings of the SIAM Conference on Data Mining (SDM10)*, 2010. Not published, yet.

[16] Y.-R. Lin, J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher. Metafac: Community discovery via relational hypergraph factorization. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge discovery and data mining (SIGKDD 2009)*, pages 527–536, Paris, France, 2009. ACM.

[17] A. Shashua and T. Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 792–799, New York, NY, USA, 2005. ACM.

[18] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383, New York, NY, USA, 2006. ACM.

[19] X. Wang, J.-T. Sun, Z. Chen, and C. Zhai. Latent semantic analysis for multiple-type interrelated data objects. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 236–243, New York, NY, USA, 2006. ACM.